La investigación reportada en esta tesis es parte de los programas de investigación del CICESE (Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California).
La investigación fue financiada por el CONAHCYT (Consejo Nacional de Humanidades, Ciencias y Tecnologías).

Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias
en Ciencias de la Computación

Clasificación de anormalidades en mastografías utilizando ensambles

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de

Maestro en Ciencias

Presenta:

Luis Guillermo Rodríguez López

Ensenada, Baja California, México

2024

Tesis defendida por

Luis Guillermo Rodríguez López

y aprobada por el siguiente Comité

Dr. Vitali Kober
Codirector de tesis

Dr. José Angel González Fraga
Codirector de tesis

Dr. Pedro Gilberto López Mariscal

Dr. César Cruz Hernández.

Dr. Pedro Gilberto López Mariscal
Coordinador del Posgrado en Ciencias de la Computación

Dra. Ana Denise Re Araujo
Directora de Estudios de Posgrado

Resumen de la tesis que presenta Luis Guillermo Rodríguez López como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Clasificación de anormalidades en mastografías utilizando ensambles

Resumen aprobado por:

Dr. Vitali Kober
Codirector de tesis

Dr. José Angel González Fraga
Codirector de tesis

El cáncer de mama es el cáncer más común en las mujeres y una de las principales causas de morbilidad y mortalidad, lo que lo convierte en un problema de salud con importancia mundial. La mamografía es una técnica de diagnóstico por imágenes altamente estandarizado para los programas de detección temprana del cáncer de mama, sin embargo, al día de hoy, la estimación de la densidad mamaria, la clasificación de calcificaciones y masas (clasificación de anormalidades), tanto en benignas como malignas (patologías) con evaluación visual sigue siendo un desafío debido al tejido adiposo de las mamas, por lo que se han creado distintas maneras de abordar este problema utilizando aprendizaje máquina. Cabe mencionar que en este estudio se empleó exclusivamente la base de datos CBIS-DDSM para realizar experimentos que incluyen la clasificación de anormalidades y la detección temprana del cáncer de mama, siendo este último la más relevante. Para ello, se propone el uso de métodos de aprendizaje máquina junto con aprendizaje profundo para mejorar la tasa de clasificación de modelos entrenados individualmente. Entre las técnicas exploradas se encuentran el aprendizaje por transferencia (*transfer learning*), el ajuste fino (*fine tuning*), el uso de redes neuronales convolucionales como extractores de características y como clasificador final, el uso de métodos de ensamble y la exploración de modelos híbridos. Como resultado, se obtuvo un modelo con un *accuraccy* del 75.95 % en la detección temprana del cáncer de mama, sin embargo, se seleccionó otro con un *accuracy* del 75.36 %, pero con una especificidad (precisión) del 81.97 %, convirtiéndose en un modelo más confiable.

Palabras clave: aprendizaje por ensamble, aprendizaje profundo, aprendizaje máquina, anormalidades de una mastografía

Abstract of the thesis presented by Luis Guillermo Rodríguez López as a partial requirement to obtain the Master of Science degree in Name of the Degree.

Classification of Abnormalities in Mammograms Using Ensembles

Abstract approved by:

Dr. Vitali Kober
Thesis Co-Director

Dr. José Angel González Fraga
Thesis Co-Director

Breast cancer is the most common cancer among women and one of the leading causes of morbidity and mortality, making it a global health concern. Mammography is a highly standardized imaging diagnostic technique used in early breast cancer detection programs. However, the visual assessment of breast density, classification of calcifications and masses (abnormality classification), both benign and malignant (pathologies), remains challenging due to the adipose tissue in the breasts. To address this issue, various approaches have been developed using machine learning. In this study, the CBIS-DDSM database was exclusively employed to conduct experiments focused on abnormality classification and early breast cancer detection. The proposed methodology integrates machine learning and deep learning techniques to enhance the classification performance of individually trained models. The explored techniques include transfer learning, fine-tuning, the use of convolutional neural networks as feature extractors and final classifiers, ensemble methods, and hybrid model exploration. As a result, a model achieving an accuracy of 75.95 % in early breast cancer detection was obtained. However, a different model, with a slightly lower accuracy of 75.36 %, was selected due to its higher specificity (precision) of 81.97 %, making it a more reliable option.

Keywords: ensemble learning, deep learning, machine learning, mammogram abnormalities.

Dedicatoria

A mi esposa, a mis padres y a todas las personas que me apoyaron y creyeron en mí, y que además no hace falta nombrarlas porque saben que les estoy agradecido por todo su apoyo, cariño y comprensión.

Agradecimientos

Agradezco al Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California (CICESE) por haberme aceptado y proporcionado un espacio para completar mi educación en el estudio de un posgrado en ciencias, el cual me llevó a realizar este trabajo de investigación.

Me permito a la vez extender mi gratitud al Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT) por brindarme el apoyo económico con el cual me fue posible realizar mis estudios de maestría. Asimismo, me permito agradecer a la Universidad Autónoma de Baja California (UABC) por haberme prestado un espacio y equipo computacional con el cual pude avanzar de manera significativa a lo largo de la experimentación realizada.

No puedo no nombrar a las personas preparadas en el área que me permitieron lograr alcanzar los objetivos en este trabajo de investigación, por lo que les agradezco a mis asesores de tesis, el Dr. José Angel González Fraga y al Dr. Vitali Kober. Más aún, me permito agradecer al Dr. Pedro Gilberto López Mariscal y al Dr.César Cruz Hernández por haber sido parte de mi comité y haber realizado la evaluación y revisión de este trabajo de investigación.

Tabla de contenido

Página

Lista de figuras

Figura Página

Figura Página

Figura Página

Figura Página

Figura Página

Figura Página

Figura Página

Lista de tablas

Tabla Página

Tabla Página

Capítulo 1. Introducción

El cáncer o la palabra cáncer, es un término que se utiliza para designar un amplio compendio de enfermedades capaces de afectar a cualquier parte de nuestro organismo. Esta enfermedad tiene que ver con la multiplicación rápida de células anormales, las cuales pueden llegar a extenderse más allá de su lugar de origen, invadiendo así partes adyacentes del cuerpo u otros órganos, tal proceso es llamado metástasis y es la principal causa de muerte a nivel mundial, siendo los de mama, pulmón, colon y recto y próstata los más comunes (Organización Mundial de la Salud., 2022).

Existen muchos tipos de cáncer y, como se sabe, este se puede desarrollar en cualquier parte de nuestro organismo, denominándose así según la parte del cuerpo en el que se origine y sin importar si ya ha ocurrido metástasis, es decir, cáncer de seno seguirá denominándose cáncer de seno sin importar si ya se esparció a otra parte del cuerpo. Esta información junto con la que se provee a continuación es por parte de American Cancer Society. (2020), en la cual se puede encontrar la distinción de dos categorías principales del cáncer.

- Cánceres hematológicos, los cuales son tipos de cáncer que se originan en los glóbulos sanguíneos, ejemplos de estos son leucemia, linfoma y mieloma múltiple.
- Los cánceres de tumor sólido son los que se llegan a desarrollar en cualquier órgano, tejido o parte del cuerpo, siendo los más comunes el de seno, próstata, pulmón y colorrectal.

Para ser más precisos, un tumor consiste en una masa o protuberancia (crecimiento), distinguiéndose entre los que son cancerosos y los que no lo son. Los tumores benignos son masas que no son cancerosas y no se propagan, mientras que los tumores malignos son aquellas masas que contienen células cancerosas.

Las causas que provocan cáncer en general son muy diversas y complejas, ya que las células cancerosas se originan a raíz de cambios en la genética de las células. El estilo de vida, herencia genética, exposición a ciertos agentes cancerígenos en el entorno, son algunas de las causas posible y en muchos casos no hay causa de apariencia atribuible (American Cancer Society., 2020).

1.1. Motivación

El cáncer más frecuente en las mujeres es el de mama y una de las principales causas de morbilidad y mortalidad, lo que lo convierte en un problema de salud con importancia mundial. El cáncer mamario,

de mama o también referido como cáncer de seno, es un tipo de cáncer que se origina y desarrolla en esta parte del cuerpo, ya sea en uno o en ambos senos. La American Cancer Society. (2021) considera importante los siguientes aspectos para la detección temprana del cáncer.

- Existen afecciones no cancerosas de seno.
- El cáncer de seno puede ocurrir también en hombres.
- Cualquier protuberancia, abultamiento o cambio en los senos, debe ser examinado por un profesional de la salud.
- La mayoría de las masas en los senos son benignas, las cuáles no se propagan más allá de los senos, aunque algunos bultos pueden aumentar el riesgo de cáncer.

El seno es un órgano que se compone principalmente por glándulas, conductos y tejido adiposo, en donde la cantidad de tejido graso determina el tamaño de cada seno (Figura 1). El seno está conformado por:

- Lobulillos: glándulas productoras de leche.
- Conductos: estos salen de los lobulillos y son canales por donde sale la leche.
- Pezón: es una abertura en la piel del seno donde los conductos se juntan, haciéndose más grandes y es por donde la leche sale del cuerpo. El pezón está rodeado por piel más oscura y gruesa, llamada areola.
- Los tejidos conectivos (estroma) y adiposos (grasa) rodean los conductos y lobulillos y el fin de estos es de mantenerlos en su lugar.
- Vasos sanguíneos y linfáticos.

Es importante mencionar que, en cada uno de estos conductos, glándulas o tejidos, se puede desarrollar cáncer, siendo unos más comunes que otros.

El identificar causas de cáncer en general es un trabajo difícil de hacer, sin embargo, con respecto al cáncer de mama, se pueden identificar ciertos patrones que indiquen la presencia de algún tumor maligno presente en alguno de los senos, los cuales se describen en breve.

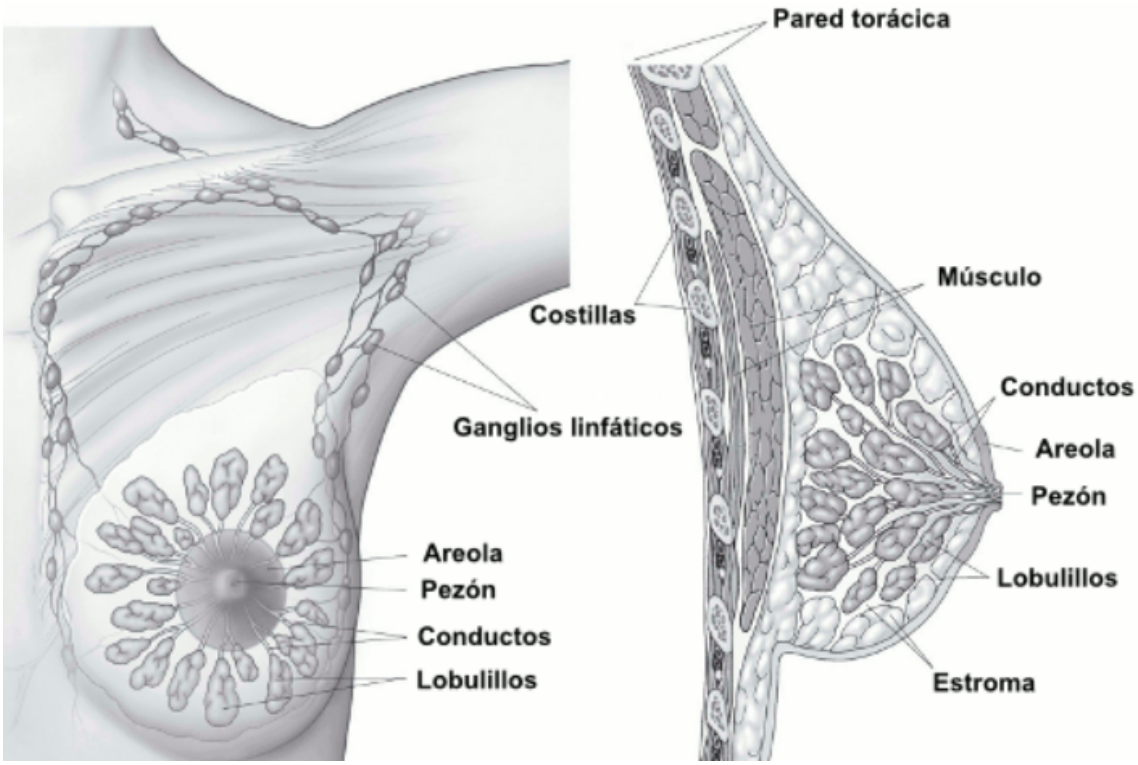


Figura 1. Tejido mamario normal. Imagen modificada de American Cancer Society. (2021).

1.2. Antecedentes

Al igual que en otros tipos de cáncer, el estado en el que se encuentre es el factor más importante para determinar el tratamiento y pronóstico, por lo que se necesitan métodos para detectar el cáncer en etapas tempranas, y en este sentido, la mastografía (mamografía) ha sido establecida como el mejor método para la detección del cáncer de mama en fases iniciales (Mračko et al., 2023). Se ha comprobado que su uso reduce la mortalidad, convirtiéndose en una 'regla de oro' (Drukteinis et al., 2013) para la evaluación por imágenes, ya que también existen ultrasonidos e imágenes por resonancia magnética de senos (MRI, por sus siglas en inglés).

La mastografía es una imagen de seno obtenido mediante rayos X. Aunque anteriormente el método usado era el de película de pantalla *screen-film*, actualmente la mastografía digital la ha reemplazado, convirtiéndose en el único método aprobado para llevar a cabo este estudio (Vinnicombe et al., 2009). En México se recomienda realizarlo anualmente a mujeres de entre 40 a 69 años, siendo su principal función el detectar anormalidades en las mamas (*breast abnormalities*), las cuales no son posibles detectar por palpación (Instituto Mexicano del Seguro Social, 2024).

Las anormalidades, mejor descritas como hallazgos, que se pueden encontrar en una mastografía son las siguientes, las cuales se pueden ver en la Figura 2:

- Distorciones arquitectónicas (*Architectural distortion*).
- Masas (*Mass*).
- Microcalcificaciones (*Microcalcifications*).
- Asimetrías (*Asymmetries*).

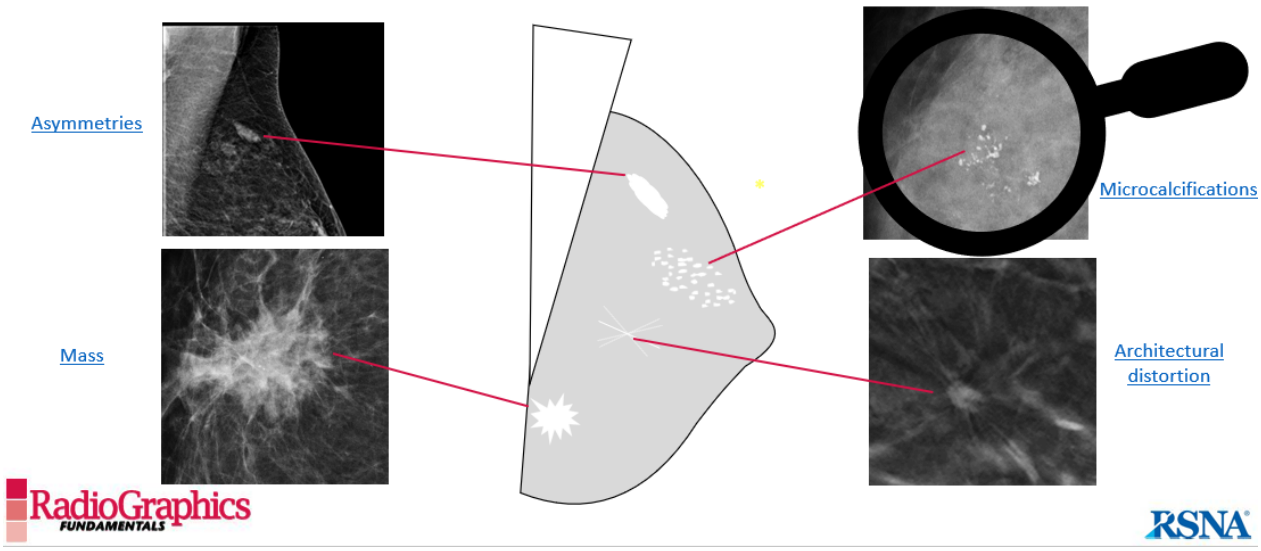


Figura 2. Posibles hallazgos ('anormalidades') en mastografías. Imagen modificada de Pesce et al. (2019)

.

La parte más importante de esto es que los hallazgos no tienen por qué ser anormales, es decir, una mujer puede presentar alguno o varios tipos de hallazgos y estar completamente sana, sin embargo, la distribución, patrones y/o el tamaño de estos suelen indicar la presencia o ausencia de algún tumor, por lo tanto, el médico que consulta al paciente debe de conocer lo que el médico radiólogo pudo observar en la mastografía para que pueda realizar un diagnóstico, de ahí es que surge un sistema estándar llamado BI-RADS, del inglés *Breast Imaging Reporting and Data System* (Tabla 1), el cual es un lenguaje de comunicación entre ambos doctores que se utiliza para medir qué tan probable es que un paciente pueda padecer cáncer (en base a estos hallazgos) y así poder llevarse a cabo un diagnóstico (American Cancer Society., 2019).

Hoy en día se sabe que la distribución, patrones y/o el tamaño de estos hallazgos, así como una alta

densidad mamaria son causas probables de cáncer, por lo que se pueden entrenar algoritmos de aprendizaje máquina y/o aprendizaje profundo para poder detectar o clasificar, ya sea el nivel de densidad mamaria presentes en las mamas, el BI-RADS al que pertenecen las mamas, así como sus anormalidades (masas y calcificaciones) y estas a su vez en su patología (benigna o maligna), entre otras.

Tabla 1. Significado de las categorías BI-RADS[1]

.

Categoría	**Recomendación**	**Probabilidad de malignidad**
0 Poco concluyente	Imágenes adicionales o comparación con exámenes anteriores	No determinada
1 Negativo	Mantener seguimiento	0 %
2 Benigno	Mantener seguimiento	0 %
3 Probablemente Benigno	Mantener seguimiento Repetir estudio en 6 meses	0 % $<$ entre $\leq$ 2 %
4. Sospecha 4A. Baja 4B. Moderada 4C. Alta	Biopsia	de 2 % a 95 % 2 % $<$ entre $\leq$ 10 % 10 % $<$ entre $\leq$ 50 % 50 % $<$ entre $<$ 95 %
5. Muy sospechoso	Biopsia	$\leq$ 95 %
6. Malignidad comprobada	Tratamiento adecuado	Diagnóstico establecido

1.3. Objetivos

En Lee et al. (2017) se menciona que pocos resultados publicados en la literatura pueden ser directamente reproducibles, ya que algunos conjuntos de datos utilizados en la experimentación son privados y aunque esta información data del año 2016-2017, lo cierto es que, hasta la fecha, lo siguen siendo, ya que muchos de estos contienen descripciones inadecuadas o incompletas y con detalles en el preprocesamiento de las imágenes no tan específicas. Además, no siempre existe código de programación compartido que pueda ayudar a la reproducibilidad de los resultados publicados. A pesar de esto, es posible experimentar con el aprendizaje profundo al utilizar algunos parámetros que se pueden encontrar en la literatura, por lo que

[1]La tabla fue modificada de: `https://www.cancer.org/es/cancer/tipos/cancer-de-seno/pruebas-de-deteccion-y-deteccion-temprana-del-cancer-de-seno/mamogramas/como-entender-su-informe-de-mamograma.html`.

se propone el uso combinado de metodologías del aprendizaje máquina para obtener mejores resultados.

1.3.1. Objetivo general

Desarrollar un algoritmo capaz de obtener una tasa de clasificación de anormalidades de una mastografía lo más alta posible con el uso de aprendizaje por ensamble.

1.3.2. Objetivos específicos

Para poder llegar a crear este algoritmo se propone:

1. Estudiar y comparar las técnicas de preprocesamiento que sean de mayor utilidad para el entrenamiento de los algoritmos de aprendizaje profundo.
2. Usar modelos de aprendizaje profundo como extractores de características y evaluarlos con diferentes modelos de aprendizaje máquina como clasificadores.
3. Definir los experimentos con base en las anormalidades y aplicar técnicas de aprendizaje por ensamble.
4. Evaluar y comparar los modelos de aprendizaje entrenados.

1.4. Estructura de la tesis

La estructura de esta tesis es la siguiente. En el capítulo 2 se describen la teoría y fundamentos teóricos detrás de los clasificadores utilizados en el aprendizaje profundo y aprendizaje máquina.

En el capítulo 3 se describe la base de datos utilizada en los experimento, así como la importancia, significado y utilidad del Aprendizaje por Transferencia (*Transfer Learning*), Ajuste Fino (*Fine Tuning*) y el Generador de imágenes (*Image Data Generator*) con respecto al uso de datos.

En el capítulo 4 se presentan los experimentos realizados y los resultados obtenidos.

En el capítulo 5 se presentan las conclusiones generales del trabajo realizado, además, se discutirán las limitaciones encontradas, así como el trabajo futuro relacionado con este tema de tesis.

Capítulo 2. Fundamentos

Cuando se habla de aprendizaje profundo o aprendizaje máquina, por lo regular, suelen aparecer conceptos como perceptrón, redes neuronales, redes neuronales convolucionales, inteligencia artificial, entre otros, sin embargo, "no hay un consenso entre los científicos e ingenieros sobre lo que es la Inteligencia Artificial, y mucho menos se ha llegado a una definición exacta y concisa que nos permita dirimir qué programas son o no inteligentes" (García, 2017, p.1), mientras que en Russell & Norvig (2010) se menciona que un agente inteligente toma la mejor acción posible en una situación.

En este capítulo se abordan la teoría y conceptos fundamentales que respaldan los experimentos realizados. Cabe mencionar que no se profundiza tan ampliamente en estos temas, abordando principalmente las redes neuronales convolucionales y algunos algoritmos (clasificadores) de aprendizaje máquina.

2.1. Aprendizaje Máquina

Géron (2019) menciona que el aprendizaje máquina es la ciencia y/o arte que les permite a las computadoras 'aprender' de un conjunto de datos conocido como conjunto de entrenamiento o *training set*. También menciona que un modelo de aprendizaje máquina es un programa, modelo matemático o una representación matemática que ha sido 'entrenado' (*training*) con algún conjunto de datos o *dataset* para 'reconocer' ciertos tipos de patrones o características. Cabe mencionar que cada ejemplo proporcionado al modelo es llamado muestra o instancia.

Se pueden destacar principalmente tres tipos de aprendizaje:

- **Aprendizaje supervisado.** En este tipo de aprendizaje se proporciona el conjunto de entrenamiento (entrada) con sus soluciones (salidas). El objetivo es lograr que el modelo 'aprenda' del conjunto de entrenamiento para asignar salidas a nuevos datos de entrada. Si el problema que se ataca es de clasificación, entonces estas salidas son conocidas como etiquetas.
- **Aprendizaje no supervisado.** En este tipo de aprendizaje el conjunto de entrenamiento no es proporcionado con sus soluciones esperadas, de hecho la idea detrás de este aprendizaje es asignar dichas salidas. Este es menos común, pero de gran utilidad dependiendo su uso.
- **Aprendizaje por refuerzo.** Aquí el entrenamiento se lleva a cabo por medio de prueba y error hasta determinar la mejor manera de solucionar cierta tarea.

Cabe decir que existen otros tipos de aprendizaje, sin embargo, los anteriormente descritos son los más conocidos, mientras que el aprendizaje profundo es un subconjunto del aprendizaje máquina.

La Figura 3 muestra que el aprendizaje profundo es un subcampo del aprendizaje máquina y éste a su vez es un subcampo de la inteligencia artificial. Más aún, los tipos de aprendizaje supervisado y no supervisado se pueden subdividir en clasificación y regresión (lo cual se puede ver en la Figura 4), diferenciándose en que los datos de entrada en tareas de clasificación (etiquetas) se tratan como valores (pueden ser vectores) discretos (números enteros), mientras que los datos de entrada del segundo bien pueden representar valores 'continuos'.

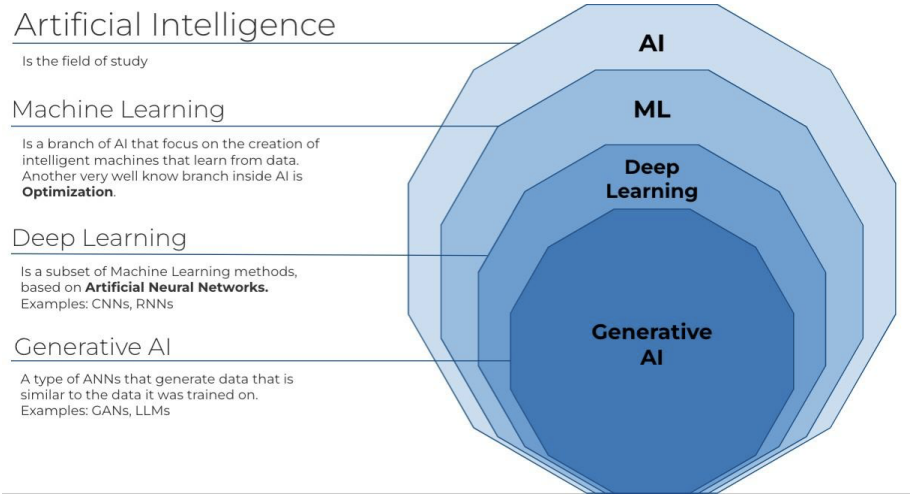


Figura 3. Subcampos de la Inteligencia Artificial, abreviado en inglés por **AI**. El Aprendizaje Máquina, abreviado en inglés por **ML**, es una de sus ramas principales y se enfoca en el estudio de modelos que aprenden a partir de los datos. Un subcampo de este es el Aprendizaje Profundo ó **Deep Learning**, abreviado en inglés por DL, el cual se basa en las redes neuronales artificiales. Actualmente otro subcampo que surgió es la Inteligencia Artificial Generativa (**Generative AI**), la cual se enfoca en la creación de contenido nuevo, como texto, imágenes, música o incluso código, a partir de modelos entrenados en grandes cantidades de datos. Imagen modificada de internet.

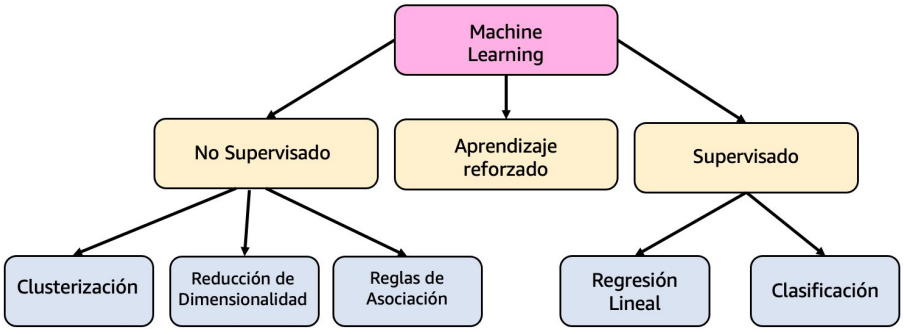


Figura 4. Subdivisión de los tipos de aprendizaje más comunes. Imagen modificada de internet.

2.1.1. Características

Como muestra la Figura4, hay dos tipos principales de algoritmos pertenecientes al aprendizaje supervisado: **regresores** y **clasificadores**. Ya sea que se quiera implementar alguno de estos mediante aprendizaje máquina o aprendizaje profundo, por lo regular, se necesita presentar los datos de entrenamiento en forma matricial. Suponga que se tienen m datos de entrenamiento (vectores renglón), llamados instancias, con sus respectivas etiquetas o valores (*targets*), y suponga que cada instancia tiene n valores, llamados características o descriptores, entonces cada instancia se puede ver como un vector fila de longitud m, de tal forma que, al 'apilar' cada una de estas, es posible crear una matriz (matriz de características) de tamaño $m \times n$.

Para ilustrar el concepto anterior, suponga m datos de entrenamiento con n descriptores (vectores renglón) $x_{11} = [a_{11}, a_{12}, ..., a_{1p}, ..., a_{1n}]$, $x_{22} = [a_{21}, a_{22}, ..., a_{2p}, ..., a_{2n}]$, ..., $x_{pp} = [a_{p1}, a_{p2}, ..., a_{pp}, ..., a_{pn}]$, ..., $x_{mm} = [a_{m1}, a_{m2}, ..., a_{mp}, ..., a_{mn}]$, entonces la matriz 1 contiene todas las instancias con sus características.

$$\begin{bmatrix} \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1p} & \cdots & a_{1n} \end{pmatrix} \\ \begin{pmatrix} a_{21} & a_{22} & \cdots & a_{2p} & \cdots & a_{2n} \end{pmatrix} \\ \vdots \\ \begin{pmatrix} a_{p1} & a_{p2} & \cdots & a_{pp} & \cdots & a_{pn} \end{pmatrix} \\ \vdots \\ \begin{pmatrix} a_{m1} & a_{m2} & \cdots & a_{mp} & \cdots & a_{mn} \end{pmatrix} \end{bmatrix}, \tag{1}$$

donde sus *targets* asociados se pueden representar como un vector m-dimensional:

$$\begin{pmatrix} (y_1) \\ (y_2) \\ \vdots \\ (y_p) \\ \vdots \\ (y_m) \end{pmatrix}. \tag{2}$$

Hay que recalcar que el concepto principal involucrado para crear un clasificador es el de características

(*features*). Por ejemplo, si se tuvieran m instancias con n características cada una y se estuviera buscando crear un clasificador de transportes públicos, al tomarse dos instancias x_1 y x_2 con sus correspondientes *targets* (distintos) $y_1 =$ carro y $y_2 =$ barco, entonces, la p-ésima característica ($1 \leq p \leq n$) representa, válgase la redundancia, una característica que comparten ambos transportes, como lo pueden ser la longitud, altitud, volumen, color (entre otros) y si se tomaran solo estas cuatro características, entonces la matriz de entrada sería de tamaño: número de instancias $\times$ número de características, es decir, $m \times 4$.

2.2. Redes Neuronales

Las redes neuronales es un paradigma de programación que permite a una computadora aprender, cuya inspiración principal se centra en la biología, específicamente en el cerebro humano, mientras que el aprendizaje profundo es un potente conjunto de técnicas que ayudan a realizar el aprendizaje en las redes neuronales (Nielsen, 2015).

2.2.1. El Perceptrón

El concepto fundamental y básico de las redes neuronales es el perceptrón, desarrollado por el científico Frank Rosenblatt en el año de 1958. Las redes neuronales artificiales "son un intento de emular la forma de trabajar del cerebro humano, y aunque estamos lejos de alcanzar su misma capacidad" (García, 2017, p.208), hoy en día sus aplicaciones han logrado importantes avances en la ciencia e ingeniería. El cerebro humano está conformado por millones de neuronas conectadas entre sí (Figura 5), por lo que en el perceptrón (Figura 6), al igual que en una neurona cerebral, llegarán señales de entrada (simulando las dendritas de una neurona) para posteriormente producir una salida.

En la Figura 6 se puede ver el esquema de una neurona artificial (perceptrón), donde sus entradas son n descriptores $x_1, x_2, ..., x_n$, sus respectivos pesos son $w_1, w_2, ...w_n$ y θ es un número real, a veces llamado umbral de disparo.

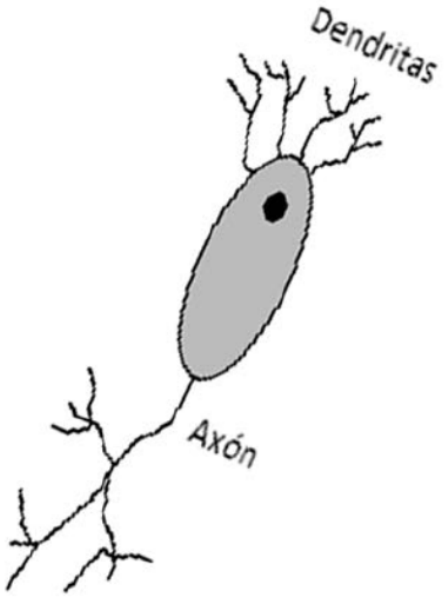


Figura 5. Figura adaptada de García (2017), en la cual él describe que "la neurona se activa y envía señales eléctricas a otras neuronas a través de los axones. Es decir, funcionan como una auténtica red con billones de conexiones que trabajan de forma paralela" (p.209).

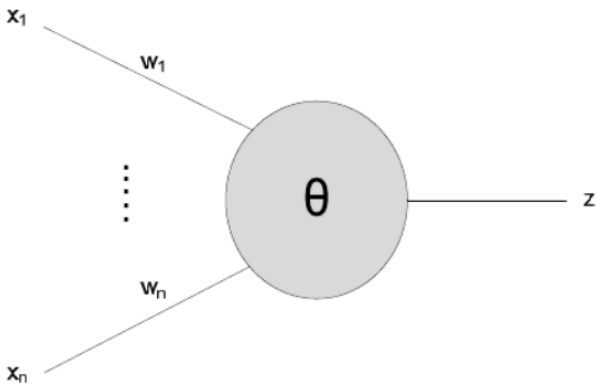


Figura 6. Se muestra un perceptrón. Figura adaptada de García (2017), en la cual él describe que este es un "modelo simple de neurona que nos va a permitir presentar los conceptos básicos para luego ir profundizando en modelos más complejos" (p. 210).

El ejemplo de la ecuación 3 muestra un perceptrón con n descriptores y sus correspondientes n pesos, un umbral θ y valores de salida (*targets*) que pueden ser 0 o 1. Si el valor de z es mayor que el umbral θ, entonces esta neurona se 'dispara', es decir, el valor *output* se fija a cualquiera de los *targets*.

$$output = \begin{cases} 0 & si \quad \sum_{j=1}^{n} x_j w_j \leq \theta \\ 1 & si \quad \sum_{j=1}^{n} x_j w_j > \theta \end{cases} . \tag{3}$$

La ecuación más comúnmente usada se obtiene al pasar restando el umbral θ al otro lado de la desigualdad, de manera que la ecuación 3 se reescribe como:

$$output = \begin{cases} 0 & si \quad \sum_{j=1}^{n} x_j w_j + b \leq 0 \\ 1 & si \quad \sum_{j=1}^{n} x_j w_j + b > 0 \end{cases}, \tag{4}$$

donde $b \equiv -\theta$ es conocido como el sesgo del perceptrón (Figura 7) y puede interpretarse como la facilidad de que la salida sea 1.

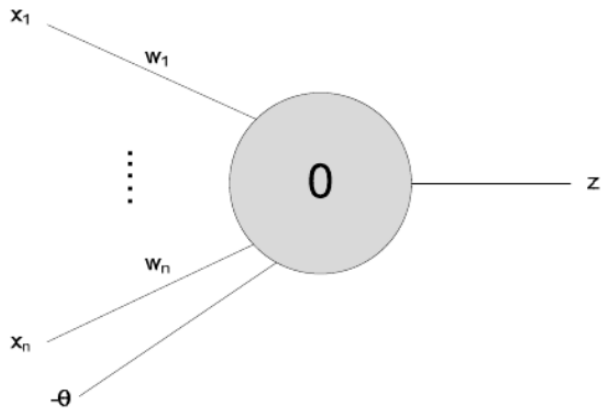


Figura 7. Se muestra el esquema alternativo de una neurona artificial. Figura adaptada de García (2017).

El objetivo es ajustar los pesos y el sesgo para que se pueda llevar a cabo una buena tarea de clasificación o regresión, sin embargo, *a priori* no es posible conocerlos, por lo que estos son inicializados de manera aleatoria y la forma de ir ajustándolos es llevada a cabo mediante un ciclo (*loop*) de entrenamiento, para lo cual se utiliza el conjunto de entrenamiento junto con sus *targets* o etiquetas (aprendizaje supervisado).

En el capítulo siete del libro de García (2017) se presenta un ejemplo en el que se utiliza un perceptrón para clasificar clientes de un banco de tal forma que estos sean aptos para que se les conceda o no una hipoteca dependiendo principalmente de dos factores: $x_1 =$ Sueldo bruto y $x_2 =$ Nivel de deuda. Tras haber hecho el entrenamiento con muchos casos previos, se logran obtener los siguientes parámetros de configuración de la neurona: $w_1 = -0.2, w_2 = 0.5$ y $\theta = 0.6$, con lo cual, sustituyendo en la ecuación 4, se obtiene $z = -0.2x_1 + 0.5w_2 - 0.6$. Se puede observar que el resultado obtenido es una recta que parte el plano en dos, de tal forma que los clientes con más riesgo son aquellos que quedan por debajo de esta, mientras que los clientes con menos riesgo son los que quedan por encima, lo cual se puede ver en la Figura 8).

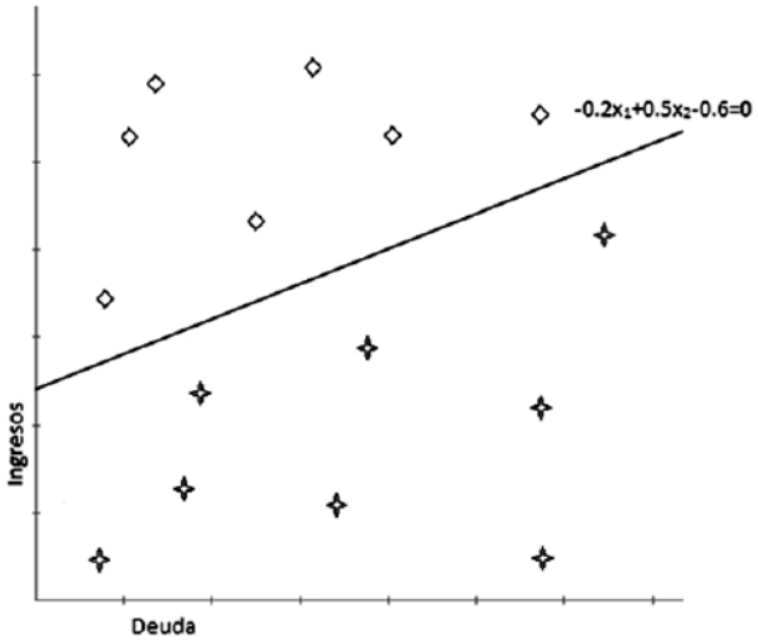


Figura 8. Clientes con más riesgo son representados por estrellas, mientras que los de menor riesgo son representados por rombos. Adaptación de García (2017).

Es posible mostrar ejemplos en el que el perceptrón puede tener cierta utilidad al ayudar a resolver problemas como el que se mencionó previamente, para lo cual, "decimos que se trata de un problema linealmente separable. En otro caso [...], hablamos de problemas que no son linealmente separables" (García, 2017, p.221), siendo estas tareas más complejas y, por lo regular, bastante más comunes.

Una posible solución para los problemas no linealmente separables es usar la salida (*output*) de una neurona como entrada a otra u otras, sin embargo y sin entrar en profundidad, esto genera particiones más complejas del plano (Figura 9), resultando así en la construcción de lo que se conoce como una red neuronal multicapa o red neuronal profunda.

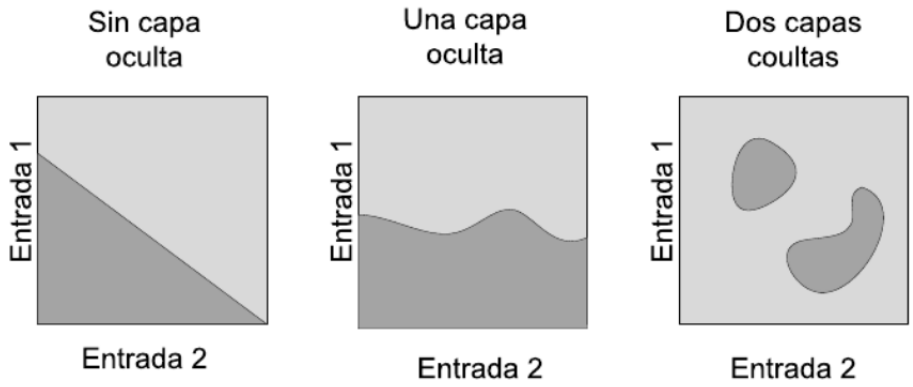


Figura 9. Particiones más complejas conforme aumenta el número de capas. Figura adaptada de García (2017).

La composición de una red neuronal multicapa (Figura 10), conocida como perceptrón multicapa, consiste de una capa de entrada (neuronas o datos de entrada), al menos dos capas ocultas (neuronas ocultas) y una capa de salida (neuronas de salida).

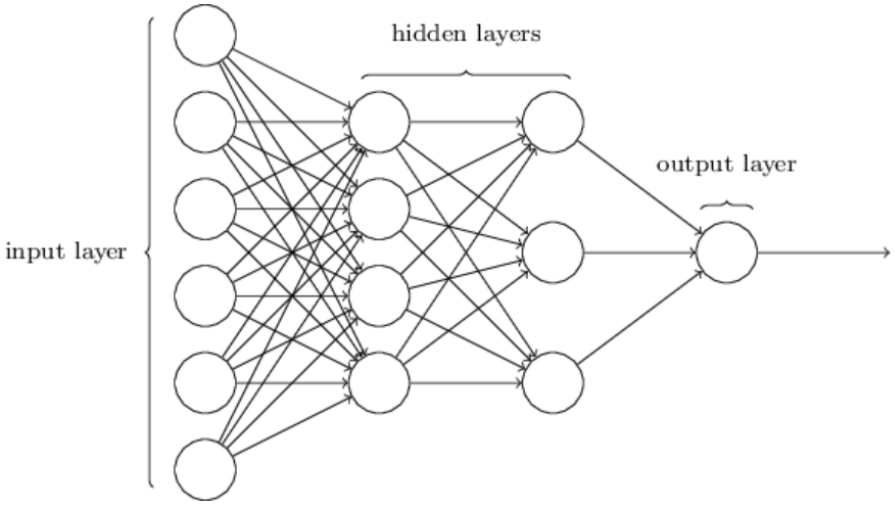


Figura 10. Perceptrón multicapa. Imagen adaptada de Nielsen (2015).

Con respecto a la sección 2.1.1, una vez que se tienen los datos en forma matricial, el número de neuronas en la capa de entrada (*input layer* de la Figura 10) se define con base al número de descriptores de la matriz de características (matriz 1), a saber, $x_1 = [a_{11}, a_{21}, ..., a_{p1}, ..., a_{m1}]$, $x_2 = [a_{12}, a_{22}, ..., a_{p2}, ..., a_{m2}]$, ..., $x_p = [a_{1p}, a_{2p}, ..., a_{pp}, ..., a_{mp}]$, ..., $x_n = [a_{1n}, a_{2n}, ..., a_{pn}, ..., a_{mn}]$, generando la matriz 5.

Observe que las matrices 1 y 5 son, en resumidas cuentas, la misma matriz, pues tienen las mismas entradas, además, *input layer* $= \begin{bmatrix} x_1 & x_2 & \cdots & x_p & \cdots & x_n \end{bmatrix}$:

$$\begin{bmatrix} x_1 & x_2 & \cdots & x_p & \cdots & x_n \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{p1} \\ \vdots \\ a_{m1} \end{pmatrix} & \begin{pmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{p2} \\ \vdots \\ a_{m2} \end{pmatrix} & \cdots & \begin{pmatrix} a_{1p} \\ a_{2p} \\ \vdots \\ a_{pp} \\ \vdots \\ a_{mp} \end{pmatrix} & \cdots & \begin{pmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{pn} \\ \vdots \\ a_{mn} \end{pmatrix} \end{bmatrix}. \tag{5}$$

El problema que surge cuando se trata de encontrar pesos y sesgos (el número de sesgos aumenta en una red multicapa) adecuados, es que un ligero cambio en los pesos $w + \Delta w$ (o sesgos) no necesariamente causa un ligero cambio en la salida *output* $+\Delta$ *output* (Figura 11), de hecho, puede ocasionar que la salida cambie totalmente, por lo cual resolver este problema no es nada trivial.

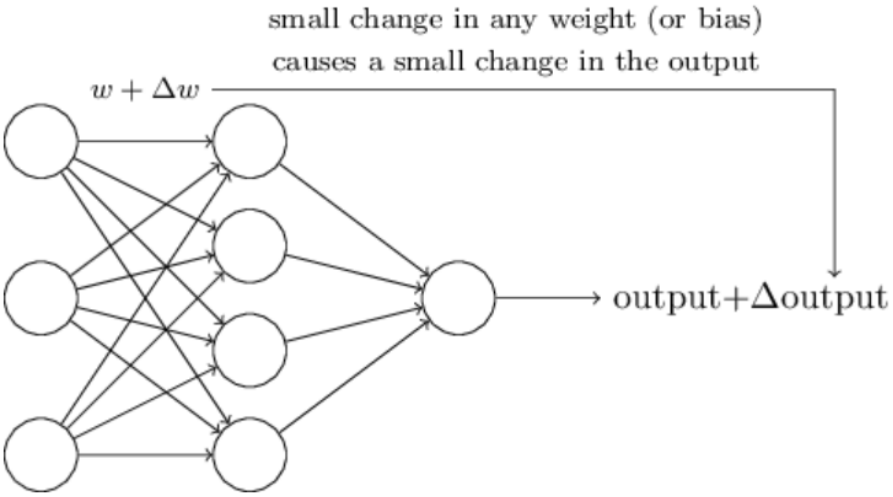


Figura 11. Pequeños cambios en la entrada no garantizan pequeños cambios en la salida. Figura adaptada de Nielsen (2015).

La forma en que se ataca este problema es utilizar funciones que no crezcan o decrezcan tan rápido. Ejemplo de estas se muestran en la Figura 12. Se puede observar que un 'pequeño' cambio en la entrada (dominio de la función) no genera un 'gran' cambio en la salida (rango de la función). Este tipo de funciones son llamadas **funciones de activación**.

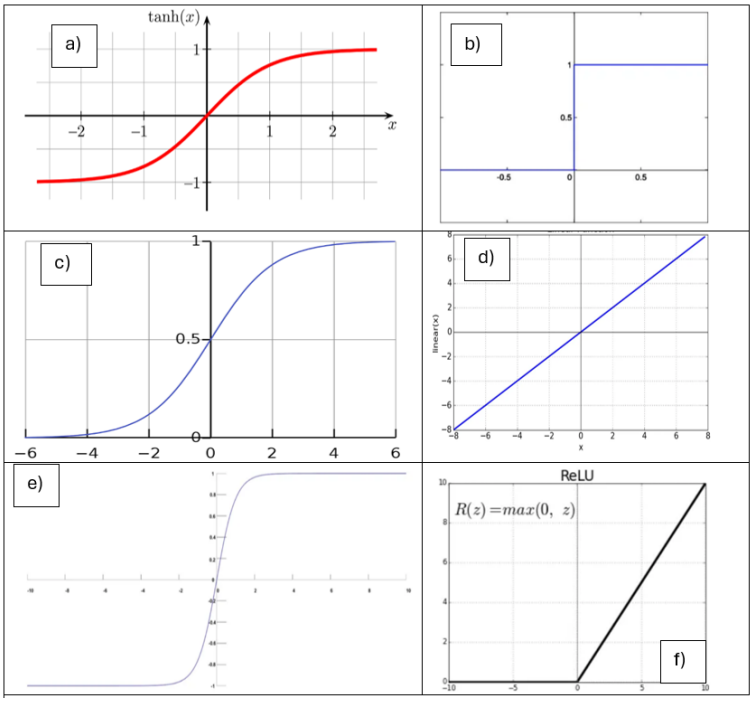


Figura 12. Funciones de activación: a) Tangente Hiperbólica $f(x) = \frac{2}{1+e^{-2x}} - 1$. b) Función Escalón H(x). c) Función Sigmoide $f(x) = \frac{1}{1+e^{-x}}$. d) Función Lineal. e) Función Softmax $S(v_i) = \frac{e^{v_i}}{\sum_j e^{v_i}}$. f) Función ReLU. Figuras adaptadas de internet.

Un vez conociendo la estructura de una red multicapa y las funciones de activación se puede considerar el siguiente ejemplo para una mejor comprensión. Considere una tarea de clasificación binaria (dos etiquetas) con un conjunto de 1000 instancias, 20 características, una capa oculta con 10 neuronas y una capa de salida con una sola neurona con *activación Sigmoide*, entonces se tienen lo siguiente:

- La suma ponderada de la neurona i-ésima (hay 10) en la capa densa aplicada a cada una de las 1000 instancias es:

$$z_i = \sum_{j=1}^{20} w_{ij} x_j + b_i, \tag{6}$$

  donde, w_{ij} son los pesos asociados a la neurona i-ésima y la característica j-ésima, x_j es el valor de la j-ésima característica perteneciente a la instancia de entrada correspondiente (hay 1000) y b_i es el sesgo de la neurona i-ésima.

- Al entrar o fijarse una instancia, se calcula z_i y después la salida de cada una de las 10 neuronas: $a_i = f(z_i)$, para i de 1 hasta 10, donde f es una función de activación, por lo tanto, la salida de toda la capa densa es un vector que contiene estas 10 neuronas de salida:

$$[a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}],$$

  las cuales ahora sirven como entrada para las siguientes neuronas y así el proceso se repite hasta llegar a la última capa.

- Dado que solo queda una neurona en la capa de salida (este suele ser el caso al utilizar *activación Sigmoide* en tareas de clasificación binaria), tenemos que el resultado final es:

$$\hat{y}_1 = f'(\sum_{j=1}^{10} w'_{ij} a_j + b'_i),$$

  donde f' es seleccionada como la función de *activacón Sigmoide* y $\hat{y}_1$ es un número entre 0 y 1, por lo que la clase se obtiene al seleccionar un umbral de clasificación t, es decir, la instancia de entrada será clasificada en la clase c_1 si $t \leq \hat{y}_1$ o será clasificada en la clase c_2 en caso contrario.

Cabe destacar que en un problema de clasificación multiclase el número de neuronas en la capa de salida es igual al número de clases, siendo la salida un vector de longitud igual al número de clases, donde cada índice del vector corresponde a una clase y cuya suma de sus elementos da como resultado 1 y en este

caso la función de activación f' que se utiliza es *Softmax*. Más aún, en este caso la instancia de entrada será clasificada en la clase correspondiente al índice con mayor probabilidad dentro del vector de salida.

Con las explicaciones anteriores, la ecuación 4 se puede reescribir de la siguiente forma:

$$\hat{y}_i = H(z_i) = H(\sum_{j=1}^{n} x_{ij}w_{ij} + b_i) = H(x_{11}w_{11} + x_{12}w_{12} + ... + x_{1n}w_{1n} + b_1), \tag{7}$$

donde $\hat{y}_i$ es 0 ó 1, H se define como $H(\text{v}) = 1$ para $v > 0$ o $H(\text{v}) = 0$ para $v \leq 0$ (Figura 12 inciso b).

En la práctica se espera que los datos sean valores pequeños, ya que computacionalmente es mejor manejarlos así, los cuales pueden ser modificados para que tomen valores dentro de ciertos rangos (como por ejemplo el [0,1]) para que mediante estas funciones de activación sea posible mantener estos valores, además, durante el ciclo de entrenamiento las instancias se pasan por lotes (*batch*) de tamaño N, por tanto, las sumas ponderadas en cada una de las capas densas se puede escribir como:

$$Z = X \cdot W + B, \tag{8}$$

donde X es la matriz de entrada de forma (N, n), la cual podría verse como una submatriz de la matriz 5 al tomar las primeras N instancias, W es la matriz de pesos de forma $(n, \#neuronas)$, B es un vector de sesgos unidimensional de longitud $\#neuronas$ y la salida Z, antes de aplicarse alguna función de activación, tiene la forma $(N, \#neuronas)$, donde cada fila representa la salida de una capa densa para cada una de las N instancias de entrada. Si bien el diseño de las capas de entrada y salida es sencillo, lo cierto es que en la literatura y en la práctica el diseño de las capas ocultas es todo un arte.

2.2.2. Aprendizaje en las redes neuronales

En la sección anterior se describió la arquitectura de las redes neuronales profundas y como es que operan en base a los números de neuronas, instancias, pesos, sesgos y clases pero no se describió como es el ajuste de pesos y sesgos, por lo que en esta sección se describe brevemente la forma en que las redes neuronales 'aprenden' a través de los datos.

Retomando las notaciones de la sección 2.1.1, si se cuenta con m instancias $x_{11}, x_{22}, ..., x_{mm}$, cada una con n características y sus correspondientes etiquetas (*targets*) $y_1, ..., y_m$, entonces el proceso de

aprendizaje es llevado a cabo mediante un *loop* de entrenamiento, en donde se van ajustando los pesos y sesgos, pero dependiendo de la comparación de las predicciones $\hat{y}_i$ con sus etiquetas, por lo que una forma de medir este acercamiento es utilizando una función de costos u objetivo como la que sigue:

$$C(w,b) \equiv \frac{1}{2n}\sum_{i=1}^{m} ||\hat{y}_i - y_i||^2. \tag{9}$$

Se escriben w y b para recordar que esta métrica depende los pesos y sesgos, sin olvidar la función de activación, además, $||\cdot||$ denota la norma euclidiana. Esta función C es conocida como error cuadrático medio (MSE por sus siglas en inglés). Cabe mencionar que dependiendo de qué tipo de problema se trate, ya sea regresión o clasificación binaria-multiclase, el esquema general es el de minimizar la métrica con la que se mide la similitud entre los *targets* y_i y predicciones $\hat{y}_i$ de cada una de las instancias de entrada, por lo que de manera más sencilla, se decidió profundizar con esta métrica, ya que en un principio se utilizó tanto en problemas de clasificación como en regresión.

Lema 2.1 *Para cualesquiera números reales x, y se cumple que*

$$x^2 + y^2 \geq 0$$

y $x^2 + y^2 = 0$ si y solo si $x = 0$ y $y = 0$.

Resumidamente el Lema 2.1 nos indica que la suma de dos números reales (se puede generalizar para n números) elevados al cuadrado siempre es positiva y, en segundo lugar, la suma de estos dos (también se puede generalizar a n sumandos) es cero, si y solamente si, ambos sumandos (los n sumandos) son cero, por lo que al observar la ecuación 9, se tiene que:

- $C(w,b)$ es no negativa (debido a cada sumando).
- $C(w,b) \approx 0$ si $\hat{y}_i - y_i \approx 0$ (por la segunda condición del Lema anterior).

La segunda condición indica que, si $C(w,b) \approx 0$ entonces $\hat{y}_i$ y y_i son casi iguales, por lo tanto, el algoritmo realizará un buen trabajo si este puede encontrar pesos y sesgos que hagan la función de costo lo más pequeña posible, es decir, que la minimicen.

En la práctica, los pesos y sesgos se inicializan con valores aleatorios pequeños (*random inicialization*)

para que gradualmente se vayan ajustando en el ciclo de entrenamiento (*training loop*), el cual trabaja como se muestra enseguida (Chollet, 2021).

Repita los siguientes pasos tanto como sea necesario:

1. No proporcione todo el conjunto de entrenamiento sino por porciones con sus respectivos *targets* y_i, es decir, si hay m instancias proporcione un lote de tamaño N (este puede ser un divisor de m).
2. Ejecute la red con el lote de datos proporcionado para obtener sus predicciones $\hat{y}_N$.
3. Calcule la pérdida obtenida con alguna función objetivo (el resultado de la función $C(w, b)$ entre y_i y $\hat{y}_i$) sobre el lote.
4. Actualice todos los pesos de la red de tal manera que reduzca ligeramente la pérdida en ese lote.

Con este ciclo eventualmente se obtendrá una red con una pérdida muy baja en sus datos de entrenamiento, por lo tanto, la red habrá aprendido a asignar *targets* de forma correcta.

Cabe decir que, aunque los pasos $1, 2$ y 3 son, en teoría fácil, la parte verdaderamente difícil es el paso 4, ya que en este se involucra el minimizar la función de costos y la manera de llevar a cabo este proceso es mediante un algoritmo bastante utilizado: **descenso de gradiente**.

Es importante mencionar que el descenso de gradiente es utilizado para minimizar funciones como la anteriormente descrita, para lo cual, se necesita calcular el gradiente ∇C con respecto a todas las direcciones involucradas (las características). También se podría buscar una función a maximizar basándose en el número correcto de valores obtenidos (o entradas bien clasificadas) en lugar de $C(w, b)$, pero se necesitaría una función dependiente de los pesos y sesgos, que sea diferenciable y que además mantenga pequeños cambios en la salida una vez realizados pequeños cambios en la entrada, lo cual no es fácil (Nielsen, 2015).

Con el conocimiento y uso del descenso de gradiente, el algoritmo anterior se puede reescribir como sigue (Chollet, 2021).

Repita los siguientes pasos tanto como sea necesario:

1. Propocione un lote de muestras de entrenamiento con sus respectivos *targets*.

2. Ejecute la red con estos datos proporcionados para obtener sus predicciones $\hat{y}$ (este paso es llamado *forward*).

3. Calcule la pérdida obtenida sobre la muestra.

4. Calcule el gradiente de la pérdida con respecto a los parámetros de la red.

5. Mueva los parámetros un poco en la dirección opuesta al gradiente (por ejemplo, $W = W - (paso \times gradiente)$), reduciendo así un poco la pérdida en el lote.

El paso 4 se refiere al uso del descenso de gradiente, mientras que la justificación del paso 5 se puede ver en el apéndice 22.4 del libro de Zhang et al. (2023). Tal paso es conocido como tasa de aprendizaje o *learning rate*.

El algoritmo anterior se llama Descenso de Gradiente Estocástico por lotes (*mini-batch stochastic gradient descent* o *mini-batch SGD*). El término estocástico se refiere al hecho de que cada lote de datos se extrae aleatoriamente, mientras que el término *mini-batch* se refiere al paso de los datos en lotes, ya que pasar todo el conjunto de datos puede llegar a ser computacionalmente intratable.

Es de suma importancia dejar en claro que el descenso de gradiente es un algoritmo de optimización que se utiliza para minimizar la función de costo, mientras que la retropropagación es un algoritmo que utiliza la regla de la cadena para calcular el gradiente negativo (paso 5) de la función de costo y así poder ajustar los pesos y sesgos. Cabe destacar que un inconveniente que surge al buscar el mínimo de la función de costo es que lo más probable es llegar a un mínimo local, por lo que actualmente se han desarrollado distintos métodos para tratar de conseguir buenos resultados a la hora de buscar minimizar esta función.

2.2.3. Redes Neuronales Convolucionales

En la sección anterior se describió el aprendizaje de las redes neuronales a partir de una matriz que posee todos los valores de las características y como es ingresada a una red neuronal. Esta forma se usa al utilizar datos tabulares, es decir, datos con forma de tabla que permita crear la matriz de características, la cual como ya se sabe, contiene características de los objetos, como lo son la altitud, longitud y peso de transportes o longitudes del sépalo y pétalo de flores, sin embargo, cuando se trabaja con imágenes se han creado varias formas de generar esta matriz o tabla de características pero no se adentrará en todas

estas. Una de ellas es llevada a cabo bajo la suposición de que cada píxel en la imagen corresponde a una sola característica, como por ejemplo, si se tiene una imagen de un solo canal y en escala de grises de tamaño 28 pixeles por 28 pixeles (matriz numérica bidimensional de forma (28,28)), entonces esta imagen es 'aplanada', es decir, esta imagen pasa de ser bidimensional a una matriz unidimensional (o vector unidimensional) de longitud 784 (784 = 28 x 28) características, obteniendo así una instancia con 784 características, por lo que ahora se pueden definir 784 neuronas en la capa de entrada. Esto se ilustra en la Figura 13.

Las redes neuronales convolucionales o *convolucional neuronal networks* (CNNs o ConvNets por sus siglas en inglés), son modelos de aprendizaje profundo bastante utilizados en aplicaciones de visión por computadora y probablemente sean estos modelos los que le dieron mayor fama a este tipo de aprendizaje. Los datos que usan de entrada son imágenes y si bien es posible ingresarlas a un modelo de red neuronal simple como se describió previamente e ilustrado en la Figura 13, lo cierto es que se han logrado obtener mejores resultados al utilizar CNNs.

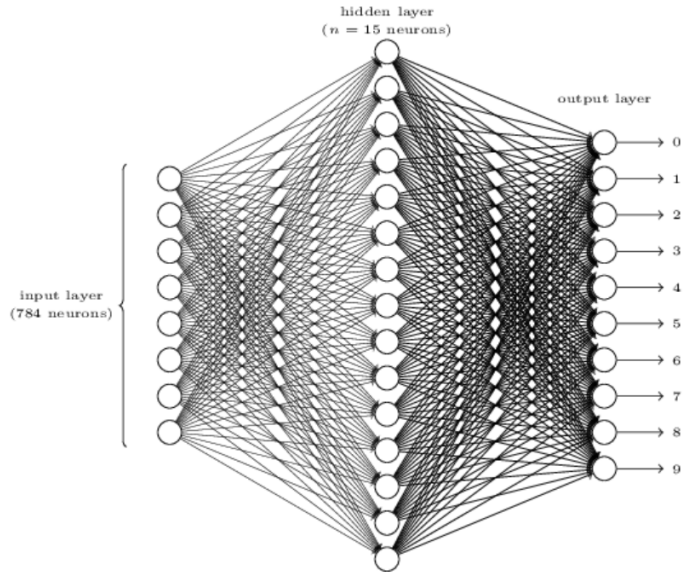


Figura 13. Se ilustra una red con capa de entrada (imagen en blanco y negro de 8 bits de tamaño $28 \times 28 = 784$), una capa oculta de 15 neuronas y una capa de salida que clasifica diez clases (0 al 9). Imagen obtenida de Nielsen (2015).

Lo que tienen en común los modelos de aprendizaje máquina y el aprendizaje profundo es que ambos generan modelos predictivos desde distintos enfoques, sin embargo, la principal diferencia entre estos es la selección y/u obtención de características. Un modelo predictivo de aprendizaje máquina necesita más que nada características, las cuales proporcionan al modelo información para que a partir de estas se encuentren patrones con los cuales pueda llevar a cabo una buena tarea de clasificación. Es en esta parte

donde se necesita de un experto que pueda obtener o crear estas características desde los datos (*feature extraction*) para posteriormente seleccionar el clasificador adecuado para su propósito, sin embargo, las CNNs combinan ambas tareas en un solo modelo, ya que la extracción de características es llevada a cabo en las primeras capas del modelo mientras que la tarea de clasificación es llevada a cabo en las últimas (Figura 14), sin olvidar que el aprendizaje es llevado a cabo mediante la retropropagación (Falconí et al., 2019).

Se puede resaltar la estructura básica de una CNN como propone Naji et al. (2022):

- **Capas convolucionales.** Estas capas son el principal elemento de una CNN. Estas generan el mayor número de cálculos en la red y se describen más adelante.
- **Capas de pooling.** Estas capas son generalmente insertadas entre las capas convolucionales, en donde su función es reducir gradualmente el tamaño espacial del volumen de los datos.
- **Retropropagación** . La retropropagación, conocida como *back-propagation*, se puede resumir como el paso del gradiente hacia atrás para ajustar los pesos del modelo y hacer que este pueda mejorar.
- **Capas completamente conectadas.** También conocidas como *Fully Connected layer* o FC (por sus siglas en inglés), son capas (puede ser una) conectadas que hacen la tarea de clasificación, las cuales vendrían siendo la composición del perceptrón multicapa.

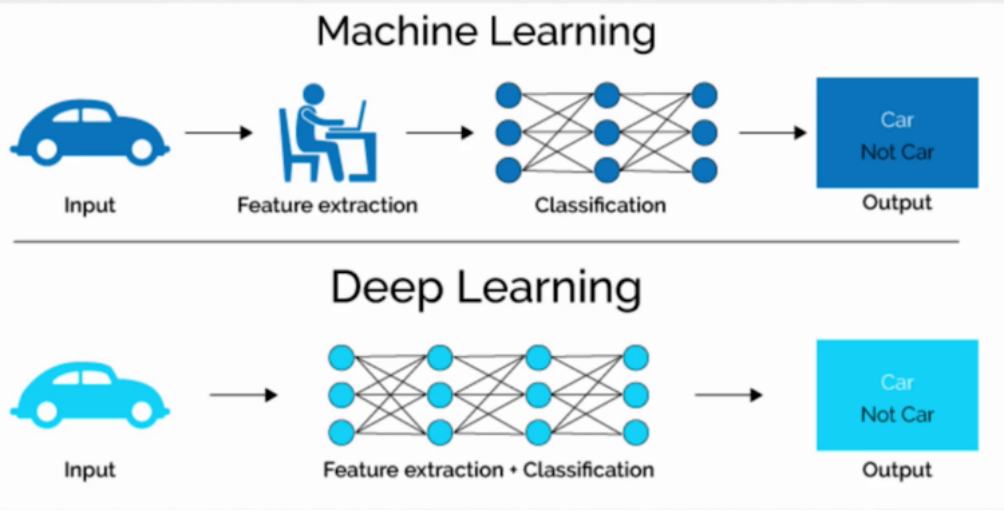


Figura 14. Se muestra la principal diferencia entre el Aprendizaje Máquina (*Machine Learning*) y Aprendizaje Profundo (*Deep Learning*): **la extracción de características**[1].

[1]Imagen modificada de: `https://openwebinars.net/blog/diferencias-entre-machine-learning-y-deep-learning/`.

Aunque hay más capas que se pueden insertar en el modelo, como las de normalización y *drop-out*, las anteriormente descritas componen la estructura básica de una CNN. Cabe mencionar que en una capa de convolución hay filtros (matrices numéricas) que realizan la operación de convolución con las imágenes de entrada. Esta operación, así como la estructura básica de una CNN se ilustran en la siguiente sección.

2.2.4. Representación de las capas de una CNN

En esta sección se presentan algunas ilustraciones e información de manera breve para comprender tanto la arquitectura como el funcionamiento de una CNN.

Una imagen en escala de grises (de un canal) de 8 bits es resumidamente una matriz numérica de tamaño $n \times m$, en donde cada elemento es llamado píxel y toma un valor de 0 a $255 = 2^8 - 1$ (estos valores representan el brillo). Un filtro es una matriz numérica más pequeña (por lo regular cuadrada) de tal forma que, al realizar la operación de 'convolución' con este filtro, también llamado *kernel*, realza ciertas características o elimina otras de la imagen original como las que se muestran en la Figura 15.

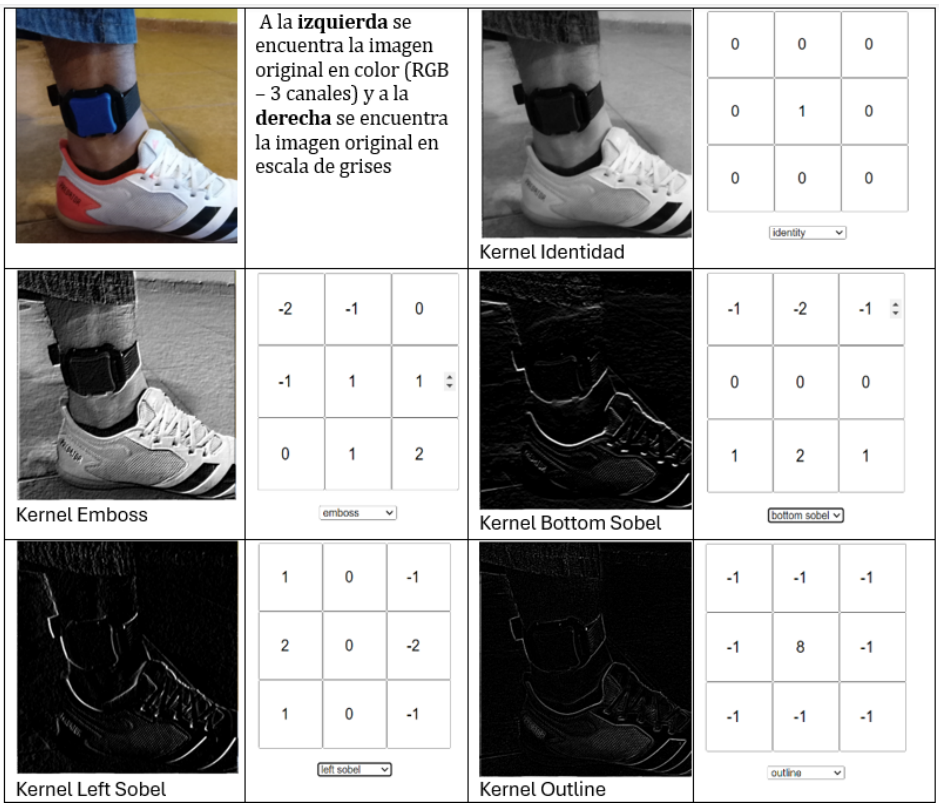


Figura 15. La figura muestra una imagen original e imágenes obtenidas mediante el proceso de convolución con la imagen original mediante el kernel situado a su derecha.

Con respecto a la descripción de la Figura 15, se comenta entre comillas simples 'convolución' porque la operación que se realiza entre la imagen y el *kernel* es un producto punto y no el que se suele encontrar en libros de procesamiento de imágenes, por lo que a continuación se muestra un ejemplo de esta operación denotada en este texto por $\odot$.

Si se define la matriz $A = \begin{bmatrix} -10 & -7 & 6 & 6 \\ -8 & 8 & 9 & -6 \\ 9 & 1 & 4 & 8 \\ 10 & 3 & -10 & 1 \end{bmatrix}$ de tamaño 4×4 y el *kernel* $k = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$ de tamaño 3×3, entonces el resultado es una matriz de tamaño 2×2, a saber, $A \odot k = \begin{bmatrix} -28 & -6 \\ 8 & 9 \end{bmatrix}$ si es que no hay *padding* (más abajo se ilustra) y el *stride* (paso) es de 1, ya que bajo estas condiciones la fórmula general indica que el tamaño de la matriz de salida de una convolución de una matriz A de tamaño $n \times n$ con un filtro k de tamaño $m \times m$ es de $(n - m + 1) \times (n - m + 1)$.

Para un mejor entendimiento de cómo realizar esta operación considere la siguiente descripción. Primero se debe 'ubicar' el número de la parte superior izquierda del *kernel* y en el ejemplo anterior es el número 1, después se 'sobrepone' el *kernel* de tal forma que el número 1 quede 'sobrepuesto' sobre el número ubicado en la parte superior izquierda de la matriz A, en este caso el número -10, por lo que quedarán 'sobrepuestos' los demás números, a saber, -7 con 0, 6 con -1, -8 con 1, 8 con 0, 9 con -1, 9 con 1, 1 con 0 y 4 con -1, así la salida es el número $k_{1,1} = (-10)(1) + (-7)(0) + (6)(-1) + (-8)(1) + (8)(0) + (9)(-1) + (9)(1) + (1)(0) + (4)(-1) = -28$.

En rojo se muestra los números de la matriz A 'sobrepuestos' por los números en verde del *kernel* k, posteriormente se obtiene el número -28 (como se mostró previamente): $\begin{bmatrix} (-10)(1) & (-7)(0) & (6)(-1) & 6 \\ (-8)(1) & (8)(0) & (9)(-1) & -6 \\ (9)(1) & (1)(0) & (4)(-1) & 8 \\ 10 & 3 & -10 & 1 \end{bmatrix}$.

Como se selecciona *stride* de 1, ahora el *kernel* se 'desplaza' un lugar a la derecha, obteniéndose $k_{2,2} = (-7)(1) + (6)(0) + (6)(-1) + (8)(1) + (9)(0) + (-6)(-1) + (1)(1) + (4)(0) + (8)(-1) = -6$, resultado del producto y suma de los número en rojo y verde: $\begin{bmatrix} -10 & (-7)(1) & (6)(0) & (6)(-1) \\ -8 & (8)(1) & (9)(0) & (-6)(-1) \\ 9 & (1)(1) & (4)(0) & (8)(-1) \\ 10 & 3 & -10 & 1 \end{bmatrix}$.

Nuevamente, como el *stride* es de 1 y ya no se puede desplazar más el *kernerl* a la derecha, se regresa a la posición inicial y ahora se desplaza un lugar hacia abajo (debido al valor del *stride*), de ahí que $k_{2,1} = (-8)(1) + (8)(0) + (9)(-1) + (9)(1) + (1)(0) + (4)(-1) + (10)(1) + (3)(0) + (-10)(-1) = 8$, mostrando como ahora está sobrepuesto el *kernel* en la matriz: $\begin{bmatrix} -10 & -7 & 6 & 6 \\ (-8)(1) & (8)(0) & (9)(-1) & -6 \\ (9)(1) & (1)(0) & (4)(-1) & 8 \\ (10)(1) & (3)(0) & (-10)(-1) & 1 \end{bmatrix}$.

Finalmente y recordando que el paso es de 1, el kernel se desplaza a la derecha, resultando el valor de $k_{2,2} = (8)(1) + (9)(0) + (-6)(-1) + (1)(1) + (4)(0) + (8)(-1) + (3)(1) + (-10)(0) + (1)(-1) = 9$, mediante la observación: $\begin{bmatrix} -10 & -7 & 6 & 6 \\ -8 & (8)(1) & (9)(0) & (-6)(-1) \\ 9 & (1)(1) & (4)(0) & (8)(-1) \\ 10 & (3)(1) & (-10)(0) & (1)(-1) \end{bmatrix}$.

Cabe mencionar que, si el paso es de dos, el proceso anterior se repite pero moviéndose a la derecha y luego posteriormente hacia abajo de dos en dos lugares.

El uso de *padding* en la matriz A bajo la convolución con el *kernel* k genera que la salida sea una matriz con las mismas dimensiones que la matriz A. Primero para realizar el *padding* lo que se hace es ampliar la matriz A, aumentando sus dimensiones y 'rellenando' con ceros los lugares vacíos generados por esta ampliación, después se realiza la convolución (proceso anteriormente descrito). Este proceso se ilustra a continuación.

Matriz $A = \begin{bmatrix} -10 & -7 & 6 & 6 \\ -8 & 8 & 9 & -6 \\ 9 & 1 & 4 & 8 \\ 10 & 3 & -10 & 1 \end{bmatrix}$, *kernel* $k = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$, matriz A con *padding*: $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -10 & -7 & 6 & 6 & 0 \\ 0 & -8 & 8 & 9 & -6 & 0 \\ 0 & 9 & 1 & 4 & 8 & 0 \\ 0 & 10 & 3 & -10 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$, tomando *stride* de 1 se tiene que $A_{padding} \odot k = \begin{bmatrix} -1 & -33 & 1 & 15 \\ -2 & -28 & -6 & 19 \\ -12 & 8 & 9 & 3 \\ -4 & 25 & 5 & -6 \end{bmatrix}$, ya que las operaciones ahora se empiezan calculando así: $\begin{bmatrix} (0)(1) & (0)(0) & (0)(-1) & 0 & 0 & 0 \\ (0)(1) & (-10)(0) & (-7)(-1) & 6 & 6 & 0 \\ (0)(1) & (-8)(0) & (8)(-1) & 9 & -6 & 0 \\ 0 & 9 & 1 & 4 & 8 & 0 \\ 0 & 10 & 3 & -10 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$. Observemos que los números de color naranja de la matriz $A_{padding} \odot k$ son las entradas de la matriz $A \odot k$.

Las capas de *pooling* lo que realizan es un submuestreo con una matriz más pequeña que A. Lo que se hace implícitamente es 'sobreponer esta matriz en la matriz A' y de los valores de los pixeles que queden cubiertos se selecciona, ya sea el mayor valor (*max-pooling*), el promedio (*average-pooling*) o el mínimo (*min-pooling*), lo cual se muestra a continuación.

Matriz $A_{padding} \odot k = \begin{bmatrix} -1 & -33 & 1 & 15 \\ -2 & -28 & -6 & 19 \\ -12 & 8 & 9 & 3 \\ -4 & 25 & 5 & -6 \end{bmatrix}$ y si se piensa en una matriz cuadrada de 2×2, la representación sería: $\begin{bmatrix} \begin{pmatrix} -1 & -33 \\ -2 & -28 \end{pmatrix} & \begin{pmatrix} 1 & 15 \\ -6 & 19 \end{pmatrix} \\ \begin{pmatrix} -12 & 8 \\ -4 & 25 \end{pmatrix} & \begin{pmatrix} 9 & 3 \\ 5 & -6 \end{pmatrix} \end{bmatrix}$ y aplicando la capa de *max-pooling* resulta la matriz $\begin{bmatrix} -1 & 19 \\ 25 & 9 \end{bmatrix}$, ya que en cada submatriz se selecciona el valor máximo (en rojo). Es importante mencionar que en la práctica elegir el máximo es el que suele tener mejores resultados.

Un concepto importante que surge debido a todas las operaciones generadas en las capas anteriores, es el de **mapa de características** (*feature map*) o **mapa de activación**, el cual es un cubo (son los filtros resultantes) generado por todas las operaciones anteriormente descritas y que posteriormente es 'aplanado' (*flatten*) para así poder ingresarlo, en el caso de CNNs, como entrada a un perceptrón multicapa o como se mencionó previamente, a las capas FC para que estás (de ser varias) realicen la tarea de clasificación mediante las funciones de activación antes descritas. Cabe mencionar que si las imágenes son proporcionadas en formato RGB (del inglés *Red,Green* y *Blue*, también llamadas de tres canales), las operaciones de convolución con los filtros definidos se realizan en los tres canales.

En la práctica se suele trabajar con imágenes RGB (como las de la Figura 16), pero también es posible hacerlo con imágenes en blanco y negro o en escala de grises. Además al utilizar una CNN, en el caso de la Figura 13, la entrada se definiría como (28,28,1) si es de un canal o (28,28,3) si es de tres canales.

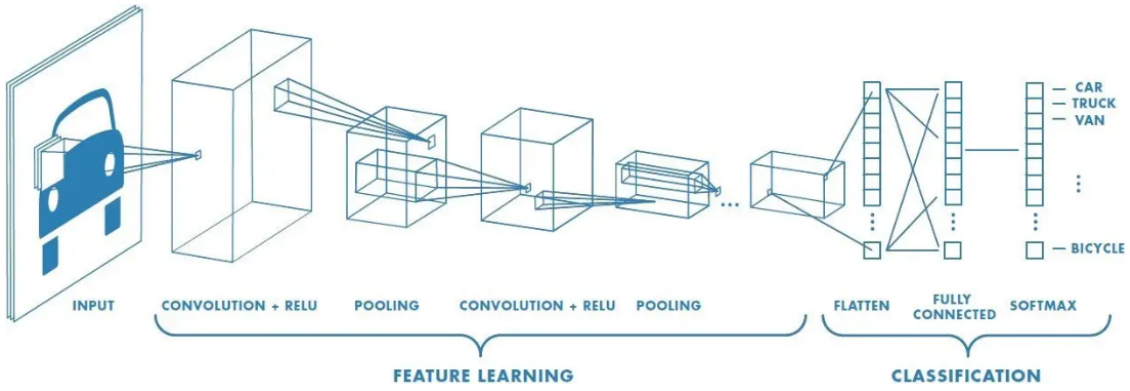


Figura 16. Se ilustra una CNN cuya entrada es una imagen RGB (tres canales), operaciones realizadas por las capas ya descritas para obtener el mapa de características: un cubo, que posteriormente es aplanado (*Flatten*) para ser ingresado como datos tabulares para que un perceptrón multicapa (FC) realice la tarea de clasificación. Imagen adaptada de Saha (2018).

Un problema que surge en los algoritmos de aprendizaje profundo, es la interpretabilidad del modelo, llamándolos muchas veces modelos de caja negra, aseverando que estos aprenden representaciones que son difíciles de explicar y presentar en 'lenguaje humano' y aunque esto es parcialmente cierto no lo es para las ConvNets, ya que es posible visualizar las representaciones que aprenden estos modelos, pues en gran parte son representaciones de conceptos visuales (Chollet, 2021). En resumen, se puede observar que lo que aprenden las ConvNets para llevar a cabo una buena clasificación son los filtros (Figuras 17 y 18).

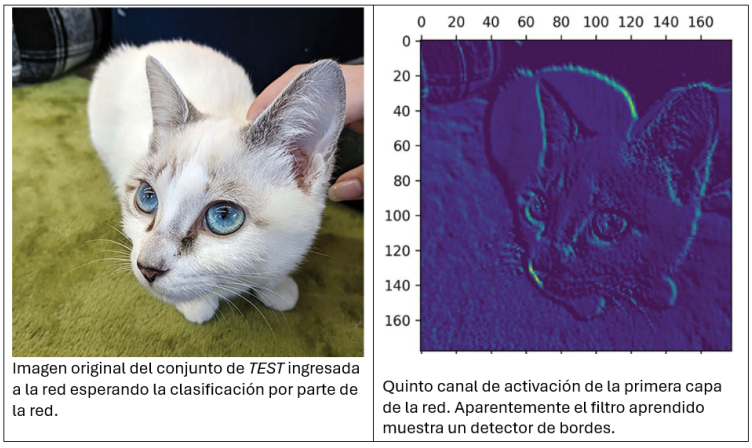


Figura 17. Aparentemente hay detección de bordes de uno de los filtros de la red,el cual aprendió a detectarlos para ayudar en la tarea de clasificación de gatos. Figura adaptada de Chollet (2021).

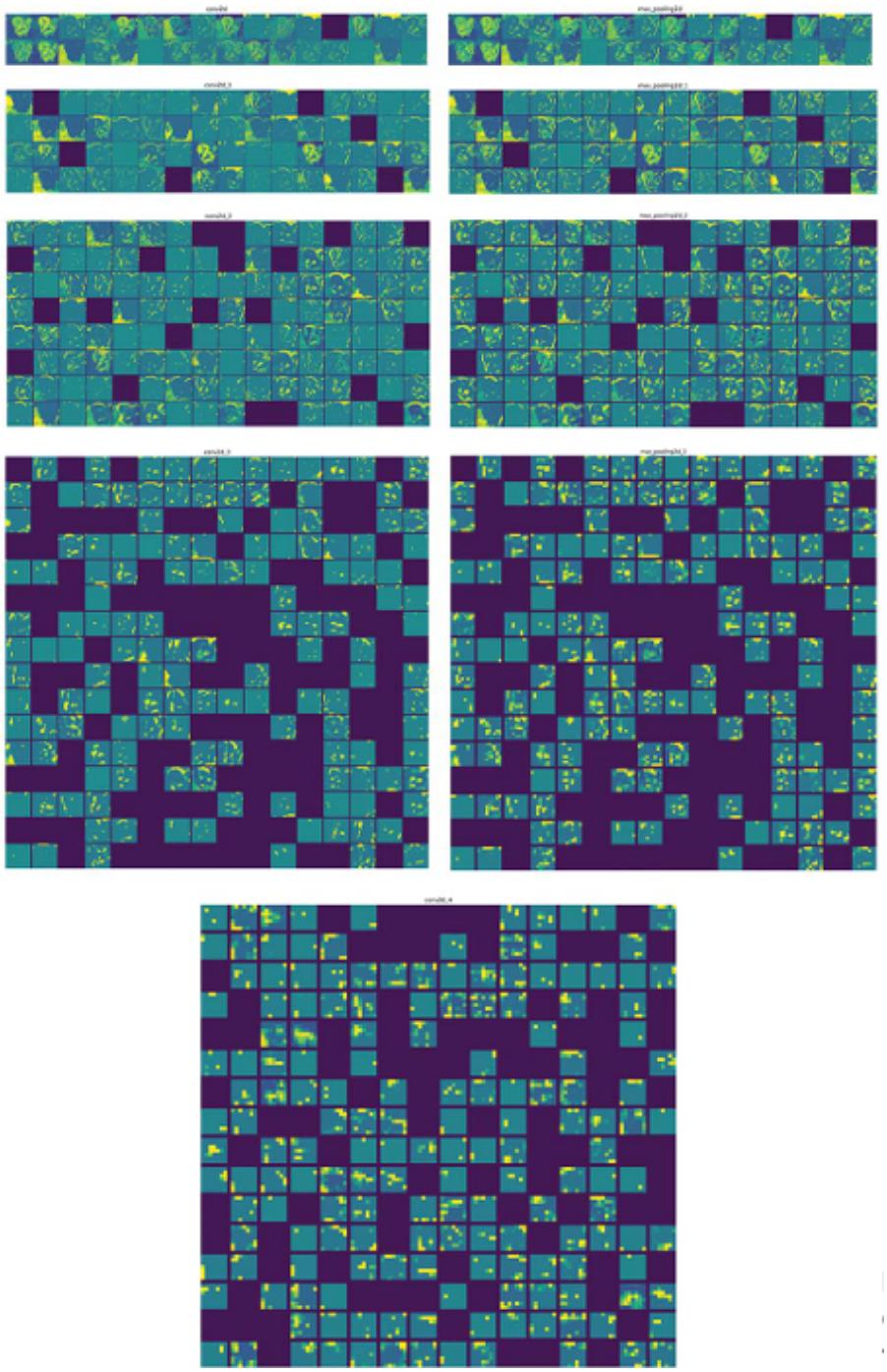


Figura 18. Se puede observar cada canal en cada capa de activación del mapa de características. Figura adaptada de Chollet (2021).

Esto significa que los parámetros entrenables de una CNN son, además de los pesos y sesgos del perceptrón multicapa, los valores de los filtros, los cuales también se inicializan de manera aleatoria, por lo que el entrenamiento suele ser más tardado con el uso de estas estructuras.

2.3. Clasificadores de Aprendizaje Máquina

En esta sección se describen brevemente algunos algoritmos de aprendizaje máquina que se utilizaron en la parte experimental. Aunque no es necesario en todos los casos comprender toda la teoría y programación detrás de estos algoritmos, el tener una buena idea de como trabajan podría ayudar a seleccionar el algoritmo adecuado para la tarea en cuestión.

2.3.1. Gaussian Naive Bayes

Los clasificadores bayesianos utilizan el teorema de Bayes con la suposición ingenua (*naive*) de la independencia condicional entre cada par de características dada la clase o etiqueta. En la clasificación bayesiana se está interesado en encontrar la probabilidad de obtener una etiqueta (L), dadas las características (*features*), escrita como $P(L|features)$. Específicamente, el teroema de Bayes se utilizaría de la siguiente manera para expresar este cálculo:

$$P(L|features) = \frac{P(features|L)P(L)}{P(features)}.$$

El modelo Gaussian Naive Bayes supone que la distribución de probabilidad de las características es normal. De ahí que una manera fácil y rápida de crear un simple clasificador es tomar estas dos suposiciones: datos con distribución normal y sin covarianza entre las características (Figura 19).

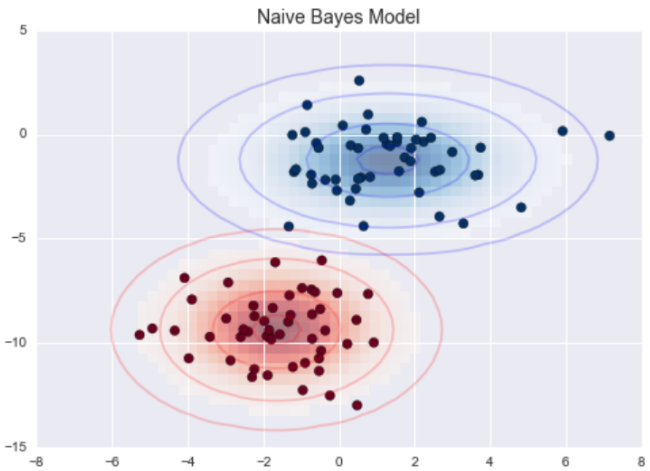


Figura 19. Datos con distribución Gaussiana sin covarianza. Figura adaptada de VanderPlas (2016).

.

2.3.2. Clasificador K-Nearest Neighbors

Aunque el clasificador previamente descrito es fácil de usar, la realidad es que casi siempre se desconoce la distribución condicional de los datos por lo que implementar el clasificador Naive Bayes no es del todo justificable, por lo que este se ha vuelto un estándar para comparar otros métodos. Algunos enfoques buscan estimar la distribución condicional de las etiquetas dados los datos, y así clasificar una instancia a la clase con la probabilidad más alta estimada. El clasificador de K-vecinos más cercanos o *K-Nearest Neighbors* (KNN) es uno de estos métodos, por lo que dado un entero positivo K y la instancia x_0, el clasificador KNN primero identifica K instancias de entrenamiento cercanos a x_0, representado este conjunto de puntos por $\aleph_0$. Después, se estima la probabilidad condicional para la clase j como el cociente de puntos 'vecinos' de clase j entre los K vecinos más cercanos:

$$P(L = j|X = x_0) = \frac{|\aleph_0|}{K}$$

.

Por último, el clasificador KNN clasifica la instancia x_0 a la clase con más alta probabilidad. Por ejemplo, considere $K = 5$, y que para una instancia **x** los 5 vecinos más cercanos estén distribuidos de la siguiente manera:

- Clase A: 3 vecinos
- Clase B: 2 vecinos

Entonces las probabilidades condicionales estimadas serían:

- $P(y = A|\mathbf{x}) = \frac{3}{5} = 0.6$
- $P(y = B|\mathbf{x}) = \frac{2}{5} = 0.4$

2.3.3. Regresión Logística

Al igual que los modelos bayesianos, este modelo es utilizado en tareas de clasificación para estimar la probabilidad de que una instancia pertenezca a una clase. Se utiliza la función logística *Sigmoide* para

mapear cualquier valor real en el rango [0,1] (Figura 12 c)). La regresión logística también es conocida como *logit regression*.

Primero para cualquier instancia i, donde $(i = 1, 2, ..., m)$, se calcula una combinación lineal de los predictores $x_i = (x_{i1}, x_{i2}, ..., x_{in})$ y los coeficientes $\theta = (\theta_0, \theta_1, ..., \theta_n)$:

$$z_i = (\theta_0 + \theta_1 x_{i1} + ... + \theta_n x_{in}) = \theta_i^\top x_i,$$

donde θ_0 es el sesgo y $\theta_1, \theta_2, ..., \theta_n$ son los coeficientes de los predictores.

Después, la función de *activación Sigmoide* se aplica a esta combinación lineal para mapear cualquier valor real al intervalo [0,1]. Esta definición es identificada como un perceptrón simple o una red de una sola capa.

Si la función de *activación Sigmoide* se representa por la función σ y t como su argumento, entonces se puede ver de la Figura 12 c) que $\sigma(t) < 0.5$ cuando $t < 0$, y $\sigma(t) \geq 0.5$ cuando $t \geq 0$. Por lo que una forma de estimar probabilidades en forma vectorizada para la i-ésima instancia se puede escribir como:

$$\hat{p}_i = h_\theta(x_i) = \sigma(\theta^\top x_i), \tag{10}$$

de ahí que las predicciones obtenidas $\hat{y}_i$ usando un umbral de probabilidad con valor de 0.5 se definen como:

$$\hat{y}_i = \begin{cases} 0 & si \quad \hat{p}_i < 0.5 \\ 1 & si \quad \hat{p}_i \geq 0.5 \end{cases}, \tag{11}$$

por lo tanto, la probabilidad será 1 si $\theta^\top x_i$ es positiva y 0 si es negativa. De hecho, es así como se obtiene la etiqueta o clasificación final al utilizar las redes neuronales en tareas de clasificación binaria con *activación Sigmoide*.

La función de costo en la regresión logística es la entropía cruzada o pérdida logística (*log-loss*), que mide la discrepancia entre las predicciones del modelo y las etiquetas reales. La función de costo $J(\theta)$ para regresión logística en tareas de clasificación binaria se define como:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \log(h_\theta(x_i)) + (1-y_i)\log(1-h_\theta(x_i))\right], \tag{12}$$

donde y_i representa la etiqueta verdadera y $h_\theta(x_i)$ la predicción, ambas correspondientes a la i-ésima instancia y, de igual manera a como se mostró en las dos secciones previas, el ajuste de sus parámetros se hace por medio del descenso de gradiente.

2.3.4. Clasificador Ridge

El modelo Ridge es una variante de la regresión Logística, pero en lugar de minimizar la función de costo mostrada en la ecuación 12, se minimiza una función de pérdida cuadrática (como en la regresión lineal) ajustada para clasificación. La función de costo que se minimiza es la suma de los errores cuadrados más un término de penalización $L2$ sobre los coeficientes del modelo. Matemáticamente, la función de costo $J(\theta)$ se define como:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 + \frac{\alpha}{2}\sum_{i=1}^{m}\theta_i^2 = \frac{1}{m}\sum_{i=1}^{m}(\theta_i^\top x_i - y_i)^2 + \frac{\alpha}{2}\sum_{i=1}^{m}\theta_i^2, \tag{13}$$

donde y_i es la etiqueta verdadera, x_i y θ cono en la sección previa 2.3.3 y α el parámetro de regularización que controla la fuerza de penalización $L2$, también conocida como regularización de *Tikhonov*.

Cabe destacar que el ajuste de parámetros no siempre se realiza con el descenso de gradiente, ya que es posible ajustarlos con soluciones matriciales o de forma estadística utilizando *máxima verosimilitud* y en este caso el ajuste de los parámetros θ se realiza mediante la resolución de la siguiente ecuación normalizada:

$$\theta = (X^\top X + \alpha I)^{-1} X^\top y, \tag{14}$$

donde X es la matriz de características de tamaño $m \times n$, y es el vector de etiquetas (ver 2) , α el parámetro de regularización y la matriz identidad de tamaño $n \times n$ es representada por I.

2.3.5. Máquina de Soporte Vectorial

De la misma manera, la Máquina de Soporte Vectorial (SVM por sus siglas en inglés), es un algoritmo de aprendizaje muy versátil, ya que es capaz de realizar regresión (SVR por sus siglas en inglés) y clasificaciones lineales o no-lineales (SVC por sus siglas en inglés). Se recomienda su implementación con *datasets* no muy grandes, ya que esta es muy costosa hablando computacionalmente, además de ser sensible a la escala de los datos, es decir, se recomienda que todos los valores de la matriz de características estén en un rango pequeño, como pro ejemplo, entre 0 y 1.

El objetivo de SVC es separar los datos como se muestra en la Figura 20. Observe que agregar más instancias de entrenamiento fuera del margen de clasificación no afecta la función de decisión, ya que esta está completamente determinada o soportada por las instancias localizadas en los bordes de las líneas a trozos. Estas instancias son llamadas vectores de soporte, los cuales se muestran en la misma figura encerrados en un círculo.

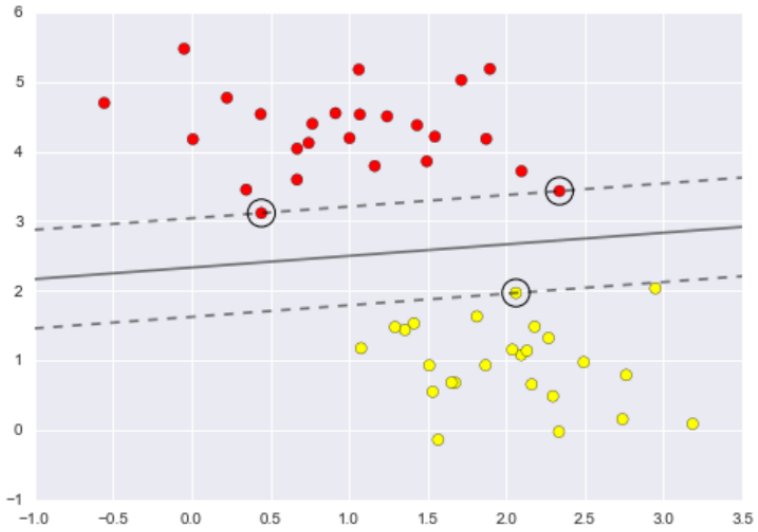


Figura 20. La línea continua que divide las clases se conoce como frontera de decisión, mientras que el espacio formado por las líneas discontínuas o por pintadas por trozos forman lo que se conoce como margen de clasificación. Figura adaptada de VanderPlas (2016).

.

Las máquinas de soporte vectoriales se vuelven muy útiles al ser combinadas con *kernels*. La idea es proyectar los datos en dimensiones superiores transformando los datos mediante funciones polinómicas o funciones de base Gaussianas denotadas por $\phi(x)$. La Figura 21 muestra perfectamente que los datos pueden ser separados pero nunca de manera lineal (usando rectas), para lo cual, el uso de funciones de similaridad como *Gaussian RBF*, suelen ser útiles al momento de separar los datos.

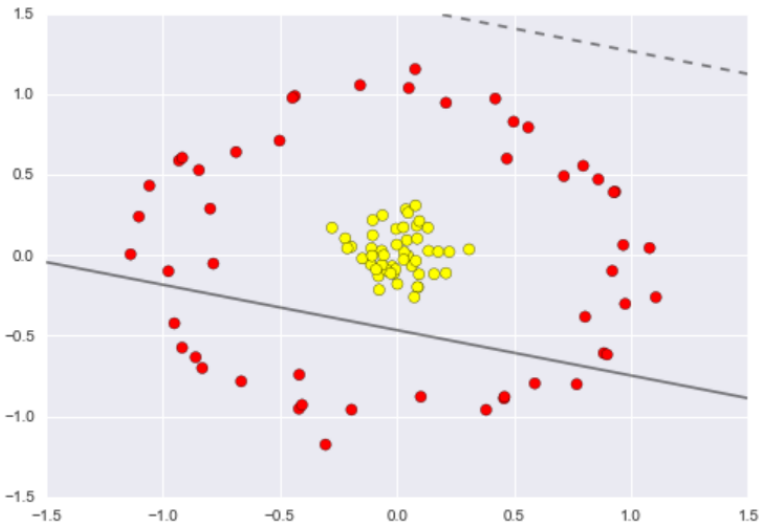


Figura 21. Los datos en la imagen no pueden ser separados por rectas estando en la misma dimensión. Figura adaptada de VanderPlas (2016).

El autor de estos ejemplos muestra que con una simple proyección de los datos usando una función de base radial (*Gaussian RBF*) centrada en medio de los datos puede ayudar a separarlos, lo cuál se muestra en la Figura 22, más aún, observe que ahora los datos son linealmente separables por el plano $r = 0.7$.

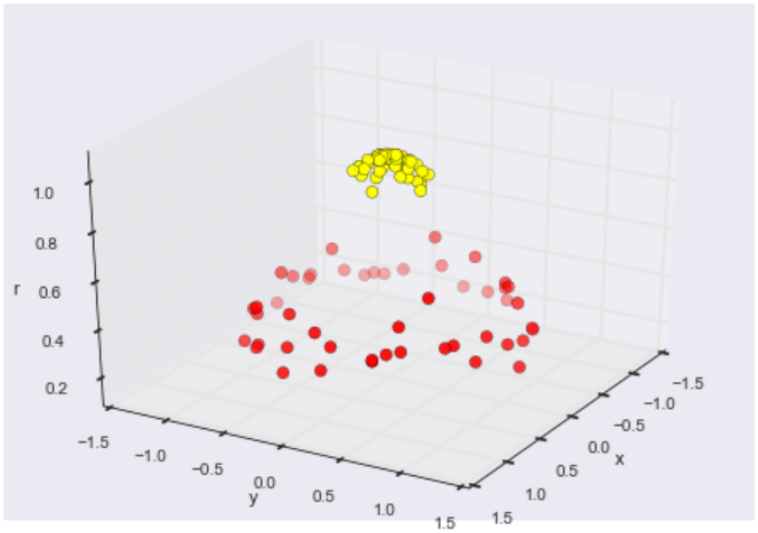


Figura 22. Los datos proyectados en una dimensión más alta ahora son linealmente separables. Figura adaptada de VanderPlas (2016).

.

Un problema que surge es encontrar el centro de la proyección, ya que el no proyectar los datos en el lugar correcto no se generará la separación lineal esperada. El enfoque llevado a cabo sería por fuerza bruta, es decir, proyectar N puntos en N dimensiones, sin embargo, el realizar esto es prácticamente intratable computacionalmente. Existe una manera de sobrepasar este problema, llamado *truco kernel*. Este

procedimiento permite realizar un ajuste implícito de los datos transformados por el kernel sin siquiera construir las representaciones N-dimensionales de los datos proyectados en base al kernel utilizado. Este método permite aprender relaciones o fronteras de decisión no-lineales como la que se muestra en la Figura 23.

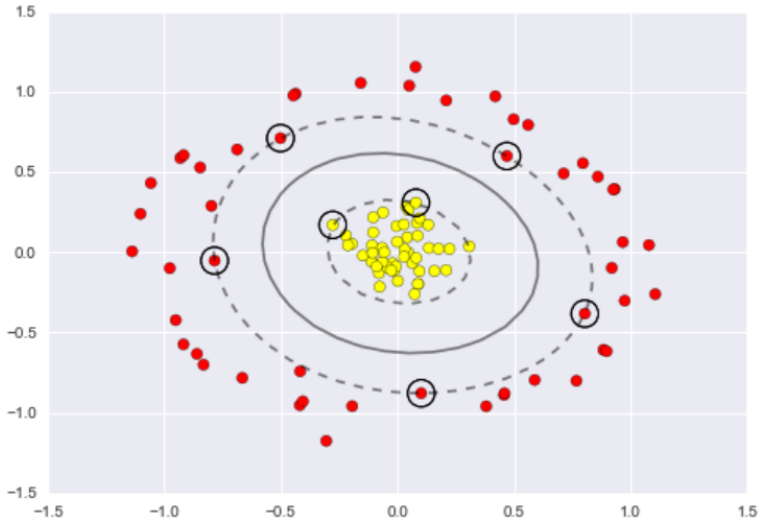


Figura 23. Con el truco kernel ahora se pueden obtener funciones o fronteras de decision no-lineales en los datos. Figura adaptada de VanderPlas (2016).

.

Brevemente se puede mencionar que el ajuste de parámetros en un SVC con kernel lineal se realiza mediante la minimización de una función de costo que equilibra principalmente dos objetivos: Maximizar el margen del hiperplano y minimizar los errores de clasificación. Además, la función de costo que se optimiza en un SVC depende del tipo de SVM que se esté utilizando (con margen duro o margen suave). Cuando los datos no son perfectamente separables, se introduce un término de regularización con penalización por errores de clasificación (o puntos dentro del margen), de ahí que se llame margen suave, por lo que la tarea de aprendizaje puede formularse como:

- Minimizar la función de costo

$$\frac{1}{2}\|W\|^2 + C\sum_{i=1}^{n}\xi_i,$$

sujeto a $y_i(W^\top X_i + b) \leq 1 - \xi_i$, para $i = 1, ..., m$ y $\xi_i \leq 0$ para cada $i = 1, ..., m$. Las variables ξ_i representan la magnitud del error para el ejemplo i, denominadas *slack variables* y C es el parámetro de regularización controla el tamaño del margen para aceptar errores, donde un alto valor de C se enfoca en minimizar los errores de clasificación, lo cual resulta en un margen más estrecho, mientras que un valor de C más pequeño indica un margen más amplio, permitiendo

más errores de clasificación en los datos.

2.3.6. Descenso de Gradiente Estocástico

El Descenso de Gradiente Estocástico (SGD por sus siglas en inglés) mencionado en la sección 2.2.2, puede ser utilizado para minimizar las funciones de costo de algunos algoritmos previamente descritos. Específicamente, SGD es un clasificador lineal implementado en **scikit-learn**[2]que utiliza el algoritmo de optimización *Stochastic Gradient Descent* para entrenar modelos lineales como la regresión logística, SVM con kernel lineal o regresión lineal. Es particularmente útil para problemas de clasificación en los que se necesita manejar grandes volúmenes de datos. Entre sus principales parámetros se pueden encontrar los siguientes:

- **loss**: Define la función de pérdida a utilizar. Las opciones 'hinge' (para SVM lineal), 'log' (para regresión logística), y 'squared loss' (para regresión lineal).
- **penalty** : Tipo de regularización ('l2', 'l1', 'elasticnet'). l2 inidica Ridge Classifier, l1 Lasso y Elasticnet una combinación de ambos.
- **alpha**: Coeficiente de regularización. Controla la importancia de la penalización en el modelo.
- **max_iter**: Número máximo de iteraciones sobre los datos.
- **learning_rate**: Estrategia de tasa de aprendizaje, como 'constant', 'optimal', o 'invscaling'.
- **tol**: Tolerancia para detener las iteraciones cuando la mejora en la función objetivo es menor que este valor.

Cabe destacar que **scikit-learn** provee Ridge Classifier, SVC, Regresión Logística y SGD CLassifier, sin embargo, algunas diferencias en su uso es la manera en que se minimiza la función de costo, como por ejemplo la Regresión Logística puede ajustar sus parámetros con el método de *máxima verosimilitud* y converger rápidamente a la solución global en conjunto de datos pequeños, mientras que el SGD Classifier puede manejar mejor cantidades más grandes de datos. En resumen, la elección entre estos depende del tamaño de los datos, la necesidad de escalabilidad, y la preferencia por la estabilidad y precisión frente a la eficiencia y flexibilidad, ya que cada uno de estos está optimizado según el enfoque que se requiera.

[2]Scikit-learn es una biblioteca de Python ampliamente utilizada para aprendizaje automático. Está diseñada para facilitar la implementación de algoritmos de machine learning: `https://scikit-learn.org/stable/index.html`.

2.3.7. Árboles de decisión

Los Árboles de decisión o *Decision Trees* son algoritmos de aprendizaje máquina muy versátiles e intuitivos, inclusive conocidos como clasificadores de *caja blanca*, ya que es posible conocer la manera en la que llevan a cabo la clasificación de instancias. Una forma bastante intuitiva de explicarlos es como el ir realizando una serie de preguntas e ir construyendo el árbol a partir de estas. Por ejemplo, si se quisiera construir un árbol para clasificar animales que se van encontrando en un recorrido, este se podría construir como el de la Figura 24.

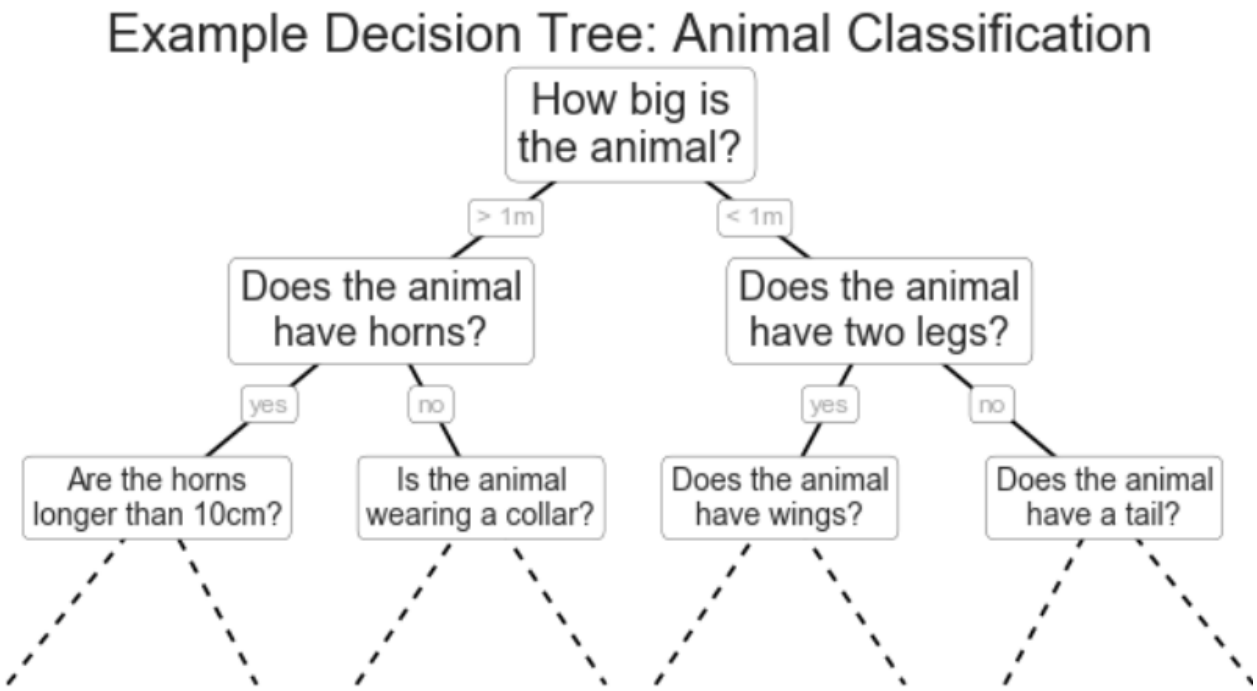


Figura 24. De arriba a bajo tenemos: ¿Qué tan grande es el animal? Se define un umbral de 1 metro, por lo que si es menor surge la pregunta ¿Tiene cuernos? (abajo izquierda) y dependiendo si los tiene o no, surgen más preguntas, pero si es mayor o igual, surge la pregunta ¿Tiene dos piernas? (abajo derecha) y dependiendo si las tiene o no, surgen más preguntas. Claramente, el objetivo es ubicar al animal dentro de un conjunto de animales que cumplan con los criterios impuestos por las preguntas. Figura adaptada de VanderPlas (2016).

En un hipotético bien construido árbol de decisión, cada pregunta reduciría el número de opciones a la mitad, por lo que la importancia recae en decidir qué o cuáles preguntas son las adecuadas para ir obteniendo esta reducción. En el aprendizaje máquina, las preguntas que se realizan toman la forma o se interpretan como las divisiones de los datos alineados con los ejes (características), es decir, cada nodo del árbol divide los datos en dos grupos, realizando el corte dentro de una de las características. Los datos se dividirán iterativamente a lo largo de alguno de los ejes, según se establezca un criterio cuantitativo, y en cada nivel asignará la etiqueta de la nueva región según una mayoría de votos de puntos dentro de

ella. Una observación importante es que conforme más se aumenta la profundidad (*depth*) se obtienen particiones de los datos cada vez más 'raras', lo cual es un claro indicador de sobreajuste y el principal problema al usar este clasificador, pues este modelo se sobreajusta fácilmente a los datos.

El *dataset* **Iris**[3]es un conjunto de datos que contiene información de tres tipos de flores de iris: Setosa, Versicolor y Virginica. Este *dataset* contiene las longitudes de sus sépalos y pétalos (características), por lo que un árbol de decisión aplicado a este conjunto de datos puede verse en la Figura 25.

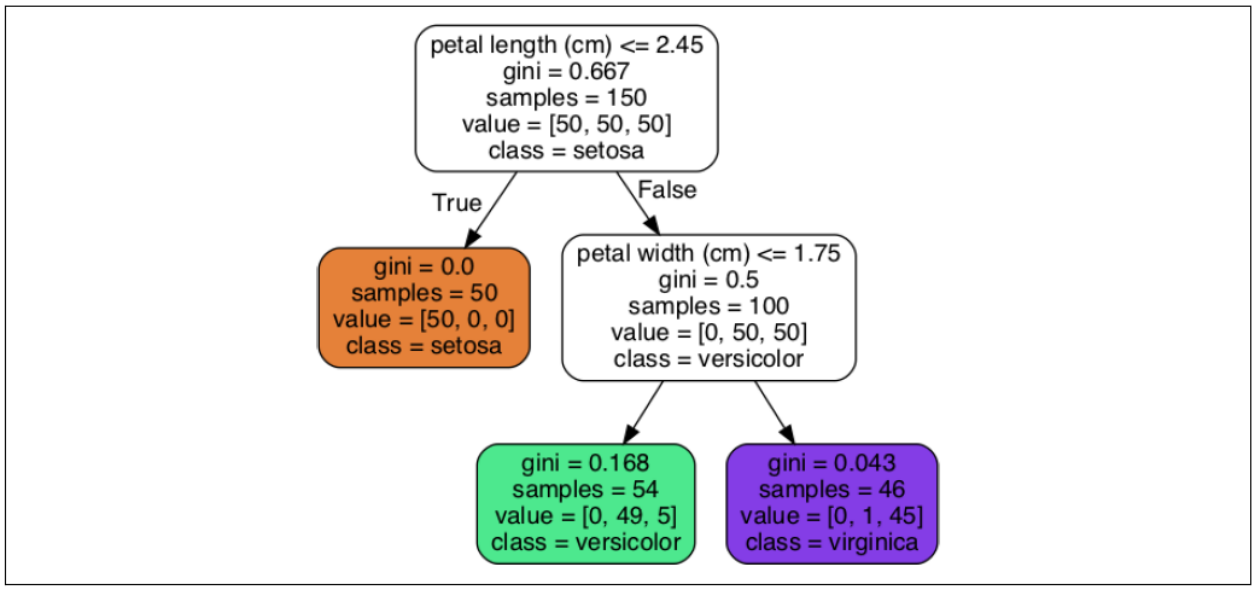


Figura 25. Árbol de Decisión formado mediante el *dataset* **Iris**. Figura adaptada de Géron (2019).

Se puede explicar la forma en la que se hacen predicciones mediante la Figura 25. Si se tiene como entrada una flor iris y se quiere clasificar basándose en sus pétalos, el nodo que se encuentra en lo más alto $(depth = 0)$, llamado nodo raíz, pregunta si la longitud de su pétalo es más pequeña que 2.45 cm, y en caso de serlo, entonces nos fijamos ahora en el nodo hijo del lado izquierdo $(depth = 1)$. Observe que ahora este nodo es un nodo hoja, es decir que no tiene hijo y por ende no hay más preguntas que hacer, por lo que el árbol de decisión predice o clasifica la flor como Iris Setosa. Observe que de haber sido la longitud de su pétalo mayor o igual a 2.45 cm, se habría hecho una segunda pregunta en relación a la misma longitud pero ahora comparando con 1.75 cm y dependiendo la respuesta se clasificaría como Iris Versicolor $(depth = 2, left)$ o Iris Virginica $(depth = 2, right)$. En la Figura 26 se pueden observar las fronteras de decisión con respecto a este *dataset*.

Las preguntas realizadas son representadas por los nodos del árbol y se busca que la forma de construirlos pueda cree la mejor separación de las clases. Los nodos se dividen en función de métricas que buscan

[3]`https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html`.

crear esta separación, por lo que las más comunes son:

- **Gini**: Para clasificación mide la impureza del i-ésimo nodo como sigue

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2, \tag{15}$$

donde $p_{i,k}$, es el cociente de instancias de la clase k entre las instancias de entrenamiento en el i-ésimo nodo.

- **Entropía**: Otra medida de impureza para clasificación en el i-ésimo nodo es la siguiente

$$H_i = -\sum_{k=1}^{n} p_{i,k} log_2(p_{i,k}), \tag{16}$$

donde $p_{i,k}$, es el cociente de instancias de la clase k entre las instancias de entrenamiento en el i-ésimo nodo y distinto de 0.

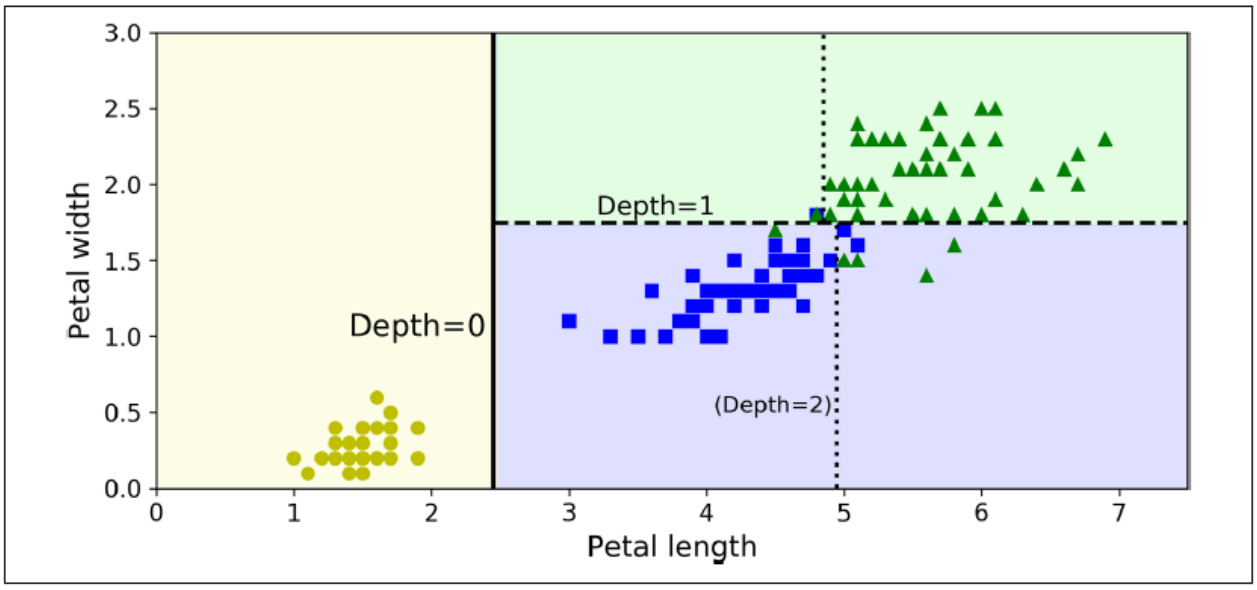


Figura 26. Particiones del *dataset* **Iris**. Figura adaptada de Géron (2019).

Cabe destacar que **scikit-learn** usa el algoritmo **CART**, el cual produce solamente árboles binarios, es decir, los nodos que no son hoja siempre tienen dos hijos, lo cual se interpreta como una respuesta binaria de sí o no a las ´preguntas' realizadas, sin embargo, existen algoritmos como el **ID3** que pueden construir árboles de decisión con nodos que tengan más de dos hijos. El algoritmo **CART** trabaja dividiendo el conjunto de entrenamiento en dos subconjuntos usando alguna característica k y un umbral t_k. Este

algoritmo busca la terna (k, t_k), que produce los subconjuntos más puros, es decir, ponderados por su tamaño. La función de costo del algoritmo CART para clasificación es la siguiente

$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}, \tag{17}$$

donde $G_{left\backslash right}$ mide la impureza del subconjunto $left\backslash right$ (izquierdo o derecho) y $m_{left\backslash right}$ es el número de instancias en el subconjunto $left\backslash right$ (izquierdo o derecho).

Este proceso de minimización se hace mediante recursión en cada nodo hijo, es decir, el árbol sigue creciendo y dividiendo los datos hasta que se cumplen ciertos criterios de detención, como por ejemplo:

- El nodo se vuelve completamente puro (es decir, Gini $= 0$).
- El número de instancias en un nodo cae por debajo de un umbral mínimo.
- Se alcanza una profundidad máxima del árbol.

2.4. Aprendizaje por Ensamble

El Aprendizaje por Ensamble o *Ensemble Learning*, es una técnica utilizada para agrupar (ensamblar) un conjunto de algoritmos de aprendizaje máquina. Con respecto a los de clasificación, un grupo o conjunto de clasificadores es llamado un *ensamble*. Se ha visto que en muchos experimentos estas técnicas han ayudado a aumentar la tasa de clasificación que con el solo uso individual de cada clasificador.

2.4.1. Clasificadores por Votos

La clasificación por votos o votación, funciona como coloquialmente conocemos las votaciones, es decir, gana o se elige la persona, objeto o cosa que tiene más votos y en el contexto de aprendizaje máquina, hay dos formas de hacer esto.

Suponga un problema de clasificación de tres clases: Manzana (M), Naranja (N) y Pera (P), contando con tres clasificadores: C1, C2 y C3. Suponga que C1 toma una instancia x_1 y la clasifica con P, C2 la

clasifica como P y C3 la clasifica como N, entonces, la clasificación final sería Pera (P). Este tipo de votación se conoce como *hard voting*.

Suponga los mismos tres clasificadores mencionados anteriormente, pero capaces de generar la probabilidad de que cada instancia de entrada pertenezca a cada una de las clases con las que se estén trabajando, Manzana (M), Naranja (N) y Pera (P) en este caso, entonces, otra manera de obtener la clasificación final correspondiente a la instancia de entrada es promediar las probabilidades correspondientes a cada una de las clases generadas por cada uno de los clasificadores asociadas con la instancia de entrada correspondiente, por tanto, la clasificación final resulta de tomar la mayor probabilidad resultante después de promediar todas las probabilidades. Este tipo de votación se llama *soft voting* y la Tabla 2 muestra un ejemplo de esta forma de clasificación.

Tabla 2. Ejemplo de *soft voting* para una instancia de entrada x_1 y sus probabilidades (**Probs.**) asociadas (CDD indica criterio de desempate).

Clasificadores	**C1**			**C2**			**C3**			**Promedio**		
Clases	M	P	N	M	P	N	M	P	N	M	P	N
Probs.	0.2	0.39	0.41	0.2	0.5	0.3	0.2	0.4	0.4	0.2	0.43	0.37
Clasificación	Max:	0.41	N	Max:	0.5	P	Max:	0.4	CDD	Max:	0.43	P

Es importante tener en cuenta las siguientes consideraciones:

- No todos los clasificadores, por defecto, pueden generar probabilidades asociadas con alguna instancia, de ahí que no siempre es posible realizar *soft voting*.
- El ejemplo previo muestra una forma de clasificación multiclase, es decir, cada clasificador regresa un vector de longitud: el número de clases con las que se esté trabajando, donde la suma de los elementos da como resultado 1. Note que la clasificación final se obtiene al tomar la clase con mayor probabilidad.
- Cabe resaltar que hay varias formas de realizar clasificación múltiple (no binaria) y una de ellas es como la que se describió previamente, aunque observe que no se describió el cómo se obtuvieron estas clasificaciones.
- Como se muestra en la Fila **Clasificación**, décima columna de la Tabla 2, CDD indica 'Criterio De Desempate', es decir, las probabilidades asociadas pueden resultar ser las mismas, por lo que es adecuado manejar un criterio de desempate.

- El *hard voting* se puede realizar siempre y cuando se maneje un criterio de desempate al utilizar un número par de clasificadores, pues la votación resultante podría resultar en empate.
- El ejemplo previo muestra como sería la metodología al usar *hard* y *soft voting* en las redes neuronales convolucionales (CNNs) en problemas multiclase o binario con *activavión Softmax* en la última capa perteneciente a la de clasificación, mientras que en clasificación binaria con *activación Sigmoide* el proceso es similar, ya que se promediarían las probabilidades y luego se aplicaría el umbral para obtener la clase final (Ver sección 2.3.3).

2.4.2. Bagging y Pasting

En general, una recomendación al combinar o ensamblar varios clasificadores es entrenarlos en diferentes conjuntos de datos, ya que cada clasificador comete errores distintos, sin embargo, esto requiere más datos. Otro enfoque que se puede seguir es usar el mismo clasificador (**clasificador base**), pero en vez de entrenarlo en todo el *dataset*, este se entrena en diferentes muestras (subconjuntos) aleatorias del *dataset*. Cuando el muestreo es hecho con reemplazo, este método es llamado *Bagging*. Cuando el muestreo es hecho sin remplazo, este método es llamado *Pasting*. Cabe destacar que la predicción final es llevada a cabo promediando el clasificador base entrenado en cada subconjunto muestral del *dataset* de entrenamiento en el caso de regresión, mientras que la clasificación final es tomada como la más frecuente, así como un *hard voting*. Uno de los algoritmos más usados y populares es el Bosque Aleatorio (*Random Forest)*, siendo este un modelo Bagging que lleva como clasificador base un Árbol de Decisión.

Al usar el modelo Bagging, muchas instancias pertenecientes al conjunto de entrenamiento podrían ser muestreadas varias veces mientras que otras podrían nunca serlo. Esto puede ser aprovechado, ya que el conjunto generado por las instancias que no fueron muestreadas podría ser usado como un **conjunto de validación**, cuyo significado se describe en posteriormente. Además, es posible muestrear características, es decir, si tenemos una matriz de características de tamaño $m \times n$, entonces es posible generar submatrices o parches aleatorios de tamaño $M \times N$, donde $1 < M < m$ y $1 < N < n$. Esta última técnica puede ser útil al manejar *datasets* con una alta dimensionalidad, tales como imágenes. El predictor Bagging Classifier perteneciente a la librería de **scikit-learn** trabaja de la siguiente forma. Considere un *dataset* de 100 instancias $(m = 100)$ y 10 características $(n = 10)$, es decir, una matriz de características de tamaño 100×10, en donde las 100 instancias son $x_1 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$, $x_2 = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2]$, así hasta $x_{100} = [100, 100, 100, 100, 100, 100, 100, 100, 100, 100]$ y defínase

5 predictores, esto a su vez genera 5 subconjuntos muestrales, donde un subconjunto le corresponde a un solo predictor como conjunto de entrenamiento, entonces, si no se muestrean las características (n se mantiene fijo), los 5 subconjuntos muestrales generan 5 matrices de características del mismo tamaño que la original, pero la instancia en el lugar 20, por seguir con este ejemplo, no sería $x_{20} = [20, 20, 20, 20, 20, 20, 20, 20, 20, 20]$, sino que podría ser alguna otra de las instancias pertenecientes al conjunto de entrenamiento (estar repetida) y podría la instancia x_{20} ni siquiera aparecer, esto siendo posible para cada uno de los 5 conjuntos muestrales.

2.4.3. Boosting

El método Boosting se refiere a cualquier método de ensamble que combine varios predictores débiles para generar uno más fuerte. La idea detrás del Boosting es entrenar predictores de manera secuencial de tal manera que el sucesor corrija a su predecesor. De los más populares e importantes son AdaBoost y Gradient Boosting.

Una manera de que un nuevo predictor corrija a su predecesor es enfocarse en las instancias utilizadas durante el entrenamiento que el predecesor no es capaz de predecir correctamente, dando como resultado un predictor enfocado más en este tipo de instancias. Esta técnica es utilizada por el algoritmo AdaBoost.

Cuando se realiza el entrenamiento del AdaBoost, primero se entrena el predictor base, creando predicciones sobre el conjunto de entrenamiento. Después AdaBoost calcula un peso para este clasificador en función de su precisión. Posteriormente el algoritmo incrementa los pesos a las instancias mal clasificadas y un segundo clasificador vuelve a ser entrenado utilizando estos pesos actualizados y otra vez vuelve a hacer predicciones sobre el conjunto de entrenamiento y el proceso se repite. Los clasificadores que cometen menos errores (instancias mal clasificadas) reciben un mayor peso en el proceso de agregación final. Una vez que todos los predictores son entrenados, este ensamble crea las predicciones finales mediante una combinación ponderada de los modelos base. Los pesos de los clasificadores dependen de su error r_j. Si un clasificador tiene un bajo error, se le asigna un mayor peso en la predicción final.

Primero, se relaciona un peso w^i con cada instancia, inicialmente con un valor de $\frac{1}{m}$. Después de entrenar el primer predictor, la tasa de error ponderada r_1 es calculada sobre el conjunto de entrenamiento. La ecuación 18 muestra tasa de error ponderada para el j-ésimo predictor r_j.

$$r_j = \frac{\sum_{\substack{i=1 \\ \hat{y}_j^i \neq y^i}}^{m} w^i}{\sum_{i=1}^{m} w^i}, \tag{18}$$

donde $\hat{y}_j^i$ es la predicción de j-ésimo predictor para la i-ésima instancia.

El peso del predictor α_j es calculado con la ecuación 19

$$\alpha_j = \eta log \frac{1 - r_j}{r_j}, \tag{19}$$

donde η es un hiperparámetro que representa la tasa de aprendizaje, cuyo valor por default suele ser 1.

Después, el algoritmo AdaBoost actualiza el pesos de las instancias usando la ecuación 20, potenciando los pesos de las instancias mal clasificadas.

$$w^i \leftarrow \begin{cases} w^i & si \quad \hat{y}_j^i = y^i \\ w^i e^{\alpha_j} & si \quad \hat{y}_j^i \neq y^i \end{cases}, \tag{20}$$

Lo que sigue es normalizar todos los pesos de las instancias al dividirlas por $\sum_{i=1}^{m} w^i$.

Por último, un nuevo predictor es entrenado usando los nuevos pesos y todo el proceso es repetido. Cabe destacar que este proceso por lo regular termina cuando el número deseado de predictores es alcanzado.

La clase predicha es la que recibe la mayoría de los votos ponderados (ecuación 21)

$$\hat{y}(x) = \arg\max_k \sum_{\substack{j=1 \\ \hat{y}_j(x)=k}}^{N} \alpha_j, \tag{21}$$

donde N es el número de predictores.

El otro algoritmo de Boosting popular es Gradient Boosting. Al igual que AdaBoost, este algoritmo también trabaja añadiendo predictores secuencialmente (ensamble), pero cada uno corrigiendo a su predecesor. Sin embargo, este algoritmo entrena al nuevo predictor con los errores residuales del predictor previo. Una forma más clara de entender este proceso es con un ejemplo de regresión.

Suponga que se tiene un Árbol de Decisión en un problema de regresión, entonces al entrenar este algoritmo con el conjunto de entrenamiento X, se ajustarían sus parámetros para posteriormente calcular

o definir la resta (residuo) $y_2 = y - \text{Pred1}(X)$, donde $\text{Pred1}(X)$ indica las predicciones del Árbol de Decisión en el conjunto de entrenamiento. Después, se entrena un segundo Árbol de Decisión ajustando sus parámetros con el mismo conjunto de entrenamiento X, pero con los *targets* y_2. Después se calcula la resta $y_3 = y_2 - \text{Pred2}(X)$, donde $\text{Pred2}(X)$ indica las predicciones de segundo Árbol de Decisión en el conjunto de entrenamiento y el proceso se repite sucesivamente (Figura 27).

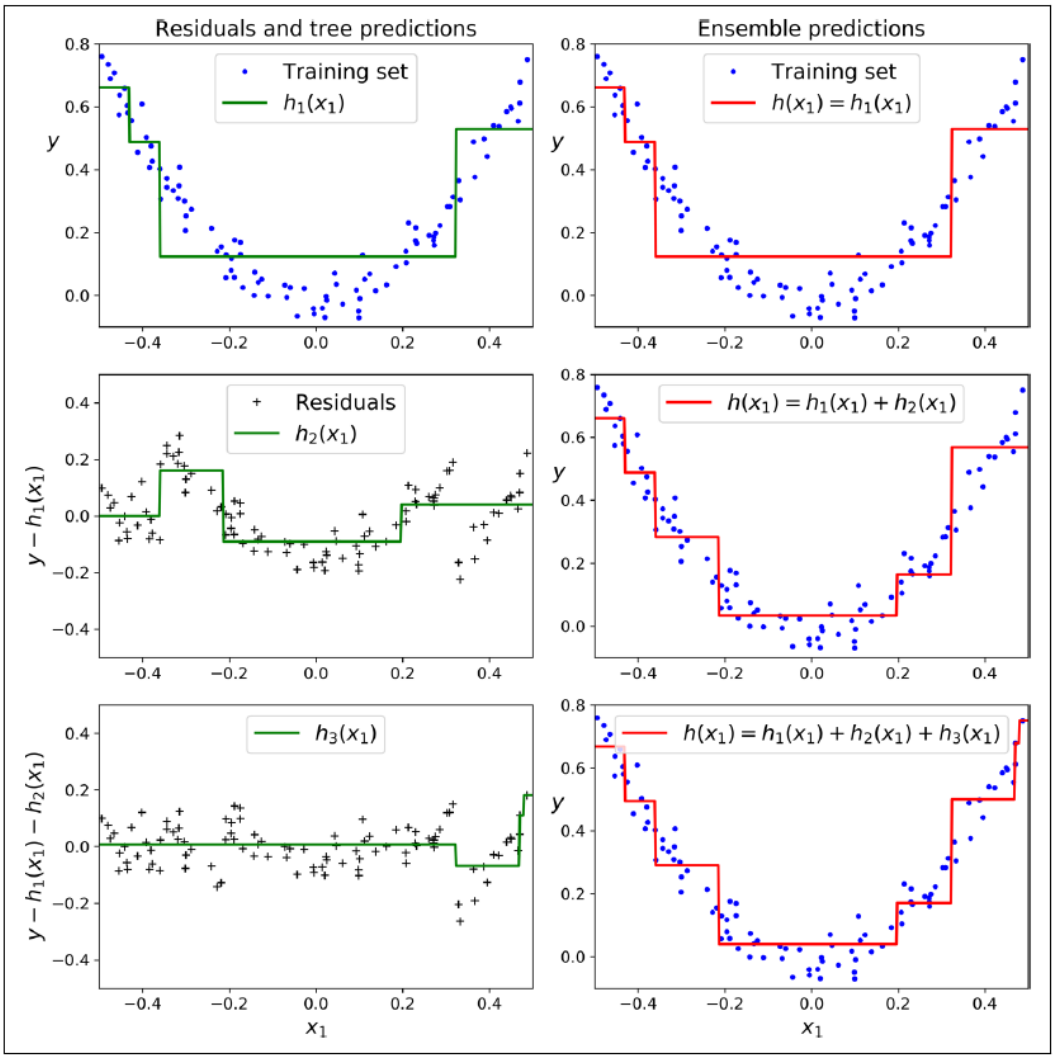


Figura 27. Representación del Gradient Boosting. Primero el predictor es entrenado (parte superior izquierda), después los predictores consecutivos son entrenados en los residuos de los predictores previos (parte en medio e inferior izquierda), mientras que la columna derecha muestra el ajuste por ensamble de las predicciones a los datos reales. Figura adaptada de Géron (2019).

Cabe mencionar que se le puede indicar a este algoritmo cuando detenerse, ya sea especificando un número de estimadores, una parada anticipada al notar que no se mejora, conocida como *early stopping*, o al llegar a una cierta tolerancia impuesta sobre en la función de pérdida.

2.4.4. Stacking

Los términos agregación (*aggregation*) y función de agregación (*aggregation function*) en el contexto de estos métodos de ensamble se refieren a la manera de combinar las predicciones de los diferentes modelos base para llegar a una predicción final. Anteriormente se describieron ya algunas como lo son el voto por mayoría o *hard voting*, promediando las probabilidades asociadas, así como el usado por el modelo AdaBoost, el cual considera los pesos asociados con cada clasificador.

El Stacking también genera una agregación, solamente que de manera diferente en comparación con los modelos anteriores. En Stacking el proceso de agregación se realiza mediante un modelo de nivel superior o meta-modelo, también conocido como *meta-learner* o meta-clasificador.

De manera sencilla la agregación en Stacking se puede explicar cómo sigue.

1. **Predicciones de modelos base o Nivel 1**
    - Varios modelos base se entrenan en el conjunto de datos de entrenamiento, como los descritos anteriormente: árboles de decisión, SVM, etc.
    - En lugar de tomar sus predicciones finales como la salida directa, las predicciones de estos modelos se utilizan como nuevas características para entrenar el *meta-learner*, este conjunto nuevo o nueva matriz de características es llamada *blending set*.
2. **Meta-modelo o Nivel 2**
    - El meta-modelo o meta-clasificador recibe como entrada (conjunto de entrenamiento) las predicciones de los modelos base y los *targets* siguen siendo los mismos.
    - Este meta-modelo aprende a agregar las predicciones de los modelos base de una manera más sofisticada, encontrando patrones y relaciones entre las salidas de los modelos base y los *targets*. Este modelo suele ser un clasificador o regresor simple como una regresión logística o regresión lineal.

Como ejemplo considere nuevamente el problema de clasificación de tres clases: Manzana (M), Naranja (N) y Pera (P), contando con los mismos tres clasificadores C1, C2, C3 capaces de generar probabilidades asociadas con cada una de las clases, así como se describieron en la sección 2.4.1. Si no nos interesa conocer su capacidad o qué tan buenos son clasificando sino solamente usar sus predicciones para

construir el *blending set*, entonces la Tabla 3 ilustra estas predicciones para cinco instancias x_1, x_2, x_3, x_4 y x_5, mientras que la Figura 28 muestra los posibles meta-clasificadores utilizados para obtener la clasificación final para cada una de las instancias utilizando el *blending set*.

Tabla 3. Clasificadores e instancias utilizados para construir el *blending set*.

Clasificadores:	**C1**			**C2**			**C3**		
Clases:	M	P	N	M	P	N	M	P	N
Probabilidad asociada con x_1:	0.3	0.4	0.3	0.1	0.8	0.1	0.71	0.2	0.09
Probabilidad asociada con x_2:	0.71	0.2	0.09	0.3	0.4	0.3	0.8	0.1	0.1
Probabilidad asociada con x_3:	0.45	0.25	0.3	0.6	0.1	0.3	0.3	0.4	0.3
Probabilidad asociada con x_4:	0.3	0.4	0.3	0.71	0.2	0.09	0.25	0.55	0.2
Probabilidad asociada con x_5:	0.2	0.3	0.5	0.4	0.05	0.55	0.3	0.4	0.3

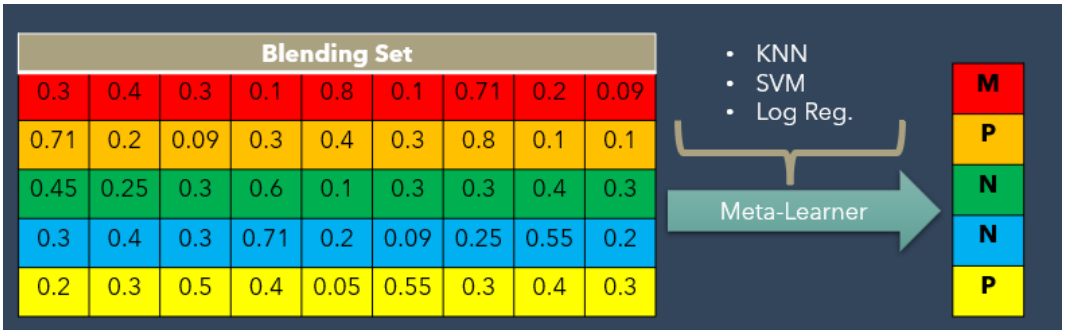


Figura 28. Matriz de características nueva conformada por las predicciones de los clasificadores de la Tabla 3 junto con algún meta-clasificador: KNN, SVM o Regresión Logística (Log Reg), el cuál provee la clasificación final para la instancia de entrada.

Algunas consideraciones al usar el Stacking son las siguientes:

- Conviene usar más este método con modelos capaces de generar probabilidades.
- Al menos hay dos enfoques para llevarlo a cabo:

  1. El usado previamente. Teniendo un conjunto de entrenamiento X_1, se generan las predicciones de los modelos base para generar el *blending set* sin conocer de antemano qué tan 'buenos' son estos modelos. Una vez generado el *blending set*, se utiliza el *meta-learner* para ajustar sus parámetros (entrenarlo) con el *blending set* para que este dé la predicción final.
  2. Otra forma de realizar el Stacking es entrenar los modelos base con un conjunto de entrenamiento X_1, evaluarlos para conocer su capacidad y después utilizar otro conjunto de datos

X_2 para generar las predicciones con este conjunto y así construir el *blending set*. Por último e igual que en el punto 1, se entrena en el *blending set* un *meta-learner*.

- Es posible construir otra capa o Nivel 3. Para esto se tendrían que utilizar varios *meta-learners* en el Nivel 2, los cuales harían el papel de clasificadores base como los del Nivel 1, después pasarían sus predicciones a un último *meta-learner* situado en el Nivel 3 para que este genere la predicción final.

- Si se puede observar, la principal desventaja en seguir el enfoque de construir una tercer capa o el enfoque de usar dos conjuntos de datos, X_1 y X_2, son los datos, ya que se recomienda usar distintos datos para entrenar el *meta-learner*, y como se sabe, no siempre es tan fácil obtener más datos, por lo que no siempre es posible construirlo de esta forma.

Para terminar conviene hacer las siguiente observaciones. Con respecto a las votaciones: si se tienen n clasificadores, ¿de cuántas formas distintas se pueden combinar los clasificadores para obtener distintos resultados al usar votaciones? Por ejemplo, considere 3 clasificadores: $C1$, $C2$ y $C3$, observe que se pueden combinar para obtener los clasificadores por votos siguientes: $V1 = \{C1, C2\}, V2 = \{C1, C3\}, V3 = \{C2, C3\}, V4 = \{C1, C2, C3\}$. De manera general, si se tienen n clasificadores distintas, se pueden generar $2^n - (n+1)$ votaciones distintas. Si ahora se considera el Stacking surge una duda similar: si se tienen n clasificadores, ¿cuántos *blending sets* distintos se pueden formar? Observe que con los mismos 3 clasificadores: $C1$, $C2$ y $C3$, los *blending sets* formados serían los siguientes: $B1 = \{P1, P2\}, B2 = \{P1, P3\}, B3 = \{P2, P3\}, B4 = \{P1, P2, P3\}$, donde $P1, P2$ y $P3$ son las predicciones de cada clasificador para generar los *blending sets*, por lo que la respuesta es la misma. Esto conviene tenerlo en cuenta, ya que más adelante se usarán las combinaciones de distintos clasificadores.

2.5. Métricas y validación

Algunas de las métricas de evaluación más comunes usadas para clasificación son las siguientes.

Matriz de Confusión. Un clasificador binario puede clasificar una instancia en alguna de las dos clases con las que fue entrenado, sin embargo, esto no significa que lo haga con un $100\,\%$ de **exactitud** (*accuracy*), por lo que hay siempre una cierta confusión en asignar a las instancias las clases correspondientes (verdaderas). Por tanto, la matriz de confusión existe como una representación visual de esta confusión y puede ilustrarse como una matriz de tamaño 2×2 (cuatro cuadrantes) en donde aparecen el número

de las instancias clasificadas por clase. Las verdaderas etiquetas se posicionan en la parte izquierda de la matriz, mientras que las que son predichas se localizan en la base o parte inferior (Figura 29). El cuadrante ubicado en la parte superior izquierda es definido como verdadero negativo (TN), el cuadrante ubicado en la parte inferior izquierda es definido como falso negativo (FN), el cuadrante superior derecho se define como falso positivo (FP), mientras que el cuadrante ubicado en la parte inferior derecha es definido como verdadero positivo (TP). De estas definiciones varias métricas surgen.

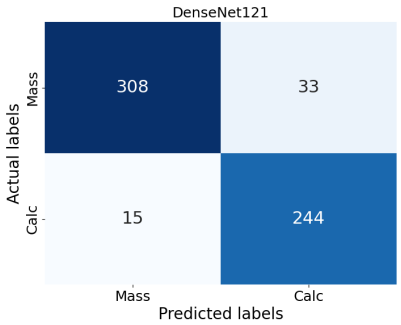


Figura 29. Ejemplo de una matriz de confusión. En este caso, el clasificador es una CNN, la tarea de clasificación se basa en distinguir Masas de Calcificaciones. En la parte izquierda (Actual labels) se representan las etiquetas verdaderas , las que son Masas y Calcificaciones, mientras que en la base se muestran las predicciones generadas por este modelo (Predicted labels) y en general, por cualquier clasificador.

Accuracy (Acc). Esta es la métrica más comúnmente usada y mide el número de instancias correctamente clasificadas. Este número es el resultado de la suma de la diagonal (TN+TP) dividida entre el número total de instancias clasificadas.

$$Acc = \frac{TN + TP}{TN + FN + FP + TP} \tag{22}$$

Precisión. Esta medida es definida como la exactitud de predicciones positivas.

$$Precision = \frac{TP}{TP + FP} \tag{23}$$

Recall. También conocida como Sensitividad o tasa de verdaderos positivos (TPR). Esta métrica mide la proporción de instancias positivas correctamente detectadas por el clasificador.

$$Recall = \frac{TP}{TP + FN} \tag{24}$$

Coeficiente de Correlación de Matthews. Este coeficiente de correlación (**MCC** por sus siglas en inglés) varia de -1 a 1, donde 1 indica el mejor acuerdo entre datos reales y predicciones. Este toma en cuenta los cuatro cuadrantes y es particularmente útil para *datasets* desbalanceados.

$$MCC = \frac{TN \times TP - FN \times FP}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{25}$$

F1-Score (F1). Esta métrica combina la Precisión y Sensitividad en una simple medida. El mejor valor posible es 1, mientras que el peor es 0.

$$F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{26}$$

Cabe destacar que aunque las métricas previamente descritas se basan en un clasificador binario, esta misma representación existe para un clasificador multiclase, pero con más cuadrantes en la matriz de confusión, lo cual resulta en una descripción más compleja. Además, siempre es posible definir métricas distintas a las descritas previamente, ya que por lo regular depende del problema con el que se esté trabajando.

Una vez que un modelo ha ajustado sus parámetros con el conjunto de entrenamiento[4], este ya está listo para generar predicciones, sin embargo, surge la duda de conocer qué tan capaz o confiable es de generar correctamente estas predicciones, por lo que una manera de observar su capacidad es la de evaluarlo con los mismos datos con lo que fue entrenado y con las métricas ya previamente descritas, sin embargo, aunque se obtenga un 100 % de exactitud en su evaluación esto se considera una mala práctica, ya que lo ideal es someter a evaluación este modelo con datos no vistos, pues el fin o el propósito de crear un modelo es para utilizarlo para la tarea para la que fue creado, por lo que siempre estará recibiendo datos nuevos, es decir, su alcance no se limita solo al conjunto de entrenamiento sino a nuevos datos, por lo que si se caen en esta mala práctica, aún teniendo una alta exactitud, puede ocurrir que al ingresarle nuevos datos su desempeño sea bajo, esto se conoce como sobreajsute.

Sobreajuste. El sobreajuste u *over-fitting* en inglés, es un término utilizado en aprendizaje máquina y este ocurre cuando un modelo se ajusta demasiado bien a los datos de entrenamiento, capturando pobremente patrones generales, además de ruido, por lo tanto, surge la necesidad de utilizar otro conjunto de datos

[4]Los modelos paramétricos necesitan de un conjunto de datos para ajustar sus parámetros y así poder generar predicciones, aunque no todos los modelos lo son, tal es el caso del clasificador KNN.

(distinto con el que fue entrenado) con el fin de evaluar su desempeño, de ahí que han surgido varias técnicas de validación.

Como se mencionó antes, la forma de evaluar un modelo es utilizar un conjunto ajeno al conjunto de entrenamiento, conocido como conjunto de prueba o *Test set* o simplemente *Test*, por lo que una forma de evaluarlo usando el *Test* es la siguiente:

1. Primero se entrena el modelo con el conjunto de entrenamiento. Recuerde que se usa la matriz de características y sus *targets* asociados para ajustar los parámetros del modelo.
2. La matriz de características generada por el *Test* (este conjunto también tiene *targets* asociados) es utilizada para generar predicciones.
3. Las predicciones son comparadas con los *targets* del *Test* y es aquí en donde surgen las métricas y la matriz de confusión como la de la Figura 29. Si las predicciones resultan ser todas iguales a los *target* (es sumamente raro que esto pase), entonces el clasificador obtendrá un 100 % de *accuracy*.

Cabe mencionar que esta forma de evaluar el modelo se llama *Hold-out validation* o simplemente *Hold-out*. Observe que esto requiere de dos conjuntos de datos: entrenamiento y prueba(*Testing*), es decir, esta validación requiere de más datos y como se ha mencionado, no siempre es fácil obtenerlos, además, es necesario mencionar que el concepto de conjunto de validación, como tal no existe, ya que un segundo enfoque que surge es el de dividir el conjunto de entrenamiento en dos subconjuntos, uno que tenga, al menos por lo visto en la práctica, un 80-85 % de los datos, el cual será utilizado para entrenamiento, mientras que el resto será utilizando como *testing*, pero en la práctica este suele llamarse conjunto de validación, más aún, pueden tenerse un conjunto de entrenamiento, un conjunto de *Test* y aún así generar un conjunto de validación a partir del conjunto de entrenamiento, por lo cuál, a veces se dice que el conjunto de validación no existe, sino que es generado a partir del de entrenamiento.

Representación de las curvas de exactitud (accuracy) y pérdida (loss). Cuando se entrena un modelo de red neuronal se pueden crear dos curvas durante el entrenamiento, una relacionada con la exactitud (*accuracy*) y otra con la pérdida *loss*. Estas representan cómo cambia la exactitud (*accuracy*) o la pérdida (*loss*) durante el entrenamiento del modelo (rendimiento del modelo) en el conjunto de datos de entrenamiento (datos que ha "visto" directamente) para ajustarse.

Accuracy de entrenamiento. El modelo calcula la precisión en cada lote durante el entrenamiento, y al final de cada época, promedia los resultados obtenidos en todos los lote de esa época para obtener el

valor de training accuracy.

Loss de entrenamiento. De manera similar, la pérdida de entrenamiento es calculada y promediada, en base a la función de costo en cada lote y luego al final de cada época se obtiene la pérdida promedio.

Además, este mismo tipo de curvas puede pueden crearse para un conjunto de validación. Estas presentan el rendimiento del modelo en este conjunto en cada época. Este conjunto no es utilizado directamente para entrenar el modelo, lo que permite evaluar su generalización. Básicamente estas indican qué tan bien generaliza (clasifica) el modelo en datos 'no vistos', por lo tanto, lo más recomendable es utilizar este conjunto de validación y plotear ambas curvas, las correspondientes al conjunto de entrenamiento y validación. Lo que se espera es que el *accuracy* de entrenamiento aumente y tienda a 1, mientras la pérdida de entrenamiento se espera que disminuya y tienda a 0. A su vez, se espera un aumento en la precisión y una disminución en la pérdida en el conjunto de validación indican que el modelo también está generalizando bien con los datos no vistos, es decir, ambas curvas en el conjunto de validación deben comportarse lo más parecido a las del entrenamiento (Figura 30).

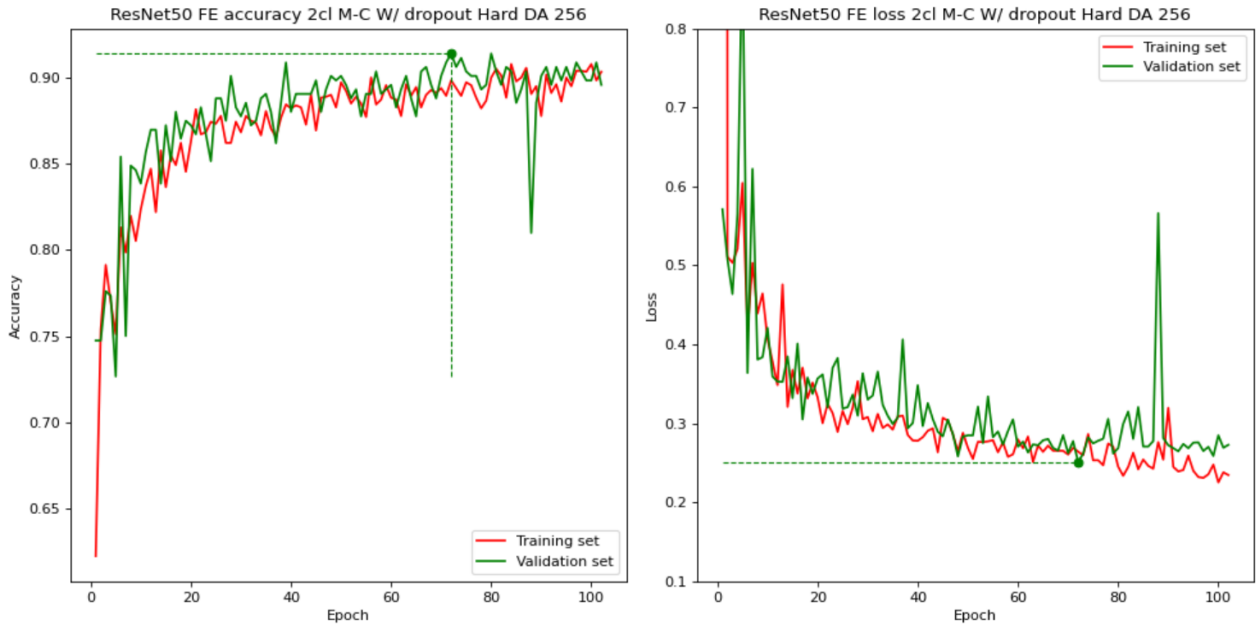


Figura 30. Entrenamiento del Modelo ResNet50 en la clasificación de masas y calcificaciones. Se muestran las curvas de entrenamiento en rojo y las de validación en verde. Observe que la curva de *accuracy* en el conjunto de entrenamiento (*Training set*) tiende a 1 (está entre 0.8 y 0.9) y la de pérdida (*loss*) tiende a 0 (está entre 0.3 y 0.2). Más aún, las curvas correspondientes al conjunto de validación (*Validation set*) se comportan de manera similar, lo cual indica que se está realizando un buen entrenamiento

Sin embargo, si la pérdida de validación comienza a aumentar mientras la pérdida de entrenamiento sigue disminuyendo, esta puede ser un signo de sobreajuste (Figura 31).

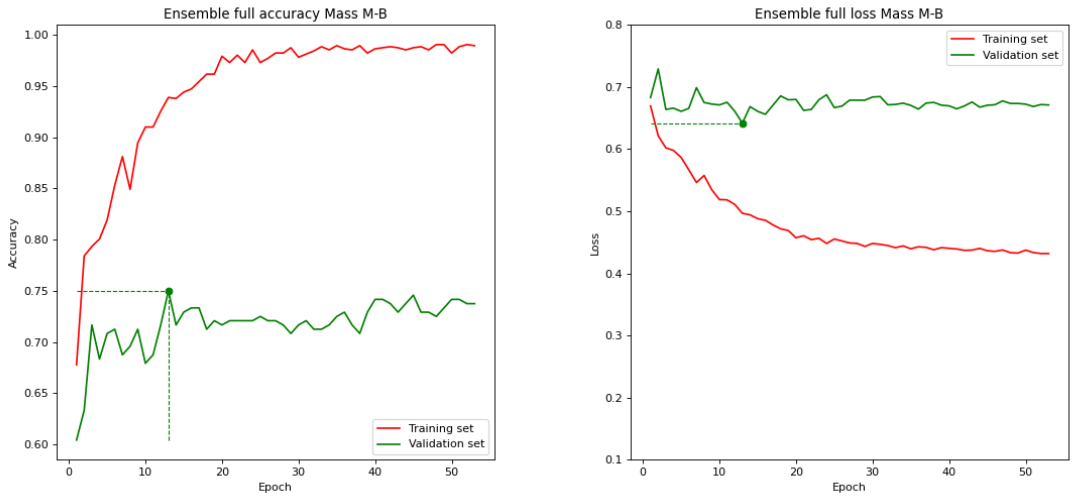


Figura 31. Se muestran las curvas de entrenamiento en rojo y las de validación en verde de la exactitud (*accuracy*) y pérdida (*loss*) de un modelo de ensamble en la clasificación de patologías. Observe que la pérdida en el conjunto de validación comienza a aumentar mientras la pérdida de entrenamiento sigue disminuyendo, esto es un signo claro de que el modelo presentará sobreajuste.

Validación por k-folds. Una de las observaciones más importantes es la siguiente. Suponga que se tienen tres conjuntos de datos X_1, X_2 y X_3 con sus respectivos *targets*, los dos primeros para entrenar un modelo y el último para evaluarlo, entonces, surge la duda de ¿con cuál de los dos conjuntos de entrenamiento conviene entrenar el modelo seleccionado para obtener mejores métricas? Una respuesta rápida es que es mejor combinarlos para obtener un conjunto más grande y así obtener un modelo que generalice mejor, es decir, que ajuste mejor sus parámetros y, aunque esta es una respuesta muy válida, la verdad es que, como se hablará en la siguiente sección, la finalidad de un modelo es ser desplegado o puesto en producción para clasificar datos como con los que fue entrenado, por lo que nunca podremos tener toda la población de datos para ajustar sus parámetros, por lo que entrenar el modelo con conjunto de datos distintos probablemente nos dará métricas distintas, pero, ¿qué tan distintas? Lo ideal sería que al entrenar el modelo con conjuntos de entrenamiento disjuntos produjeran las mismas métricas porque siguen siendo datos pertenecientes a una misma población pero en la práctica se sabe que esto no ocurre, de hecho, lo esperado sería que las métricas variaran pero no tanto. Esto anteriormente tiene una semejanza con las muestras aleatorias (no lo son), ya que representan una parte de la población total pero nunca es posible estudiar toda. En la práctica, para tener una noción más general del comportamiento de un modelo, lo que se hace es utilizar la validación por *k folds*, también conocida como *k-fold cross validation*. Esta validación se ilustra en la Figura 32 y funciona de la siguiente manera:

- Se divide el conjunto de entrenamiento en k partes para entrenar k modelos.

- Suponiendo que se ordenan los k subconjuntos o divisiones, el primer modelo usará el primer subconjunto como validación y entrenará con los restantes $k-1$ subconjuntos, el segundo modelo usará el segundo subconjunto como validación y usara el primero y los $k-2$ restantes como conjuntos de entrenamiento y así sucesivamente hasta entrenar y evaluar el k-ésimo modelo.

- Después se recomienda promediar las métricas de interés así como calcular una varianza para analizar el alcance del modelo con los datos de entrenamiento.

  Cabe destacar que el utilizar este tipo de validación da una idea más general del alcance que tendrá el modelo al entrenarlo con todos estos datos pero esto también requiere, además de una buena cantidad de datos, un aumento en el poder de cómputo al entrenar los k modelos.

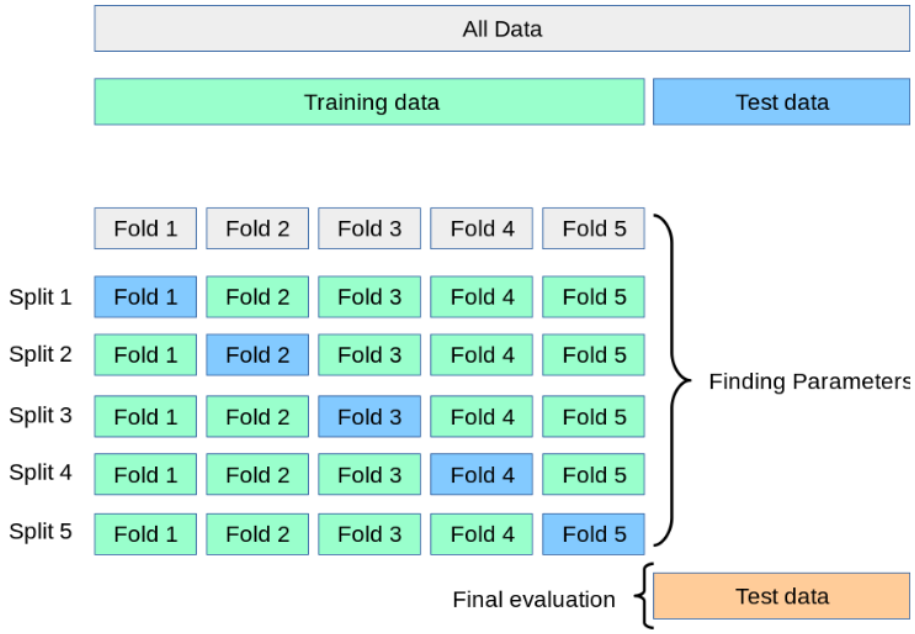


Figura 32. Representación de la técnica *k-folds*. Se muestran 5 particiones distintas de los datos de entrenamiento para entrenar 5 predictores como se explicó previamente, observe que incluso se considera aplicar la técnica de *Hold-out*, ya que se muestra un conjunto de Test para la evaluación final.[5]

Para terminar este capítulo, cabe mencionar que un problema que surge al entrenar modelos de aprendizaje máquina es que se tengan mucho más datos de una clase que de otras, lo cual se conoce como datos desbalanceados. Esto, aunque no lo parezca, termina perjudicando o sesgando el comportamiento de los modelos a la hora de entrenarlos. Por ejemplo, si se sigue el enfoque de crear un conjunto de validación a partir del conjunto de entrenamiento, ¿se puede asegurar que en el conjunto de entrenamiento y validación exista la misma proporción de las clases para no sesgar tanto al modelo? De manera resumida

[5]Figura adaptada de: `https://scikit-learn.org/stable/modules/cross_validation.html`.

la respuesta es que sí es posible y, más aún, se puede asegurar que la forma de estos no es única. Una forma de realizar esta separación es mediante el uso de la función `train_test_split` de **scikit-learn**, indicando `stratify=y`, en donde `y` representa las clases (etiquetas) de todo el conjunto de entrenamiento. Cabe destacar que cuando se utiliza `stratify=y` se está indicando que la división de los datos debe mantener la misma proporción de las clases en los conjuntos de entrenamiento y validación, lo cual es especialmente importante en problemas de clasificación donde las clases están desequilibradas. Por otro lado, el parámetro `random_sate = número fijo`, devuelve la misma partición (correspondiente al conjunto de entrenamiento y validación) de todas las posibles particiones estratificadas (*stratify*), por lo que cambiar este parámetro garantiza otra partición estratificada. Cabe destacar también que durante la experimentación, el uso de **semillas**, como en este caso, garantiza más la reproducibilidad y/o replicabilidad del experimento.

Capítulo 3. Datos

Para que cualquier algoritmo de aprendizaje máquina sea capaz de realizar una buena tarea de clasificación (regresión) es necesario que se tengan 'buenos datos', es decir, se necesita de un experto o alguna persona capaz de obtenerlos, para después conocerlos,es decir, ver sus características, visualizarlos, hacer un resumen estadístico (*data exploration*), limpiarlos y preparar los datos para su uso (*data cleaning)*, procesos abreviados como *data analysis*. También se suele buscar características de mayor relevancia (*insights*) o creación de nuevas características a partir de las que se tienen (ingeniería de características-*feature engineering*), ya que muchas de las bases de datos existentes no fueron creadas con la idea de alimentar modelos de aprendizaje máquina, pues el interés por la implementación o despliegue (*deployment*) de estos algoritmos ha aumentado en los últimos años debido al auge del poder computacional.

Los datos en este caso son mastografías, a las cuáles se les pueden extraer regiones de interés (ROI´s en inglés) para su posterior estudio, ya que no se necesita de toda la mastografía para llevar a cabo la implementación de un algoritmo de aprendizaje máquina-profundo. El examen de mastografía consiste en cuatro imágenes, dos proyecciones para cada seno: la proyección craneocaudal (CC) y la proyección oblicua medio lateral (MLO). Estas se pueden ver en la Figura 33.

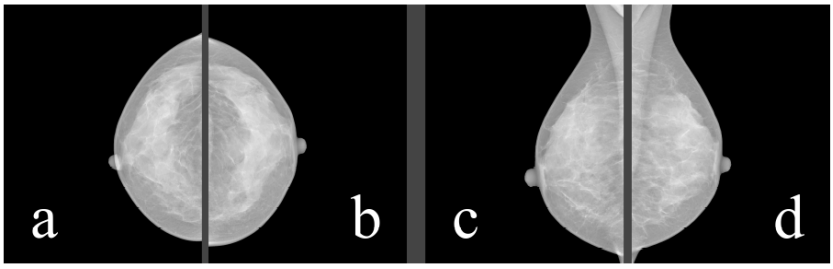


Figura 33. Cuatro vistas: (a) CC derecho, (b) CC izquierdo, (c) MLO derecho, (d) MLO izquierdo. Fuente: Moreira et al. (2012).

En el año 2016-2017, Lee et al. (2017) hablaban de que pocos resultados pueden ser reproducidos, ya que la mayoría de bases de datos en los que fueron desarrollados y evaluados sistemas de detección, conocidos como CAD, fueron en datos (mastografías) privados o en porciones indefinidas de bases de datos públicas, provocando una imposibilidad en la comparación de los resultados obtenidos, sumando a esto la poca existencia de bases de datos públicas 'bien-curadas', por lo tanto, se busca el uso de bases de dates públicas de uso común para que exista una mejor comparación y reproducibilidad de los resultados obtenidos por los investigadores.

Al momento de revisar la literatura sobre esta temática es común encontrar artículos, como se mencionó

previamente, con resultados de uso total o parcial de bases de datos públicas, dificultando la comparación y/o reproducibilidad de los resultados, además, comparando varios de estos artículos se suele encontrar un número distinto de imágenes con respecto a la misma base de datos utilizada, por lo que el artículo de Mračko et al. (2023) ofrece una buena orientación con respecto a las estadísticas e inconvenientes al momento de usar alguna de las bases de datos públicas disponibles. Cabe mencionar que este artículo sirvió de apoyo para respaldar la información que a continuación se describe.

3.1. CBIS-DDSM

CBIS-DDSM es la base de datos que se usó en estos experimentos y es una de las más grandes de datos públicas con fácil acceso, la cual se puede descargar desde: `https://www.cancerimagingarchive.net/collection/cbis-ddsm/`. Se tuvo acceso a su material el 20 Octubre de 2023, unos meses después de la publicación de Mračko et al. (2023) pero más de un año desde que accedieran a la base de datos (10 de Marzo del 2022). Esta es una base datos actualizada de la base de datos original DDSM liberada en 1997, sin embargo, esta versión actualizada provee la segmentación de ROI´s o parches de sus anormalidades (masas y calcificaciones), las máscaras para obtener los parches, así como toda la mastografía completa en formato DICOM (del inglés *Digital Imaging and Communications in Medicine*) junto con sus diccionarios de datos en formato csv. Cabe mencionar que las mamografías y parches están en 16 *bits*. Este dataset también provee información sobre la densidad mamaria de cada mama, pero estandarizada por el Colegio Americano de Radiología (ACR por sus siglas en inglés), cuya evaluación corresponde a un nivel de BI-RADS, entre otros. Lo más relevante de esta base de datos son las patologías Benigna (*Benign*), Maligna(*Malignant*) y *Benign Without Callback* (abreviado BWC) de las anormalidades encontrados (masas y calcificaciones), ya que estas están verificadas, es decir, se les realizaron biopsias a los pacientes para determinarlas. *Benign Without Callback* indica que la mastografía del paciente debe ser descartada, es decir, el médico experto (radiólogo) indicó que a la mama del paciente no se le debía de realiza una biopsia, pues la considera benigna, sin embargo, debe ser monitoreada.

Cabe mencionar que la base de datos se puede descargar por partes, es decir, carpetas que contiene las mastografías completas, de las cuales se extrajeron las masas y/o calcificaciones de interés (ROI´s), mientras que otras carpetas contienen estos ROI´s junto con las máscaras binarias con las que se extrajeron (Figuras 34 y 35). Además las imágenes se proveen en carpetas divididas en imágenes de

entrenamiento (*Training*) y prueba (*Test*) como en la Figura 36.

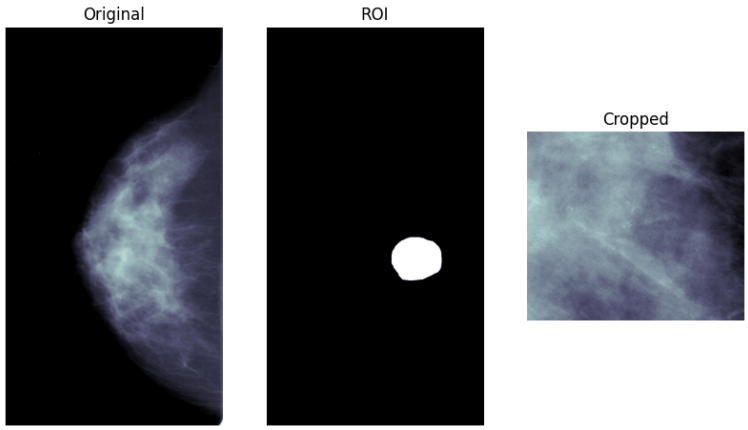


Figura 34. De izquierda a derecha: Mastografía completa. Máscara binaria con la que se obtiene la región de interés. Calcificaciones obtenidas ('cropped file'- parche).

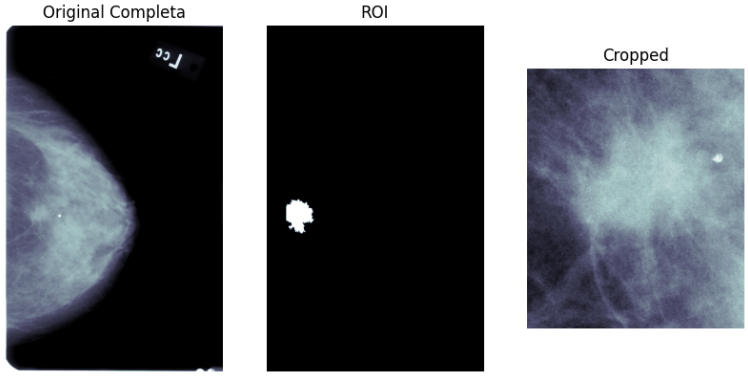


Figura 35. De izquierda a derecha: Mastografía completa. Máscara binaria con la que se obtiene la región de interés. Masa obtenida ('cropped file'- parche).

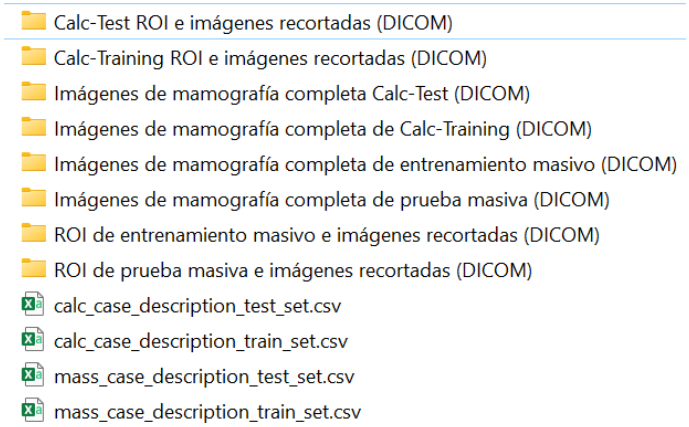


Figura 36. Las carpetas que lleven en su nombre 'mamografía completa' contiene solamente la imagen completa, mientras que en las que llevan su nombre ROI contienen las máscaras binarias junto con los parches obtenidos con estas.

Algunos inconvenientes que presenta esta base de datos y que indica Mračko et al. (2023) son las siguientes:

- Imágenes reflejadas.
- Inconsistencia en el nombre de las imágenes con respecto al del archivo csv.
- Algunas máscaras en el conjunto de datos tienen diferentes resoluciones que las imágenes originales.
- Elementos redundantes en las mastografías completas (marcas, letras, etc.)
- Máscaras de gran tamaño que enmarcan de más en la mastografía que solo las calcificaciones existentes.

En este trabajo de tesis se utilizaron directamente los parches (masas y calcificaciones) proporcionados por la base datos, sin embargo, se observó una discrepancia en la forma en la que estaban distribuidas las imágenes en las carpetas, es decir, como la máscara binaria y el parche están juntos dentro de la misma carpeta, solo hay que 'cargar' las imágenes correspondientes a los parches para trabajar con ellos, por lo que al ingresar a la primer carpeta se pudo constatar que la terminación de la imagen DICOM perteneciente al parche es **1-1.dcm** y la de la máscara binaria es **1-2.dcm**, por lo que se 'sobrentiende' que todos los parches tienen esa misma terminación, sin embargo, no siempre ocurría eso, pues la terminación de estos se permutaba con las de las máscaras binarias en otras carpetas (Figura 37). Esto es un problema debido a que es posible ingresar máscaras binarias a cualquier modelo de aprendizaje máquina sin darse cuenta, por lo que se exploraron dos formas de obtener los parches de manera correcta.

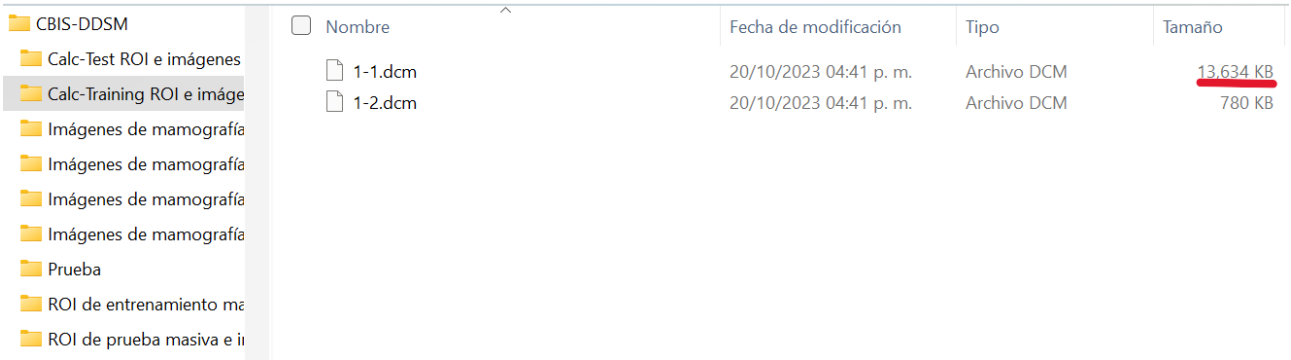


Figura 37. Las máscaras binarias se localizan en la misma carpeta que los parches (ROI´s y *cropped files*), sin embargo, la terminación de varios archivos esta permutada en otras carpetas.

Una forma de evitar el problema antes mencionado es quedarse con las imágenes mediante un umbral de peso en los archivos, ya que las máscaras binarias, en su mayoría, pesan más de 10 *Megabytes* (MB)

como la de la Figura 37, pues se observa que el archivo con terminación **1-1.dcm** pesa poco más de 13 MB, sin embargo, al utilizar este umbral siguen quedando pocas imágenes mezcladas, es decir, existen máscaras binarias que pesan menos que estos 10 MB, por lo que resta ir bajando de a poco el umbral, visualizarlas e ir capturando poco a poco los parches restantes y más en el caso de masas. La forma más fácil de obtener todos los parches es mediante el uso de su formato DICOM. En **python** existe una librería llamada **pydicom**[1], con la cual se pueden leer estos archivos y obtener su información, además de ayudar a visualizarlos (Figuras 34 y 35). Ya que los parches están descritos como `cropped images`, es más fácil ubicarlos de esta forma al utilizar el método `SeriesDescription` del archivo (Figura 38). Después de capturar los datos de esta segunda manera, se compararon las listas obtenidas entre sí y se verificó que ambas tenían el mismo número de elementos.

```
[26]  calc_train_particular.loc[343,:]

...   Series UID              1.3.6.1.4.1.9590.100.1.2.323173986211744534717...
      Collection                                                      CBIS-DDSM
      3rd Party Analysis                                                    NaN
      Data Description URI          https://doi.org/10.7937/K9/TCIA.2016.7O02S9CY
      Subject ID                          Calc-Training_P_00474_LEFT_MLO_1
      Study UID               1.3.6.1.4.1.9590.100.1.2.185043095411645186738...
      Study Description                                                     NaN
      Study Date                                                     09-21-2017
      Series Description                                        cropped images
      Manufacturer                                                          NaN
      Modality                                                               MG
      SOP Class Name                        Secondary Capture Image Storage
      SOP Class UID                               1.2.840.10008.5.1.4.1.1.7
      Number of Images                                                        1
      File Size                                                        2.41 MB
      File Location           .\CBIS-DDSM\Calc-Training_P_00474_LEFT_MLO_1\0...
      Download Timestamp                                2023-10-20T19:11:50.918
      Name: 343, dtype: object
```

Figura 38. Se muestran parte de las características o datos que provee el archivo DICOM de las matografias.

Lo anteriormente descrito muestra que de no haber realizado un análisis exploratorio de los datos se pudieron haber ingresado las imágenes con una sola terminación y así haber afectado al modelo. Cabe señalar que hay autores que utilizan o crean sus máscaras binarias para extraer sus propios parches de las mastografías completas, sin embargo, se pudo constatar que existe un número mayor de parches que máscaras binarias, ya que al cargar algunas imágenes en formato DICOM, en el campo `Series Description`, este se encontraba vacío, lo cual hace más difícil rastrear todas las máscaras binarias (Figura 39). También, otro pequeño detalle surgió en el caso de las calcificaciones, ya que el archivo csv localizado en la carpeta de 'mamografía completa' indica un total de 1547 parches de calcificaciones, pero el archivo csv localizado en la carpeta ROI correspondiente a las calcificaciones indica 1546, por lo

[1]Fuente: `https://pydicom.github.io/`

que después se pudo corroborar que la anormalidad si existía dentro de las carpetas pero no era posible señalar su patología. Este pequeño análisis puede indicar que este sea el motivo por el que algunos autores reportan menos (inclusive más) imágenes en sus experimentos al momento de utilizar esta base de datos, sin embargo, para este trabajo de investigación se puedo verificar que el número de parches obtenidos coincide de igual manera a las estadísticas que se muestran en la Figura 40 y la manera en que se distribuyen se redacta en la sección 4.1.

```
[39]
...  Dataset.file_meta -------------------------------
     (0002, 0000) File Meta Information Group Length  UL: 194
     (0002, 0001) File Meta Information Version       OB: b'\x00\x01'
     (0002, 0002) Media Storage SOP Class UID         UI: Secondary Capture Image Storage
     (0002, 0003) Media Storage SOP Instance UID      UI: 1.3.6.1.4.1.9590.100.1.2.104927815413
     (0002, 0010) Transfer Syntax UID                 UI: Implicit VR Little Endian
     (0002, 0012) Implementation Class UID            UI: 1.2.40.0.13.1.1.1
     (0002, 0013) Implementation Version Name         SH: 'dcm4che-1.4.35'
     -------------------------------------------------
     (0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
     (0008, 0016) SOP Class UID                       UI: Secondary Capture Image Storage
     (0008, 0018) SOP Instance UID                    UI: 1.3.6.1.4.1.9590.100.1.2.104927815413
     (0008, 0020) Study Date                          DA: '20170906'
     (0008, 0023) Content Date                        DA: '20160503'
     (0008, 0030) Study Time                          TM: '083408'
     (0008, 0033) Content Time                        TM: '114207.115000'
     (0008, 0050) Accession Number                    SH: ''
     (0008, 0060) Modality                            CS: 'MG'
     (0008, 0064) Conversion Type                     CS: 'WSD'
     (0008, 0090) Referring Physician's Name          PN: ''
     (0008, 103e) Series Description                  LO: 'ROI mask images'
     (0010, 0010) Patient's Name                      PN: 'Calc-Training_P_00474_LEFT_MLO_1'
     (0010, 0020) Patient ID                          LO: 'Calc-Training_P_00474_LEFT_MLO_1'
     (0010, 0030) Patient's Birth Date                DA: ''
     (0010, 0040) Patient's Sex                       CS: ''
```

Figura 39. La máscara binaria se muestra en `Series Description` por el nombre `ROI mask images`, por lo que se puede ubicar de esta manera.

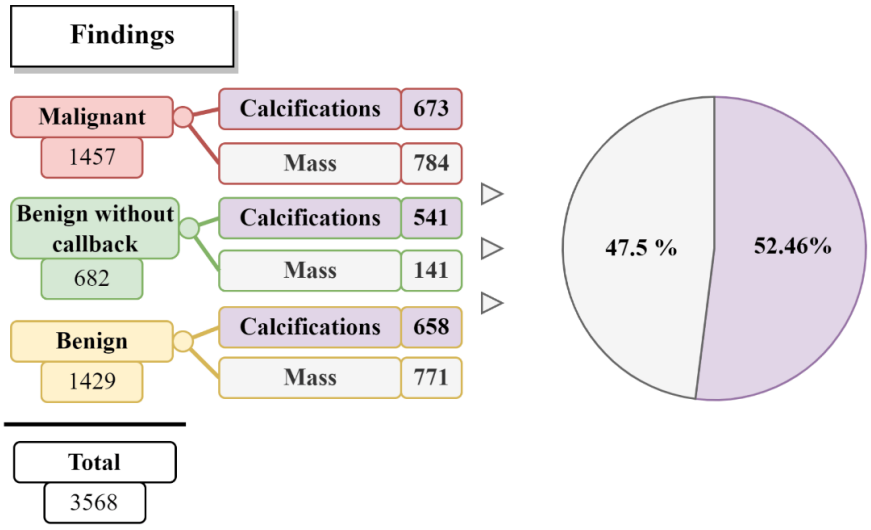


Figura 40. Estadísticas obtenidas por Mračko et al. (2023).

3.2. Aprendizaje por Transferencia y Ajuste Fino

Cuando se planea la implementación de un algoritmo de inteligencia artificial para abordar cierta problemática, se debe realizar todo el preprocesamiento descrito al inicio de este capítulo, sin embargo, en la práctica existen más problemáticas, como lo son la suposición de distribución estadística sobre los datos e independencia estadística, existencia de datos faltantes, datos repetidos, entre otros, pero dos de las principales son, en algunos casos, el costo o fuerte inversión económica requerido para adquirir ciertos tipos de datos, mientras que en este caso es la poca existencia de bases de datos públicas y con esto una poca existencia de imágenes médicas, esto debido a la información sensible del paciente, lo cual es comprensible.

Si bien el Aprendizaje por Transferencia (*Transfer Learning*) y Ajuste Fino (*Fine Tuning*), abreviados en este texto por TF y FT respectivamente, son técnicas utilizadas en el aprendizaje profundo para aumentar la tasa de clasificación y obtener mejores resultados, ya que en principio se cuenta con un número de datos correspondientes al de la Figura 40, los cuales se consideran pocos para entrenar modelos de aprendizaje profundo, por lo que el aumento de datos (*data augmentation*) puede también ayudar a elevar la tasa de clasificación y obtener un modelo mejor validado, ya que esta técnica combate mejor el sobreajuste.

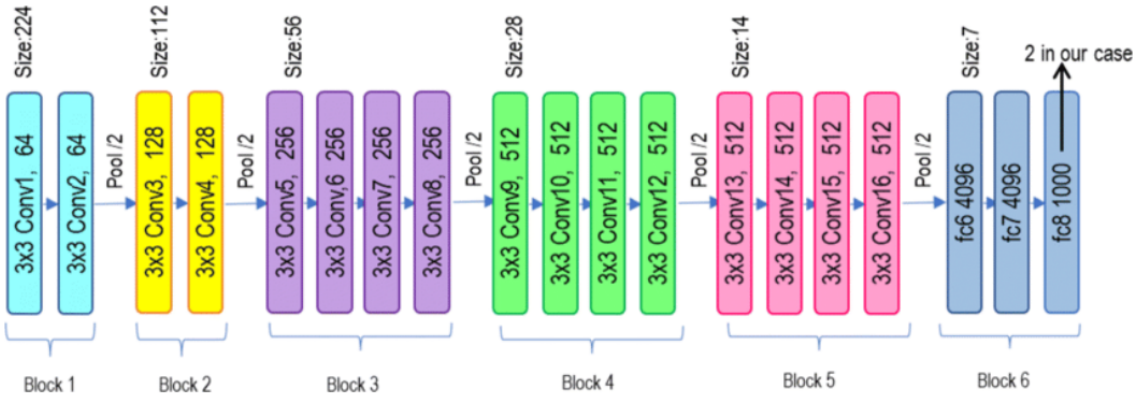
Una vez que se tiene creado un algoritmo para clasificar ciertos objetos o *targets* pertenecientes a un conjunto Ω_0 (conjunto universal de *targets*) de interés etiquetas, este realizará una buena tarea de clasificación sobre el conjunto Ω_0, suponiendo que le haya ido bien en su conjunto de entrenamiento D_0 (conjunto de características con las que se entrenó), sin embargo, ¿qué pasa si se entrena un algoritmo con características del conjunto D_0 pero con *targets* de un conjunto distinto al de Ω_0? Pues sin ninguna duda, la tarea de clasificación baja, pero ¿qué tanto?, es decir, ¿es útil crear o entrenar un algoritmo con características en D_0 asociadas a *targets* en Ω_0 para clasificar *targets* en otro conjunto distinto a Ω_0? , dígase ¿Ω_1? Observe que la pregunta no es tan absurda como pareciera ya que esto podría ser útil o exitoso si las etiquetas pertenecientes a los conjuntos Ω_0 y Ω_1, compartieran algunas características. Lejos de adentrarse más en este tema y caer en la filosofía, cabe señalar que se puede encontrar una explicación más rigurosa al tratar de formalizar lo anteriormente descrito en Weiss et al. (2016), ya que se ha visto que en la práctica ésta metodología (aprendizaje por transferencia) puede funcionar.

En visión por computadora cuando se habla de aprendizaje por transferencia se suele referir al uso de CNNs ya entrenadas (preentrenadas) con ciertas imágenes asociadas a sus etiquetas para clasificar otros

tipos de imágenes distintas con las que fueron entrenadas, y el hecho de que se tengan buenos resultados, al menos en esta temática, es que a pesar de que las mamografías son diferentes a los carros, gatos, frutas, entre otros, las propiedades visuales de un objeto tales como sus formas, texturas, bordes, entre otros, en una imagen son generales (Falconí et al., 2019), siendo esta una de las razones por las que estas CNNs preentrenadas pueden ayudar a obtener una buena tasa de clasificación.

Una CNN preentrenada ya tiene sus parámetros fijos, por lo que se dice que estos están congelados (*frozen*). Estos pesos fueron obtenidos durante el entrenamiento con datos de particular interés o para realizar la tarea con la que fue desarrollada (clasificación, segmentación, entre otros). Para utilizar estas redes en datos distintos con las que fueron entrenadas (TF), lo que se hace es utilizar las capas que están antes de su perceptrón multicapa (capaz convolucionales, *drop-out*, etc.), por lo que un enfoque utilizado es quitar este último y crear el propio, es decir, construir el que se desee, por lo que los pesos y sesgos que se modifican en el *loop* de entrenamiento serán los del perceptrón multicapa creado. Una observación importante es que esto requiere poder computacional ya que dependiendo la computadora una sola ejecución puede tardar horas o días pues se llegan a tener, por lo regular, millones de parámetros entrenables.

La Figura 41 muestra la arquitectura **VGG19**. Este modelo y **VGG16** son modelos de redes neuronales convolucionales diseñados para clasificación de imágenes y fueron propuestos por el Visual Geometry Group de la Universidad de Oxford en 2014. VGG16 contiene 16 capas, 13 capas convolucionales de 3×3 y 3 capas FC, en donde capas de *max-pooling* de 2×2 siguen después de las capas convolucionales. Por su parte VGG19 es una variante de VGG16 con 19 capas de profundidad.



VGG-19 Architecture [39]. VGG-19 has 16 convolution layers grouped into 5 blocks. After every block, there is a Maxpool layer that decreases the size of the input image by 2 and increases the number of filters of the convolution layer also by 2. The dimensions of the last three dense layers in block 6 are 4096, 4096, and 1000 respectively. VGG classifies the input images into 1000 different categories. As there are two output classes in this study the dimension of fc8 is set to two

Figura 41. Khattar & Quadri (2022) describen la arquitectura VGG19. Cabe señalar que la utilizan para clasificar dos clases.

Como se mencionó antes, las propiedades visuales ayudan a poder reutilizar estas CNNs existentes, sin embargo, estas mismas propiedades, a pesar de ser lo suficientemente generales para ayudar a empezar a atacar cierta problemática de clasificación, no son lo suficientemente particulares para realizar un excelente trabajo, por lo que la técnica de ajuste fino (FT) puede ayudar a mejorar esta tarea y para esto lo que se hace es descongelar *unfrozen* sus parámetros desde alguna de las capas de la CNN para que así los valores de los filtros sean también ajustables (entrenables) desde esta capa. La manera de realizar el entrenamiento con FT es de al menos dos formas. Primero se puede definir todo el modelo y de una vez seleccionar desde qué capa realizar el ajuste fino y la segunda es entrenar primero la CNN preentrenada (TL), evaluarla y después seleccionar la capa desde la cual se desea hacer este ajuste (FT) para volverlo a entrenarla. La segunda forma de hacer FT se hace principalmente porque al ser una red entrenada con otros tipos de datos, no se asegura una buena tasa de clasificación a mayor profundidad de la capa seleccionada, pues recuerde que estas redes no fueron entrenadas con el fin de resolver nuestra problemática.

En la Figura 41 se pueden observar principalmente cinco bloques (el sexto es el perceptrón multicapa), por lo que el FT podría realizarse de la siguiente manera. La metodología indica descongelar los pesos de las últimas capaz, es decir, los pesos de la capa '3×3 Conv6 512' del bloque 5, para posteriormente entrenar la red con el perceptrón que se haya construido. Cabe señalar que el incremento de la tasa de clasificación puede o no aumentar. De hecho, se pueden descongelar más pesos y sesgos de las capas anteriores a esta última capa, haciendo esto gradualmente capa por capa o como se mencionó, descongelar desde un inicio tantas capas como se desee. Es importante señalar que no todas las redes preentrenadas tienen capas como esta red, de hecho algunas son más complejas, sin embargo, se puede seguir realizando FT pero señalando el número de la capa desde la cuál se desea descongelar los pesos y sesgos, ya que algunas suelen tener más de 100.

Cabe mencionar que en la práctica existen técnicas para ir observando el comportamiento del modelo de tal forma que se puede guardar el modelo en el momento en el que este pueda alcanzar un mínimo (posiblemente local) y detener el entrenamiento si después de este la métrica no mejora. En **TensorFlow**[2]esto se puede llevar a cabo mediante una combinación de métodos conocidos como **early-stopping** y **checkpoint**.

[2]TensorFlow es una biblioteca de código abierto para el aprendizaje automático y el aprendizaje profundo (deep learning) desarrollada por Google. Está orientada al diseño, entrenamiento y despliegue de redes neuronales profundas y otros modelos de *machine learning*: `https://www.tensorflow.org/?hl=es-419`.

3.3. Image Data Generator

El generador de imágenes o **Image Data Generator** abreviado en este texto como **IDG**, es una clase de Keras[3]que contiene varios métodos para preprocesar imágenes de manera eficiente y realizar aumento de datos al aplicar diversas transformaciones de manera dinámica y en tiempo real durante el entrenamiento de modelos, lo cual se conoce como *data augmentation*. Este se puede utilizar desde **TensorFlow**, específicamente de

`tensorflow.keras.preprocessing.image.ImageDataGenerator`.

Sus principales características son las siguientes:

1. **Aumento de datos**. Este generador permite aplicar transformaciones aleatorias a las imágenes, lo que ayuda a aumentar la diversidad del conjunto de datos y mejorar la capacidad de generalización del modelo. Algunas transformaciones incluyen:
   - Rotación aleatoria.
   - Zoom.
   - Traslación horizontal y vertical.
   - Volteo horizontal.
   - Cambios en brillo, contraste y saturación.
   - Corte (shearing)
2. **Normalización**. Se puede aplicar normalización a las imágenes, ya que como se mencionó antes, lo más común es trabajar con imágenes de 8 bits, por lo que es necesario escalar los valores de los píxeles. Esto se hace de manera automática antes de que las imágenes se pasen a la red.
3. **Carga eficiente de datos**. En lugar de cargar todo el conjunto de datos en memoria, el generador permite la carga en tiempo real de las imágenes por lotes (*batch*) durante el entrenamiento. Esto es útil cuando se trabaja con grandes cantidades de datos, ya que como se describió en la sección 2.2.2, la carga completa de los datos puede llegar a ser computacionalmente intratable al ocupar toda la memoria que se esté utilizando.

[3]Keras es una biblioteca de Redes Neuronales de Código abierto escrita en Python capaz de ejecutarse sobre TensorFlow: `https://keras.io/`

4. **Lectura desde directorios**. El punto anterior es llevado a cabo mediante la lectura de las imágenes desde un directorio por lotes, lo que es útil cuando los datos están organizados en subcarpetas por clases o etiquetas (lo cuál es recomendable) con los métodos `flow`, `flow_from_dataframe` y `flow_from_directory`.

Cabe destacar que este aumentador de datos, con el método `flow`, puede trabajar con todas las imágenes cargadas al mismo tiempo y no estar organizadas por carpetas.

Aunque es muy útil este aumentador de datos, la principal limitación es que las opciones para las transformación de las imágenes están limitadas, por lo que es mejor opción el obtener más datos para permitir al modelo aprender más formas de identificar las imágenes o utilizar otras librerías de programación para obtener transformaciones más exactas o posibles que las que provee este aumentador de datos.

3.4. Extracción de características (Feature Extraction)

De las secciones 2.2.3 y 2.2.4 se sabe que una CNN está compuesta por distintas capas y que los elementos que aprenden son los filtros (características) para que posteriormente estos sean ingresados (después del aplanado o *flatten*) al perceptrón multicapa para que realice la tarea de clasificacón, sin embargo, es posible utilizar otro clasificador como alguno de los mencionados en la sección 2.3. Esta técnica o metodología es llamada Extracción de Características o *Feature Extraction*.

La forma de realizar esta técnica es utilizar el aplanado de todo el mapa de características como nuevo conjunto de datos de entrenamiento, aunque cabe mencionar que se puede utilizar solo una parte de este. Por ejemplo, si se considera un conjunto de datos de diez mil imágenes y se desea utilizar las características generadas por los bloques convolucionales de la red (los filtros que aprendió), entonces primero se selecciona el bloque desde el que se quiere obtener estas características, que por lo regular es el último, es decir, todo el mapa de características. Si se hace referencia al de la Figura 41, el último bloque seleccionado sería el: '3×3 Conv16 512' del bloque 5. Este regresa un mapa de características de tamaño $7 \times 7 \times 512$, 7×7 unidades espaciales y 512 canales de características, lo que da un total de $25,088$ características al aplanarlo (*flatten*). Una vez que se conoce este número, la nueva matriz de características tendrá un tamaño de $(10000, 25088)$, pues este corresponde al número de instancias (imágenes) junto con las características (25088) obtenidas por cada imagen.

3.4.1. Extracción de las capas FC

La idea es la misma que la del *Feature Extraction*, sin embargo, en vez de utilizar todo el mapa de características, lo que se puede hacer es utilizar las neuronas de salida de alguna de las capas FC del perceptrón multicapa con el fin de utilizar otro clasificador para obtener la predicción final. Esta idea es válida porque en la sección 2.2.1, antes de iniciar la 2.2.2, se describe que las salidas de las neuronas es una suma ponderada de las características, para ser más específico, se describe que '... la salida Z, antes de aplicarse alguna función de activación, tiene la forma $(N, \#neuronas)$, donde cada fila representa la salida de una capa densa para cada una de las N instancias de entrada', por lo que si se seleccionan las neuronas de salida de la primer capa de densa del bloque 6 de la Figura 41, a saber el 'fc6 4096', se podría construir otra matriz de características de tamaño $(10000, 4096)$, si se consideran las diez mil imágenes previas, por lo que lo consiguiente sería utilizar un clasificador como los mencionados en la sección 2.3.

Por último cabe mencionar que esta técnica será referenciada como *FC Extraction* en este trabajo de tesis.

Capítulo 4. Experimentos y Resultados

Como se mencionó en el capítulo anterior, la base de datos con la que se trabajó fue CBIS-DDSM. Es importante mencionar se coincidió en número con las estadísticas descritas en la Figura 40, dando un total de 3568 imágenes recuperadas, pero se descartaron las 682 imágenes etiquetadas como BWC, dando un total de 2886 imágenes. Como se sabe, este número de imágenes es relativamente muy poco en comparación a los experimentos encontrados dentro y fuera de la literatura, ya que se sabe que se necesitan miles de imágenes para obtener mejores resultados, por lo que el uso del *Image Data Generator* podría ayudar a combatir esta falta de datos, aunque lo más recomendable sería obtener más datos y aplicar el generador de imágenes. Estas imágenes fueron guardadas en carpetas distintas a las que originalmente se encontraban, ya que como se mencionó en la sección 3.1, existieron errores al explorar la base de datos. Las imágenes en estas carpetas ya se encuentran distribuidas de manera más ordenada para su posterior carga y preprocesamiento, ya que se guardaron las rutas o *paths* hacia estas carpetas para llamarlas de manera más fácil. Cabe mencionar que la base de datos **INbreast** se descartó debido a la falta de confirmación en las patologías de las regiones de interés por parte de expertos radiólogos, sin embargo, muchos investigadores optan por usarla como conjunto de validación, mientras que otros como conjunto de entrenamiento para sus modelos de aprendizaje profundo (Mračko et al., 2023).

4.1. Experimentos Principales

Se utilizaron las regiones de interés (parches) que la base de datos CBIS-DDSM proporciona sin la necesidad de realizar segmentación alguna de las mastografías. Dentro de todos los experimentos realizados y descritos en este capítulo, cabe destacar que se centran alrededor de los cinco experimentos siguientes:

1. **Experimento Masas**. Clasificación de masas en benignas y malignas, es decir, patologías de masas. Para este experimento se utilizaron 1214 Imágenes para entrenamiento (577 benignas y 637 malignas) y 341 de *Test* (194 benignas y 147 malignas). La etiqueta `Masa Maligna` fue representada por el número 0, mientras que la etiqueta de `Masa Benigna` fue representada por el número 1. La Figura 42 muestra el tipo de imágenes utilizadas en este experimento.

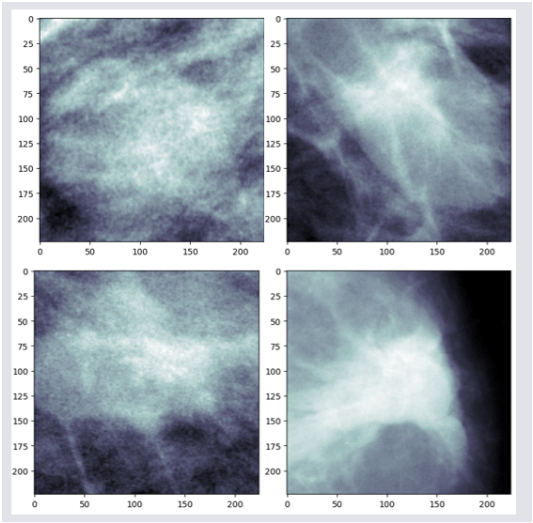


Figura 42. Las imágenes en la parte superior corresponden a masas benignas, mientras que las inferiores corresponden a masas malignas

2. **Experimento Calcificaciones**. Clasificación de calcificaciones en benignas y malignas, es decir, patologías de calcificaciones. Para este experimento se utilizaron 1072 Imágenes para entrenamiento (528 benignas y 544 malignas) y 259 de *Test* (130 benignas y 129 malignas). La etiqueta `Calcificación Maligna` fue representada por el número 0, mientras que la etiqueta de `Calcificación Benigna` fue representada por el número 1. La Figura 43 muestra el tipo de imágenes utilizadas en este experimento.

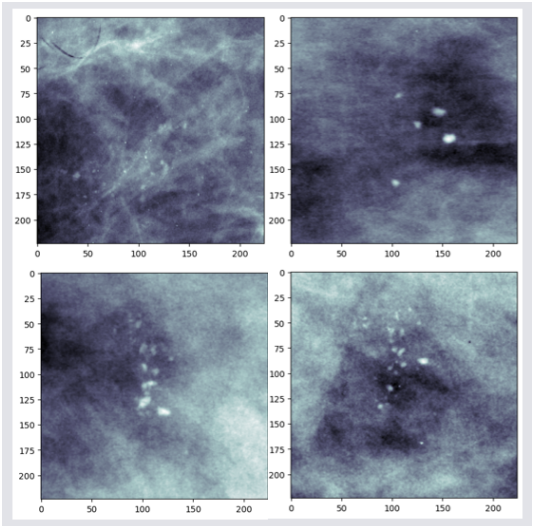


Figura 43. Las imágenes en la parte superior corresponden a calcificaciones benignas, mientras que las inferiores corresponden a calcificaciones malignas.

3. **Experimento Multiclase**. El algoritmo recibe un parche con la región de interés de la masto-

grafía, sea masa o calcificación y regresa un de las cuatro opciones posibles: `Masa Benigna` o `Masa Maligna` o `Calcificación Benigna` o `Calcificación Maligna`, básicamente es la combinación de los dos experimentos anteriores, experimentando a la vez con todas las imágenes. Para este experimento se utilizaron las 2886 imágenes con la distribución ya descrita. La Figura 44 muestra el tipo de imágenes utilizadas en este experimento. Las etiquetas `Calcificación Benigna`, `Calcificación Maligna`, `Masa Benigna` y `Masa Maligna` fueron representadas por los números 0,1,2 y 3 respectivamente, pero estas etiquetas se tuvieron que vectorizar para hacer uso de la *función de activación Softmax*:

- $0 \rightarrow [1, 0, 0, 0]$
- $1 \rightarrow [0, 1, 0, 0]$
- $2 \rightarrow [0, 0, 1, 0]$
- $3 \rightarrow [0, 0, 0, 1]$

Para realizar esta vectorización se utilizó la función `to_categorical` de **Tensorflow**, específicamente `tensorflow.keras.utils.to_categorical`.

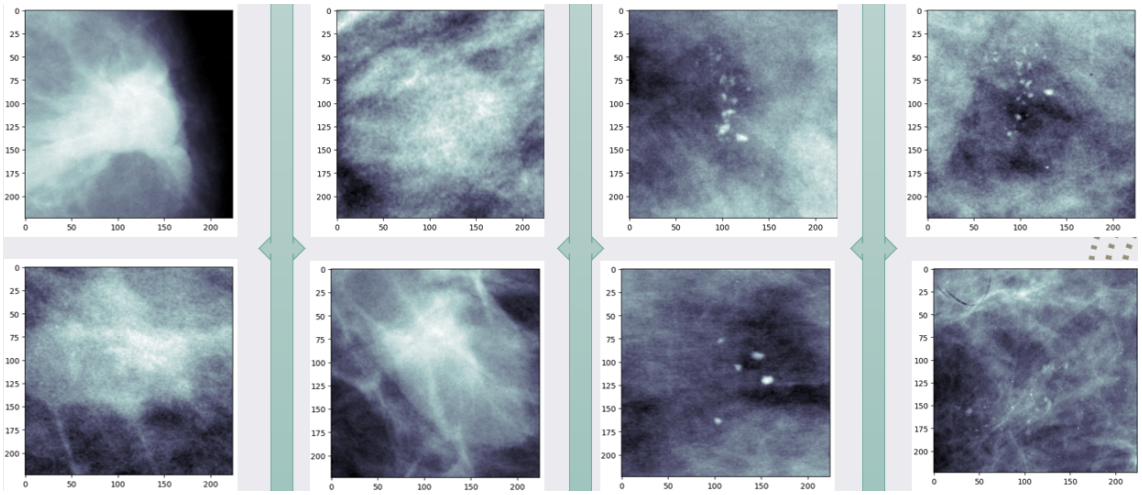


Figura 44. Se muestran las cuatro opciones posibles al recibir una región de interés o parche de una mastografía, observe que son las mismas imágenes que se muestran en la parte de arriba pero ordenadas o distribuidas en un de cuatro clases posibles.

4. **Experimento M-B**. En esta parte del experimento se combinan las imágenes en patologías para entrenar un algoritmo que sea capaz de clasificarlas, sin importar si provienen de masas o calcificaciones, por ende se utilizaron las 2886 imágenes, pero 1105 imágenes benignas para entrenamiento (577 masas benignas y 528 calcificaciones benignas), 1181 imágenes malignas para entrenamiento (637 masas malignas y 544 calcificaciones malignas) y las 600 restantes como *Test*, dando un

total de 324 benignas (194 masas benignas y 130 calcificaciones benignas) y 276 malignas (147 masas malignas y 129 calcificaciones malignas). Las etiquetas correspondientes a `Malignas` fueron representadas por el número 0, mientras que las `Benignas` por el número 1. La Figura 45 muestra el tipo de imágenes utilizadas en este experimento.

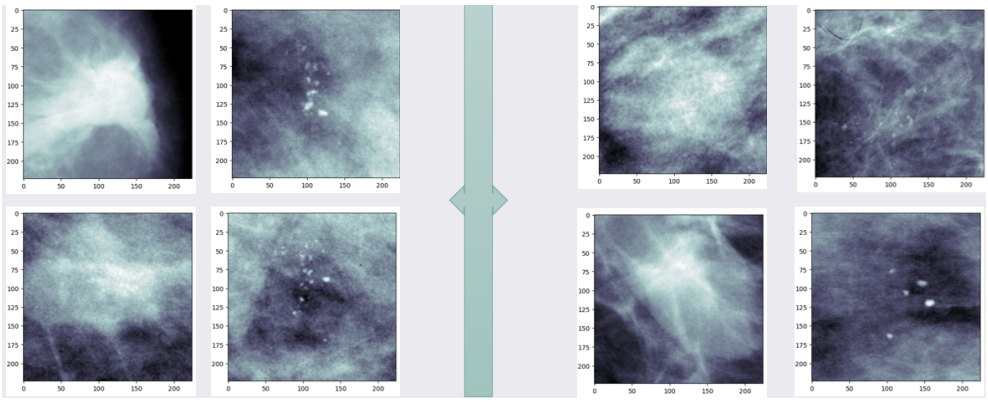


Figura 45. En la parte izquierda se muestras patologías malignas y en la derecha benignas sin importar si son masas o calcificaciones. Observe que son las mismas imágenes de los dos primeros experimentos pero distribuidas de otra forma.

5. **Experimento M-C**. Se utilizaron todas las imágenes para entrenar un algoritmo que sea capaz de clasificar o distinguir entre masas y calcificaciones sin tomar en cuenta sus patologías. Se utilizaron un total de 1214 imágenes de masas para entrenamiento (577 benignas y 637 malignas), 1072 imágenes de calcificaciones para entrenamiento (528 benignas y 544 malignas) y las 600 restantes como *Test*, dando un total de 341 masas (194 benignas y 147 malignas) y 259 calcificaciones (130 benignas y 129 malignas). Las etiquetas correspondientes a `Masas` fueron representadas por el número 0, mientras que `Calcificaciones` fueron representadas por el número 1. La Figura 46 muestra el tipo de imágenes utilizadas en el este experimento.

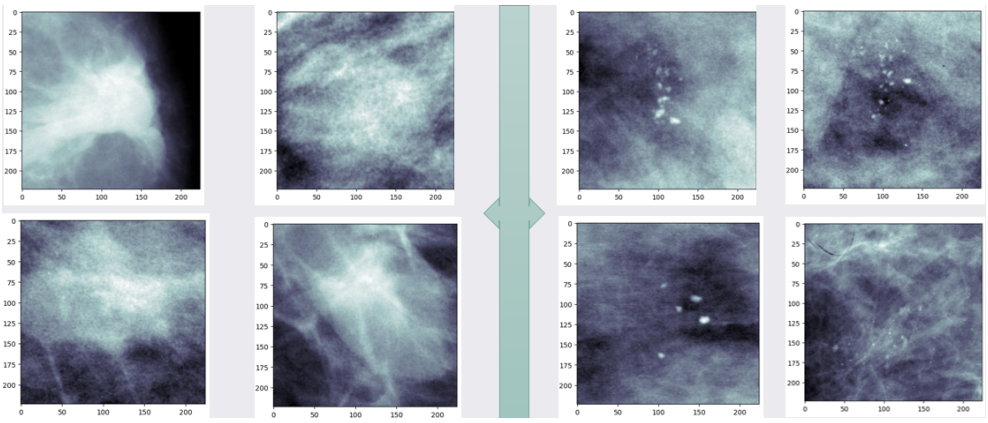


Figura 46. Muestra de Imágenes utilizadas en el **Experimento M-C**. En la parte izquierda se muestran un tipo de anormalidades correspondientes a las masas, mientras que en el lado derecho se muestra otro tipo de anormalidad correspondiente a calcificaciones. Observe que son las mismas imágenes de los dos primeros experimentos pero distribuidas de otra forma

4.2. Experimentación (Primera Parte)

La principal idea es trabajar con algunas redes neuronales convolucionales preentrenadas encontradas en la literatura, por lo que *inicialmente* se optó por utilizar las redes con las que experimentó Jaamour et al. (2023), a saber: **DenseNet121**, **VGG19**, **ResNet50**, **InceptionV3** y **MobileNetV2**. Cabe mencionar que los parámetros que utilizó no fueron explorados, ya que sus experimentos realizados no superaron el $70\,\%$ de *accuracy*, además, su código de programación puede verse en **GitHub**. Otra de las redes que se encuentran en la literatura es el modelo **VGG16**, la cuál utilizó Lai (2021) en distintos experimentos, pero sin poder superar también el $70\,\%$ de *accuracy* en ellos, cabe destacar que realizó el **Experimento M-C**, logrando obtener un $91.7\,\%$ de *accuracy*, por lo cual, se decidió utilizar sus parámetros. Su código de programación está disponible en su **GitHub**

DenseNet121. DenseNet121 es una arquitectura de red neuronal convolucional propuesta en 2017 como parte de la familia de modelos DenseNet[1](Dense Convolutional Networks). La principal innovación de DenseNet121 son las conexiones densas entre sus capas. DenseNet conecta cada capa con todas las capas siguientes en el mismo bloque. Esto significa que cada capa recibe como entrada las salidas de todas las capas anteriores en su bloque, lo cual permite un flujo de información y de gradientes más efectivo.

ResNet. ResNet50 es una arquitectura de red neuronal convolucional creada por investigadores de Microsoft en 2015. Este modelo es parte de la familia de redes ResNet (Residual Networks), que introdujeron las conexiones residuales. Como su nombre lo indica, este contiene 50 capas de profundidad, además, los bloques residuales, también conocidos como *skip connections*, permiten que la salida de una capa se sume directamente a la salida de una capa más adelante, formando lo que se llama un bloque residual. Esta técnica ayuda a combatir el problema de *vanishing gradients* (desvanecimiento de gradiente), permitiendo que los gradientes fluyan sin dificultad incluso en redes muy profundas.

Inception. InceptionV3 es una red neuronal convolucional creada por Google como parte de la serie de arquitecturas Inception y fue introducida en 2015. A pesar de tener bastantes capas convolucionales, tiene pocos parámetros entrenables, esto debido a su factorización de largas capas convolucionales.

MobileNetV2. Esta es una arquitectura de red neuronal convolucional diseñada para dispositivos móviles y de baja potencia, presentada por Google en 2018. Su objetivo es lograr un alto rendimiento en tareas

[1]DenseNet fue introducida en el artículo "Densely Connected Convolutional Networks"por Gao Huang y colaboradores: `https://arxiv.org/abs/1608.06993`.

de visión artificial manteniendo un bajo uso de recursos computacionales, lo que la hace ideal para aplicaciones en dispositivos con limitaciones de procesamiento y memoria.

La forma en la que se trabajó fue la de utilizar las seis redes preentrenadas previamente mencionadas, utilizando los parámetros de Lai (2021). Para ser más preciso, se utilizaron los enfoques de TL y FT. La técnica de TL se utilizó inicialmente con los pesos obtenidos del conjunto de datos **ImageNet**. Esta consistió en eliminar las capas densas y de salida de cada una de las CNN utilizadas para posteriormente personalizar nuevas. Esto nos permite reutilizar la parte del modelo encargada de la extracción de características, mientras añadimos nuevas capas adecuadas para la nueva tarea de clasificación correspondiente con dada experimento. Las capas que añadimos fueron consistentes en todas las CNN: una capa de *drop-out* con un valor de 0.5, una capa densa y una capa de clasificación con *activación Sigmoide* para la tarea de clasificación binaria, mientras que la capa de clasificación con *activación Softmax* se utilizó para la tarea de clasificación multiclase. La forma de las capas completamente conectadas FC se muestran en la Tabla 4.

Tabla 4. Salidas empleadas en el enfoque de aprendizaje por transferencia.

Flatten
Drop-Out de 0.5
FC con N número de Neuronas
Capa con *activación (Sigmoide o Softmax)*

El único parámetro que varió fue el número N de Neuronas en la capa densa de cada CNN. Inicialmente, se experimentó con 64 neuronas, luego 128 y así se continuo duplicando el número hasta llegar a 1024. Esto da como resultado 5 experimentos principales (los anteriormente descritos) que multiplicados por las 6 redes utilizadas y además, las 5 modificaciones de la capa FC en el número de neuronas N: 64,128,256,512 y 1024 (aunque en algunos se utilizó 2048 Neuronas), da un total de 150 experimentos mínimos hasta el momentos.

Activación Sigmoide o Softmax. Una observación importante es que las redes que fueron entrenadas para resolver la tarea de clasificación binaria utilizaron *activación Sigmoide*, regresando un valor de salida que representa la probabilidad de que la entrada pertenezca a la clase positiva (o clase 1), según sea la que se haya definido con este número, sin embargo, esta no regresa un vector de longitud 2 como lo haría la *activación Softmax*, por lo que la clasificación final es obtenida al pasar un umbral de probabilidad como en la sección 2.3.3 ecuación 11. Cabe mencionar que este enfoque es el más utilizado o común al tratar

de resolver tareas de clasificación binaria con redes neuronales, sin embargo, es posible atacar problemas de clasificación binaria con activación *Softmax*, regresando en este caso un vector de probabilidades de longitud 2, en donde en cada posición del vector se sitúa la probabilidad de que la instancia de entrada corresponda a una de las dos clases.

Carga de Datos. Las rutas o *paths* de las imágenes fueron cargadas y posteriormente se crearon las etiquetas numéricas como se mencionó previamente. De suma importancia indicar que se crearon dos conjuntos a partir del conjunto de entrenamiento perteneciente a cada uno de los 5 experimentos principales: uno de entrenamiento y otro de validación. Esto con el fin de supervisar y visualizar la existencia de sobreajuste de la CNN utilizada. La partición de los conjuntos de entrenamiento fue de manera estratificada:

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, stratify=y,
                                 random_state=42,
```

en donde los conjuntos con `X` son las rutas hacia las imágenes y las `y`'s las etiquetas. Observe que ahora se tienen tres conjuntos, uno de entrenamiento (*Training*), uno de validación (*Validation*) y otro de prueba (*Test*). Cabe señalar que en el **Experimento Masas** y **Experimento Calcificaciones** se utilizó un `test_size=0.15`. Posteriormente, las imágenes fueron cargadas y de estas se obtuvieron sus pixeles con **pydicom**, para posteriormente obtenerlas en forma de tensores (que es son los elementos con los que trabaja **Tensorflow**) en escala de grises y así poder aplicarles un redimensionamiento (*resize*) de 224×224. Después, estas fueron apiladas, convertidas a 8 *bits* (se dividieron sus pixeles por 65535, lo cual representa el mayor número posible alcanzable por un pixel de 16 *bits* y luego se multiplicaron por 255) y por último se replicaron los canales de estas para convertirlas en imágenes RGB, ya que las redes neuronales preentrenadas trabajan con este tipo de imágenes.

Entrenamiento - TL. El entrenamiento total se definió para 200 épocas. Las redes se 'compilaron' con estos parámetros:

```
optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'],
```

esto en el caso de tener clasificación binaria y estos:

```
optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy']
```

en el caso multiclase. Después, se realizó el entrenamiento de las redes preentrenadas (TL) con las modificaciones previamente descritas, posteriormente, los modelos 'optimo' y 'final' obtenidos durante y al final del entrenamiento se guardaron completamente (no solamente los pesos) para que al final del entrenamiento se cargaran y comparan su *score* (se refiere a algunas de las métricas descritas, siendo la más común el *accuracy*) en el conjunto de *Test*. Cabe mencionar que la forma de guardar el modelo 'optimo' es mediante la combinación de `tf.keras.callbacks.EarlyStopping` y `tf.keras.callbacks.ModelCheckpoint` proporcionadas por **Tensorflow**.

Los valores utilizados para `EarlyStopping` fueron los siguientes:

```
monitor='val_loss', mode='min', patience=50, verbose=1.
```

Por otro lado, los valores utilizados para el `ModelCheckpoint` fueron:

```
path,monitor='val_loss', mode='min', verbose=1,save_best_only=True,
save_freq='epoch'.
```

De manera resumida se puede decir que `EarlyStopping` detiene el entrenamiento cuando la función de pérdida monitoreada (`monitor='val_loss'`) no mejora, esto después de un número definido de épocas de paciencia (`patience=50`), mientras que `ModelCheckpoint` guarda el modelo (`path`) con el mejor rendimiento (`mode='min'`) en base a una métrica definida (`val_loss`). Cabe señalar que la paciencia aplicada en el **Experimento Masas** y el **Experimento Calcificaciones** fue de 30 épocas.

Entrenamiento - FT. Para esta técnica se realizó el segundo enfoque descrito en la sección 3.2. Primero se cargaron los modelos con mejor *score* en el *Test*, después se descongelaron los pesos desde alguna capa de cada una de las redes y así mantener el resto de los pesos congelados. Lo que sigue después de descongelar desde cierta capa del modelo y volverlo entrenable es compilarlo. Esto se hizo de manera similar a la forma previamente indicada, pero siguiendo la recomendación de bajar la tasa de aprendizaje, por lo que la compilación fue la siguiente:

```
optimizer = RMSprop(learning_rate=0.0001), loss = 'binary_crossentropy',metrics=['
uracy']),
```

y solo se cambió `'binary_crossentropy'` por `'categorical_crossentropy'` en el caso multiclase. Cabe mencionar que también se utilizaron `EarlyStopping` y `ModelCheckpoint` de manera similar pero

con una paciencia de 30, ya que se conoce que este enfoque ayuda pero no lo suficiente. El entrenamiento límite continuó en 200 épocas.

Lo que se hizo fue realizar FT a 'tres niveles de profundidad', es decir, después de terminar el entrenamiento con TL, se cargaba el modelo con mejor *accuracy* en el *Test*, después se descongelaba una capa *L1* a partir de la cual se modificaban pesos y sesgos, así el modelo se compila y se entrena, supervisando y guardando los modelos para posteriormente ser evaluados con el *Test*. Después de este primer entrenamiento con FT, se cargó el modelo con mejor *accuracy* en el *Test*, después se descongeló otra capa *L2* (*L2* menor o más profunda que *L1*) y el proceso se repite hasta otra capa *L3* de profundidad de ser posible para algunos modelos (*L3* menor o más profunda que *L2*). Dependiendo el modelo o red preentrenada es como se selecciona la capa a partir de la cuál descongelar sus pesos, más aún, las capas descongeladas a 'tres niveles de profundidad' desde la cual se modificaron estos en cada modelo se muestran en la Tabla 5.

Tabla 5. Niveles de profundidad de Fine Tuning experimentados.

Modelo	Nivel 1 de profundidad	Nivel 2 de profundidad	Nivel 3 de profundidad
DenseNet121	313	141	-
ResNet50	143	81	-
InceptionV3	280	249	-
MobileNetV2	143	134	-
VGG16	'block5_conv3'	'block5_conv2'	'block5_conv1'
VGG19	'block5_conv4'	'block5_conv3'	'block5_conv2'

La razón por la que no se hizo FT en una tercer nivel de profundidad en algunos modelos fue porque no se notaban mejores resultados en la capa anterior (MobileNetV2 e InceptionV3) o de plano el modelo resultaba demasiado 'pesado' para entrenarlo, por lo que rápidamente se mostraba el mensaje `OOM` (fuera de memoria), haciendo imposible su entrenamiento (DenseNet121 y ResNet50).

También cabe destacar que debido a la profundidad de las capas del modelo VGG, con esta red se llevó hasta 5 niveles de profundidad en los experimentos **M-C, M-B** y **Masas**, ya que este proceso de FT fue mejorando gradualmente. Los bloques que se descongelaron en la red VGG16 fueron: 'block5_conv3', 'block5_conv2', 'block5_conv1', 'block4_conv3' y 'block4_conv2', mientras que los bloques descongelados en la red VGG19 fueron: 'block5_conv4', 'block5_conv3', 'block5_conv2', 'block5_conv1' y 'block4_conv'.

4.2.1. Variantes Experimentales

El logro y éxito alcanzados en tareas visuales, no solamente en imágenes médicas, vino con el auge computacional por el uso de redes neuronales convolucionales, pero como se describió antes, se necesitan de datos limpios, curados o preprocesados para que los modelos utilizados puedan alcanzar una buena tasa de clasificación, además de bastantes datos, por lo que en el caso de imágenes, aplicar un buen preprocesamiento suele ayudar bastante a las CNNs para realizar una mejor tarea. Algunos métodos encontrados en la literatura correspondiente a imágenes médicas son: normalización de contraste global (GCN abreviado en inglés), normalización de contraste local y segmentación por umbral de Otsu.

Dado que este conjunto de datos (CBIS-DDSM) no es lo suficientemente grande, se utilizó aumento, el cual es utilizado por muchos investigadores y en este trabajo se realizó por medio del **IDG** (en todos los experimentos anteriormente descritos), siendo la rotación y recorte las técnicas más comúnmente utilizadas, sin embargo, la técnica de rotación suele distorsionar las imágenes. Por esta razón, la rotación por ángulos rectos es más recomendada (Falconí et al., 2020).

Debido a que el **IDG** espera ciertos parámetros y fue utilizado por Lai (2021), primero se plotearon algunas transformaciones resultantes en una imagen con los parámetros que él utilizó, las cuales se muestran en la Figura 47, mientras que las transformaciones con otros parámetros se muestra en la Figura 48.

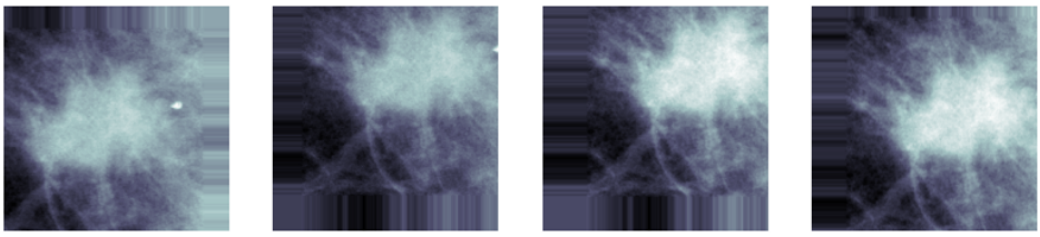


Figura 47. Ejemplo del IDG con Parámetros Iniciales: *shearing* de 0.2.

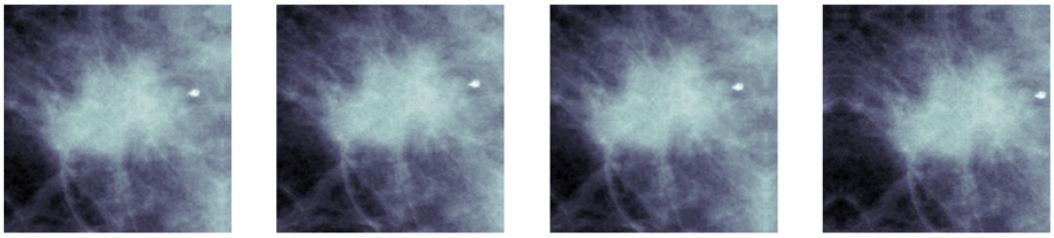


Figura 48. Ejemplo del IDG con Parámetros Propuestos: *shearing* de 0.1.

En la sección previa no se mencionó nada sobre el generador de datos **IDG**, ya que no es parte de la arquitectura de alguna de las CNNs, sin embargo, es una herramienta que ayuda a combatir el sobreajuste de las CNNs y a elevar un poco la tasa de clasificación. Debido a que se utilizó en todos los experimentos, se optó por usar los parámetros que utilizó Lai (2021), así como los otros que tratan de distorsionar menos las imágenes, esto con el fin de observar qué tanto podía afectar la tasa de clasificación, por lo que para no desviar la atención del lector sobre los 5 experimentos principales realizado, nos referiremos a estos experimentos como 'Variantes'.

Variante Parámetros Iniciales. Los parámetros iniciales hacen referencia a los utilizados por Lai (2021) en el **IDG**. La Figura 47 muestra algunas transformaciones aplicadas al parche original (Figura 35).

Variante Parámetros Propuestos. Como las imágenes de la Figura 47 muestran un poco de distorsión, se propuso modificar los parámetros que utilizó Lai (2021), por lo que la Figura 48 muestra las mismas transformaciones aplicadas a la imagen original (Figura 35) mientras que la Figura 49 es una muestra de las transformaciones obtenidas del generador con los parámetros propuestos aplicados a la imagen original.

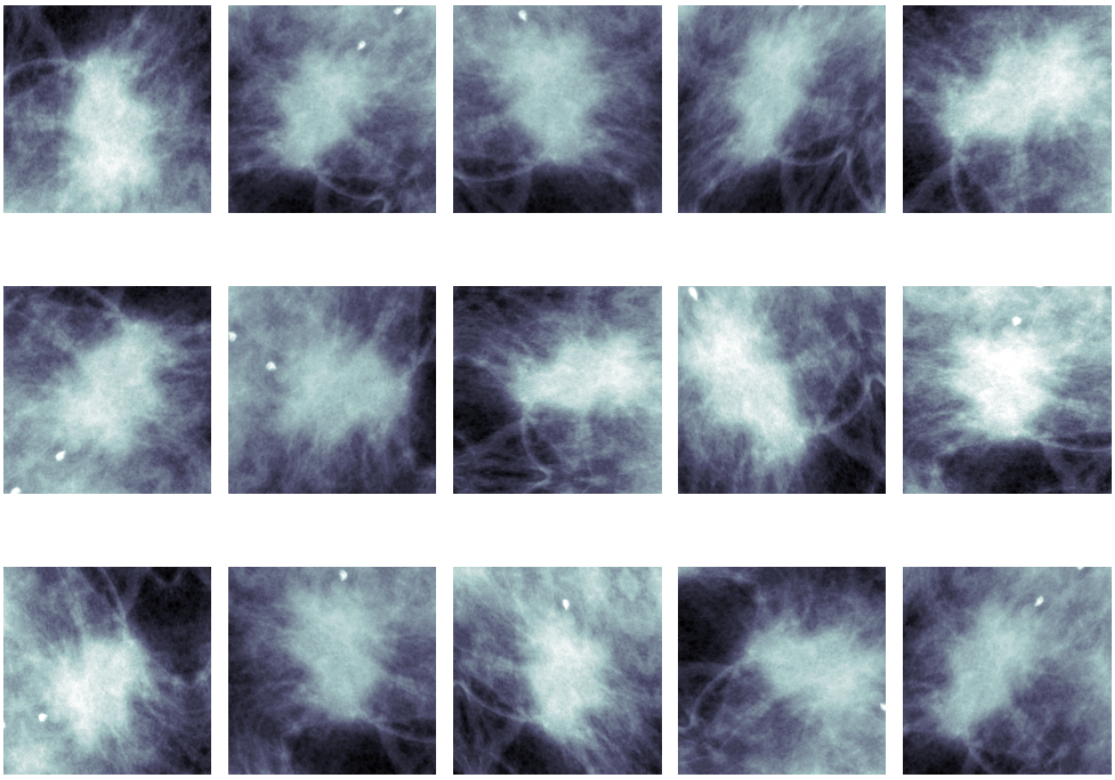


Figura 49. Distintas transformaciones aplicadas al parche original Figura 35. Estas son una combinación de las distintas transformaciones aplicadas a la imagen, rotación más brillo, rotación más *shearing*, etc.

Variante Global AveragePooling2D (GAvgP). Esta capa se puede usar desde **Tensorflow** y se puede aplicar directamente utilizando `tf.keras.layers.GlobalAveragePooling2D`. Recuerde que durante el proceso de entrenamiento las características que se van generando quedan capturadas en el mapa de características (el cubo de la Figura 16). La función de esta capa es la de reducir la dimensionalidad de este con el fin de no ingresar demasiada información que pueda sobreajustar el modelo utilizado, ya que en la sección 3.4, se describió que al usar la red VGG19 como extractor de características genera un cubo formado por 7×7 unidades espaciales y con 512 canales de características, el cuál debe ser aplanado antes de ser ingresado a un modelo de aprendizaje máquina. Pues la forma en la que actúa la capa `GlobalAveragePooling2D` es la de ir promediando los valores obtenidos en cada unidad espacial, es decir, que en lugar de obtener una unidad espacial de tamaño 7×7, se obtendrá solo un valor de estas, a saber, el promedio de todos los 49 elementos que conforman dicha unidad, por lo cuál, al final se obtiene un vector unidimensional de tamaño 512 de longitud, por lo que ya no debe de ser aplanado para ser ingresado a un modelo de aprendizaje máquina o al perceptrón multicapa mismo. La forma en la que trabaja esta capa se ilustra en la Figura 50.

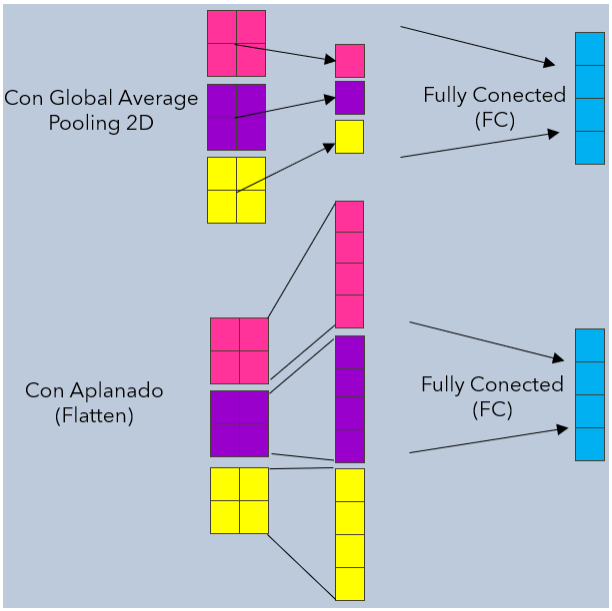


Figura 50. Representación del uso de las capas `Flatten` y `GlobalAveragePooling 2D`. En el uso de la capa `GlobalAveragePooling 2D`, en lugar de tomar cada elemento de los mapas características resultantes para crear el mapa final, solo se toma el promedio de cada unidad o 'rebanada', mientras que con el uso de `Flatten`, todo el mapa de características (el cubo resultantes) es aplanado para ser utilizado.

Es importante no perder de vista que en esta variante también se requiere el uso del **IDG**, es decir, como cada uno de los experimentos usarán esta capa en vez de la capa *Flatten* y se requiere el uso del **IDG** para el aumento de datos, se menciona que los parámetros utilizados en este son los propuestos.

Variante CLAHE. Se aplicó un preprocesamiennto a las imágenes: *Contrast Limited Adaptive Histogram Equalization*, abreviado por sus siglas en inglés CLAHE. Este es un método utilizado en el procesamiento de imágenes de ajuste del contraste usando el histograma de las imágenes para mejorar la calidad de estas (Zuiderveld, 1994). Esta técnica fue utilizada por Falconí et al. (2020), quienes trabajaron con la mastografía completa y las máscaras binarias para obtener los parches, normailzarlos (valores de 0 a 255), aplicarles CLAHE y luego un *resize* en este orden. Cabe decir que utilizaron una medida (*aspect ratio*) que relaciona el cociente de la altura y anchura de las imágenes obtenidas con las máscaras binarias y, aquellas que proporcionaban un *aspect ratio* fuera del rango [0.4,1.5], fueron descartadas. Cabe señalar que usó aumento de datos con distintas transformaciones, generando 60 mil imágenes, logrando obtener un *accuracy* de 81. %. Para los experimentos realizados en este trabajo de tesis, el CLAHE fue aplicado a los parches antes de redimensionarlos, sin descartar ninguna imagen.

Las Figuras 51 y 52 muestran los resultados de aplicar CLAHE con distintos valores. Cabe mencionar que este fue aplicado utilizando la librería **OpenCV**[2]usando el parámetro `clip_limit = 40` (valor predeterminado) en los experimentos.

Cabe mencionar que, al igual que la variante previamente descrita, se requiere el uso del **IDG** para el aumento de datos, por lo que los parámetros usados fueron los propuestos.

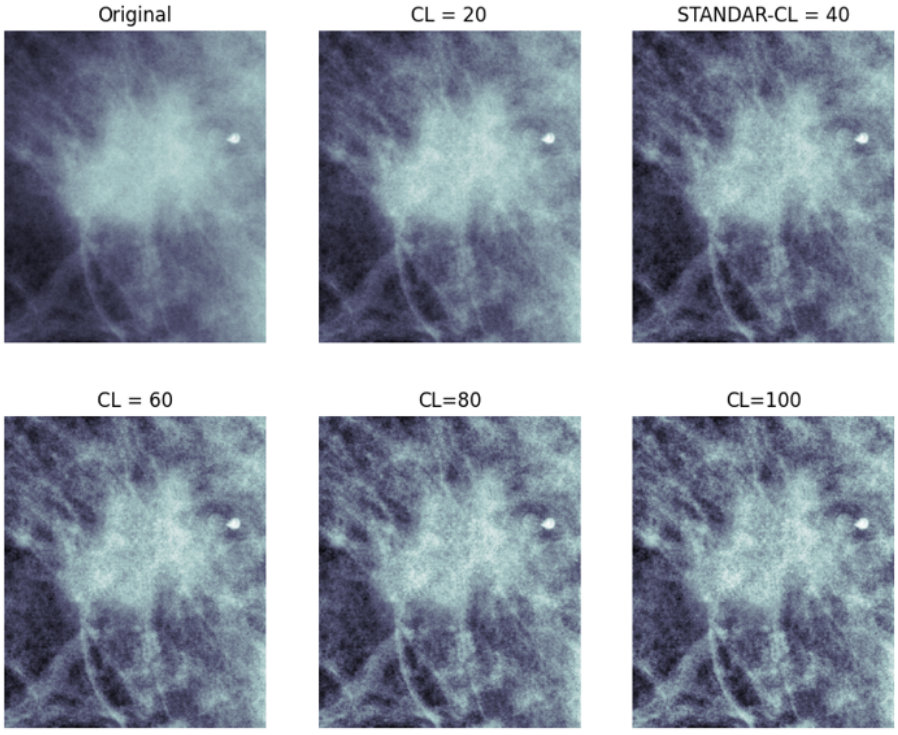


Figura 51. Niveles de CLAHE aplicados a la Figura 35 con el parámetro `clip_limit` (CL).

[2]https://opencv.org/

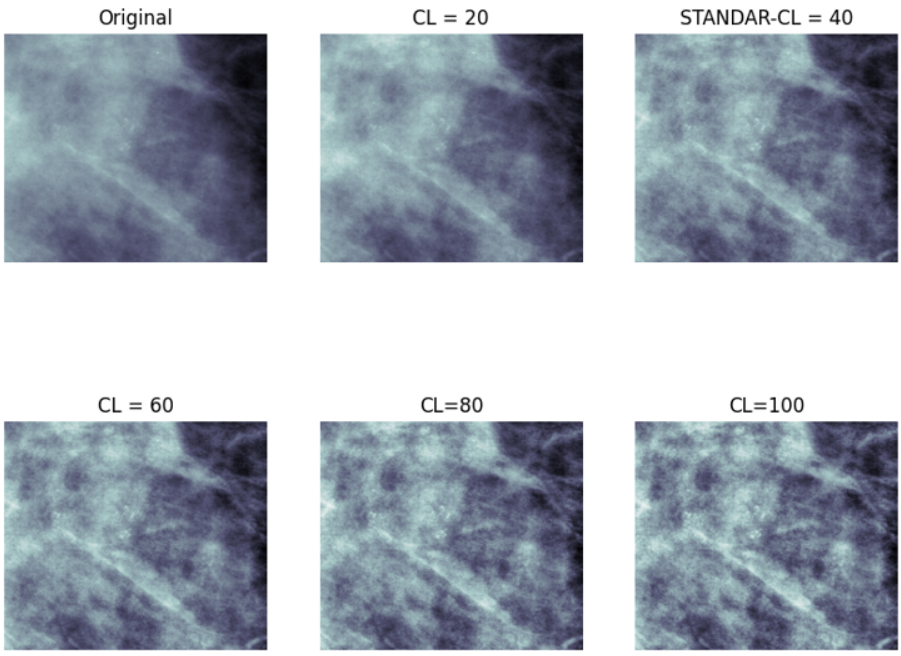


Figura 52. Niveles de CLAHE aplicados a la Figura 34 con el parámetro `clip_limit` (CL).

4.2.2. Primeros Resultados

Si se observa bien, los 150 experimentos propuestos se combinaron con las cuatro variaciones anteriores, dando un resultado de al menos 600 experimentos. Para ser más preciso considere lo siguiente:

- **Experimento Masas**. Se usaron las 4 variantes en el experimento con cada una de las 6 CNNs y cambiando el número de neuronas 5 veces. Esto da como resultado 120 experimentos.
- **Experimento Calcificaciones**. Se usaron las 4 variantes en el experimento con cada una de las 6 CNNs y cambiando el número de neuronas 5 veces. Esto da como resultado 120 experimentos.
- **Experimento Multiclase**. Se usaron las 4 variantes en el experimento con cada una de las 6 CNNs y cambiando el número de neuronas 5 veces. Esto da como resultado 120 experimentos.
- **Experimento M-B**. Se usaron las 4 variantes en el experimento con cada una de las 6 CNNs y cambiando el número de neuronas 5 veces. Esto da como resultado 120 experimentos.
- **Experimento M-C**. Se usaron las 4 variantes en el experimento con cada una de las 6 CNNs y cambiando el número de neuronas 5 veces. Esto da como resultado 120 experimentos.

Aunque resulte repetitivo, se puede entender con más claridad que el número de experimentos o veces en correr los entrenamientos TL y FT fue de al menos 600 veces.

Antes de mostrar los resultados obtenidos en cada uno de los experimentos, se puede decir que **no hubo un experimento que sobresaliera tanto del resto**, sin embargo, los resultados en los experimentos en los que se utilizó la capa GAvgP fueron casi siempre ligeramente menores con respecto al *accuracy* obtenido, mientras que en los experimentos restantes este se mantuvo más parejo.

A continuación se muestra un diagrama (Figura 53) que resume el trabajo realizado en esta primera parte.

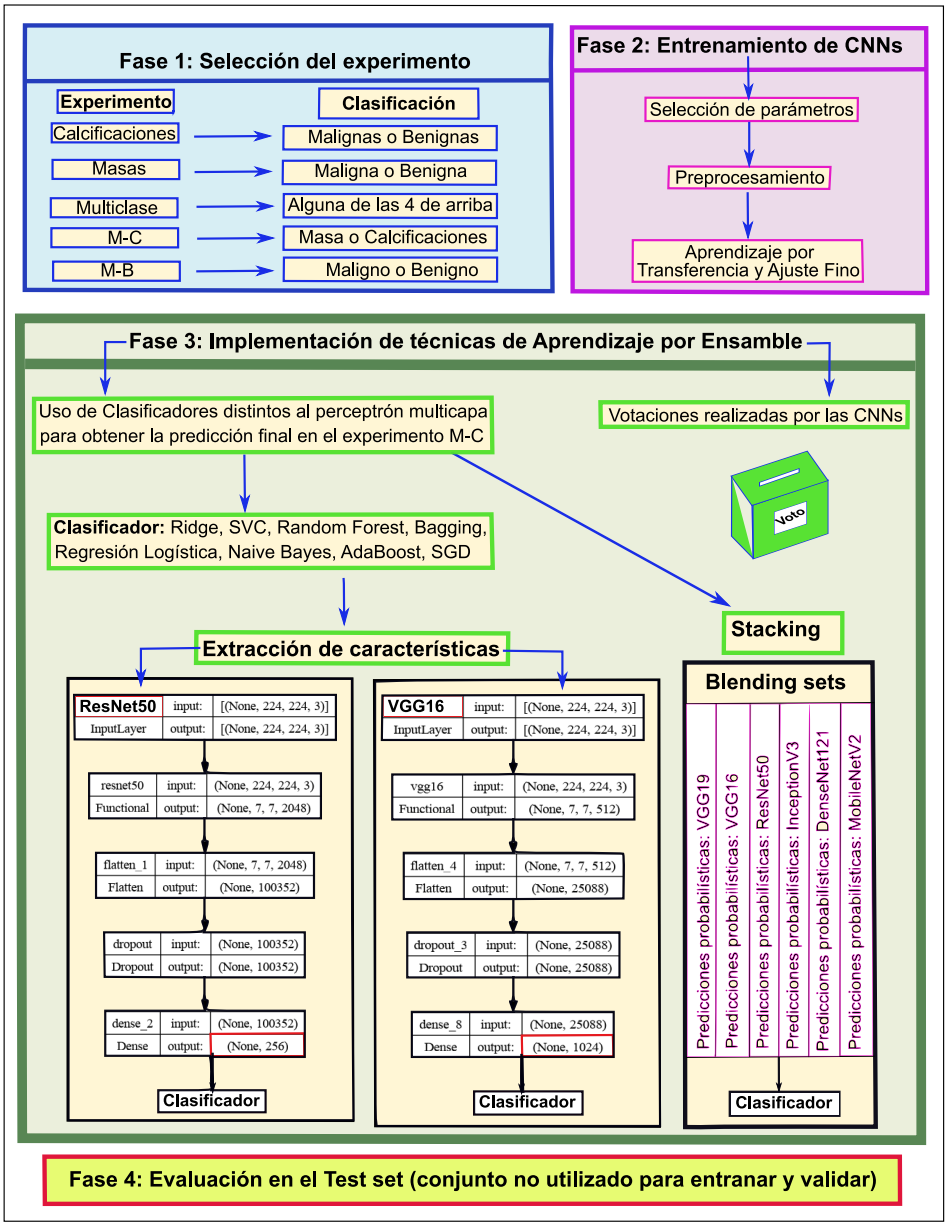
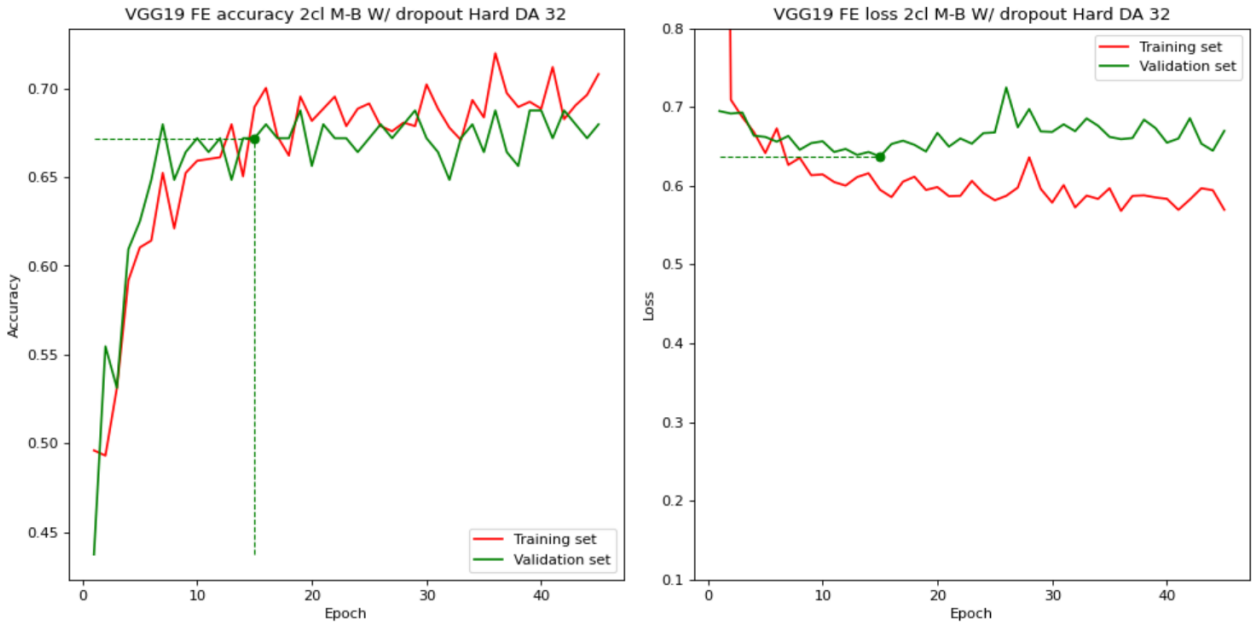


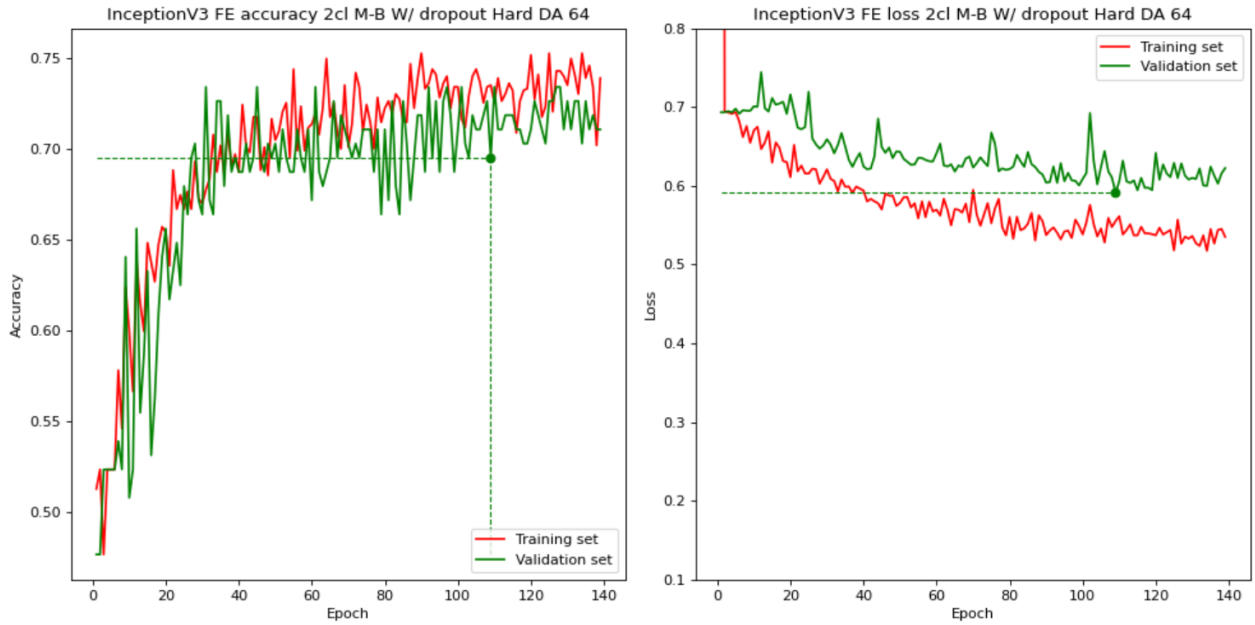
Figura 53. Diagrama que resume el trabajo realizado en esta primera parte.

4.2.3. Experimento Masas (Variante Parámetros Iniciales)

Como se mencionó antes, se experimentó en algunos modelos desde 16 y 32 neuronas hasta 2048. El mejor resultado al utilizar la **Variante Parámetros Iniciales** fue usar 32 neuronas con el modelo VGG19 y 64 en el modelo InceptionV3. Las gráficas de exactitud y pérdida de estos modelos se pueden ver en la Figura 54. Los resultados se muestran en la Tabla 6 y la Figura 55.



Gráfica de entrenamiento y validación (*accuracy* y *loss*) de la red VGG19.



Gráfica de entrenamiento y validación (*accuracy* y *loss*) de la red InceptionV3.

Figura 54. Gráficas de entrenamiento y validación de los modelos InceptionV3 y VGG19, los cuales obtuvieron el mismo accuracy en el conjunto de *Test*: 0.7448 (Tabla 6).

Tabla 6. Resultados asociados con las matrices de confusión de la Figura 55.

Modelo	Número de Neuronas	Accuracy	Imágenes mal clasificadas
DenseNet121	512	0.7097	99
ResNet50	128	0.7067	100
InceptionV3	64	0.7448	87
MobileNetV2	256	0.6891	106
VGG16	64	0.7214	95
VGG19	32	0.7448	87

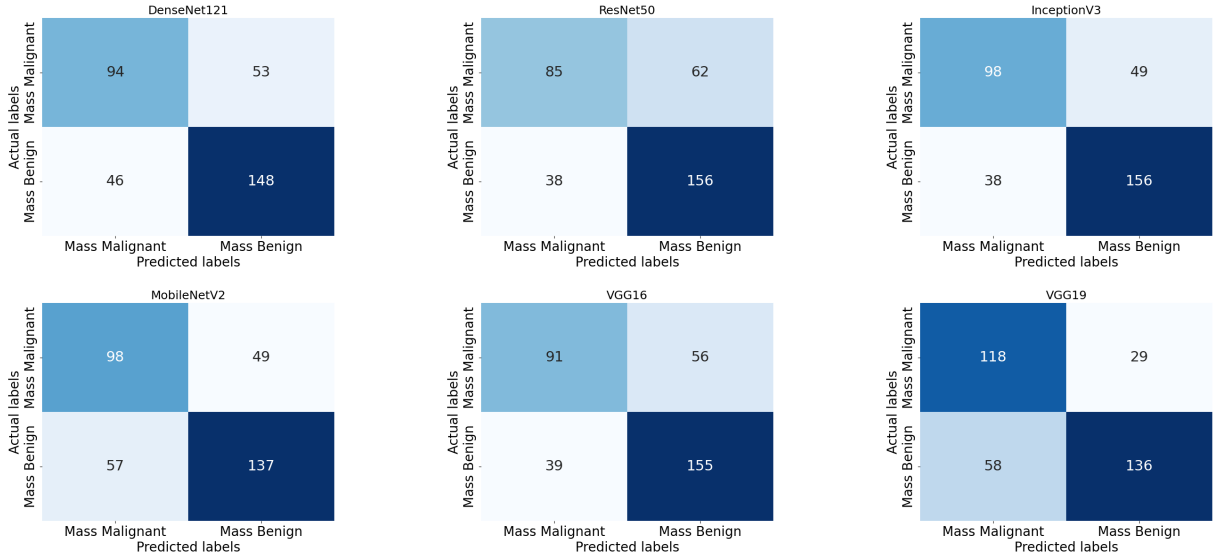


Figura 55. Resultados individuales obtenidos por cada una de las CNNs en el *Test* (341 imágenes), las cuales corresponden al uso de la **Variante Parámetros Iniciales**.

4.2.4. Experimento Calcificaciones (Variante CLAHE)

En este experimento se obtuvieron resultados similares en todas las combinaciones, sobresaliendo ligeramente la aplicación de la **Variante CLAHE**. La Tabla 7 presenta estos resultados respaldados por las matrices de confusión de la Figura 56, mientras que la gráfica de exactitud y pérdida del mejor modelo (DenseNet121) se muestra en la Figura 57.

Tabla 7. Resultados asociados con las matrices de confusión de la Figura 56.

Modelo	Número de Neuronas	Accuracy	Imágenes mal clasificadas
DenseNet121	256	0.6756	84
ResNet50	1024	0.6679	86
InceptionV3	1024	0.6216	98
MobileNetV2	256	0.6409	93
VGG16	512	0.6563	89
VGG19	1024	0.6486	91

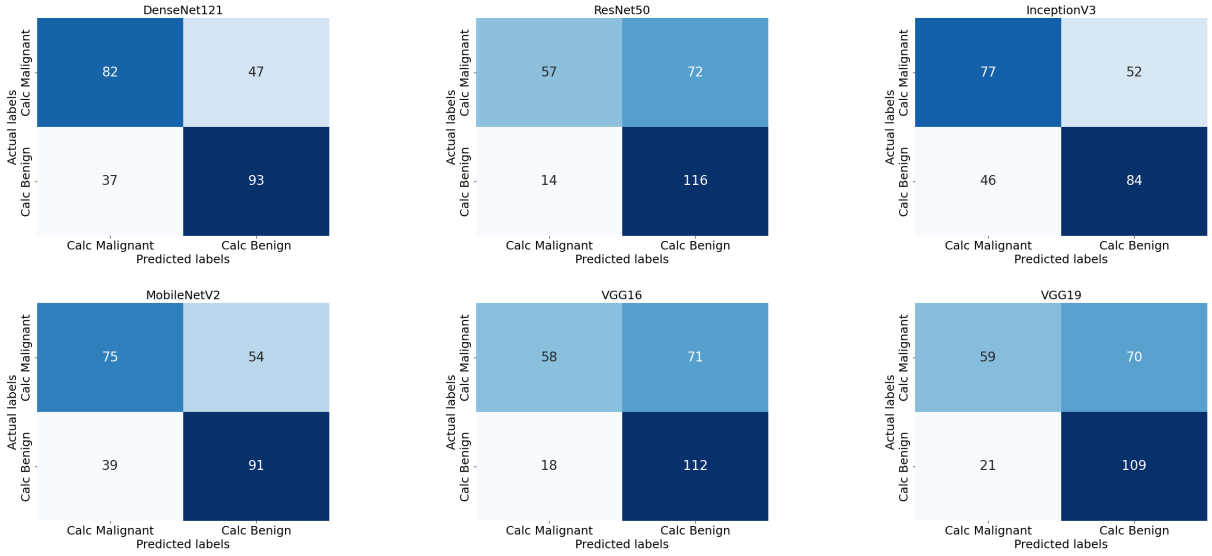


Figura 56. Resultados individuales obtenidos por cada una de las CNNs en el *Test* (259 imágenes), las cuales corresponden al uso de la **Variante CLAHE**.

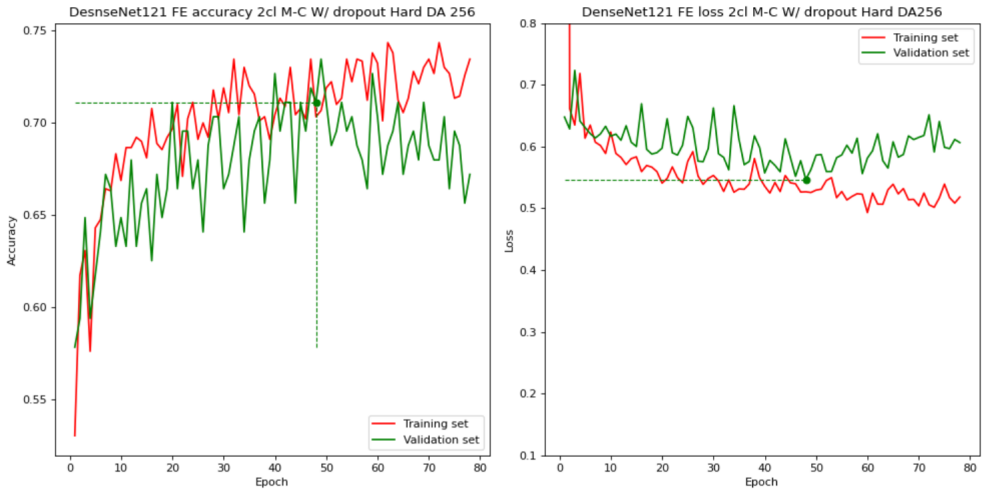


Figura 57. Gráfica de entrenamiento y validación (*accuracy* y *loss*) obtenida por la red DenseNet121, la cual obtuvo un 0.6756 en el *Test* (Tabla 7).

4.2.5. Experimento Multiclase (Variante Parámetros Iniciales)

De todos los experimentos realizados este fue en el que menor *accuracy* se obtuvo, por lo que no se continuó la experimentación. Solo se muestran las matrices de confusión obtenidas por todos los modelos (Figura 58) junto con las curvas de exactitud y pérdida del modelo (ResNet50) con mayor exactitud lograda (Figura 59).

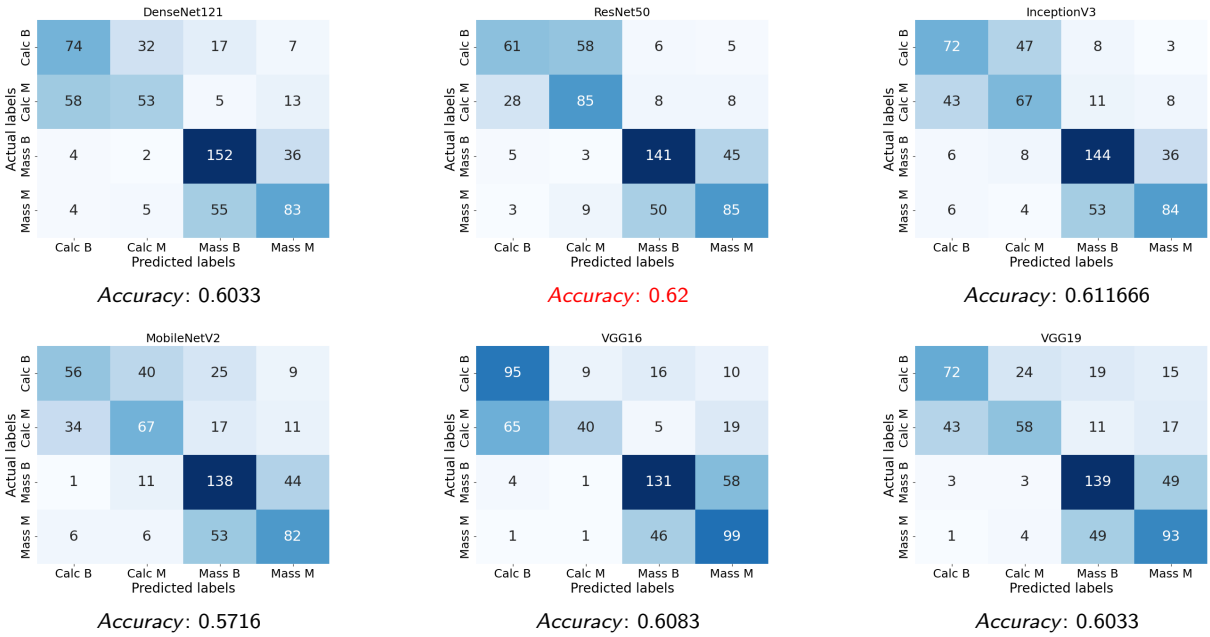


Figura 58. Resultados individuales obtenidos por cada una de las CNNs en el *Test* (600 imágenes), las cuales corresponden al uso de la **Variante Parámetros Iniciales**.

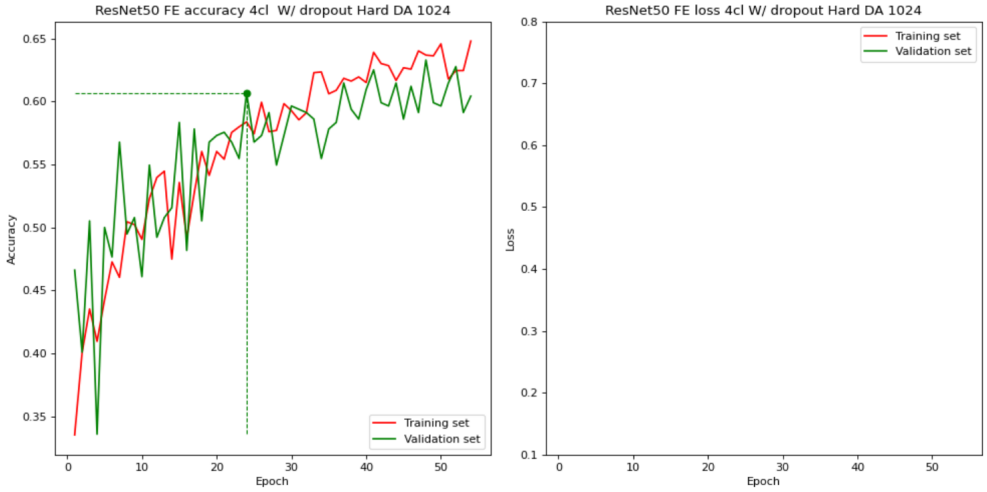


Figura 59. Gráfica de entrenamiento y validación (*accuracy* y *loss*) del modelo ResNet50. En esta se puede ver que se usaron 1024 neuronas en la capa FC, además de que la gráfica de pérdida o loss (derecha) no se ve, ya que no tiende a 0, lo cual indica que el modelo no está obteniendo resultados lo suficientemente buenos.

4.2.6. Experimento M-B (Variantes: Parámetros Iniciales y CLAHE)

En este experimento sobresalieron las variantes con **Parámetros Iniciales** y **CLAHE**. Los resultados correspondientes a la **Variante Parámetros Iniciales** se ilustran en la Figura 60 con la gráfica de exactitud y pérdida (Figura 61) del modelo VGG16, el cual alcanzó la mayor exactitud (Tabla 8).

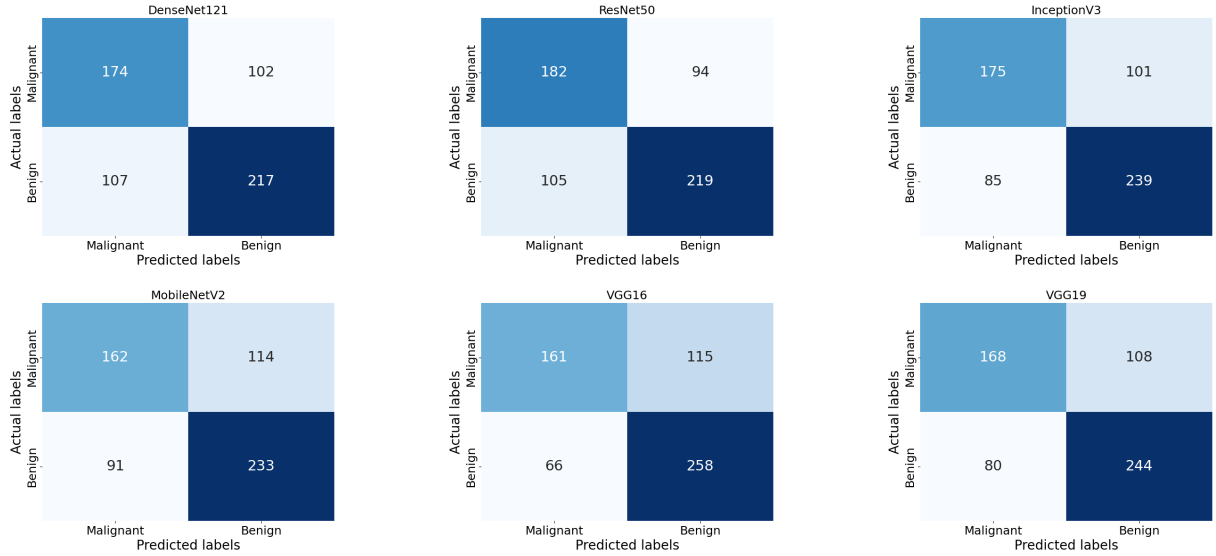


Figura 60. Resultados individuales obtenidos por cada una de las CNNs en el *Test* (600 imágenes) con el **Experimento M-B**, correspondientes al uso de la **Variante Parámetros Iniciales**.

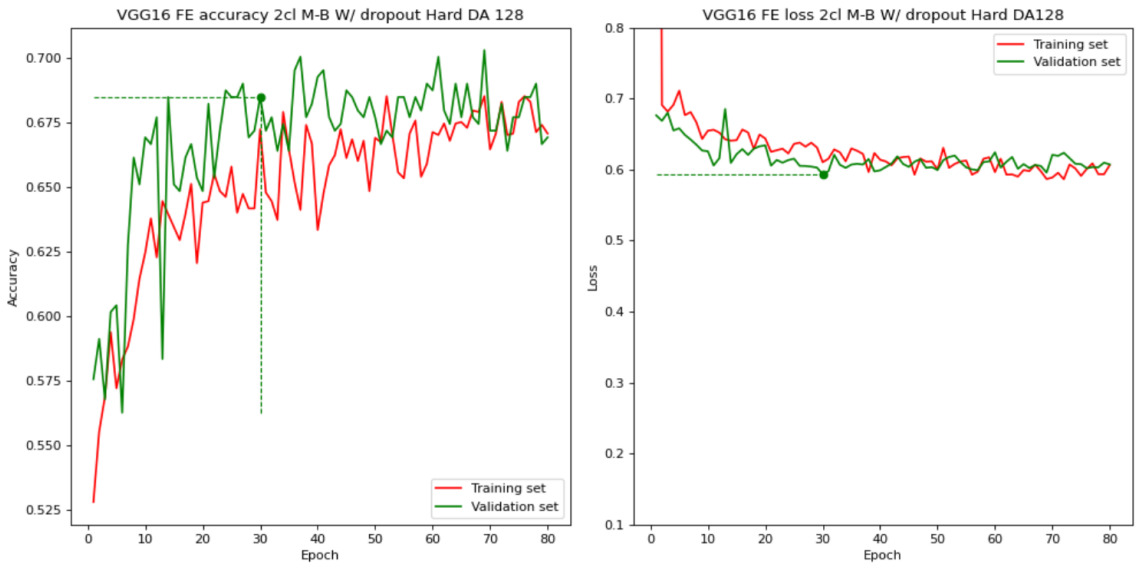


Figura 61. Gráfica de entrenamiento y validación (*accuracy* y *loss*) del modelo VGG16 en el experimento con la **Variante Parámetros Iniciales**, el cual obtuvo un *accuracy* en el *Test* de: 0.6983 (Tabla 8).

Tabla 8. Resultados asociados con las matrices de confusión de la Figura 60.

Modelo	Número de Neuronas	Accuracy	Imágenes mal clasificadas
DenseNet121	1024	0.6516	209
ResNet50	1024	0.6683	199
InceptionV3	256	0.69	186
MobileNetV2	256	0.6583	205
VGG16	128	0.6983	181
VGG19	256	0.6866	188

Con respecto a la **Variante CLAHE**, la Tabla 9 muestra los resultados obtenidos asociados con las matrices de confusión de la Figura 62, además se muestra la gráfica de exactitud y pérdida (Figura 63) del modelo (VGG19) que alcanzó la mayor exactitud.

Tabla 9. Resultados asociados con las matrices de confusión de la Figura 62.

Modelo	Número de Neuronas	Accuracy	Imágenes mal clasificadas
DenseNet121	2048	0.6916	185
ResNet50	512	0.6816	191
InceptionV3	256	0.675	195
MobileNetV2	256	0.6616	203
VGG16	256	0.6916	183
VGG19	512	0.7016	179

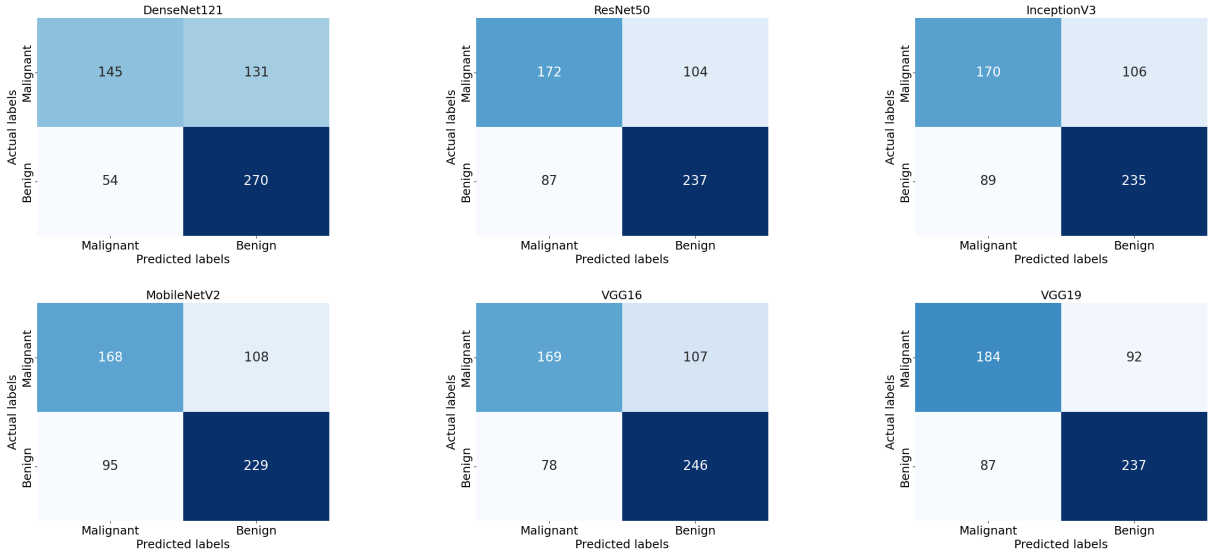


Figura 62. Resultados individuales obtenidos por cada una de las CNNs en el *Test* (600 imágenes) con en el **Experimento M-B**, correspondientes al uso de la **Variante CLAHE**.

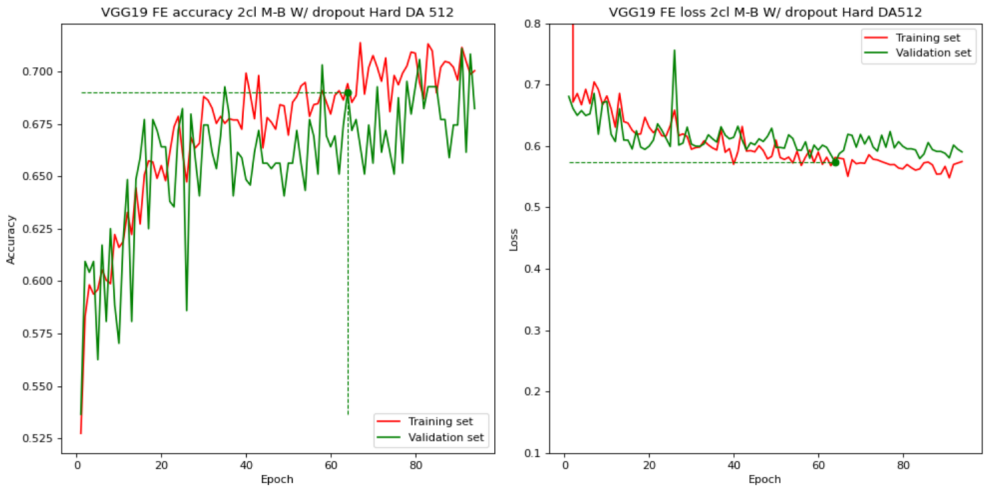


Figura 63. Gráfica de entrenamiento y validación (*accuracy* y *loss*) del modelo VGG19 en el experimento con **Variante CLAHE**, el cual obtuvo un *accuracy* en el *Test* de: 0.7016 (Tabla 9)

4.2.7. Experimento M-C (Variante Parámetros Iniciales)

Este experimento lo realizó Lai (2021) y, en efecto, con estos parámetros fueron con los que se obtuvieron mejores resultados, los cuales se ilustran en las matrices de confusión (Figura 64), mientras que las curvas de exactitud y pérdida del modelo ResNet50 se pueden observar en la Figura 65. Este modelo alcanzó la mayor exactitud (Tabla 10).

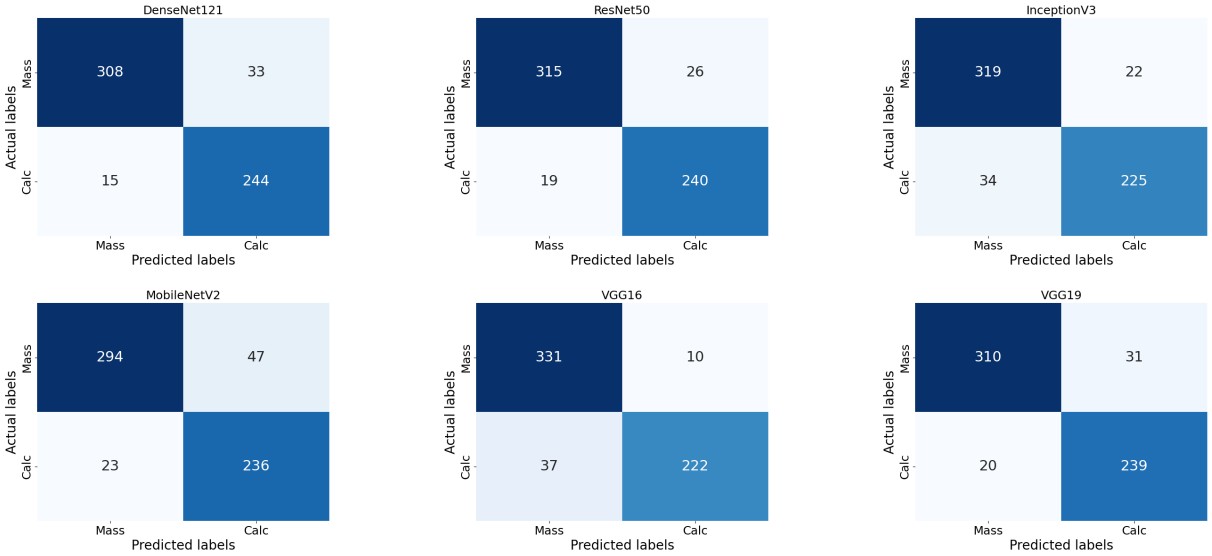


Figura 64. Resultados individuales obtenidos por cada una de las CNNs en el *Test* (600 imágenes) con el **Experimento M-C**, correspondientes al uso de la **Variante Parámetros Iniciales**.

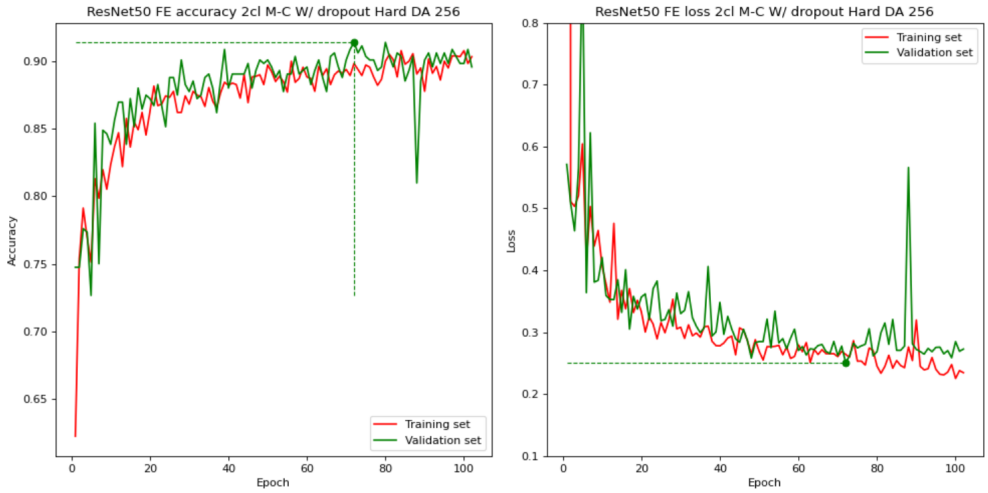


Figura 65. Gráfica de entrenamiento y validación (*accuracy* y *loss)* del modelo ResNet50, el cual obtuvo un *accuracy* en el conjunto de *Test* de: 0.925 (Tabla 10).

Tabla 10. Resultados asociados con las matrices de confusión de la Figura 64.

Modelo	Número de Neuronas	Accuracy	Imágenes mal clasificadas
DenseNet121	1024	0.92	48
ResNet50	256	0.925	45
InceptionV3	256	0.9066	57
MobileNetV2	1024	0.8833	70
VGG16	256	0.9216	47
VGG19	64	0.915	51

4.2.8. Resumen Experimental de los Primeros Resultados

A continuación se muestra un resumen de los resultados antes mostrados (Tabla 11), los cuales se compararon con los obtenidos por Lai (2021) y Jaamour et al. (2023), ya que siguieron la misma estrategia, es decir, utilizar solamente el conjunto de entrenamiento y *test* que provee la base de datos CBIS-DDSM, pero aún así, cabe señalar que ambas referencias tienen aproximadamente el mismo número de imágenes que el que se reporta en este documento. Además, Jaamour et al. (2023) no utilizó el **IDG**, por el contrario, realizó otro tipo de aumento de datos con las imágenes, así como experimentar con tamaños de 224, 512 y 1024 pixeles.

Tabla 11. Resumen experimental de la sección 4.2.1.

Experimento	Red con accuracy en Test	Jaamour et al. (2023) accuracy en Test	Lai (2021) accuracy en Test
Masas	Varias:0.7448	VGG19: 0.6435	No realizado
Calcificaciones	DenseNet121: 0.6756	VGG19: 0.6705	No realizado
Multiclase	ResNet50: 0.625	No realizado	No realizado
Experimento M-B	VGG19: 0.7016	MobileNetV2: 0.6708	VGG16: 0.69
Experimenot M-C	ResNet50: 0.925	No realizado	VGG16: 0.9166

De la Tabla 11 se puede concluir que los experimentos realizados superan ligeramente los resultados obtenidos por Lai (2021) y Jaamour et al. (2023), excepto el obtenido en Masas, pues es más alto. Lo que sigue es tratar de elevar la tasa de clasificación utilizando el Aprendizaje por Ensamble, pero no de todos los experimentos, ya que como se mencionó antes, esto requiere más tiempo computacional.

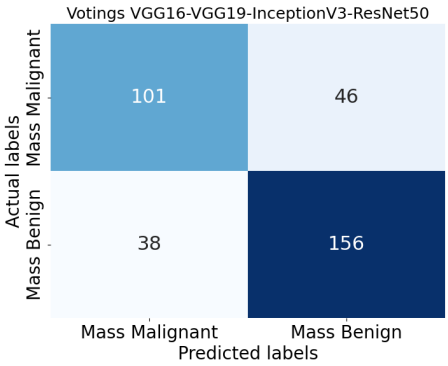
4.2.9. Aplicación del Aprendizaje por Ensamble

En esta sección se aplican las técnicas descritas en el capítulo 2, comenzando con las votaciones de las CNNs entrenadas en la sección anterior.

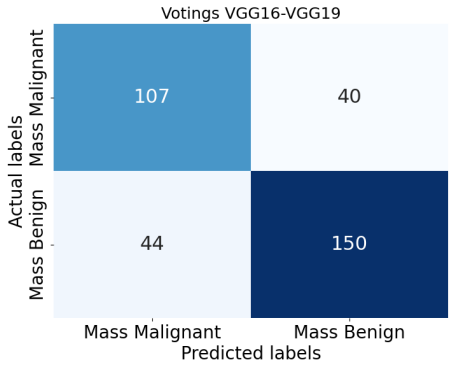
4.2.9.1. Votaciones de los Primeros Resultados

Las CNNs entrenadas, dependiendo la tarea de clasificación binaria o multiclase (con *activación Sigmoide* o *Softmax* respectivamente), son clasificadores probabilísticos (descritos en la sección 4.2, por lo que se podría realizar votaciones a partir de estas. Dado que se experimentó con 6 CNNs, el número de combinaciones distintas que se pueden formar es de 57 (sección 2.4.4), pero a continuación se presentan solo los mejores resultados para cada uno de los experimentos mostrados en la sección previa.

La Figura 66 muestra las matrices de confusión de las dos mejores combinaciones por votaciones en el **Experimento Masas** con la **Variante de Parámetros Iniciales**, ambas obteniendo un *accuracy* del 75.36 %.



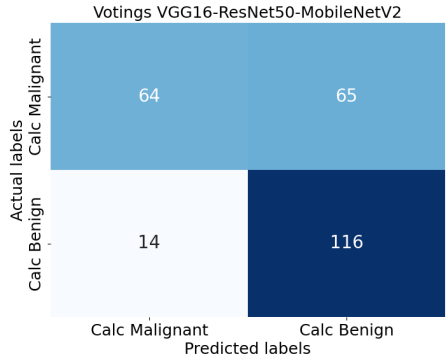
La votación de estos 4 clasificadores genera un *accuracy* del 0.7536, al clasificar mal 84 imágenes de 341.



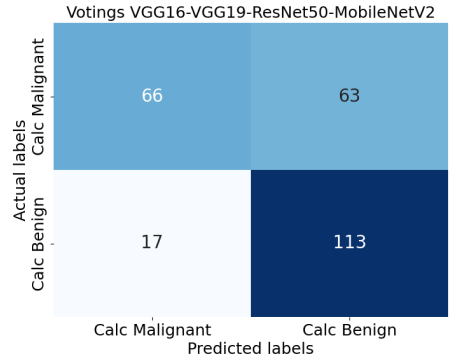
La votación de estos 2 clasificadores genera un *accuracy* del 0.7536, al clasificar mal 84 imágenes de 341.

Figura 66. Votaciones de las redes del **Experimento Masas** con la **Variante Parámetros Iniciales**: ambas combinaciones generan un *accuracy* ligeramente superior al obtenido sin este método, el cuál era de 0.7448 clasificando mal 87 imágenes de 341.

La Figura 67 muestra las matrices de confusión de las dos mejores combinaciones por votaciones en el **Experimento Calcificaciones** con la **Variante CLAHE**, una obteniendo un *accuracy* del 96.4 % y la otra un 96.1 %.



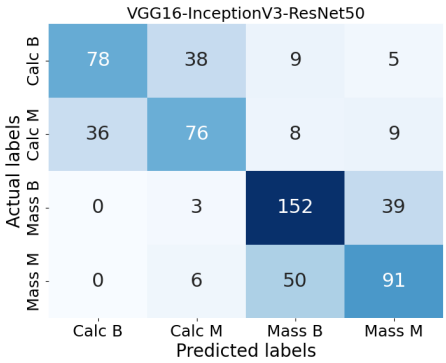
La votación de estos 3 clasificadores genera un *accuracy* del 0.694, al clasificar mal 79 imágenes de 259.



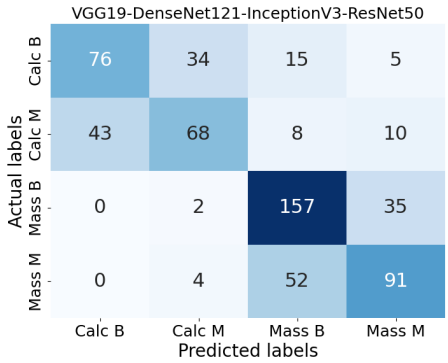
La votación de estos 4 clasificadores genera un *accuracy* del 0.691, al clasificar mal 80 imágenes de 259.

Figura 67. Votaciones de las redes del **Experimento Calcificaciones** con la **Variante CLAHE**: ambas combinaciones generan un *accuracy* ligeramente superior al obtenido sin este método, el cuál era de 0.6756, clasificando mal 84 imágenes de 259.

La Figura 68 muestra las matrices de confusión de las dos mejores combinaciones por votaciones en el **Experimento Multiclase** con la **Variante Parámetros Iniciales**, una obteniendo un *accuracy* del 66.16 % y la otra un 65.33 %.



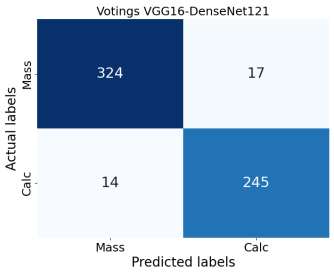
La votación de estos 3 clasificadores genera un *accuracy* del 0.6616.



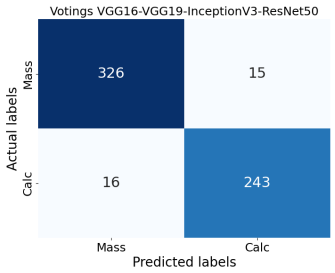
La votación de estos 4 clasificadores genera un *accuracy* del 0.6533.

Figura 68. Votaciones de las redes del **Experimento Multiclase** con la **Variante Parámetros Iniciales**: ambas combinaciones generan un *accuracy* ligeramente superior al obtenido sin este método, el cuál era de 0.62.

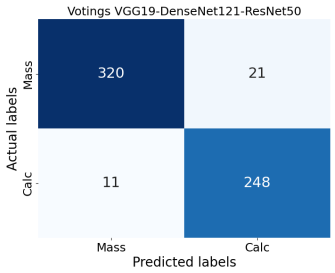
La Figura 69 muestra las matrices de confusión de las tres mejores combinaciones por votaciones en el **Experimento M-C** con la **Variante Parámetros Iniciales**, dos de ellas obtuvieron un *accuracy* del 94.83 %, mientras que la restante obtuvo un 94.66 %.



La votación de estos 2 clasificadores genera un *accuracy* del 0.9483, al clasificar mal 31 imágenes de 600.



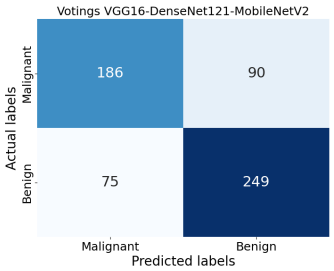
La votación de estos 4 clasificadores genera un *accuracy* del 0.9483, al clasificar mal 31 imágenes de 600.



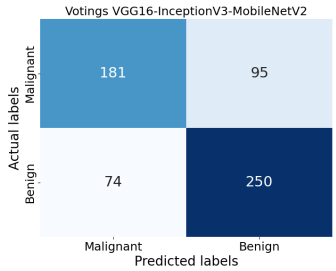
La votación de estos 3 clasificadores genera un *accuracy* del 0.9466, al clasificar mal 132 imágenes de 600.

Figura 69. Votaciones de las redes del **Experimento M-C** con la **Variante Parámetros Iniciales**: ambas combinaciones generan un *accuracy* ligeramente superior al obtenido sin este método, el cuál era de 0.925.

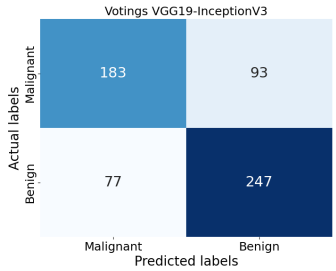
La Figura 70 muestra seis matrices de confusión. La parte superior muestra tres matrices de confusión de las tres mejores combinaciones por votaciones en el **Experimento M-B** con la **Variante Parámetros Iniciales**, alcanzando un *accuracy* la mejor combinación del 72.5 %, mientras que la parte inferior muestra tres matrices de confusión de las tres mejores combinaciones por votaciones en el **Experimento M-B** con la **Variante CLAHE**, alcanzando un *accuracy* la mejor combinación del 71.33 %.



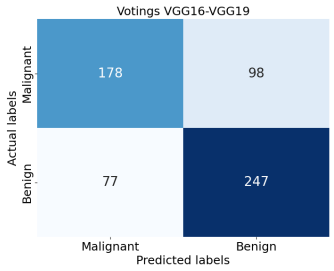
La votación de estos 3888 clasificadores genera un *accuracy* del 0.725, al clasificar mal 165 imágenes de 600.



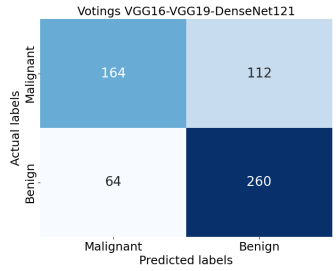
La votación de estos 3 clasificadores genera un*accuracy* del 0.7183, al clasificar mal 169 imágenes de 600.



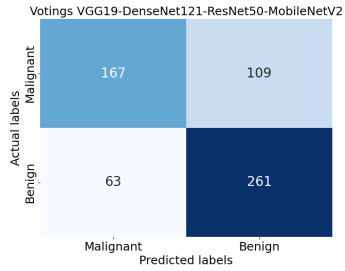
La votación de estos 2 clasificadores genera un *accuracy* del 0.7166, al clasificar mal 170 imágenes de 600.



La votación de estos 2 clasificadores genera un *accuracy* del 0.7083, al clasificar mal 175 imágenes de 600.



La votación de estos 3 clasificadores genera un *accuracy* del 0.7133, al clasificar mal 176 imágenes de 600.



La votación de estos 4 clasificadores genera un *accuracy* del 0.7066, al clasificar mal 172 imágenes de 600.

Figura 70. Ambas combinaciones generan un *accuracy* ligeramente superior al obtenido sin este método. Las matrices en la parte superior son las votaciones referentes a los clasificadores que utilizaron la **Variante Parámetros Iniciales**, de las cuáles el mejor *accuracy* obtenido fue de 0.6983 al clasificar mal 181 imágenes, mientras que las matrices inferiores son las votaciones referentes a los clasificadores que utilizaron la **Variante CLAHE**, de las cuáles el mejor *accuracy* obtenido fue de 0.7016 al clasificar mal 179 imágenes.

Conclusiones. Claramente el uso de votaciones incrementó la tasa de clasificación en todos los experimentos sin excepción, sin embargo, los resultados obtenidos no son tan asombrosos, ya que la tasa de clasificación no aumenta ni en un 5 % de *accuracy* en el mejor de los casos. Por otro lado, una observación interesante es que los resultados individuales obtenidos por las CNNs en el **Experimento M-B** no llegan al 70 % de *accuracy* al utilizar la **Variante Parámetros Iniciales**, mientras la red VGG19 alcanza a llegar al 70 % de *accuracy* al utilizar la **Variante CLAHE**, por lo que se esperaría que al realizar las votaciones esta misma variante obtenga el mayor *accuracy*, sin embargo, al revisar la Figura 70, la matriz de confusión con más alto *accuracy* es la generada por la combinación o votaciones de las redes VGG16, DenseNet121 y MobileNetV2 pero pertenecientes a la **Variante Parámetros Iniciales**, alcanzando un *accuracy* del 72.5 %, mientras que la combinación mas alta perteneciente a la **Variante CLAHE** es de 71.33 %. En el **Experimento Masas**, correspondiente a los resultados relacionados con la **Variante Parámetros Iniciales**, se esperaba que con la sola combinación de las redes InceptionV3 y VGG19 se obtuviera el *accuracy* más alto, ya que el más alto obtenido de manera individual correspondía a estas redes, lo cual no ocurrió.

4.2.9.2. Sobre el Experimento M-C

En esta parte se presentan los resultados obtenidos al aplicar la técnica de **FC Extraction** a la red **ResNet50**, la cual había obtenido un *accuracy* de 0.925 con 256 neuronas (Tabla 10), por lo que las 971 imágenes con las que se realizó el entrenamiento (el conjunto de validación correspondiente al 20 % se mantuvo a parte) fueron utilizadas para generar un nuevo conjunto de entrenamiento de forma (971,256), el cuál fue utilizado para entrenar modelos de aprendizaje máquina. Cabe destacar que se usó el conjunto de *Test* de 600 imágenes para evaluar estos algoritmos, formando un conjunto de *Test* de tamaño (600,256). Los resultados se muestran en la Tabla 12.

Tabla 12. Resultados de clasificadores que superaron el*accuracy* de 0.92

Clasificador	Accuracy
Ridge CI	0.92833
Random Forest	0.9266
Bagging(DTree)	0.93
Best Params SVC	0.9266
Best Params Ridge	0.9283
Best Params Bagging(DTree)	0.9283
Best Params SGD	0.9266
Best Params Bagging(BP SVC)	0.93

También se obtuvieron las votaciones de los clasificadores utilizados en esta parte de la Experimentación. Los resultados se muestran en la Tabla 13 y la matriz de confusión asociada al resultado en color azul se muestra en la Figura 71.

Tabla 13. Resultados de votaciones de algunos clasificadores que superaron el *accuracy* de 0.92.

Clasificador	Accuracy
BP SVC - Bagging(DTree) - Ridge stdr	0.935
BP SVC - Bagging(DTree) - Gradient Boosting stdr	0.935
Bagging(DTree) - BP Bagging(BP SVC) - Gradient Boosting stdr	0.935
Bagging(DTree) - Ridge stdr - Gradient Boosting stdr	0.9366
BP SVC - Bagging(DTree) - BP Bagging(BP SVC) - Ridge stdr - Gradient Boosting stdr	0.935

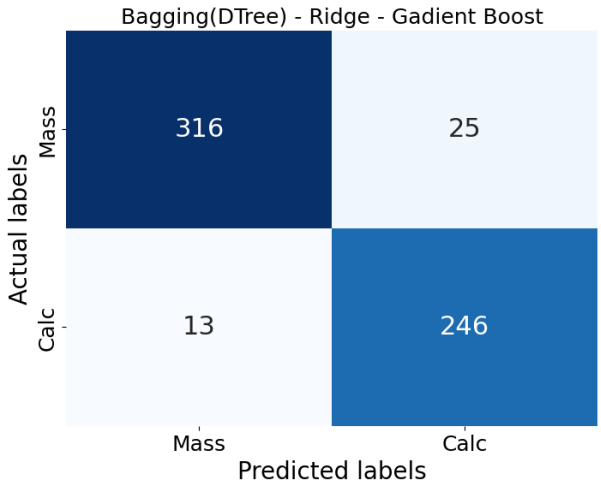


Figura 71. Se muestra la matriz de confusión obtenida de la combinación marcada en azul de la Tabla 13, la cuál superó el *accuracy* de 0.92 siendo la más alta entre todas.

La técnica de Stacking también fue explorada. Para esto, los *blending sets* fueron construidos a partir del conjunto de validación y al contar con 6 CNNs es posible construir, como ya se mencionó, $2^6 - (6+1)$ combinaciones, sin embargo, solo se muestran los mejores resultados en la Tabla 14 y el mejor resultado en rojo, cuya matriz de confusión de este meta-clasificador se puede ver en la Figura 72.

Tabla 14. Resultados de meta-clasificadores y *blending sets* que superaron el *accuracy* de 0.92

Blending set	**Meta-clasificador**	**Accuracy**
VGG16 - VGG19 - DenseNet121 - ResNet50	Naive Bayes	0.945
VGG16 - VGG19 - DenseNet121	SGD	0.945
VGG19 - DenseNet121	SVC	0.9383
VGG16 - DenseNet121	Logistic Regression	0.9466
VGG19 - DenseNet121	KNN	0.9383
VGG16 - VGG19 - DenseNet121 - ResNet50	Ada Boost (Naive Bayes)	0.945
VGG16 - VGG19 - DenseNet121	Ada Boost (SGD)	0.9416
VGG19 - DenseNet121	Bagging (SVC)	0.94
VGG16 - DenseNet12	Ada Boost(Logistic Regression)	0.9466
VGG19 - DenseNet121	Gradient Boost CL	0.935
VGG16 - DenseNet121	Ridge stdr	0.9483

Conclusiones. Con respecto a este experimento es claro que las técnicas propuestas también mejoran la tasa de clasificación (ligeramente), ya que se logra pasar de un 92.5 % de *accuracy* a un 98.83 %, el cual ya había sido obtenido en las votaciones anteriormente realizadas. Por último, se menciona que en los Anexos se obtiene un mayor *accuracy*.

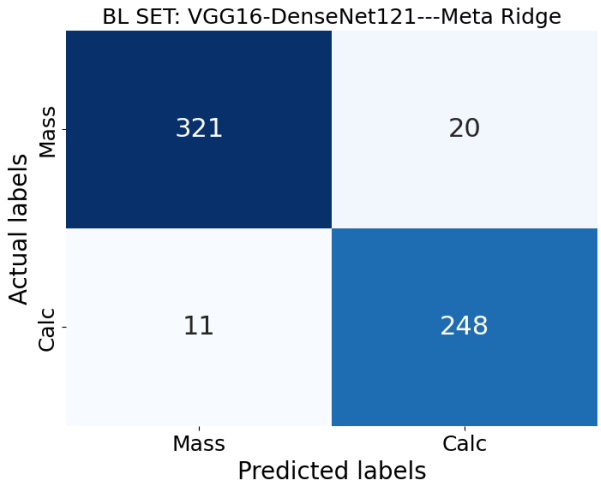


Figura 72. Se muestra la matriz de confusión obtenida de la combinación marcada en rojo de la Tabla 14 y el meta-clasificador, la cuál superó el *accuracy* de 0.92 siendo la más alta entre todas.

4.2.10. Resumen Experimental de los Primeros Resultados (después del Aprendizaje por Ensamble)

En la sección pasada se pudo corroborar que las técnicas aplicadas mejoran el *accuracy* obtenido, por lo que en la Tabla 15 se comparan los resultados obtenidos antes y después de aplicar técnicas de ensamble.

Tabla 15. Resumen experimental después del uso de Ensambles.

Experimento	Resumen Experimental con Ensambles Accuracy en Test	Resumen Experimental Tabla11 Accuracy en Test
Masas	VGG16-VGG19: 0.7536	VGG19 e InceptionV3: 0.7448
Calcificaciones	VGG16-ResNet50-MobileNetV2: 0.694	DenseNet121: 0.6756
Multiclase	VGG16-InceptionV3-ResNet50: 0.6616	ResNet50: 0.625
M-B	VGG16-DenseNet121-MobileNetV2: 0.725	VGG19: 0.7016
M-C	VGG16-DenseNet121: 0.9483	ResNet50: 0.925
M-C (Anexos)	ResNet50-DenseNet121: 0.9533	ResNet50: 0.945

Si bien los resultados obtenidos mejoraron ligeramente, en las secciones posteriores se tratará de elevar

más las tasas de clasificación explorando más a fondo el aprendizaje por ensamble, enfocándose más en la clasificación de patologías, es decir, de cáncer de mama.

Nota: Estos experimento fueron realizados en un entorno virtual de python (venv) en Windows con TensorFlow versión 11 y una GPU de NVIDIA RTX 4080.

4.3. Aprendizaje por Ensamble en la clasificación de cáncer de mama

Aunque los resultados obtenidos en el **Experimento M-C** fueron excelentes, observe que este no detecta o clasifica cáncer, ya que solo distingue entre masas y calcificaciones, por lo que se planeó seguir experimentando con los cuatro experimentos restantes para tratar de mejorar la tasa de clasificación de cáncer, ya que estos sí clasifican las anormalidades en benignas o malignas, sin embargo, experimentar con los cuatro requiere demasiado tiempo, por lo que solo se eligió experimentar con dos de ellos: **M-B** y **Masas**. Para esto, se exploraron otros parámetros y técnicas vistas en la literatura principalmente relacionada con técnicas de ensamble para la clasificación de cáncer, ya que este es más comúnmente encontrado en textos relacionados con el aprendizaje máquina y no tanto en aprendizaje profundo. Dos de los textos seleccionados relacionados con la clasificación de tejido maligno que utilizan la técnica de Ensamble y la base de datos CBIS-DDSM son los siguientes.

Nemade et al. (2024) entrenaron las redes VGG16, VGG19 e InceptionV3 con función de activación Softmax para la clasificación binaria de tejido maligno y tejido benigno, después, utilizaron regresión logística y un perceptrón multicapa para la agregación (Stacking), además, comparte algunos parámetros con los cuales se experimentó, sin embargo, aunque sus resultados son muy sorprendentes, menciona que obtuvo 55,890 imágenes entre las bases de datos **CBIS-DDSM** y la base de datos original **DDSM**, lo cual dista increíblemente con el número de imágenes disponibles, por lo que se siguió utilizando el **IDG** con los parámetros propuesto para el aumento de datos.

Por su parte, Baccouche et al. (2022) utilizaron un ensamble de modelos **ResNetV2** con CLAHE y aumento de datos rotando los parches extraídos en 0°, 90°, 180° y 270°. Primero entrenaron los modelos **ResNet50V2, ResNet101V2** y **ResNet152V2**, después realizaron Stacking de estos modelo con un perceptrón multicapa, sin embargo, es más una combinación entre el enfoque de extracción de características y Stacking, ya que entrenaron el perceptrón multicapa con las características extraídas de las CNNs en todo el conjunto de entrenamiento, de hecho Nemade et al. (2024) lo definen como un *modelo*

híbrido, señalando que existen otros en la literatura. Los modelos ResNetsV2 no han sido utilizados en nuestros experimentos, por lo que se decidió trabajar con ellos y seguir los parámetros e ideas de ambos textos. Se menciona también que el número de imágenes dista también en este caso, ya que obtuvieron 8802 imágenes, pues las imágenes etiquetadas con **BWC** son consideradas benignas e imágenes con nivel de **BI-RADS** mayor a 3 son consideradas malignas y benignas en otro caso. Cabe mencionar que usaron el **IDG**, sin embargo, no comparten los parámetros utilizados, por lo cual se decidió utilizar los parámetros propuestos para realizar el aumento de datos.

ResNetV2. Es una versión mejorada de la familia ResNet, introducida en 2016. ResNetV2 se diseñó con ajustes específicos en los bloques residuales, mejorando el flujo de información y el rendimiento en redes profundas, especialmente en arquitecturas muy profundas como ResNet101V2 y ResNet152V2. En ResNetV2 el orden de las operaciones dentro de cada bloque residual cambia. A diferencia de ResNet original, que aplica primero convoluciones y luego normalización y activación, ResNetV2 invierte este orden. Los bloques residuales de ResNetV2 aplican primero una *batch normalization* (normalización por lotes) y una *activación ReLU* antes de la convolución (pre-activación). Esto ayuda a estabilizar el entrenamiento en redes profundas, ya que permite que el flujo de gradientes pase mejor por toda la red.

4.3.1. Parámetros

Descripción 1. Nemade et al. (2024) mencionan los siguientes parámetros utilizados en el aprendizaje por transferencia (recuerde que se quita el perceptrón de la red y se crea uno propio):

1. Capa FC de 1024 Neuronas con activación ReLU.
2. Capa de *drop-out*.
3. Última capa con función de *activación Softmax* para la clasificación binaria.

Además utilizaron la función de pérdida *categorical cross-entropy* y optimizador Adam en las redes VGG16, VGG19 e InceptionV3 para posteriormente realizar Stacking con regresión logística. Cabe señalar que los valores utilizados en cada una de estas funciones no fueron compartidos y además no se menciona el uso de ajuste fino.

Descripción 2. En el caso del uso de los modelos ResNetV2, se menciona y se ilustra el uso del ajuste

fino y del aprendizaje por transferencia en cada uno de los modelos ResNetV2, además comparten los parámetros utilizados por el perceptrón multicapa. A pesar de que Baccouche et al. (2022) experimentaron con varios parámetros, comparten los que proporcionaron su mejor resultado, los cuales se muestran en la Tabla 16.

Tabla 16. Valores de los mejores hiperparámetos

Hiperparámetro	Valor con mejor resultado	Descripción
Batch size	32	Tamaño del lote de entrenamiento
Épocas	30	Número de épocas de entrenamiento
Drop-out	30 %	% de neuronas "apagadas"
LR	10^{-2}	Tasa de aprendizaje
Smoothing	25 %	% de suavizado de las etiquetas

Cabe señalar que el código de programación es compartido, pero solo la parte del Ensamble del perceptrón multicapa con los modelos ResNets, sin embargo, no está claro si se utilizó el **IDG** para el aumento de datos. En general, este modelo parece ser más un modelo híbrido, ya que entrena el perceptron con las características extraídas de las capas de *drop-out* de los modelos ResNetV2 y no con las predicciones probabilísticas, esto es como realizar **FC Extraction** de las capas *drop-out* de los modelos ResNetV2 para después concatenarlas y pasarlas al perceptrón.

A continuación se presentan los resultados obtenidos divididos en dos partes, en la sección 4.4 se presentan los resultados obtenidos por nueve CNNs en el **Experimento Masas** y en el **Experimento M-B**, cabe señalar que el CLAHE no fue aplicado en ninguno de los experimentos. Seis CNNs fueron las que ya se han utilizado y se entrenaron con los parámetros de la **Descripción 1** y las tres restantes (modelos ResNetV2) con los parámetros de la **Descripción 2** (no hubo extracción de características). En la sección 4.5 se presentan resultados enfocados en el **Experimento Masas** con los parámetros y técnicas mencionados por la **Descripción 2** (extracción de características). En ambos experimentos se utilizó un conjunto de validación estratificado del 20 %.

4.4. Experimentación (Segunda Parte)

En esta parte la experimentación se realizó de la siguiente manera. Primero se entrenaron las nueve CNNs, después se realizó votaciones de tosas las combinaciones de CNNs (502 combinaciones) y por

último se cambió la manera de hacer esta agregación al realizar Stacking con el conjunto de validación y con el uso de distintos meta-clasificadores. El diagrama del trabajo realizado en esta sección se puede ver en la Figura 73.

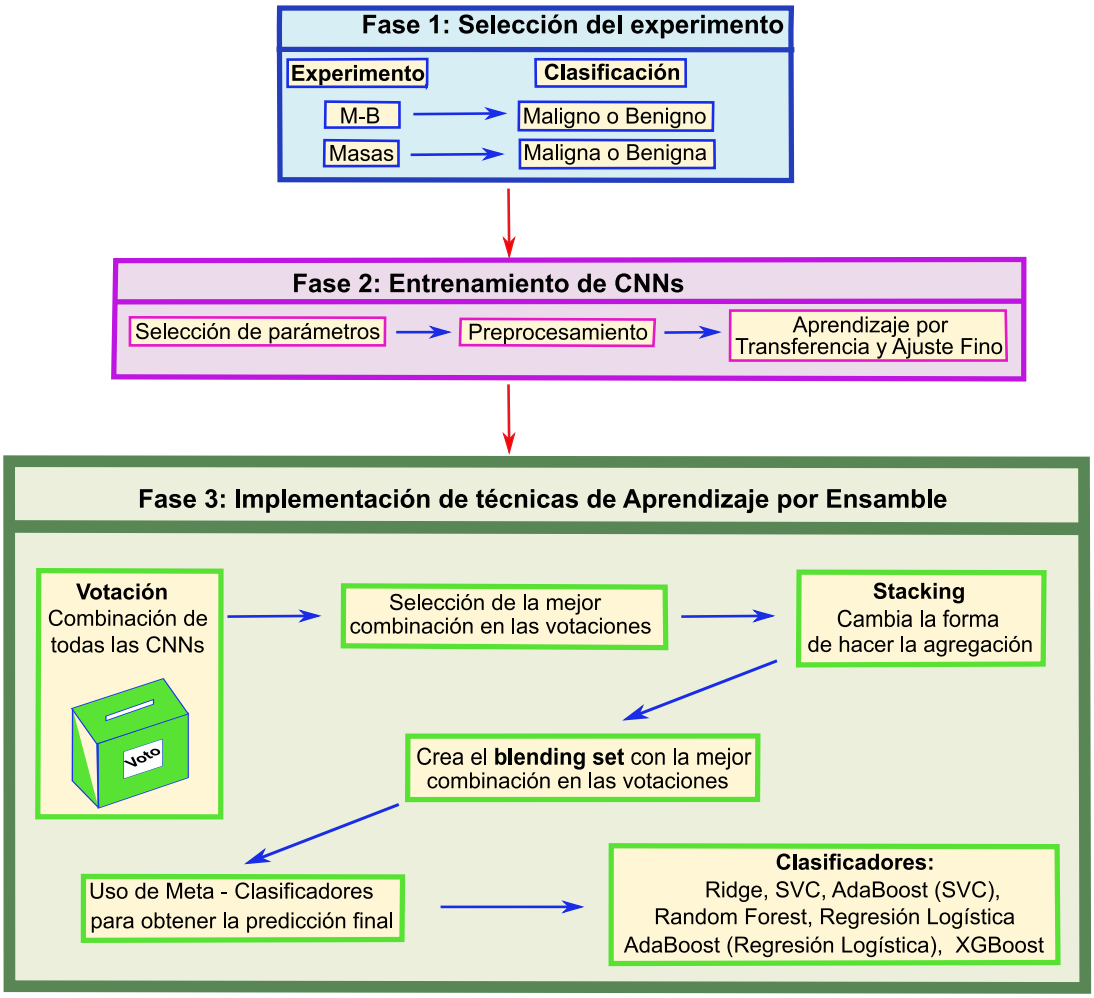


Figura 73. Diagrama que resume el trabajo realizado en esta segunda parte.

4.4.1. Resultados Experimento M-B

Los resultados de este experimento se muestra en la Figura 74 y las métricas asociadas en la Tabla 17. Además, una observación que no se hizo anteriormente en este experimento es que, aunque el *accuracy* es bajo, no se sabe realmente cual de las dos clases le cuesta más clasificar a cada una de las CNN, es decir, las métricas relacionadas con el *Test* de la Tabla 17, mezclan las dos clases, por lo que al evaluar cada una de las CNNs en cada clase (masas y calcificaciones) se tendrá un mejor panorama de lo que

está ocurriendo, ya que la razón de realizar este experimento es ver si al juntar estas clases se puede elevar la tasa de clasificación del cáncer.

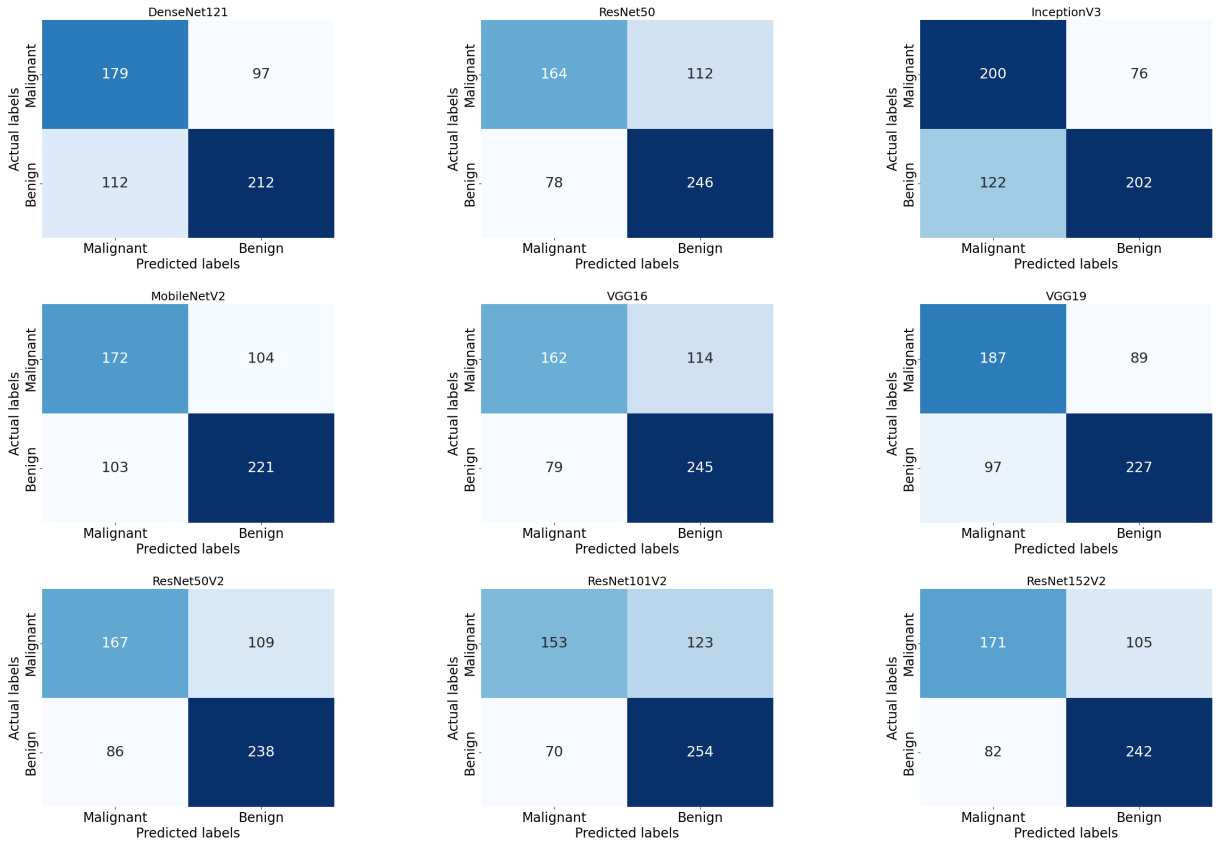


Figura 74. Matrices de confusión de los resultados individuales obtenidos por cada una de las CNNs en el *Test* (600 imágenes) con el **Experimento M-B**.

Tabla 17. Métricas obtenidas en base a la Figura 74.

Modelo	Accuracy	f1	MCC	Precisión	Recall	Imágenes mal clasificadas
DenseNet121	0.6516	0.6698	0.302	0.686	0.6543	209
ResNet50	0.6833	0.7214	0.3591	0.6871	0.7592	190
InceptionV3	0.67	0.671	0.3479	0.7266	0.6234	198
MobileNetV2	0.655	0.681	0.3053	0.68	0.682	207
VGG16	0.6783	0.7174	0.3488	0.6824	0.7561	193
VGG19	0.69	0.7093	0.3774	0.7183	0.7006	186
ResNet50V2	0.675	0.7093	0.3427	0.6858	0.7345	195
ResNet101V2	0.6783	0.7246	0.3489	0.6737	0.7839	193
ResNet152V2	0.6883	0.7213	0.3698	0.6974	0.7469	187

Los resultados al evaluar individualmente cada clase se muestran en las Tablas 18 y 19 mientras que sus matrices de confusión se muestran en las Figuras 75 y 76 respectivamente.

Tabla 18. Métricas obtenidas en base a la Figura 75.

Modelo	Accuracy	f1	MCC	Precisión	Recall	Imágenes mal clasificadas
DenseNet121	0.637	0.6466	0.2742	0.6323	0.6615	94
ResNet50	0.6447	0.6891	0.3006	0.6144	0.7846	92
InceptionV3	0.6139	0.5726	0.2331	0.6442	0.5153	100
MobileNetV2	0.6293	0.6521	0.2603	0.6164	0.6923	96
VGG16	0.6795	0.7108	0.3666	0.6496	0.7846	83
VGG19	0.6795	0.6937	0.3601	0.6666	0.723	83
ResNet50V2	0.6254	0.6643	0.2568	0.6037	0.7384	97
ResNet101V2	0.61	0.6688	0.2336	0.5828	0.7846	101
ResNet152V2	0.6525	0.6959	0.3167	0.6204	0.7923	90

De la Tabla 18 se puede ver que la red VGG19 se mantiene como la que mejor clasifica calcificaciones, al igual que al red VGG16, mientras que el *accuracy* de los modelos ResNet50 y ResNet152V2 cae.

Tabla 19. Métricas obtenidas en base a la Figura 76.

Modelo	Accuracy	f1	MCC	Precisión	Recall	Imágenes mal clasificadas
DenseNet121	0.6627	0.6866	0.3266	0.7283	0.6494	115
ResNet50	0.7126	0.7461	0.415	0.75	0.7422	98
InceptionV3	0.7126	0.7336	0.4265	0.7758	0.6958	98
MobileNetV2	0.6744	0.7024	0.3458	0.7318	0.6752	111
VGG16	0.6774	0.7222	0.3383	0.7079	0.7371	110
VGG19	0.6979	0.7208	0.3961	0.76	0.6855	103
ResNet50V2	0.7126	0.7434	0.4172	0.7553	0.7319	98
ResNet101V2	0.7302	0.7676	0.4468	0.7524	0.7835	92
ResNet152V2	0.7155	0.7413	0.4274	0.7679	0.7164	97

De la Tabla 19 se puede ver que se supera el 70 % de *accuracy*, siendo la red ResNet101V2 la mas alta con un 73 % de *accuracy*, pero por debajo del 74.48 % logrado por los modelos InceptionV3 y VGG19 en la primera parte de la experimentación (Tabla 6) al clasificar mal 87 imágenes.

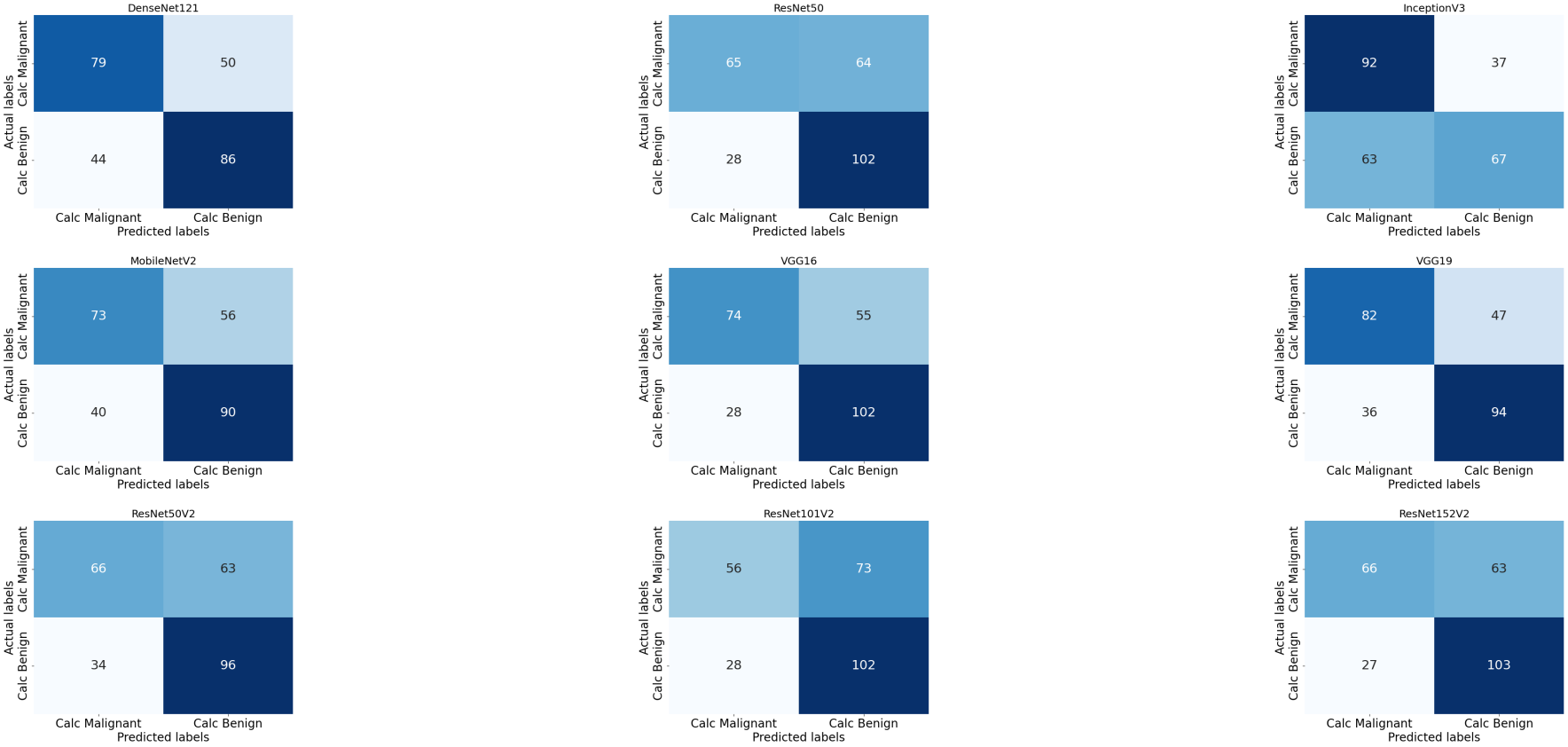


Figura 75. Matrices de confusión de los resultados individuales obtenidos por cada una de las CNNs en el *Test* (259 imágenes de calcificaciones) con el **Experimento M-B**.

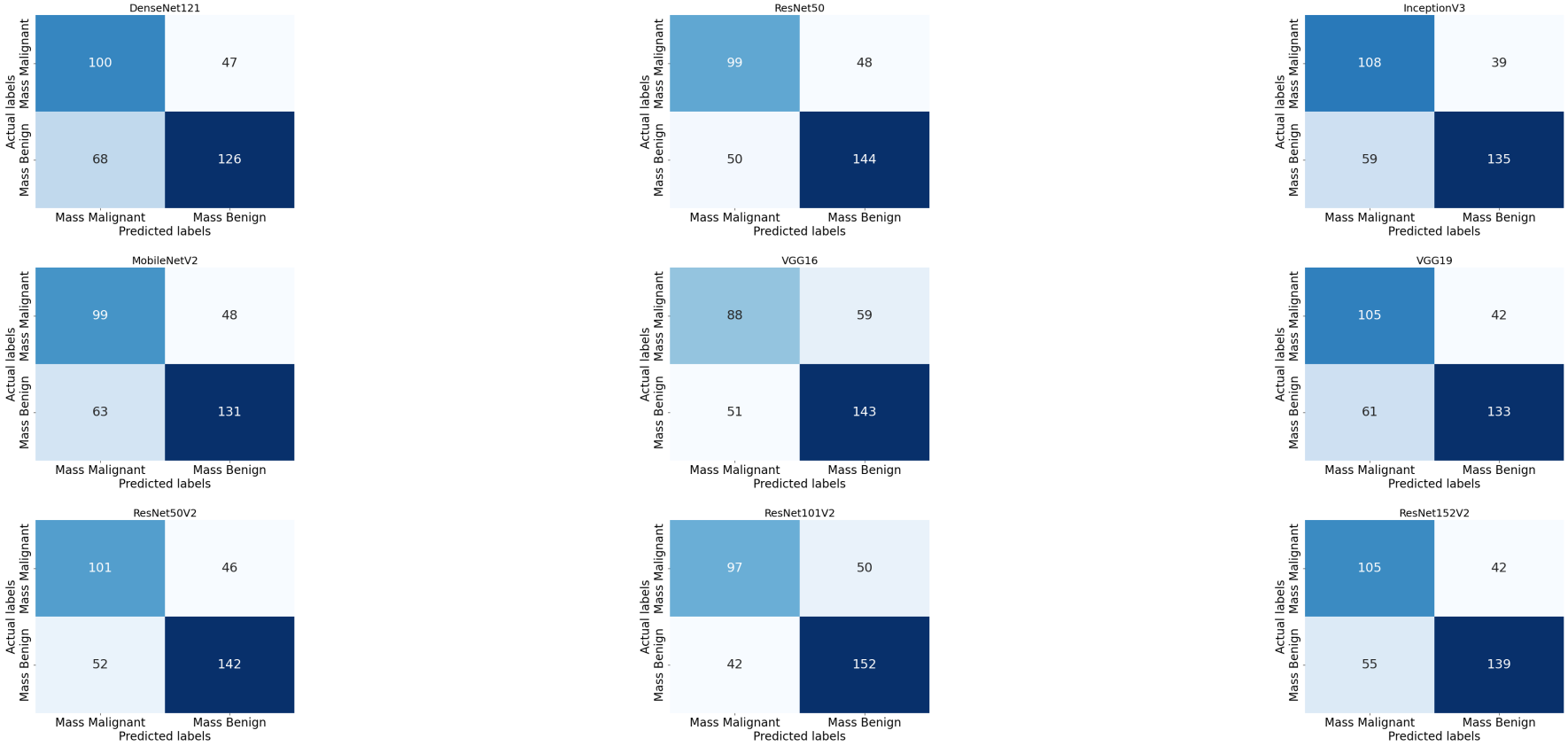


Figura 76. Matrices de confusión de los resultados individuales obtenidos por cada una de las CNNs en el *Test* (341 imágenes de masas) con el **Experimento M-B**.

Podemos observar que el uso de estos parámetros no ayudan a superar los resultados obtenidos en la primera parte de la experimentación, ya que los resultados son similares.

Votaciones. Como ahora se tienen entrenados 9 clasificadores, el número de combinaciones distintas de votaciones que se pueden realizar es de $2^9 - (9 + 1) = 502$. Cabe señalar que este número de combinaciones se hizo 3 veces, una de manera general, es decir, que toma en cuenta ambas clases (votaciones de CNNs de la Tabla 17), otra para calcificaciones (votaciones de CNNs de la Tabla 18) y otra para masas (votaciones de CNNs de la Tabla 19), por lo que los resultados más altos se muestran en la Tabla 20.

Tabla 20. Resultado de las votaciones de cada una de las CNNs por clase: ambas, masas y calcificaciones

Votaciones de:	Accuracy	f1	MCC	Precisión	Recall	Imágenes mal clasificadas
VGG16-InceptionV3-ResNet152V2 (Ambas)	0.7166	0.7439	0.4278	0.7264	0.7623	170 de 600
VGG16-InceptionV3-ResNet152V2 (Masas)	0.7448	0.768	0.4868	0.7955	0.7422	87 de 341
VGG16-InceptionV3 (Calcificaciones)	0.6988	0.7153	0.3997	0.6805	0.7538	78 de 259

De la Tabla 20 se puede observar que hubo una ligera mejora en todos las clasificaciones, además de que la combinación de VGG16-InceptionV3-ResNet152 aparece dos veces. Cabe señalar que los datos en rojo son los más altos obtenidos en la clasificación de cancer a partir de calcificaciones.

Stacking. Después de realizar las votaciones se buscó cambiar la forma de hacer la agregación, por lo que se utilizaron las CNNs VGG16, InceptionV3 y ResNet152V2 para formar un *blending set* sobre el conjunto de validación para realizar Stacking, ya que esta apareció dos veces en la tablas de votaciones 20. Cabe señalar que el Stacking se hizo sobre el conjunto de validación conformados por ambas clases, ya que se requiere observar el comportamiento de la tasa de clasificación al utilizar ambas clases. Se usaron distintos meta-clasificadores, siendo evaluados en el *Test set* correspondiente de manera general (las 600 imágenes Tabla 21), en calcificaciones (259 imágenes Tabla 22) y en masas (341 imágenes Tabla 23), sin embargo, las tablas muestran que cambiar la forma de agregación al realizar Stacking no

mejora los resultados obtenidos por las votaciones.

Tabla 21. Resultados de Stacking en el *Test* al clasificar todas las 600 imágenes.

Meta-Learner	Accuracy	f1	MCC	Precisión	Recall
XGBoost	0.645	0.6613	0.2896	0.6819	0.6419
Regresion Logística	0.695	0.7188	0.3855	0.7155	0.7222
Ridge	0.6933	0.7195	0.3815	0.7108	0.7283
Random Forest	0.625	0.6317	0.2545	0.6724	0.5956
SVC	0.6883	0.7162	0.371	0.7044	0.7283
Ada(SVC)	0.685	0.7191	0.3629	0.6934	0.7469
Ada(Regresion Logística)	0.6933	0.716	0.3827	0.716	0.716

Tabla 22. Resultados de Stacking en el *Test* al clasificar las 259 imágenes de calcificaciones.

Meta-Learner	Accuracy	f1	MCC	Precisión	Recall
XGBoost	0.6033	0.6222	0.2057	0.6405	0.6049
Regresion Logística	0.6716	0.7001	0.3376	0.6906	0.7098
Ridge	0.6783	0.7097	0.3501	0.692	0.7283
Random Forest	0.5766	0.6043	0.1492	0.61	0.5987
SVC	0.6816	0.729	0.356	0.6745	0.7932
Ada(SVC)	0.6583	0.674	0.3163	0.695	0.6543
Ada(Regresion Logística)	0.6633	0.6833	0.3243	0.6942	0.6728

Tabla 23. Resultados de Stacking en el *Test* al clasificar las 341 imágenes de masas.

Meta-Learner	Accuracy	f1	MCC	Precisión	Recall
XGBoost	0.645	0.6613	0.2896	0.6819	0.6419
Regresion Logística	0.695	0.7188	0.3855	0.7155	0.7222
Ridge	0.6933	0.7195	0.3815	0.7108	0.7283
Random Forest	0.625	0.6317	0.2545	0.6724	0.5956
SVC	0.6883	0.7162	0.371	0.7044	0.7283
Ada(SVC)	0.685	0.7191	0.3629	0.6934	0.7469
Ada(Regresion Logística)	0.6933	0.716	0.3827	0.716	0.716

4.4.2. Resultados Experimento Masas

La forma de experimentar fue la misma que la anteriormente descrita, es decir, se experimentó con las nueve CNNs, después se realizaron votaciones y posteriormente Stacking.

Resultados individuales. Los resultados individuales se muestran en la Figura 77 mientras que las métricas asociadas en la Tabla 24.

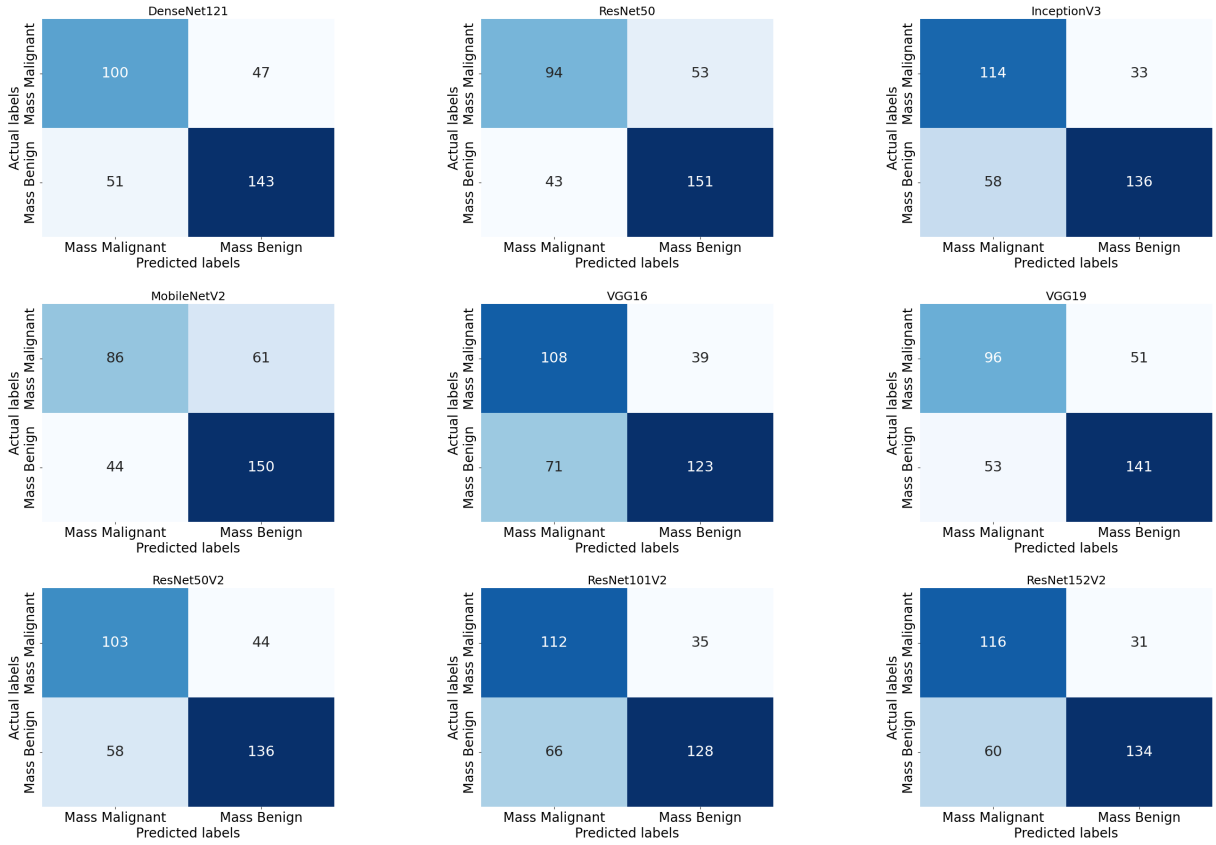


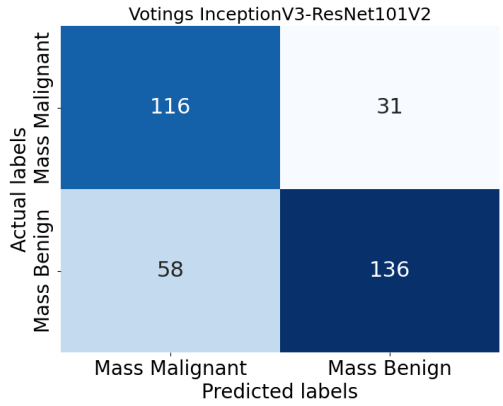
Figura 77. Matrices de confusión de los resultados individuales obtenidos por cada una de las CNNs en el *Test* (341 imágenes de masas) con en el **Experimento Masas**.

Como se puede observar de la Tabla 24, estos resultados no parecen superar los obtenidos en la primera parte de la experimentación, sin embargo, se puede observar que el modelo InceptionV3 aparece otra vez como uno de los mejores modelos al realizar esta tarea de clasificación.

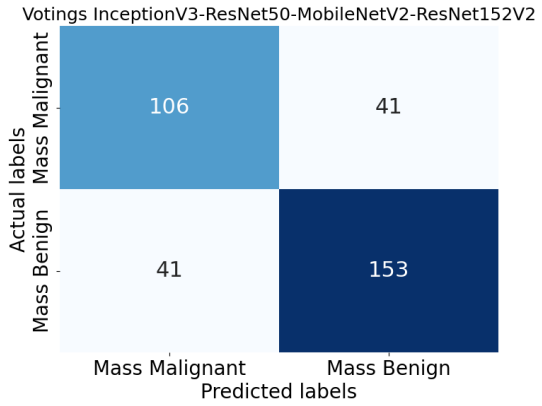
Tabla 24. Métricas obtenidas en base a la Figura 77.

Modelo	Accuracy	f1	MCC	Precisión	Recall	Imágenes mal clasificadas
DenseNet121	0.7126	0.7447	0.4161	0.7526	0.7371	98
ResNet50	0.7184	0.7587	0.422	0.7401	0.7783	96
InceptionV3	0.7331	0.7493	0.472	0.8047	0.701	91
MobileNetV2	0.692	0.7407	0.3652	0.7109	0.7731	105
VGG16	0.6774	0.691	0.3656	0.7592	0.634	110
VGG19	0.695	0.7305	0.3792	0.7343	0.7268	104
ResNet50V2	0.7008	0.7272	0.3984	0.7555	0.7010	102
ResNet101V2	0.7038	0.717	0.418	0.7852	0.6597	101
ResNet152V2	0.7331	0.7465	0.4755	0.8121	0.6907	91

Votaciones. Los resultados en las votaciones mejoraron ligeramente e incluso superaron el *accuracy* obtenido en las votaciones para clasificar masas en el **Experimento M-B**, además, aparecieron varias combinaciones con el mismo *accuracy* de 75.36 %, teniendo como base los modelos ResNet50 y ResNet152V2, sin embargo, el modelo con mayor *accuracy* alcanzó un 75.95 % , es decir, de no pasar el 73.5 % de *accuracy*, se logró llegar a un casi 76 %. Cabe señalar que este modelo no ganó en todas las métricas, ya que un segundo modelo lo superó en precisión al llegar a un 81.2 %, pero obtuvo un *accuracy* del 73.3 %. Ambos modelos se muestran en la Figura 78 y sus métricas en la Tabla 25.



Modelo con 81.2 % de precisión y 73.3 % de *accuracy*.



Modelo con 75.95 % de *accuracy* y 78.86 % de precisión.

Figura 78. Matrices de confusión de los resultados obtenidos por votaciones en el *Test* de 341 imágenes de masas.

Como era de esperarse, los resultados obtenidos en las votaciones mejoran la tasa de clasificación, no de una manera drástica, pero ayudan a identificar cuál es la mejor combinación de modelos para tenerlos en cuenta en futuros experimentos.

Tabla 25. Métricas obtenidas en base a la Figura 78.

Votaciones de:	Accuracy	f1	MCC	Precisión	Recall	Imágenes mal clasificadas
InceptionV3-ResNet50-MobileNetV2-ResNet152V2	0.7595	0.7886	0.5097	0.7886	0.7886	82 de 341
InceptionV3-ResNet101V2	0.739	0.7534	0.4855	0.8143	0.701	89 de 341

Stacking. La combinación con que se obtuvo el mayor *accuracy* comprende los modelos InceptionV3, ResNet50, MobileNetV2 y ResNet152V2, por lo que se creó un *blending set* con las predicciones de estos modelos en el conjunto de validación para realizar Stacking (cambiar la manera de hacer la agregación). Los resultados se muestran en la Tabla 26 y la Figura 79 muestra las matrices de confusión del modelo en rojo y azul.

Tabla 26. Resultados de Stacking.

Meta-Learner	Accuracy	f1	MCC	Precisión	Recall
XGBoost	0.6803	0.7046	0.3606	0.7428	0.6701
Regresion Logística	0.7507	0.7683	0.5043	0.8150	0.7268
Ridge	0.7448	0.7603	0.4956	0.8165	0.7113
Random Forest	0.7302	0.7486	0.4636	0.7965	0.7061
SVC	0.739	0.7507	0.4892	0.822	0.6907
Ada(SVC)	0.736	0.75	0.4805	0.8132	0.6958
Ada(Regresion Logística)	0.7536	0.7704	0.5110	0.8197	0.7268

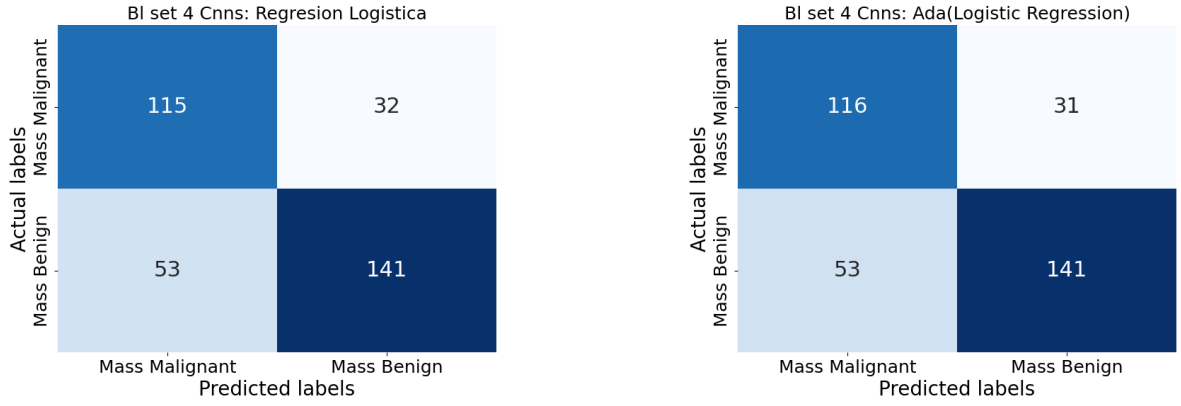


Figura 79. Se muestran las matrices de confusión obtenidas de los dos mejores clasificadores de la Tabla 26.

Los resultados obtenidos no alcanzan a superar los obtenidos en las votaciones, ya que el mejor modelo (ver color rojo y azul de la Tabla 26) alcanza un 75.36 % de *accuracy*, pero cabe señalar que hasta el momento solo se ha tomado en cuenta el*accuracy* para seleccionar el mejor modelo, sin embargo, conviene a veces sacrificar una métrica por otra, ya que un análisis más profundo se puede obtener de la Tabla 27 y la Figura 80. Observe que el mejor modelo con un 75.95 % de *accuracy* clasifica mal 82 de 341 imágenes, sin embargo, el modelo obtenido por Stacking, usando como meta-clasificador AdaBoost con estimador base una Regresión Logística, a pesar de estar dos imágenes abajo, se sitúa 10 imágenes más arriba en cuestión de falsos negativos (**en términos clínicos**), por lo tanto, este modelo con deja pasar menos este tipo de datos, por tanto, podemos concluir que conviene más seleccionar este modelo obtenido por Stacking con un 81.97 % de precisión (**en términos de la matriz de confusión**).

Tabla 27. Resumen experimental de los mejores modelos.

Ensamble:	**Accuracy**	**f1**	**MCC**	**Precisión**	**Recall**	**Imágenes mal clasificadas**
Votaciones InceptionV3-ResNet50-MobileNetV2-ResNet152V2	0.7595	0.7886	0.5097	0.7886	0.7886	82 de 341
Votaciones InceptionV3 ResNet101V2	0.739	0.7534	0.4855	0.8143	0.701	89 de 341
Stacking Ada(Reg. Log.)	0.7536	0.7704	0.511	0.8197	0.7268	84 de 341

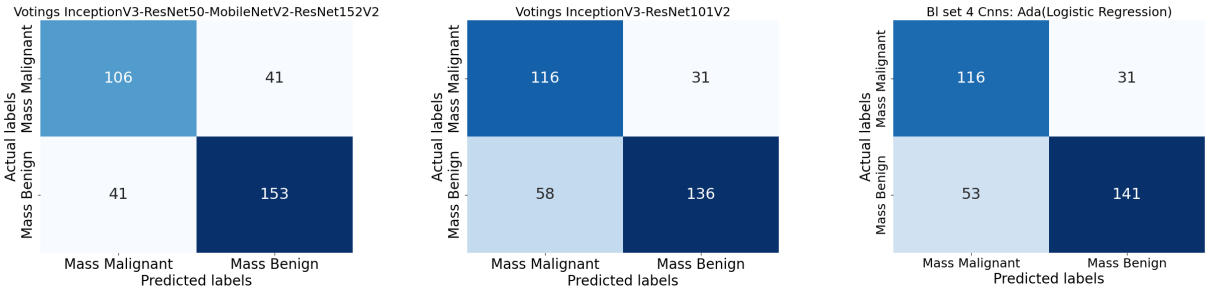


Figura 80. Matrices de confusión pertenecientes a los mejores modelos, relacionados con la **Tabla 27**.

Nota: Estos experimento fueron realizados en un entorno virtual de Anaconda Linux con TensorFlow versión 17 y una GPU de NVIDIA RTX 4080.

4.5. Experimentación (Tercera Parte)

Esta última parte se centra únicamente en el **Experimento Masas** y se trabaja con la extracción de características de diferentes maneras, principalmente siguiendo los métodos y parámetros de la **Descripción 2** (Baccouche et al., 2022). El diagrama que resume el trabajo realizado ene sta sección se puede ver en la Figura 81.

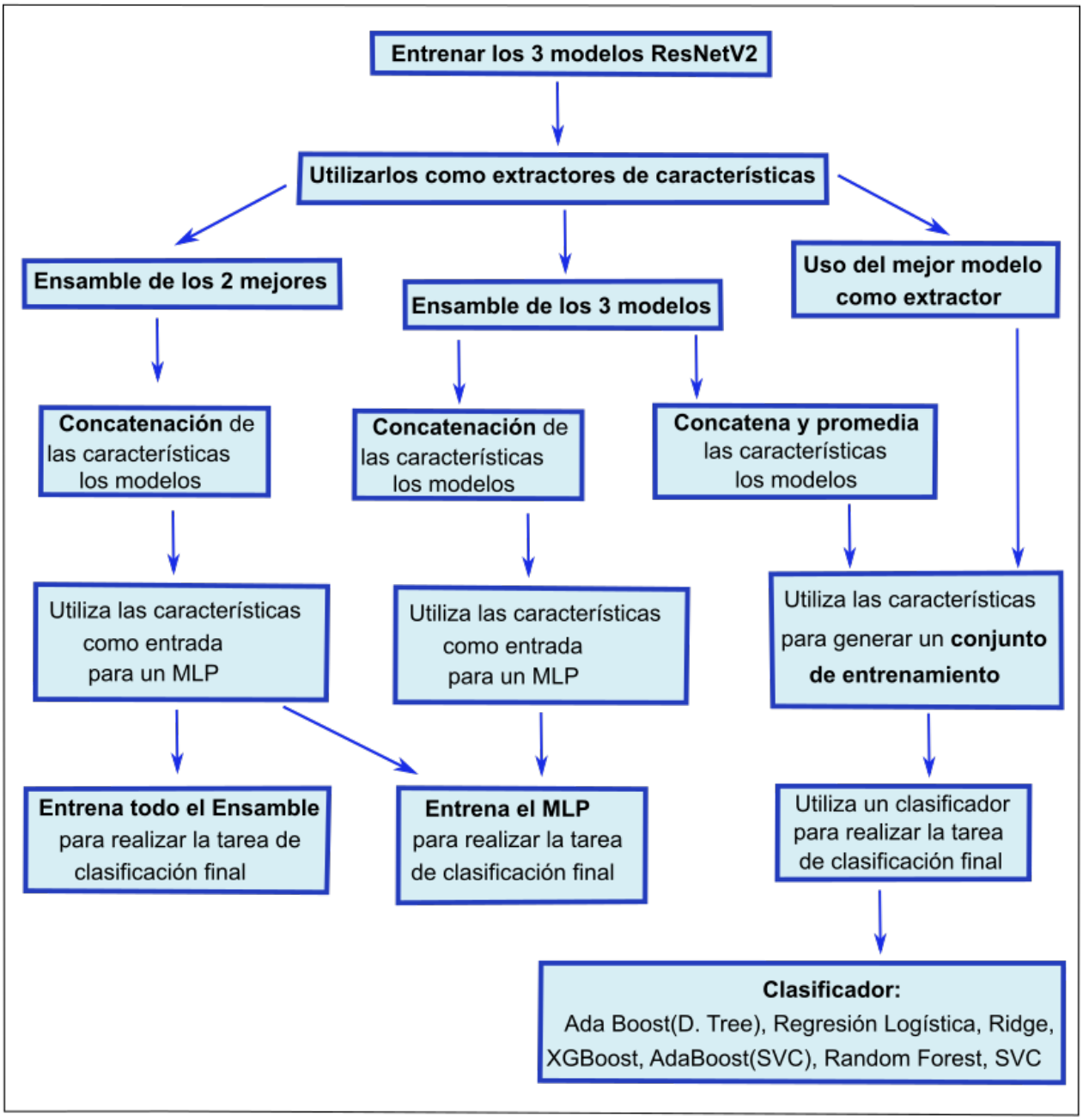


Figura 81. Diagrama que resume el trabajo realizado en esta tercera parte. **Nota: MLP**, del inglés *multi layer perceptron*, significa perceptrón multicapa

4.5.1. Resultados Individuales.

Como se menciona en la **Descripción 2**, primero se entrenaron los modelos ResNetV2 con los parámetros proporcionados, cabe señalar que aunque en el experimento anterior fueron entrenadas estas redes, la versión de **TensorFlow** que se utilizó fue la 17, la última hasta la fecha en la que se entrega este documento, por lo que existe un problema para obtener las características de estos modelos desde cualquiera de sus capas previas a la que realiza la clasificación (la última con *activación Softmax*), por lo que se tuvo que reentrenar estos modelos con parámetros similares en **TensorFlow** 15. A continuación se muestran los resultados obtenidos (Tabla 28 y Figura 82), así como sus gráficas de entrenamiento (Figura 83) y la estructura de cada modelo utilizado (Figura 84), lo cual es importante para explicar todo lo que sigue.

Tabla 28. Resultados individuales de los modelos ResNetV2.

Modelo	Accuracy	f1	MCC	Precisión	Recall	Imágenes mal clasificadas
ResNet50V2	0.7008	0.7371	0.3901	0.7371	0.7371	102
ResNet101V2	0.7126	0.7379	0.4222	0.7666	0.7113	98
ResNet152V2	0.7419	0.7634	0.4828	0.7977	0.7319	88

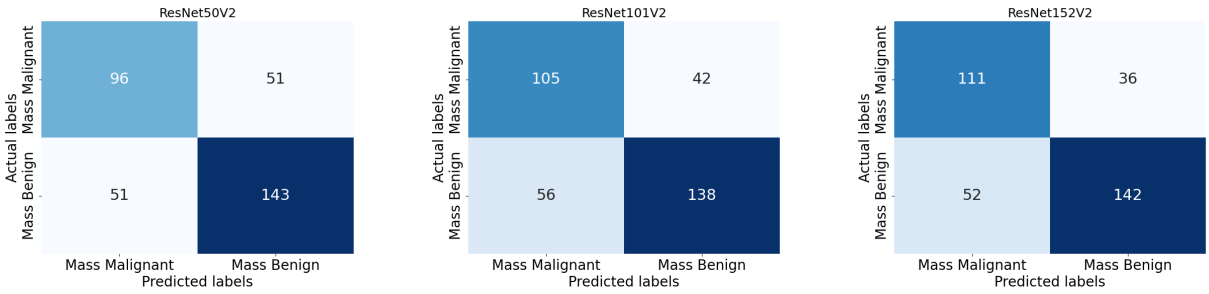


Figura 82. Matrices de confusión de los modelos ResNetV2 obtenidos en el *Test* (341 imágenes de masas).

De estos primeros resultados al utilizar los modelos ResNetV2 podemos observar que no superan mucho las tasas de clasificaciones previas, sin embargo, el modelo ResNet152V2, al igual que el modelo InceptionV3, sigue apareciendo con uno de los más altos resultados.

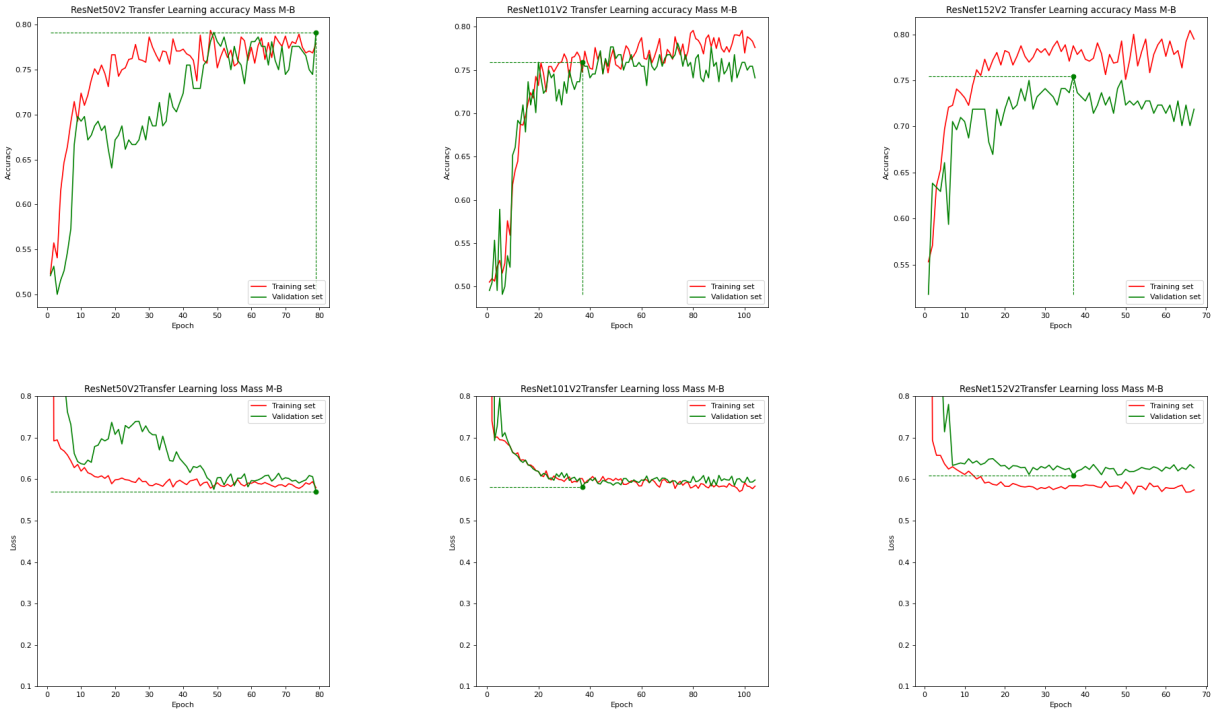


Figura 83. Gráficas de *accuraccy* y pérdidas de los modelos ResNetV2.

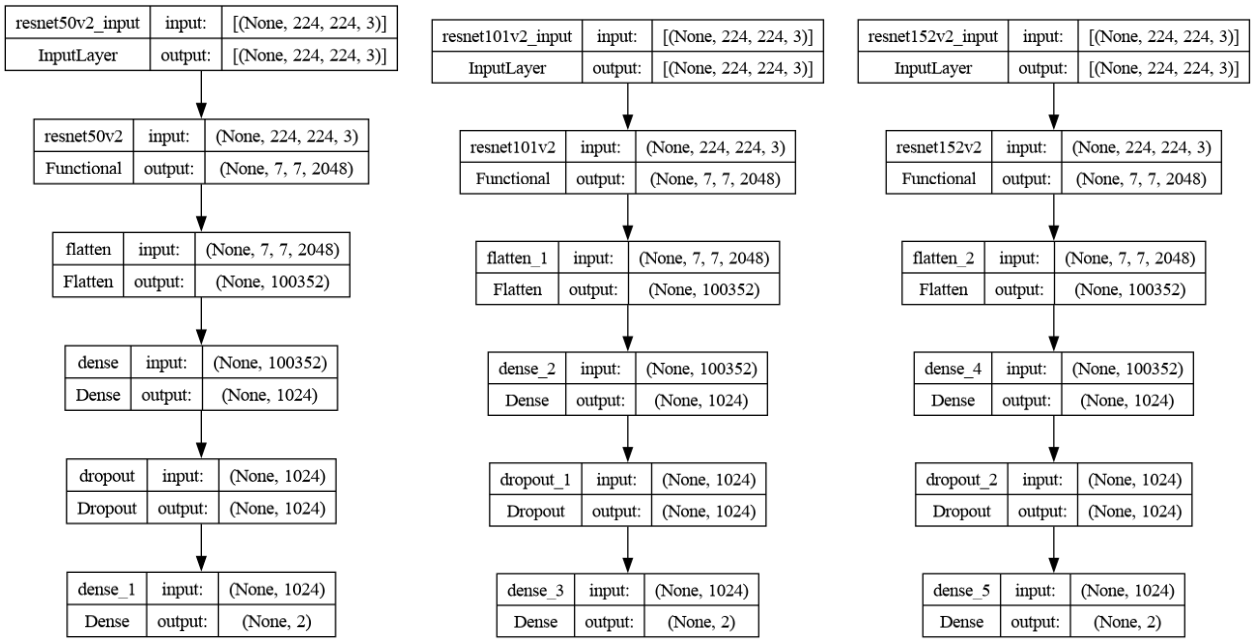


Figura 84. Estructuras resultantes de los modelos ResNetV2 utilizados en aprendizaje por transferencia y ajuste fino.

4.5.2. Aprendizaje por Ensamble de los modelos ResNetV2

Después de entrenar los modelos por separado lo que siguió es construir un perceptrón multicapa para realizar Stacking, sin embargo, la metodología no fue la que se describió en la sección 2.4.4, es decir, en lugar de tomar las predicciones probabilísticas provenientes de la útima capa de clasificación binaria con *activación Softmax*, lo que Baccouche et al. (2022) hicieron fue quitarla en todos los modelos ResNetsV2, y a partir de esta, ensamblar un perceptron multicapa para que este diera la clasificación final. La Figura 85 muestra los modelos ResNetV2 (extractores) pero sin la última capa de clasificación con *activación Softmax* y la Tabla 29 muestra las capas del perceptron multicapa ensamblado, por lo que la representación final utilizada se puede observar en la Figura 86.

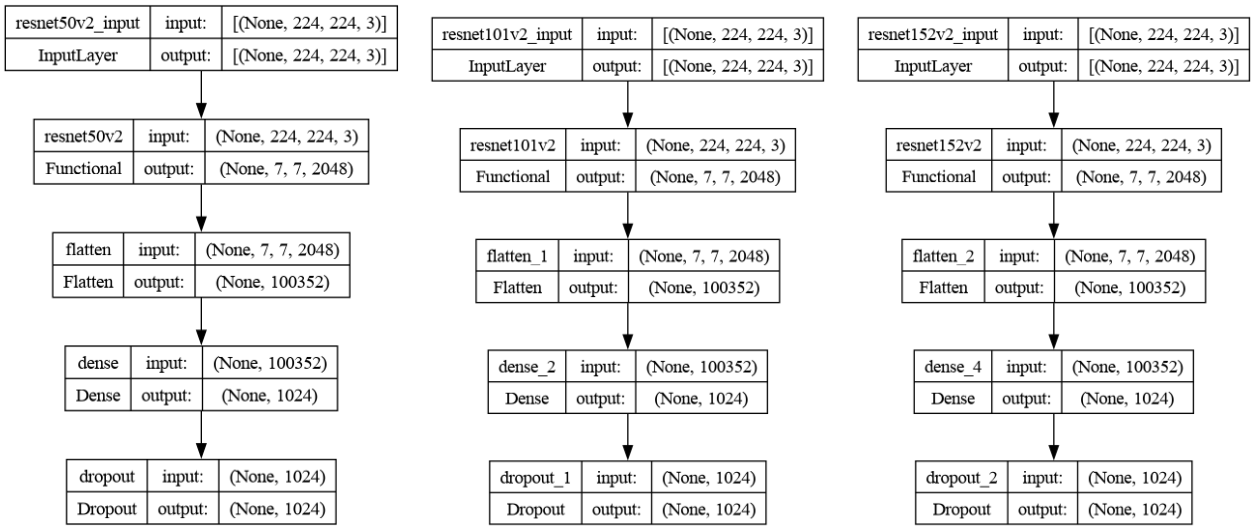


Figura 85. Modelos ResNetV2 utilizados como extractores de características hasta la capa de *drop-out* de cada uno, observe que son los mismos modelos de la Figura 84, pero sin la última capa de clasificación de dos neuronas.

Tabla 29. Capas empleadas en el perceptrón multicapa.

Flatten	Concatenación de salidas *drop-out* de los modelos ResNetV2
FC con 1000 Neuronas	*activación Sigmoide*
FC con 100 Neuronas	*activación ReLU*
FC con 10 Neuronas	*activación Sigmoide*
FC con 2 Neuronas	*activación Softmax*

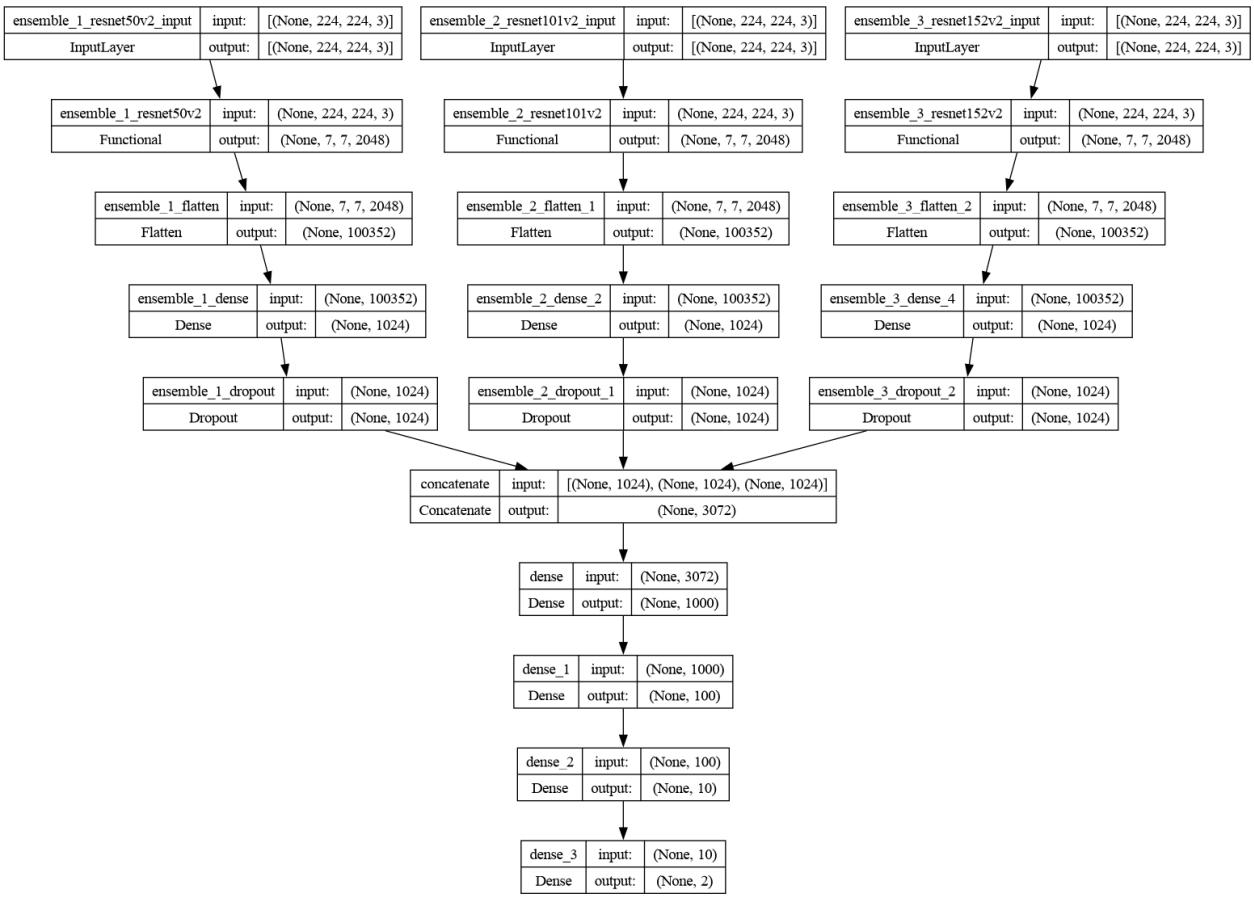


Figura 86. Modelo resultantes del ensamble de los extractores (modelos ResNetV2) con el perceptrón multicapa propuesto.

Es importante aclarar que hay dos formas de proceder: entrenar solamente el perceptrón multicapa o entrenar el todo el modelo desde ciertas capas (o todo el modelo completo). Observe que cualquier forma de continuar no es tal cuál el uso de Stacking y va más allá de simplemente extraer características, por lo tanto, se suele llamr a este tipo de modelos como *modelos híbridos*.

Cabe señalar que se hizo lo siguiente. Se entrenó el perceptrón multicapa con el ensamble de los tres modelos ResNetV2. Se intentó reentrenar todo el modelo en conjunto, es decir, se descongelaron los pesos de los tres modelos de ResNetV2 desde la capa donde inicialmente se entrenaron de manera individual, pero rápidamente ocurrió una falta de memoria `OOM`, por lo que se ensamblaron los dos mejores modelos ResNetV2 para poder realizar dicho entrenamiento: ResNet101V2 y ResNet152V2 (Figura 87). Al ensamblar estos dos modelos se realizaron dos entrenamientos: el entrenamiento individual del perceptrón multicapa y en conjunto, es decir, iniciar el entrenamiento desde las capas donde inicialmente se entrenaron de manera individual estos dos modelos. Es importante mencionar que el entrenar todo el modelo en conjunto resultó ser muy pesado computacionalmente hablando, ya que tardaba en carga y

evaluar los modelos, no tanto en entrenar debido a los parámetros y configuraciones utilizados. A continuación se presentan las matrices de confusión (Figura 88) de todos los resultados obtenidos (Tabla 30) junto con las gráficas de entrenamiento (Figura 89) del perceptrón multicapa, ya que el entrenamiento completo de los dos modelos ensamblados junto con el perceptrón generó un sobreajuste demasiado notable (Figura 90), por lo que su tasa de clasificación bajó bastante.

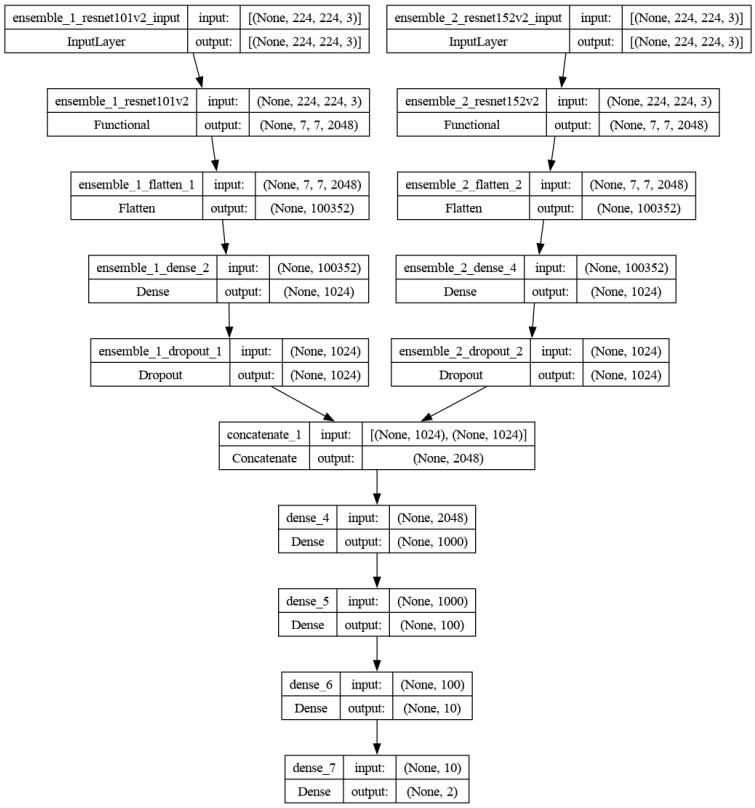


Figura 87. Ensamble de los modelos ResNet101V2 y ResNet152V2.

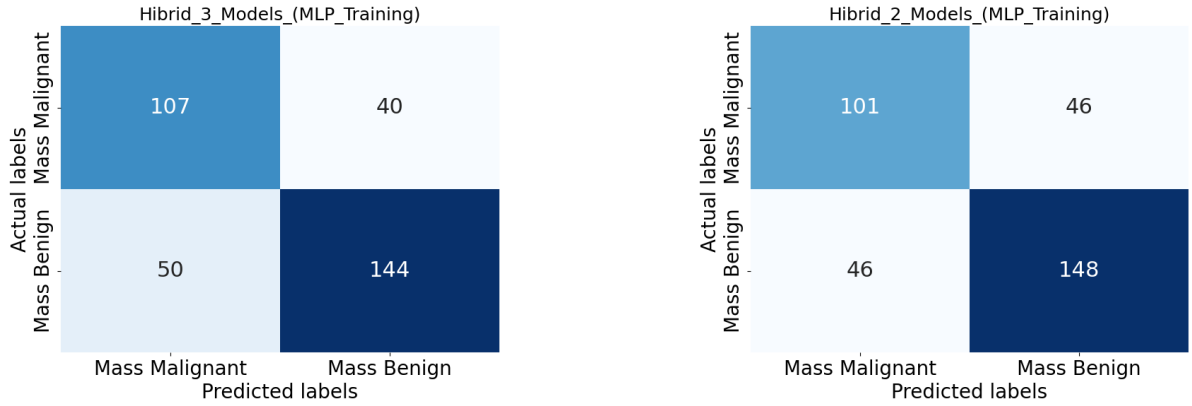
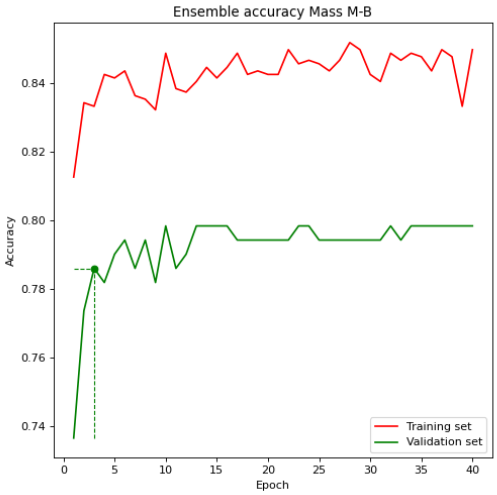


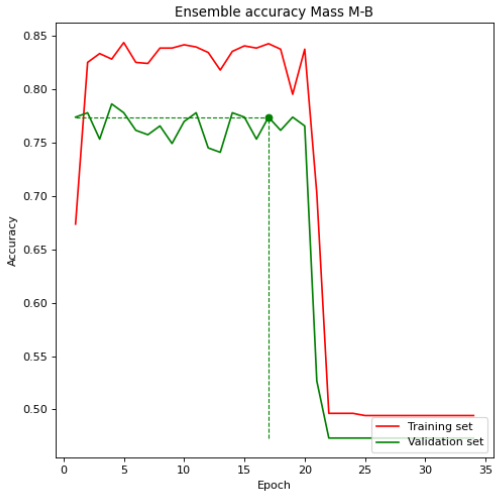
Figura 88. Matrices de confusión de los Modelos por Ensamble. En ambos casos el entrenamiento fue realizado solamente en el perceptrón multicapa.

Tabla 30. Métricas obtenidas en base a la Figura 88.

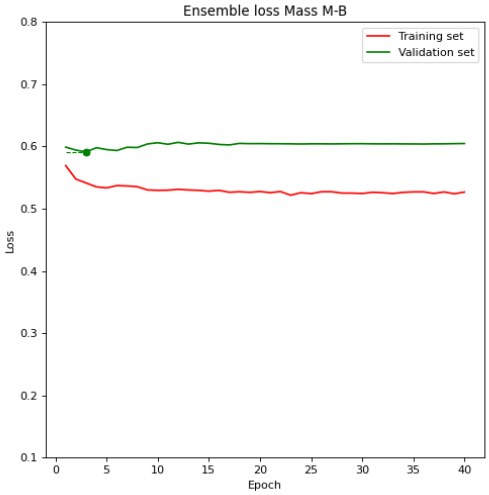
Modelo	Accuracy	f1	MCC	Precisión	Recall	Imágenes mal clasificadas
Ensamble 3 CNNs (entrenamiento del perceptrón multicapa)	0.736	0.7619	0.4671	0.7826	0.7422	90
Ensamble 2 CNNs (entrenamiento del perceptrón multicapa)	0.7302	0.7628	0.4499	0.7628	0.7628	92



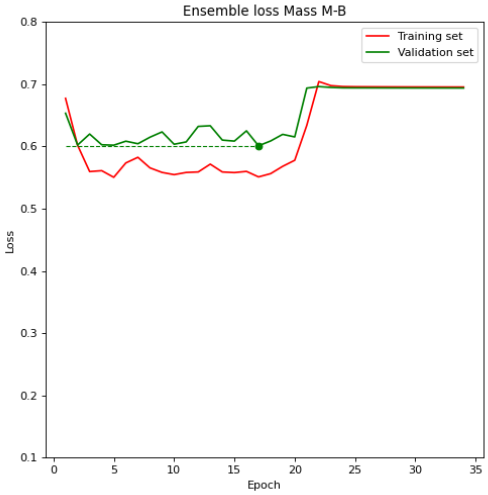
Gráfica de *accuracy* del preceptron multicapa de la Figura 86.



Gráfica de *accuracy* del preceptron multicapa de la Figura 87.



Gráfica de pérdida del perceptrón multicapa de la Figura 86.



Gráfica de pérdida del perceptrón multicapa de la Figura 87.

Figura 89. Gráficas de *accuraccy* y pérdidas del perceptrón multicapa de los dos modelos ensamblados (Figuras 86 y 87).

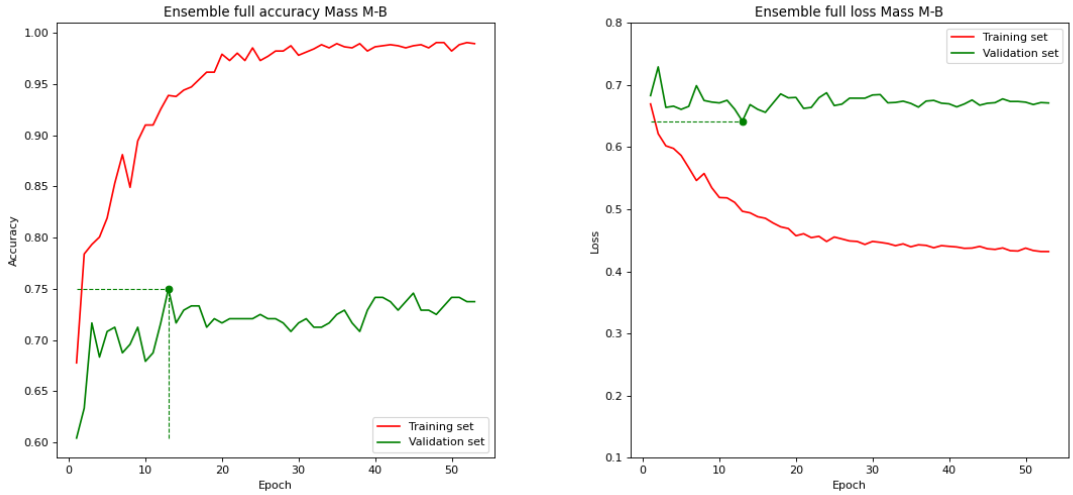


Figura 90. Graficas de *accuracy* y pérdida del entrenamiento del ensamble completo: los dos modelos ResNetV2 y el perceptrón multicapa (Figura 87).

La Tabla 30 y la Figura 88 muestran que el aprendizaje por ensamble, al entrenar solamente el perceptrón multicapa, supera los resultados obtenidos individuales por los modelos ResNet50V2 y ResNet101V2, sin embargo, no logra superar el resultado del modelo ResNet152V2, sin embargo, de la Figura 89, se puede observar que el modelo no realiza un buen aprendizaje y la existencia de sobreajuste. Cabe señalar que por más parámetros que se experimentó con estos modelos no se fue capaz de mejorar el comportamiento de estas curvas.

Por último, se menciona que no se obtuvieron buenos resultados al realizar el entrenamiento completo de los dos modelos ensamblados con el perceptrón multicapa (Figura 90).

4.5.3. Uso de los extractores ResNetV2

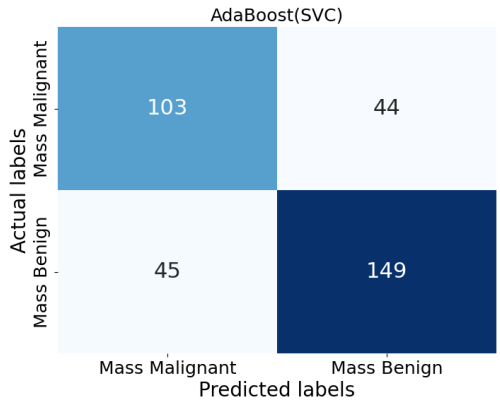
Se utilizaron los modelos ResNetV2 (Figura 85) como extractores de características. Primero se promediaron las características extraídas del el conjunto de entrenamiento de los tres extractores, formando un conjunto de 1024 características, mientras que en el segundo experimento solo se utilizaron las 1024 características de salida del extractor correspondiente al del modelo ResNet152V2, ya que este obtuvo el mayor *accuracy* individual durante la experimentación. Con estos dos conjuntos se entrenaron algoritmo de aprendizaje máquina, los resultados se muestran en las Tablas 31 y 32, mientras que la Figura 91 muestra los modelos marcados en rojo que aparecen en las tablas.

Tabla 31. Resultados obtenidos al combinar las predicciones de los tres extractores.

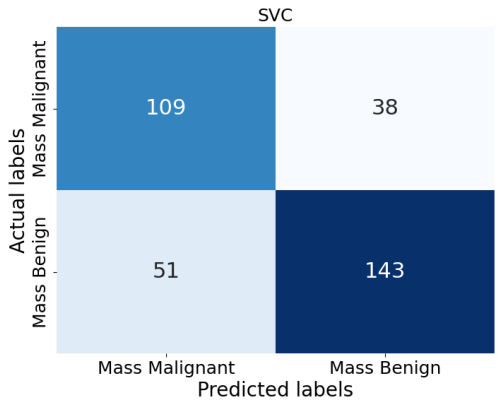
Clasificador	Accuracy	f1	MCC	Precisión	Recall
Random Forest	0.7272	0.7506	0.4525	0.7821	0.7216
AdaBoost(D. Tree)	0.7126	0.7393	0.4208	0.7637	0.7164
Regresión Logística	0.6686	0.6937	0.3369	0.7314	0.6597
Ridge	0.6803	0.7046	0.3606	0.7428	0.6701
SVC	0.7272	0.7506	0.4525	0.7821	0.7216
AdaBoost(SVC)	0.739	0.77	0.4683	0.772	0.768
XGBoost	0.6832	0.7112	0.3629	0.7388	0.6855

Tabla 32. Resultados obtenidos al utilizar las características del modelo ResNet152V2 como extractor.

Clasificador	Accuracy	f1	MCC	Precisión	Recall
Random Forest	0.7331	0.7586	0.4618	0.7814	0.7371
AdaBoost(D. Tree)	0.692	0.7075	0.3925	0.7696	0.6546
Regresión Logística	0.7184	0.7433	0.434	0.7722	0.7164
Ridge	0.7214	0.7466	0.4393	0.7734	0.7216
SVC	0.739	0.7626	0.4749	0.79	0.7371
AdaBoost(SVC)	0.736	0.7554	0.4739	0.7988	0.7164
XGBoost	0.7126	0.7336	0.4265	0.7758	0.6958



Matriz de confusión correspondiente al modelo AdaBoost con SVC por modelo o estimador base de la Tabla 31.



Matriz de confusión correspondiente al modelo SVC de la Tabla 32.

Figura 91. Matrices de confusión del mejor modelo en cada experimento.

Nota: Estos experimento fueron realizados en un entorno virtual de Anaconda Linux con TensorFlow versión 15 y una GPU de NVIDIA RTX 4080.

4.6. Sobre los resultados encontrados en la literatura

La Tabla 33 muestra algunos resultados del estado del arte en tareas de clasificación de cáncer con el uso de la base de datos DDSM y CBIS-DDSM.

Tabla 33. M,C indica si se realiza clasificación de Masas y/o Calcificaciones.

Referencia	Métodos	M,C	Métricas
Das et al. (2023)	Varios	M,C	89 % acc.
Ansar et al. (2020)	BreastNet18 (VGG16)	M	98 % acc.
Nasir Khan et al. (2019)	4-Views, VGG-like- network	M,C	77 % acc., 76 % AUC.
Fraga et al. (2022)	VGG16	M	80 % acc.
Ragab et al. (2019)	AlexNet-SVM	M	87 % acc.
Hassan et al. (2020)	AlexNet	M	100 % acc.
Zahoor et al. (2022)	Varios	M	94 % acc.
Nazir et al. (2022)	CNN-InceptionV4	M	99.2 % acc.
Shu et al. (2020)	RGP-CNN, GGP-CNN	M,C	76 % acc.
Shen et al. (2019)	Patch to whole image CNN	M,C	91 % AUC
Mobark et al. (2022)	ConoNet (Xception)	M,C	94 %acc.
Arora et al. (2020)	Varios	M,C	88 % acc., 88 % AUC
Lai (2021)	VGG16	M-B	69 % acc.
Jaamour et al. (2023)	VGG19, MobileNetV2	M; C; M-B	64.35 % acc.; 67.05 % acc., 67.08 % acc.,
Baccouche et al. (2022)	ResNetV2	tejido maligno	95.13 % acc.
Falconí et al. (2020)	VGG16	M	81 % acc., 84 % AUC
Nemade et al. (2024)	Ensamble VGG16-19-InceptionV3	tejido maligno	98.1 % acc.,

Trabajo Realizado	Stacking	M;	75.3 % acc., 81.9 % prec.;
	Votaciones	C;	69.88 % acc.
	Votaciones	M-B	72.5 % acc.

Muchos de los resultados encontrados en la literatura (Tabla 33) presentan algunos retos para tratar de obtener o reproducir los mismos resultados, ya que algunos no suelen ser tan precisos o detallados en la forma de preprocesar o hacer el aumento de datos en las imágenes, incluso suele existir un número distinto de imágenes reportadas en algunos artículos relacionados con la base de datos, pues no muestran estadística alguna de los datos con los que experimentan. La falta de código de programación y la falta de los parámetros utilizados en las redes también dificulta el entender o reproducir los resultados mencionados, sin embargo, la parte más importante es la falta de un conjunto de *Test* con el que todos se puedan comparar, ya que a pesar de que existe uno (el proporcionado por la base de datos), el número de imágenes es realmente pequeño como para obtener buenos resultados con solo el conjunto de entrenamiento, por lo que algunos autores juntan todos los datos para realizar aumento de datos y evalúan después en un lote de imágenes distinto al que solamente ellos tienen acceso, incluso, a veces, se suele sospechar de la inexistencia de este conjunto de *Test*, ya que se puede evaluar en el conjunto de datos que se utilizó para entrenar o supervisar el aprendizaje de las CNNs (conjunto de validación), lo cual no es recomendable, y mucho menos lo es el evaluar los modelos en el conjunto de datos con el que se entrenaron, ya que esto puede ocasionar una tasa falsa de clasificación del 100 % de *accuracy*, pero afortunadamente, no todos los resultados encontrados parecen mostrar este tipo de prácticas.

Capítulo 5. Conclusiones

A través de este trabajo de investigación se implementaron distintas metodologías del aprendizaje máquina en combinación con el aprendizaje profundo, convergiendo en la realización de distintos experimentos relacionados con la clasificación de anormalidades en una mastografía y las distintas formas de clasificar sus patologías (detección temprana del cáncer). Para esto, fue necesario explorar distintos parámetros y preprocesamientos encontrados en la literatura con el fin de poder obtener tasas de clasificaciones lo más altas posibles por cada una de las CNNs en cada uno de los experimentos realizados.

Dentro de los experimentos realizados, el experimento que distingue (clasifica) entre masas y calcificaciones resultó ser el más alto, llegándose a obtener un 95 % de *accuracy*, y como claramente lo muestran las matrices de confusión del experimento multiclase, la dificultad yace en la clasificación temprana del cáncer de mama, ya que estas muestran también una clara distinción entre masas y calcificaciones, pero no así en la clasificación de patologías (benignas o malignas). Al adentrarse más en los experimentos realizados, se mostró varias veces que cuesta más el detectar el cáncer de mama utilizando calcificaciones, por lo que se prestó más atención al uso de masas para realizar esta tarea. Los resultados obtenidos por cada una de las CNNs superó el 70 % de *accuracy*, pero nunca se llegó al 75 % de manera individual. Explorando distintos enfoques, como el de extracción de características de las capas de los perceptrones multicapa, se logró superar ligeramente los resultados obtenidos por las CNNs en algunos experimentos, siendo los clasificadores por votos los que superaron siempre ligeramente los resultados obtenidos por cada una de ellas. Esto indica que al entrenar distintos clasificadores es recomendable realizar la clasificación por votos. Además, la técnica de Stacking, al igual que la clasificación por votos, terminó mejorando ligeramente los resultados obtenidos, ya que con el uso de esta se obtuvieron los mejores modelos, uno con un 75.95 % de *accuracy* y el mejor modelo seleccionado obtuvo un 75.36 % de *accuracy*, pero con un 81.97 % de precisión, es decir, se mantuvo como el segundo mejor modelo, pero clasificó menos número de masas benignas como malignas, lo cual lo convierte en el modelo más confiable.

Además, se exploraron metodologías más avanzadas, como el uso de modelos híbridos para elevar más la tasa de clasificación de patologías y, aunque los resultados muestran una mejora con dos de los modelos ensamblados, el coste computacional y las tasas de clasificación de estos modelos provocaron que no se lograra superar la tasa de clasificación del mejor modelo en el ensamble.

Lo anteriormente descrito permite concluir que las CNNs son los modelos más importantes, ya que los resultados finales dependen mucho de las tasas de clasificación obtenidas mediante el uso de estos modelos, por lo que se debería de enfocar más en lograr una alta tasa de clasificación con estas, además,

los "errores que cometen" estos modelos pueden ser ligeramente corregidos con el uso de ensambles, principalmente con el uso de votaciones y Stacking, por lo que estos métodos deberían de utilizarse después de entrenar varias CNNs.

5.1. Limitaciones y Trabajo Futuro

Sin duda alguna la mayor limitación que se tiene corresponde al número de los datos, ya que como se sabe, el uso del aprendizaje profundo depende mucho de este número, llegándose a atacar problemas pequeños de 10 mil imágenes de entrenamiento, lo cual representa 5 veces más el tamaño de los datos utilizados, además, existen defectos encontrados en la base de datos, a la cual se le tuvo que realizar una pequeña exploración para que las estadísticas encontradas en Mračko et al. (2023) coincidieran lo mejor posible. De hecho, esta limitación ha hecho que distintos autores generen más datos utilizando el conjunto de *Test* como entrenamiento, pero a veces sin realizar una validación que permita medir de manera significativa la capacidad del modelo entrenado, por lo que en este trabajo de tesis se respeto la división general que la base de datos provee.

Para trabajo futuro se propone realizar la técnica de *Feture extraction* con alguno de los modelos VGG o con cualquier otro modelo, ya que aunque se tomó en cuenta realizar esta técnica con el modelo ResNet152V2, el número proporcionado de características por el *Flatten* supera las 100 mil y para un conjunto de datos de entrenamiento no mayor a 1500 imágenes podría generar problemas como la "maldición de la dimensionalidad" o un alto coste computacional. También se propone seguir algunos de los enfoques en los que se combinan las bases de datos, es decir, utilizar las tres bases de datos públicas disponibles: CBBIS-DDSM, InBreats y miniMias, con el fin de aprovechar más la data, como el utilizado por Baccouche et al. (2022), pero conociendo las limitaciones que estas puedan proveer (patologías no verificadas como en el caso de la base de datos INbreast).

Literatura citada

American Cancer Society. (2019). Mamogramas (mamografías). https://www.cancer.org/es/cancer/tipos/cancer-de-seno/pruebas-de-deteccion-y-deteccion-temprana-del-cancer-de-seno/mamogramas.html.

American Cancer Society. (2020). ¿Qué es el cáncer? https://www.cancer.org/es/cancer/entendimiento-del-cancer/que-es-el-cancer.html.

American Cancer Society. (2021). ¿Qué es el cáncer de seno? https://www.cancer.org/es/cancer/tipos/cancer-de-seno/acerca/que-es-el-cancer-de-seno.html.

Ansar, W., Shahid, A. R., Raza, B., & Dar, A. H. (2020). Breast cancer detection and localization using mobilenet based transfer learning for mammograms. In Brito-Loeza, C., Espinosa-Romero, A., Martin-Gonzalez, A., & Safi, A., editors, *Intelligent Computing Systems*, 11–21, Cham. Springer International Publishing. https://doi.org/10.1007/978-3-030-43364-2_2.

Arora, R., Rai, P., & Raman, B. (2020). Deep feature–based automatic classification of mammograms. *Medical Biological Engineering Computing*, *5*8. https://doi.org/10.1007/s11517-020-02150-8.

Baccouche, A., Garcia-Zapirain, B., & Elmaghraby, A. S. (2022). An integrated framework for breast mass classification and diagnosis using stacked ensemble of residual neural networks. *Scientific Reports*, *1*2(1). https://doi.org/10.1038/s41598-022-15632-6.

Chollet, F. (2021). *Deep Learning with Python*, (2a ed.). Manning Publications Co.

Das, H. S., Das, A., Neog, A., Mallik, S., Bora, K., & Zhao, Z. (2023). Breast cancer detection: Shallow convolutional neural network against deep convolutional neural networks based approach. *Frontiers in Genetics*, *1*3. https://doi.org/10.3389/fgene.2022.1097207.

Drukteinis, J. S., Mooney, B. P., Flowers, C. I., & Gatenby, R. A. (2013). Beyond Mammography: New Frontiers in Breast Cancer Screening. *The American Journal of Medicine*, *1*26(6), 472–479. https://doi.org/10.1016/j.amjmed.2012.11.025.

Falconí, L. G., Maria Perez, W. G. A., & Conci, A. (2020). Transfer learning and fine tuning in breast mammogram abnormalities classification on CBIS-DDSM database. *Advances in Science, Technology and Engineering Systems*, *5*(2), 154–165. https://doi.org/10.25046/aj050220.

Falconí, L. G., Pérez, M., & Aguilar, W. G. (2019). Transfer learning in breast mammogram abnormalities classification with mobilenet and nasnet. In *2019 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 109–114. https://doi.org/10.1109/IWSSIP.2019.8787295.

Fraga, J. A. G., Kober, V., Gutierrez-Lopez, E., & Gonzalez-Sarabia, J. A. (2022). Convolutional neural networks for automatic detection of breast pathologies. In Tescher, A. G. & Ebrahimi, T., editors, *Applications of Digital Image Processing XLV*, volume 12226. International Society for Optics and Photonics, SPIE. https://doi.org/10.1117/12.2633449.

García, A. (2017). *Inteligencia Artificial. Fundamentos, práctica y aplicaciones*, (2a ed.). Alfaomega Grupo Editor.

Géron, A. (2019). *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, (2a ed.). O'Reilly Media.

Hassan, S. A., Sayed, M. S., Abdalla, M. I., & Rashwan, M. A. (2020). Breast cancer masses classification using deep convolutional neural networks and transfer learning. *Multimedia Tools and Applications*, *7*9(41), 30735–30768. https://doi.org/10.1007/s11042-020-09518-w.

Instituto Mexicano del Seguro Social (2024). La Mastografía. https://www.imss.gob.mx/salud-en-linea/cancer-mama/mastografia#:~:text=Es%20un%20estudio%20de%20rayos,la%20observaci%C3%B3n%20o%20la%20palpaci%C3%B3n.

Jaamour, A., Myles, C., Patel, A., Chen, S. J., McMillan, L., & Harris-Birtill, D. (2023). A divide and conquer approach to maximise deep learning mammography classification accuracies. *PloS One*, *18*(5), e0280841. https://doi.org/10.1371/journal.pone.0280841.

Khattar, A. & Quadri, S. (2022). "Generalization of convolutional network to domain adaptation network for classification of disaster images on twitter". *Multimedia Tools and Applications*, *8*1. https://doi.org/10.1007/s11042-022-12869-1.

Lai, L. (2021). Medicalcnn: Abnormality detection in mammogram images using deep convolutional neural networks. https://github.com/leoll2/MedicalCNN.

Lee, R. S., Gimenez, F., Hoogi, A., Miyake, K. K., Gorovoy, M., & Rubin, D. L. (2017). A curated mammography data set for use in computer-aided detection and diagnosis research. *Scientific data*, *4*. https://doi.org/10.1038/sdata.2017.177.

Mobark, N., Hamad, S., & Rida, S. Z. (2022). Coronet: Deep neural network-based end-to-end training for breast cancer diagnosis. *Applied Sciences*, *12*(14). https://doi.org/10.3390/app12147080.

Moreira, I. C., Amaral, I., Domingues, I., Cardoso, A., Cardoso, M. J., & Cardoso, J. S. (2012). Inbreast: Toward a full-field digital mammographic database. *Academic radiology*, *19*(2), 236–248. https://doi.org/10.1016/j.acra.2011.09.014.

Mračko, A., Vanovčanová, L., & Cimrák, I. (2023). Mammography Datasets for Neural Networks—Survey. *Journal of Imaging*, *9*(5). https://doi.org/10.3390/jimaging9050095.

Naji, M., Alyassine, W., Nizamani, Q. U. A., Zhang, L., Wei, X., Xu, Z., Braytee, A., & Anaissi, A. (2022). Deep learning algorithm based support vector machines. In Daimi, K. & Al Sadoon, A., editors, *Proceedings of the ICR'22 International Conference on Innovations in Computing Research*, 281–289, Cham. Springer International Publishing.

Nasir Khan, H., Shahid, A. R., Raza, B., Dar, A. H., & Alquhayz, H. (2019). Multi-view feature fusion based four views model for mammogram classification using convolutional neural network. *IEEE Access*, *7*, 165724–165733. https://doi.org/10.1109/ACCESS.2019.2953318.

Nazir, M. S., Khan, U. G., Mohiyuddin, A., Al Reshan, M. S., Shaikh, A., Rizwan, M., & Davidekova, M. (2022). A novel cnn-inception-v4-based hybrid approach for classification of breast cancer in mammogram images. *Wireless Communications and Mobile Computing*, *2*022(1). https://doi.org/https://doi.org/10.1155/2022/5089078.

Nemade, V., Pathak, S., & Dubey, A. K. (2024). Deep learning-based ensemble model for classification of breast cancer. *Microsystem Technologies*, *3*0(5), 513–527. https://doi.org/10.1007/s00542-023-05469-y.

Nielsen, M. (2015). *Neural Networks and Deep Learning*. Determination Press. https://books.google.com.mx/books?id=STDBswEACAAJ.

Organización Mundial de la Salud. (2022). Cáncer. https://www.who.int/es/news-room/fact-sheets/detail/cancer.

Pesce, K., Orruma, M. B., Hadad, C., Bermúdez Cano, Y., Secco, R., & Cernadas, A. (2019). Bi-rads terminology for mammography reports: What residents need to know. *RadioGraphics*, *3*9(2), 319–320. https://doi.org/10.1148/rg.2019180068.

Ragab, D. A., Sharkas, M., Marshall, S., & Ren, J. (2019). Breast cancer detection using deep convolutional neural networks and support vector machines. *PeerJ*, *7*, e6201. https://doi.org/10.7717/peerj.6201.

Russell, S. & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*, (3a ed.). Prentice Hall.

Saha, S. (2018). A Guide to Convolutional Neural Networks — the ELI5 way. https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/.

Shen, L., Margolies, L. R., Rothstein, J. H., Fluder, E., McBride, R., & Sieh, W. (2019). Deep learning to improve breast cancer detection on screening mammography. *Scientific Reports*, *9*(1). https://doi.org/10.1038/s41598-019-48995-4.

Shu, X., Zhang, L., Wang, Z., Lv, Q., & Yi, Z. (2020). Deep neural networks with region-based pooling structures for mammographic image classification. *IEEE Transactions on Medical Imaging*, *3*9(6), 2246–2255. https://doi.org/10.1109/TMI.2020.2968397.

VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly Media. https://jakevdp.github.io/PythonDataScienceHandbook/.

Vinnicombe, S., Pinto Pereira, S. M., McCormack, V. A., Shiel, S., Perry, N., & Dos Santos Silva, I. M. (2009). Full-field digital versus screen-film mammography: Comparison within the uk breast screening program and systematic review of published data. *Radiology*, *2*51(2), 347–358. https://doi.org/10.1148/radiol.2512081235.

Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. *Journal of Big Data*, *3*, 9. https://doi.org/10.1186/s40537-016-0043-6.

Zahoor, S., Shoaib, U., & Lali, I. U. (2022). Breast cancer mammograms classification using deep neural network and entropy-controlled whale optimization algorithm. *Diagnostics*, *1*2(2). https://doi.org/10.3390/diagnostics12020557.

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023). *Dive into Deep Learning*. Cambridge University Press. https://D2L.ai.

Zuiderveld, K. (1994). *Contrast limited adaptive histogram equalization*, 474–485. Academic Press Professional, Inc., USA.

Anexos

Anexo A. Exploración del Experimento M-C

Dado que el experimento M-C fue el más alto, se pretendió profundizar en este para presentarlo en el **SPIE** (*Society of Photo-Optical Instrumentation Engineers*), el cuál se llevó a cabo los días del 18-22 de Agosto de 2024. Esto sirvió para experimentar con la teoría ya descrita en los capítulos 2 y 3, ya que se aplicaron las técnicas de *Feature Extraction*, *FC Extraction* junto con la teoría de aprendizaje máquina. Se utilizaron las siguientes tres redes neuronales convolucionales del experimento de la **Variante Parámetros iniciales**:

1. DenseNet121 con 1024 neuronas, la cual obtuvo un accuracy del 0.8783.
2. VGG16 con 1024 neuronas, la cual obtuvo un accuracy del 0.8983.
3. VGG19 con 128 neuronas, la cual obtuvo un accuracy del 0.9033.

Observe que estas redes no son las descritas en la Tabla 10, ya que DenseNet121 tiene 1024 unidades, las mismas que la red VGG16, mientras que la red VGG19 tiene 128. El objetivo del experimento es poder obtener un mejor *accuracy*, a partir de la arquitectura VGG16, que el obtenido de la red VGG19 utilizando *Feature y FC Extraction* así como la combinación de las tres en conjunto.

En la Figura 92 se muestra la arquitectura VGG16 (abajo) así como la modificación realizada (arriba) para la tarea de clasificación entre masas y calcificaciones, cuyas modificaciones corresponden a las descritas en la Tabla 4. Observe que de la red modificada se extrajeron dos vectores de características. El primero está relacionado con el concepto de *Feature Extraction*, el cuál es el vector *Feature Vector 1*. Cabe mencionar que este se extrajo de la última capa convolucional de la arquitectura VGG16, cuyo aplanado o *Flatten* produjo 25088 características. El segundo vector esta relacionado con el concepto de *FC Extraction* es el *Feature Vector 2*. Este vector tiene por número de características el mismo número de neuronas, a saber 1024.

A continuación se muestran los resultados obtenidos en tablas y matrices de confusión. Cabe mencionar que se marcan en azul los resultados iguales o mejores a los obtenidos por la red VGG19, y en rojo el modelo con menos imágenes mal clasificadas.

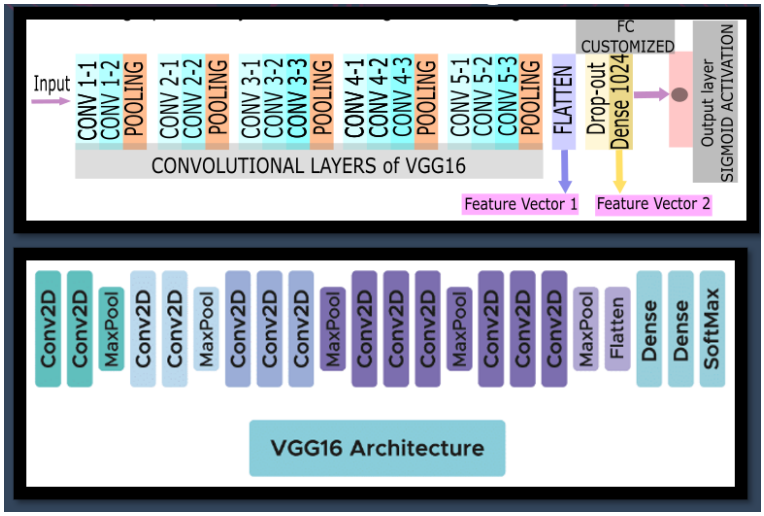


Figura 92. En la parte de arriba se puede observar la arquitectura VGG16 modificada para nuestra tarea de clasificación, mientras que abajo se muestra la arquitectura completa de VGG16 con 4096 unidades en cada capa densa y *activación ReLu*, capaz de clasificar 1000 clases. Imagen modificada de internet.

A.1 Resultados individuales

Los resultados individuales son las métricas e imágenes mal clasificadas que se obtuvieron en base a las redes descritas al inicio de esta sección, por lo que se presentan las matrices de confusión (Figura 93) y las métricas asociadas a estas (Tabla 34).

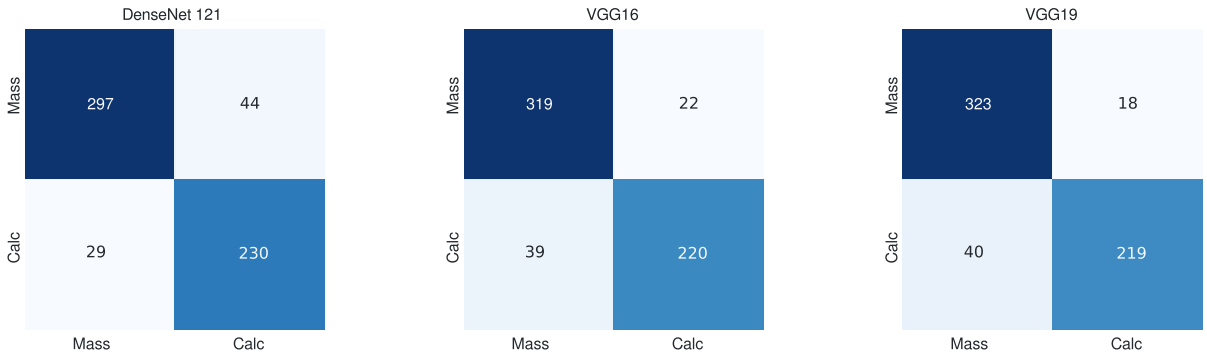


Figura 93. Resultados individuales de cada CNN descrita previamente.

Tabla 34. Resultados Individuales relacionados a la Figura 93.

Clasificador	Accuracy	Recall	F1	MCC	Precisión	Imágenes mal Clasificadas
DenseNet 121	0.8783	0.8880	0.8635	0.7547	0.8394	73
VGG16	0.8983	0.8494	0.8782	0.7924	0.9090	61
VGG19	0.9033	0.8455	0.8830	0.8032	0.9240	58

A.2 VGG16 como Extractor de Características

En esta parte se muestran las métricas y matrices de confusión obtenidos al utilizar los vectores de características *Feature Vector 1* y *Feature Vector 2* para generar un nuevo conjunto de datos (por cada vector) para entrenar algoritmos de aprendizaje máquina.

Con respecto al *Feature Vector 1*, la Tabla 35 muestra los mejores resultados obtenidos y la Figura 94 sus matrices de confusión asociadas.

Tabla 35. Resultados Relacionados con la Figura 94.

Clasificador	Accuracy	Recall	F1	MCC	Precisión	Imágenes mal Clasificadas
Naive Bayes	0.6516	0.8764	0.6847	0.3774	0.5618	209
AdaBoost (Naive Bayes)	0.8133	0.7876	0.7846	0.6199	0.7816	112

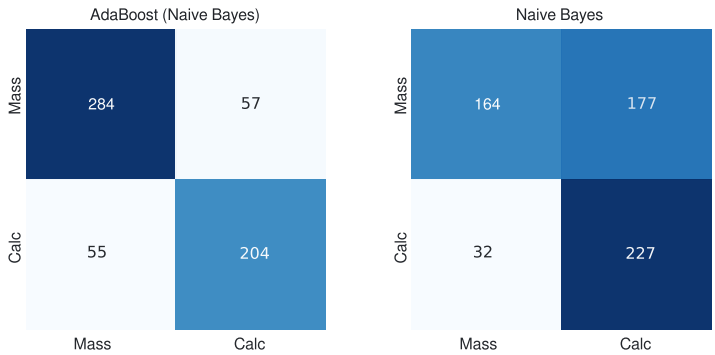


Figura 94. Resultados con modelos Naive Bayes utilizando *Feature Vector 1*.

Con respecto al *Feature Vector 2*, la Tabla 36 muestra los mejores resultados obtenidos y la Figura 95 sus matrices de confusión asociadas.

Tabla 36. Resultados relacionados con la Figura 95.

Clasificador	Accuracy	Recall	F1	MCC	Precisión	Imágenes mal Clasificadas
AdaBoost (SVC)	0.5683	0.000	0.000	0.000	0.000	259
SGD	0.6833	0.9961	0.7308	0.5021	0.5771	190
Regresión Logística	0.9033	0.8455	0.8830	0.8032	0.9240	58

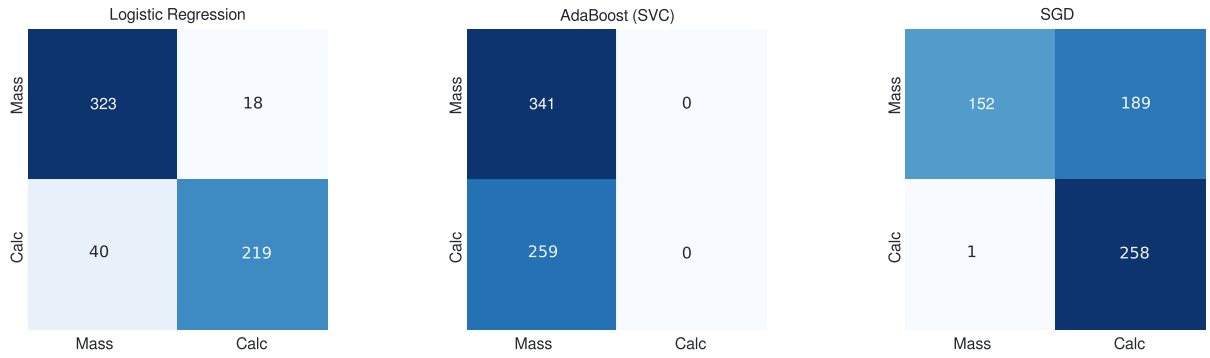


Figura 95. Regresión Logística y modelos sesgados hacia cada una de las clases: AdaBoost(SVC) y SGD. Estos resultados se obtuvieron utilizando *Feature Vector 2*.

La Tabla 37 muestra más resultados al utilizar el *Feature Vector 2*. Además de otro clasificador se muestran los resultados obtenidos or votaciones de los modelos utilizados, los cuales se pueden observar, junto con sus matrices de confusión, en la Figura 96.

Tabla 37. Resultados relacionados con la Figura 96.

Clasificador	Accuracy	Recall	F1	MCC	Precisión	Imágenes mal Clasificadas
BP Random Forest	0.8800	0.8764	0.8631	0.7566	0.8500	72
Voting Cl 1	0.9033	0.8455	0.8830	0.8032	0.9240	58
Voting Cl 2	0.9083	0.8378	0.8875	0.8146	0.9434	55

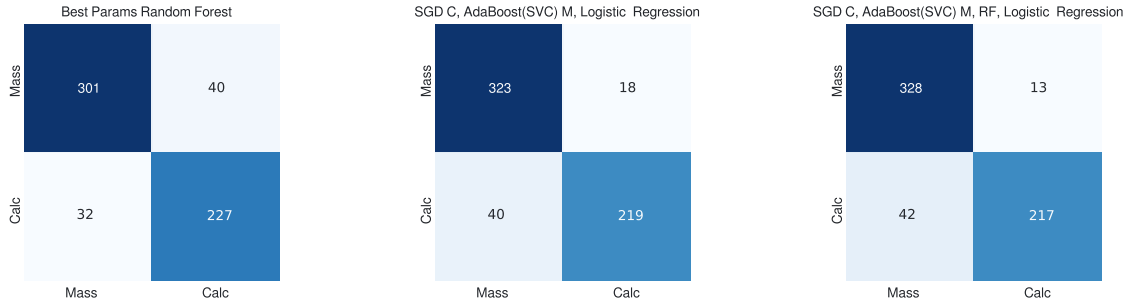


Figura 96. Se muestra el Random Forest pero con ajuste de Hiperparámetros, los clasificadores por votaciones CL1 y Cl2. Estos resultados se obtuvieron utilizando *Feature Vector 2*.

A.3 Stacking y Votaciones usando las CNNs

En esta última parte se trabajaron directamente con los resultados individuales obtenidos por las CNNs. La Figura 97 muestra las combinaciones por votos de los tres modelos y los resultados se observan en la Tabla 38.

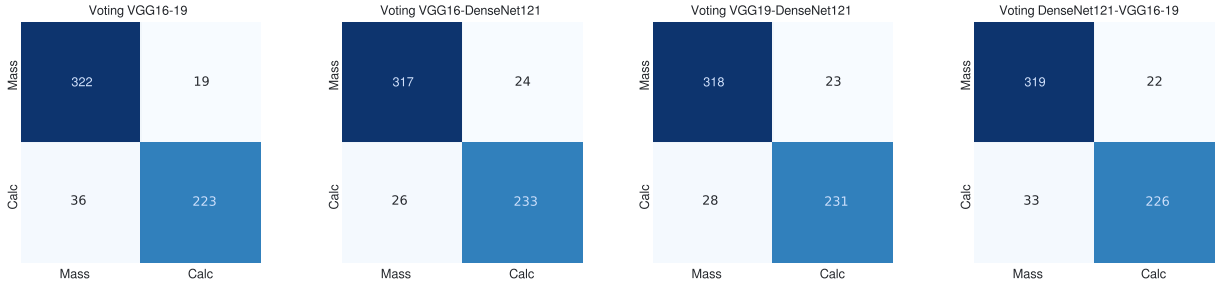


Figura 97. Combinación de los modelos VGG16, VGG19 y DenseNet121 para crear Clasificadores por votos.

Tabla 38. Resultados de la Figura 97.

Clasificadores por votos	Accuracy	Recall	F1	MCC	Precisión	Imágenes Clasificadas
VGG16-19	0.9083	0.8610	0.8902	0.8130	0.9214	55
VGG16-Dens	0.9166	0.8996	0.9029	0.8301	0.9066	50
VGG19-Dens	0.915	0.8918	0.9005	0.826	0.9094	51
All CNNs	0.9083	0.8725	0.8915	0.8127	0.9112	55

La Tabla 39 muestra los resultados de los *blending sets* conformados por las CNNs junto con algunos meta-clasificadores, además, se pueden observar sus matrices de confusión asociadas en la Figura 98.

Tabla 39. Mejores resultados obtenidos con cada *blending set* y meta-clasificador relacionados con la Figura 98.

Blending Set	Meta-Learner	Accuracy	Recall	F1	MCC	Precisión	Imágenes mal Clasificadas
VGG16-19	Bagging(LogReg)	0.9116	0.8648	0.8942	0.8199	0.9256	53
VGG16-Dens	RidgeCL	0.9166	0.9034	0.9034	0.8300	0.9034	50
VGG19-Dens	Logistic Reg.	0.9133	0.8957	0.8992	0.8232	0.9027	52
All CNNs	Bagging(SVC)	0.9116	0.9110	0.8990	0.8207	0.8872	53

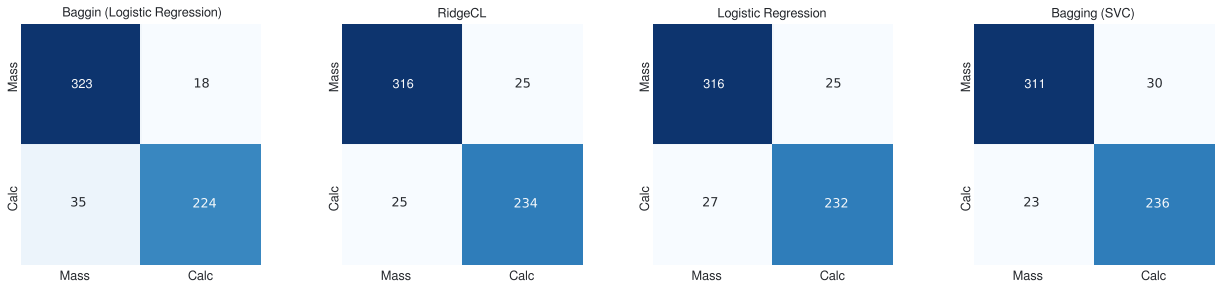


Figura 98. Meta-Clasificadores: (a) Bagging(Logistic Regresion), (b) RidgeCL, (c) Logistic Regression, (d) Bagging(SVC).

Anexo B. Experimento M-C con 5 semillas

El objetivo de este experimento fue obtener indicios de qué redes y ensambles de estas aparecen más si se cambia la partición de los datos, ya que es bien sabido en el ámbito científico que el uso de ciertas particiones, a pesar de ser 'estratificadas', pueden generar ligeramente mejores resultados.

Se utilizaron 5 distintas particiones del conjunto de datos (partición de los datos en entrenamiento y validación estratificados) cambiando el parámetro `random_state` de `train_test_split`. Utilizando los números 13,24,42,56 y 87 se generaron estas distintas particiones 'estratificadas'.

Para cada una de las 5 semillas se hizo lo siguiente. Primero se entrenaron las 6 redes mostradas en la Tabla10 utilizando una partición estratificada distinta, es decir, 5 veces cada una. Después en cada semilla se seleccionaron las mejores CNNs, es decir, las que tuvieron más lato *accuracy*. Posteriormente en cada semilla se asigno un umbral a superar, a saber el máximo accuracy obtenido de entre todas estas. Después, para cada semilla y con todas las redes se realizó el experimento Stacking y votaciones, además, por separado se realizó el experimento **FC Extraction** aplicado a ResNet50 con 256 neuronas, ya que esta había sido la CNN con más alto *accuracy* en este experimento.

El principal objetivo es ver que conjuntos de redes y/o Meta-Clasificadores aparecen más frecuentemente superando el umbral anteriormente mencionado sin importar la semilla, por lo que los resultados de manera muy resumida se muestran a continuación.

Votaciones. En este caso, las combinaciones que superaron su umbral en 4 de 5 semillas se muestran en la Figura 99. Más aún, el *accuracy* más alto se obtuvo con la semilla 24 al combinar las redes DenseNet121 y ResNet50 con un *accuracy* de 0.9533, mejorando el previo de 0.9483 al clasificar mal 28 imágenes (Figura 100).

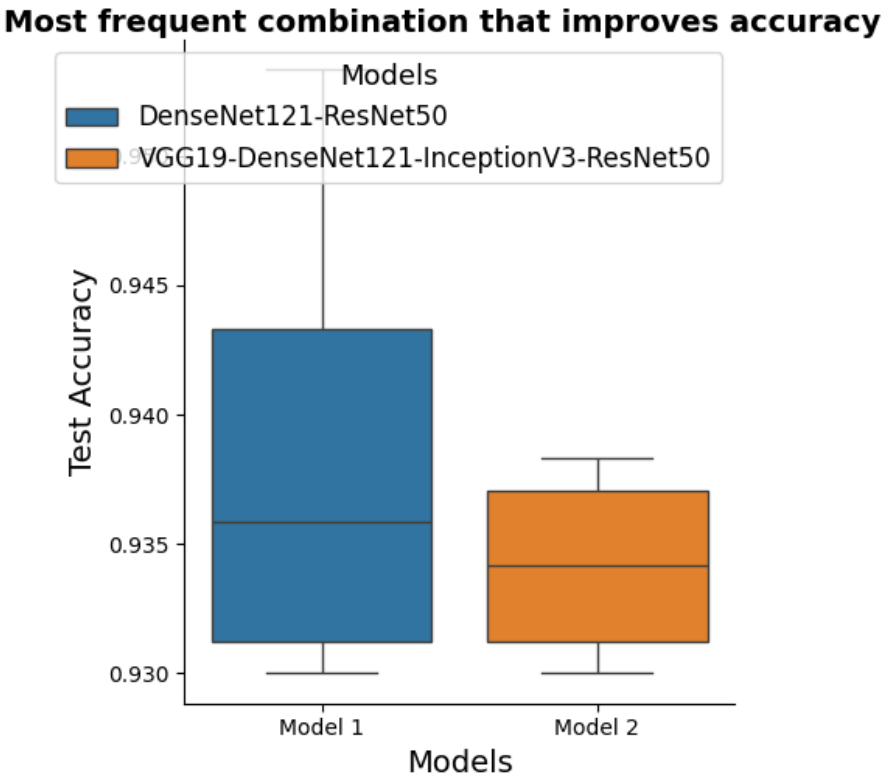


Figura 99. Se muestran las combinaciones más frecuentes de los modelos que superaron su umbral en 4 de 5 ocasiones.

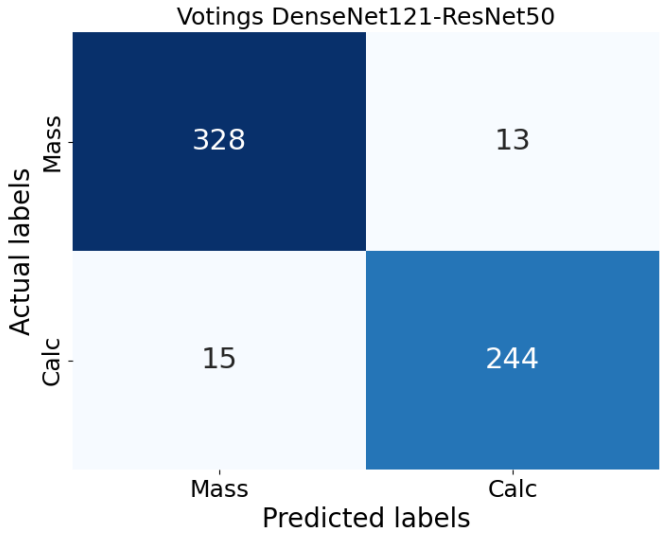


Figura 100. Se muestran la matriz de confusión del clasificador por votaciones conformado por las CNNs DenseNet121 y ResNet50 obtenidas en la semilla 24. Cabe destacar que el umbral a superar obtenido en esta semilla fue de 0.945, el cual le correspondió a la de ResNet50.

Stacking. En este experimento se rebasó el umbral por varios *meta-learners* con distintos *blending sets*, pero no superaron el *accuracy* obtenido por votaciones, por lo que solo se muestran algunos (Figura

101).

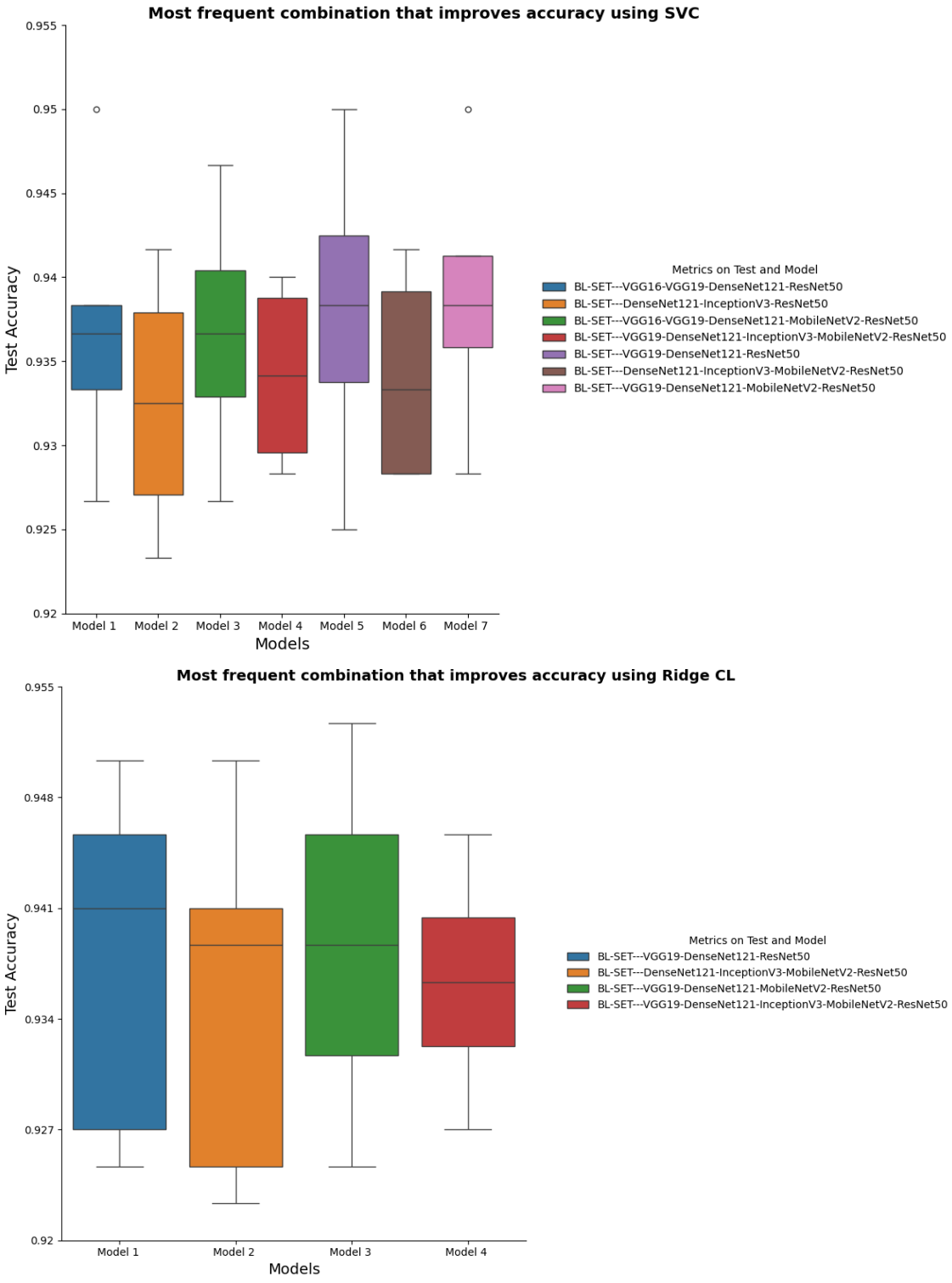


Figura 101. Se muestran las combinaciones más frecuentes de varios *blending sets* y *meta-learners* (SVC y Ridge Cl) que superaron su umbral correspondiente.

FC Extraction con ResNet50. Este experimento no fue concluyente, es decir, los *accuracys* suben o bajan dependiendo el clasificador o las combinaciones de estos, de los cuales a lo mucho una combinación aparece 2 veces.

Conclusiones

1. Las redes **Densenet12**1 y **ResNet50** deberían considerarse al realizar esta tarea de clasificación, pues son las que más frecuentemente aparecen.
2. Las votaciones es el método con el que se alcanzó el máximo *accuracy* en este experimento.

Por último, cabe destacar que dentro de todos los experimentos realizados (**M-C, M-B**, etc.) el que más mejoró al realizar FT fue el **Experimento M-C** y no solamente eso, sino que el modelo final parecía devolver mejores resultados, es decir, como se describió en la sección 4.2 se guardaron dos modelos durante la experimentación de las 5 semillas, uno 'óptimo' y el modelo correspondiente al finalizar el entrenamiento, recordando que este último corresponde a la falta de mejora de la función de pérdida. Curiosamente se observó que en la mayoría de los casos el modelo final devolvía mejores resultados. Esto parece indicar que se debería de entrenar por mayor tiempo o aumentar la paciencia en esta experimentación. El resultado de este comportamiento se ilustra en la Figura 102.

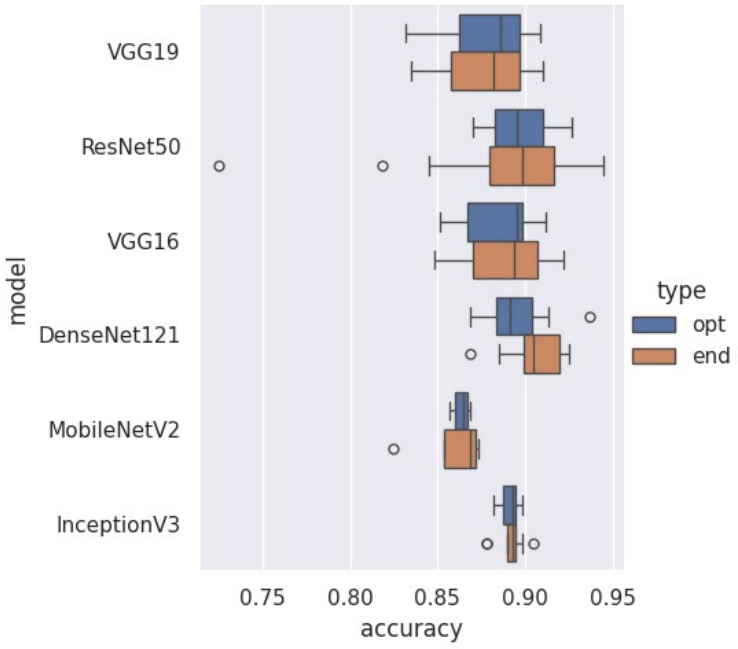


Figura 102. Se muestran los *accuracys* obtenidos de los modelos 'óptimo' y 'final' de todas las redes en estos cinco experimentos. Observe que el modelo final es bastante competitivo en obtener un mejor *accuracy* que el modelo 'óptimo' capturado por la combinación de `EarlyStopping` y `ModelCheckpoint`