UNIVERSIDAD
DE GRANADA

ETS Ingeniería Informática y de Telecomunicación

Máster en Ciencia de Datos e Ingeniería de Computadores

trabajo de fin de máster

Mejora de la Interpretabilidad en Modelos de Clasificación de Lesiones Cancerosas en Biopsias de Próstata mediante Técnicas de XAI.

Presentado por:
Carlos Lara Casanova

Tutor:
Francisco Herrera Triguero
Departamento de Ciencias de la Computación e Inteligencia Artificial
Mentor:
Iván Sevillano-García
Departamento de Ciencias de la Computación e Inteligencia Artificial

Curso académico 2024-2025

Mejora de la Interpretabilidad en Modelos de Clasificación de Lesiones Cancerosas en Biopsias de Próstata mediante Técnicas de XAI.

Carlos Lara Casanova

Carlos Lara Casanova *Mejora de la Interpretabilidad en Modelos de Clasificación de Lesiones Cancerosas en Biopsias de Próstata mediante Técnicas de XAI.*.

Trabajo de fin de Máster. Curso académico 2024-2025.

Responsables de tutorización

Francisco Herrera Triguero
Departamento de Ciencias de la Computación e Inteligencia Artificial

Iván Sevillano-García
Departamento de Ciencias de la Computación e Inteligencia Artificial

Máster en Ciencia de Datos e Ingeniería de Computadores

ETS Ingeniería Informática y de Telecomunicación

Universidad de Granada

Declaración de originalidad

D./Dña. Carlos Lara Casanova

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Máster (TFM), correspondiente al curso académico 2024-2025, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 30 de agosto de 2025

Fdo: Carlos Lara Casanova

Índice general

Agradecimientos

Gracias a Francisco e Iván por brindarme la oportunidad de trabajar con ellos y guiarme durante la realización de este trabajo. Gracias a mi familia por apoyarme durante el tiempo que me llevó desarrollar este trabajo.

Summary

KEYWORDS: neural networks, xai, explainable ai, prediction, explainability, artifical inteligence, machine learning, computer vision, deep learning, prostate cancer, gleason score, gleason groups.

Aquí va el resumen en inglés

Resumen

PALABRAS CLAVE: redes neuronales, xai, ia explicable, clasificación, explicabilidad, inteligencia artificial, aprendizaje automático, visión por computador, aprendizaje profundo, cáncer de próstata, gleason score, gleason groups.

Resumen en español.

1 Introducción

En 2020, el **cáncer de próstata**, (*Prostate Cancer, PCa*) fue el segundo tipo de cáncer más frecuente, y el quinto más mortal, en varones [HJR+21]. Una **biopsia de próstata** es una prueba que consiste en la extracción de pequeños tejidos de la próstata para examinar posibles signos de cáncer (Ver Figura 1.1). El **Sistema de Puntuación de Gleason** (*Gleason Score*) es una calificación, en el rango de 2 a 10, que se da a biopsias de la próstata tras ser examinadas bajo un microscopio [Sha24]. Valores más altos indican cánceres más agresivos y de crecimiento más rápido. En 2014 la **Sociedad Internacional de Patología Urológica** (*International Society of Urological Pathology, ISUP*) propuso un nuevo sistema basado en la Puntuación de Gleason, que propone cinco grupos ordenados (**Gleason Groups, GGs**) [EARH17]:

- **GG1:** Cáncer de grado bajo. Puntuación de gleason 6 o inferior.
- **GG2:** Cáncer de grado medio. Puntuación de gleason 7.
- **GG3:** Cáncer de grado medio pero más agresivo que GG2. Puntuación de gleason 7 pero percibido más agresivo.
- **GG4:** Cáncer de grado alto. Puntuación de gleason 8.
- **GG5:** Cáncer de grado alto. Puntuación de gleason 9 o 10.

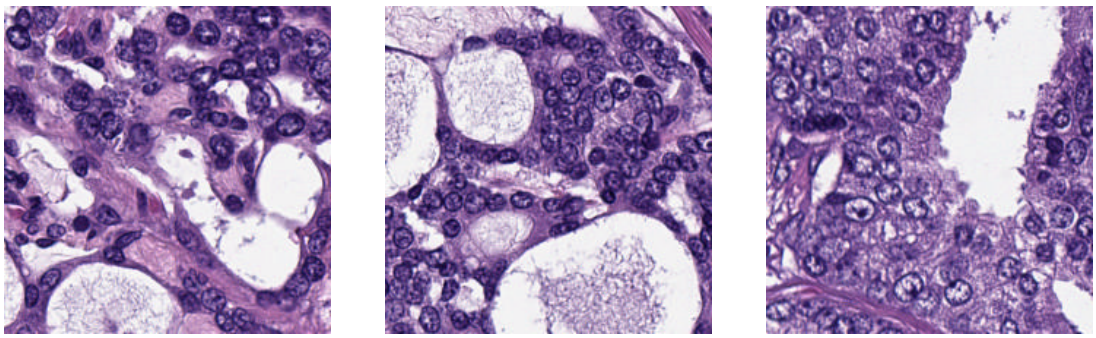

Figura 1.1: Tres imágenes correspondientes a tres biopsias distintas.

En el campo de la **Inteligencia Artificial (IA)**, la necesidad de asegurar transparencia y confiabilidad a dado pie a acuñar el término de **Inteligencia Artificial eXplicable** (*eXplainable Artificial Inteligence, XAI*) [LBC+24]. El uso de XAI es especiamente importantes en ámbitos donde la toma de decisiones repercute directamente en la vida de las personas, como es el caso de la medicina. La Unión Europea ha propuesto una regulación para el uso de IA asistida en estos aspectos, entre ellos, que estos sistemas sean capaces de explicar sus decisiones de forma clara y comprensible.

Dentro del campo de la XAI, hay diferentes propuestas para la evaluación y la mejora de las explicaciones generadas. Por un lado, herramientas como **REVEL** [SGLH23a] permiten analizar la calidad de las mismas de manera robusta. Por otro, enfoques como **X-SHIELD** [SGLH23b] buscan mejorar las explicaciones mediante técnicas de regularización que aseguran el buen comportamiento de la generación de explicaciones. Aplicar estas técnicas en imagen médica puede ayudar a mejorar la interpretación de las personas profesionales de la salud de las decisiones de una IA que asista.

En este contexto este Trabajo de Fin de Máster trata de resolver el problema de **construir un modelo de IA de clasificación para lesiones cancerosas en biposias de próstata** y de la **mejora en interpretabilidad del modelo mediante técnicas de XAI**. Para esto se vamos a utilizar las herramientas REVEL y XSHIELD, y propondremos posibles mejoras de XSHIELD.

El TFM se estructura del siguiente modo. En primer lugar vamos a introducir los fundamentos teóricos necesarios para el correcto entendimiento del trabajo realizado (Capítulo 2). Después se va a hacer un estudio del estado del arte (Capítulo 3). A continuación presentamos las propuestas de este TFM (Capítulo 4). En el Capítulo 5 se describe el conjunto de datos con el que trabajamos, los distintos experimentos que realizamos y discutimos los resultados . Finalmente en el Capítulo 6 se incluyen conclusiones y posibles trabajos futuros.

1.1. Objetivos

El objetivo principal del TFM es la utilización (y propuesta) de técnicas de evaluación y mejora de explicaciones para modelos de clasificación de imágenes para detección de tejidos cancerosos en biopsias de próstata. Para el desarrollo del TFM, se divide el objetivo en distintos subobjetivos:

- Revisión del estado del arte.

- Estudio del código que implementa [SGLH23a, SGLH23b] para su correcto entendimiento.
- Proposición de nuevos métodos XAI para mejora de explicaciones para modelos de explicación.
- Modificación del código para implementar dichas propuestas.
- Diseño de los experimentos a realizar e implementación de estos.
- Evaluación de los resultados y subsecuente discusión.

Aquí introduciría el objetivo principal del TFM y lo dividiría en algunos sub-objetivos (revisión del estado del arte, estudio de las implementaciones de las métricas ReVE-L/regularización X-Shield, proposición de mis regularizaciones, implementación de estas, experimentación y finalmente evaluación/discusión de estos).

1.2. Planificación

En esta sección hablo de cómo me he planificado el proyecto según estudio del problema/diseño de mis modelos/implementación/experimentación (haría una tabla). Luego haría una comparación de mi planificación vs cómo se ha repartido finalmente y haría una estimación final de cuántas horas me ha llevado el TFM y en cuánto se estima el coste del proyecto final.
POR HACER

2 Fundamentos teóricos

2.1. Aprendizaje automático

El **aprendizaje automático** (AA) o **machine learning** (ML) es una rama de la **inteligencia artificial** que se encarga del desarrollo de algoritmos que aprendan de datos o experiencias con el fin de mejorar su rendimiento en ciertas tareas [Alp20, AMMIL12, Bis06, Mur22]. Estos algoritmos disponen de un **modelo** definido por ciertos parámetros y dichos parámetros serán los que se aprendan a través del **entrenamiento** o **aprendizaje**.

Para saber a qué nos referimos cuando se habla de que el algoritmo *"aprende"* conviene destacar la siguiente definición proveniente de [Mit97]: *"Un programa se dice que aprende de una experiencia E con respecto a una clase de tareas T y a una medida de rendimiento T, si su rendimiento en las tareas en T, medido por P, mejora con la experiencia E."*

El AA se utiliza habitualmente para resolver problemas complejos que no se pueden o se saben resolver con algoritmos tradicionales diseñados por humanos. Además se requiere tener datos de los que poder aprender. Algunos de los campos donde se utilizan el AA para resolver este tipo de problemas son el procesamiento de lenguaje natural, el reconocimiento del habla o la visión por computador, entre muchos otros.

Los algoritmos de AA se clasifican en distintos paradigmas dependiendo de los datos o experiencia de la que se disponga. Existen tres paradigmas principales: el **aprendizaje supervisado**, el **aprendizaje no supervisado** y el **aprendizaje por refuerzo** [AMMIL12, GBC16].

El aprendizaje supervisado hace referencia a los algoritmos de AA en los que se tiene un conjunto de datos agrupados en pares dato-**etiqueta**. La etiqueta se corresponde con la salida que se requiere del algoritmo de AA cuando tenga de entrada dicho dato. Un ejemplo típico de conjunto de datos de este tipo sería un conjunto de imágenes de números escritos a mano y para cada imagen el número que está escrito en dicha imagen.

El aprendizaje no supervisado hace referencia a los algoritmos de AA en los que se disponen de datos que no están etiquetados. En este tipo de algoritmos no se quiere aprender algo específico de los datos sino más bien encontrar patrones o estructurar los datos de entrada. Por ejemplo se dispone de distinta información relativa a libros (autor, título, número de páginas, etc) y el algoritmo categoriza los libros en distintas categorías, aunque no se sepa a qué se corresponde cada categoría.

En el aprendizaje por refuerzo no se disponen de datos de entrada, sino de una entrada y una *recompensa* correspondiente a cada salida posible para dicha entrada. Este tipo de AA es especialmente útil para aprendizaje de juegos o tareas relacionadas con la teoría de control.

Para este TFG es relevante el aprendizaje supervisado. En particular serán relevantes los dos siguientes problemas típicos de aprendizaje supervisado.

2.1.1. Problema de clasificación

Este problema trata de clasificar las entradas de un tipo concreto en un número de clases k. Es decir, para una entrada el programa deberá clasificar dicha entrada en una de las k clases. Por lo tanto el algoritmo produce una función $f : \mathbb{R}^n \to \{1, ..., k\}$ que a cada entrada $\mathbf{x}$ le asocie una clase $y = f(\mathbf{x})$. Dicha función tratará de aproximar una función objetivo g que clasifique perfectamente todas las entradas.

Para este problema se disponen de una serie de datos de entrada $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$ con $\mathbf{x}_i \in \mathbb{R}^n$ y sus clases correspondientes $\{y_1, ..., y_N\}$ con $y_i \in \{1, ..., k\}$.

Algunas formas de resolver estos problemas son mediante árboles de clasificación, máquinas de vectores-soporte o mediante redes neuronales.

Un caso particular de este problema que es más relevante para este trabajo es el caso en el que hay dos clases. Este problema se denomina **problema de clasificación binaria**. En este TFG se tratará de determinar visibilidad de puntos de la cara de una persona en una imagen luego las dos clases se corresponden con visible o no visible.

2.1.2. Problema de regresión

Este problema es similar al de clasificación, pero la función que aprende el algoritmo es de la forma $f : \mathbb{R}^n \to \mathbb{R}^m$. Por lo tanto ahora el programa debe predecir un valor numérico (o varios si $m > 1$) para una entrada concreta. Al igual que ocurría en el

problema de clasificación se disponen de una serie de datos de entrada y el resultado que se espera del programa para dichas entradas, es decir $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$ con $\mathbf{x}_i \in \mathbb{R}^n$ y $\{\mathbf{y}_1, ..., \mathbf{y}_N\}$ con $\mathbf{y}_i \in \mathbb{R}^m$.

Algunas formas de resolver este tipo de problemas son mediante la regresión lineal, máquinas de vectores-soporte o redes neuronales.

2.1.3. Optimización

Ya se ha hablado de algunos problemas de AA. En ellos se quiere mejorar el rendimiento de una función f para una tarea. La forma en la que se mide el rendimiento es mediante otra función denominada **función de error**, **función de pérdida** o **función de coste** [GBC16]. Esta función mide el error de la aproximación de la salida de f para una entrada $\mathbf{x}$. Por lo tanto se busca minimizar dicha función.

Por ejemplo, una función de coste habitual es el error cuadrático medio:

$$ECM = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Donde $\hat{y}_i$ es la predicción de nuestro modelo para un dato x_i y y_i es el valor correcto de salida para esa entrada.

El método principal en el AA para minimizar dicha función es el **gradiente descendente**. Este método iterativo se basa en que, para una función multivariable dos veces diferenciable, el gradiente de dicha función en ese punto (que es un vector) apunta en la dirección contraria al ascenso más empinado. Por lo tanto tomando pasos en la dirección contraria al gradiente, debería disminuir la función de coste y por lo tanto dar un mejor rendimiento.

Se recuerda que el gradiente de una función $E : \mathbb{R}^n \to \mathbb{R}$ en un punto no es más que le vector compuesto de las derivadas parciales respecto a cada variable en dicho punto:

$$\nabla E(\mathbf{w}) = \left[\frac{\partial E}{\partial x_1}(\mathbf{w}), ..., \frac{\partial E}{\partial x_n}(\mathbf{w})\right]^\top$$

Normalmente, el método utilizado para resolver el problema de AA hace que las funciones que se pueden representar en la solución estén parametrizadas por un vector de parámetros o **pesos w**. Por ejemplo, en un problema de regresión con función objetivo $f : \mathbb{R}^n \to \mathbb{R}$ que se plantea resolver mediante regresión lineal se tiene que la función que se aprende toma la forma:

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} \qquad , \mathbf{x} \in \mathbb{R}^n, \mathbf{w} \in \mathbb{R}^n$$

Donde el vector **w** son los parámetros que se pueden cambiar a la función para mejorar su rendimiento.

La función de coste dependerá de dichos parámetros. Luego el gradiente descendiente cambiará el valor de los parámetros en cada paso buscando minimizar la función de coste. Estos cambiarán según la siguiente regla:

$$\mathbf{w}_n = \mathbf{w} - \eta \nabla E(\mathbf{w})$$

Donde $\mathbf{w}_n \in \mathbb{R}^l$ son los nuevos parámetros, $\mathbf{w} \in \mathbb{R}^l$ son los parámetros del paso actual, E es el función de coste, ∇E representa el gradiente de E y η es un hiperparámetro (parámetro del algoritmo) en $\mathbb{R}$ llamado **learning rate**(lr).

El lr determina el tamaño de los pasos tomados por el algoritmo en cada iteración. Un valor del lr alto puede hacer que el algoritmo converja más rápido pero también puede hacer que el algoritmo no llegue a converger o se salte soluciones. Por otro lado un valor bajo del lr hace que el algoritmo converja demasiado lento a un mínimo de la función. En la práctica se suelen utilizar métodos que hacen que el lr varíe a lo largo del entrenamiento, normalmente de mayor a menor.

El algoritmo del gradiente descendente inicia los pesos **w** a un valor aleatorio y empieza a iterar con la regla anteriormente descrita buscando reducir el error en cada paso durante un número de **épocas** (cada época itera sobre todos los datos de entrenamiento), aunque hay otros criterios de parada del algoritmo. En caso de ser satisfactorio el algoritmo para en un mínimo local (o global idealmente) del error de pérdida.

El cálculo del gradiente del error conlleva utilizar todos los datos de entrenamiento lo que en la práctica resulta muy costoso. Por ello se suele utilizar el **descenso de gradiente estocástico** (SGD), que en cada paso calcula una estimación del gradiente real a partir de un subconjunto aleatorio de los datos de entrenamiento denominado **minibatch**. Destacar que los datos de entrenamiento no se deben repetir entre *minibatches*

hasta que se hayan utilizado todos los de entrenamiento entre cada repetición.

Sobreentrenamiento

Cuando se optimiza un modelo, puede ocurrir que el modelo se ajuste muy bien para los datos de entrada con los que se ha entrenado, pero tenga mal rendimiento para datos con los que no se ha entrenado y por lo tanto no generaliza bien lo aprendido de los datos de entrenamiento al problema. Cuando esto ocurre se dice que el modelo se ha **sobreentrenado** o **sobreajustado**. Una forma de identificar este problema es guardándonos un número de datos con los que no se va a entrenar el modelo y comparar el error del modelo a lo largo del entrenamiento de los datos de entrenamiento con los que nos se han guardado. El sobreentrenamiento se puede observar cuando el error en los datos con los que no se entrena empieza a aumentar a medida que se sigue entrenando el modelo como se puede ver en la Figura 2.1.

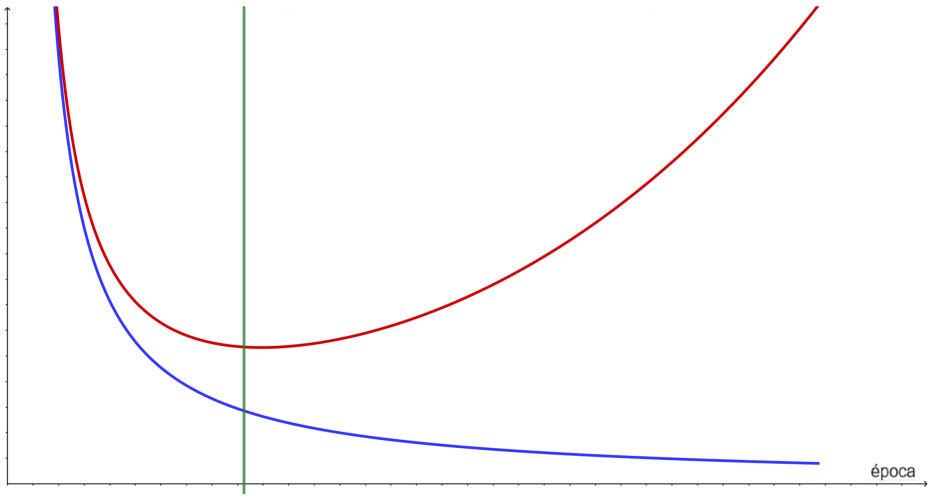

Figura 2.1: Esta figura muestra el error en entrenamiento en azul y el error en los datos con los que no se entrena el modelo en rojo a lo largo del entrenamiento. Se puede ver como a partir de la línea verde el modelo sigue aprendiendo de los datos de entrada pero empeora el poder de generalización con los datos con los que no entrena.

2.2. Deep Learning

Deep learning(DL) [BB23, GBC16, LBH15, Pri23, Sch15] es un subcampo del AA determinado por los métodos con los que aprende y tipo de datos con los que trata. Los algoritmos de DL pueden tratar datos poco estructurados como pueden ser imágenes o textos. Este tipo de algoritmos automatizan la extracción de características, es decir, determinan qué información de los datos es relevante para la resolución del problema. Esto elimina mucha de la intervención necesaria por los humanos a la hora de crear el algoritmo.

El DL en los últimos años ha supuesto un avance significativo en muchos campos en los que apenas se avanzaba con otras técnicas de IA como el reconocimiento del habla o el procesamiento de lenguaje natural.

Redes neuronales

Una **red neuronal** (*artificial neural network*, ANN) [Bis95, Hay98, MOM12] es un modelo de AA que consiste en una red conectada de **nodos** o **neuronas**. Cada neurona recibe una serie de números reales (señales) de las neuronas conectadas a esta, hace una operación con dichos números que resulta en otro número real y lo manda a los nodos conectados a esta.

La operación que realiza una neurona con $n \in \mathbb{N}$ entradas es la siguiente:

$$y = \sum_{i=1}^{n} \sigma(w_i x_i + w_0) \qquad , w_i, x_i \in \mathbb{R} \tag{2.1}$$

Dónde x_i son las entradas de la neurona, w_i son los **pesos** de la neurona, en particular a w_0 se le denomina sesgo, y σ es una función de los números reales a los números reales y se le denomina **función de activación** [SSA20]. Al conjunto de los pesos de todas las neuronas de una red se les denomina, naturalmente, **pesos de la red**. A través de la modificación de dichos pesos es como se entrenará a una red para realizar una tarea concreta.

Existen dos tipos de ANN según el flujo de la información dentro de la red. Nos centramos en el caso en el que este flujo es una única dirección, en cuyo caso se tiene una **red neuronal hacia delante** (*feedforward neural network* o *multilayer perceptron*,

MLP).

Las MLP son el modelo fundamental del *Deep Learning*. Estas están compuestas por distintas **capas**, cada una compuesta de varias neuronas en paralelo. La primera capa es la **capa de entrada** que recibe la entrada a la red y la propaga a la siguiente capa. Luego puede haber una o más **capas ocultas**, donde cada una de estas están compuestas de neuronas que reciben las señales de la capas anterior y propagan su salida a las neuronas de la siguiente capa. Por último está la **capa de salida** que está compuesta por uno o varias neuronas cuya salida se considera la salida de la red. El número de capas ocultas determina la **profundidad** de la red. Se puede ver un ejemplo de un modelo de este tipo en la figura 2.2.

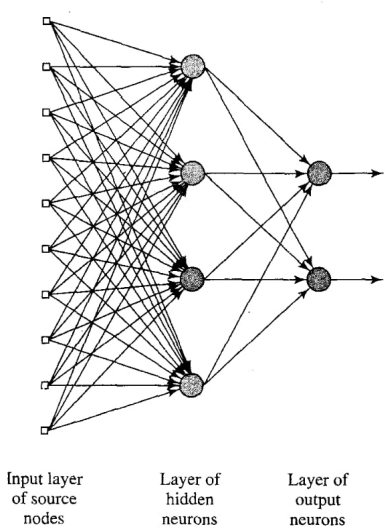

Figura 2.2: Ejemplo de ANN con una única capa oculta y con dos salidas. Imagen extraída de [Hay98].

Si se considera ahora la función f que implementa un red MLP, se tiene que esta depende de los pesos de la red. Una vez se tengan unos datos con los que entrenar la red y una función de coste que evalúe el rendimiento de la red, se puede aplicar el gradiente descendente estocástico para entrenar la red modificando los pesos de esta. Sin embargo, el cálculo del gradiente de la función de error en un red puede resultar muy costoso, especialmente para el cálculo de las derivadas parciales del error respecto de los pesos de las neuronas en las capas menos profundas. Es por esto que se suele utilizar un algoritmo, denominado **backpropagation**, para el cálculo del gradiente.

Este algoritmo está basado en la aplicación de la regla de la cadena que dicta que si se tiene una función f que depende de otra función g que a su vez depende de otra función h, entonces se cumple que:

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial z}$$

De esta forma se puede calcular la parcial de la función de coste respecto a los pesos de una capa utilizando las parciales respecto de los pesos de la siguiente. Luego aplicando esto desde el final al principio de la red, se consigue agilizar el cálculo del gradiente de la función de coste.

Funciones de activación

Como se comenta en la ecuación (2.1), la salida de una neurona es una suma ponderada de las entradas más un sesgo, a las que se le aplica una función de activación. La inclusión de este tipo de funciones es para evitar que la red sea lineal. Si la ecuación (2.1) fuese idéntica sin aplicar la función de activación entonces la salida de la red sería una función lineal respecto de la entrada, esto es, un polinomio de primer grado. Esto haría que la red no pudiera implementar funciones complejas lo que es necesario para aprender patrones más complejos presentes en los datos. Es por esto que normalmente las funciones de activación deben ser funciones no lineales. Destacar que la función de activación debe ser derivable para poder calcular el gradiente de la función de coste con el algoritmo *backpropagation* cuando se entrene la red.

Existen varias funciones de activación que se pueden utilizar, sin embargo son especialmente interesantes la función de activación **ReLU** (*rectifier linear unit*) definida como $f(x) = max\{0, x\}$ y la **sigmoidal** definida como $f(x) = \frac{1}{1+e^{-x}}$. Se pueden ver sus gráficas en la figura 2.3 y la figura 2.4, respectivamente.

La función de activación ReLU es la que se suele usar en las neuronas de las capas ocultas pues contrarresta el **problema de desvanecimiento del gradiente**. Un problema que afecta al entrenamiento de las redes neuronales que causa que cuando se calcula el gradiente del error para actualizar los pesos este tome valores muy pequeños haciendo que los pesos de la red cambien muy lentamente. Esto dificulta, o incluso impide, el entrenamiento de la red. Además esta función de activación es la más usada normalmente y tiene mejor rendimiento en la mayoría de los casos.

La función de activación sigmoidal se suele utilizar en la capa de salida en los

problemas de clasificación binarios.

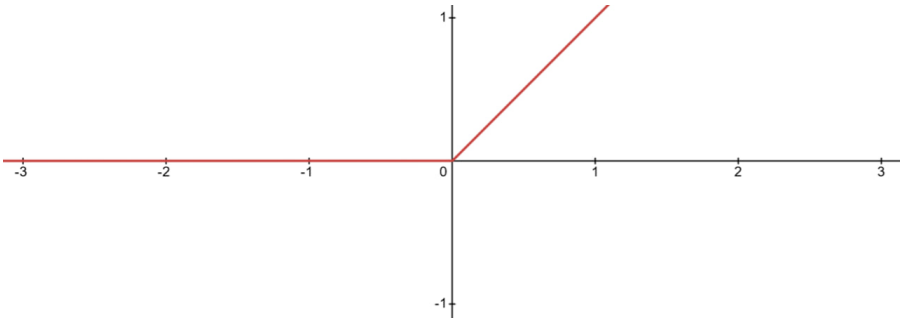

Figura 2.3: Función de activación ReLU.

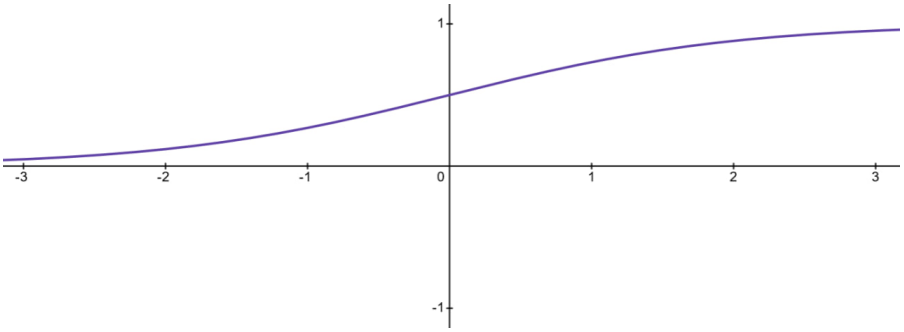

Figura 2.4: Función de activación sigmoidal.

Batch normalization

Batch normalization [IS15] es una método consistente en normalizar los datos de entrada de cada capa. De esta manera se consigue independizar la distribución de los datos de entrada de cada capa de la red, con los parámetros aprendidos por la red. Está probado que este método hace que el entrenamiento de la red sea más estable y más rápido. Además también se sabe que tiene un efecto regularizador en la red, lo que disminuye la probabilidad de que la red sobreentrene.

Dropout

Dropout [HSK+12] se refiere a la técnica utilizada en redes neuronales consistente en ignorar algunas neuronas de forma aleatoria durante el periodo de entrenamiento. Usualmente se le da a cada neurona una probabilidad p de ser ignorado, y en cada

paso del algoritmo de entrenamiento se desconecta cada neurona de la red con dicha probabilidad. De esta forma se busca que ninguna neurona sea muy dependiente de la salida de otra neurona específica. Esta técnica regulariza la red lo que hace que la red sea menos propensa a sobreentrenar.

2.2.1. Redes convolucionales

La **redes convolucionales** (*Convolutional neural networks*, CNN) [LLY+22] son un tipo de red neuronal feedforward diseñadas para procesar datos en forma de múltiples arrays. En nuestro caso nos centraremos en el caso en el que la entrada son imágenes, que son arrays en 2 dimensiones (matrices) que contienen tres **canales**, uno por cada color. La estructura las CNNs es una capa de entrada, seguida de varias capas ocultas y una capa de salida. Las capas ocultas suelen estar compuestas por una o varias **capas convolucionales** seguidas de una **capa de pooling**, esto repetido varias veces. Además al final de la red se suelen colocar una o varias **capas totalmente conectadas** hasta la capa de salida.

Capas convolucionales

Una **convolución** es un operador matemático que recibe dos funciones $f, g : \mathbb{R} \to \mathbb{R}$ y la transforma en otra función $f * g$ definida como sigue:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(x)g(t-x)dx$$

Basado en este operador, existe la **convolución discreta** en 2 dimensiones, que se define como sigue:

$$(f \star g)[i,j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m,n)g(i-m, j-n)$$

Este operador no es más que la adaptación de la convolución al caso en el que f y g sean funciones discretas, es decir, definidas sobre los números enteros, $\mathbb{Z}$. Este es el operador que aplican las capas convolucionales a sus entradas. En el caso de las convoluciones aplicadas en una CNN f se corresponde con la entrada de la capa convolucional y g con el **núcleo**, **kernel** o **filtro**. de la convolución. El *kernel* no es

más que una matriz de dimensiones $n \times m$ de valores reales. Véase la Figura 2.5 para ver un ejemplo de una convolución.

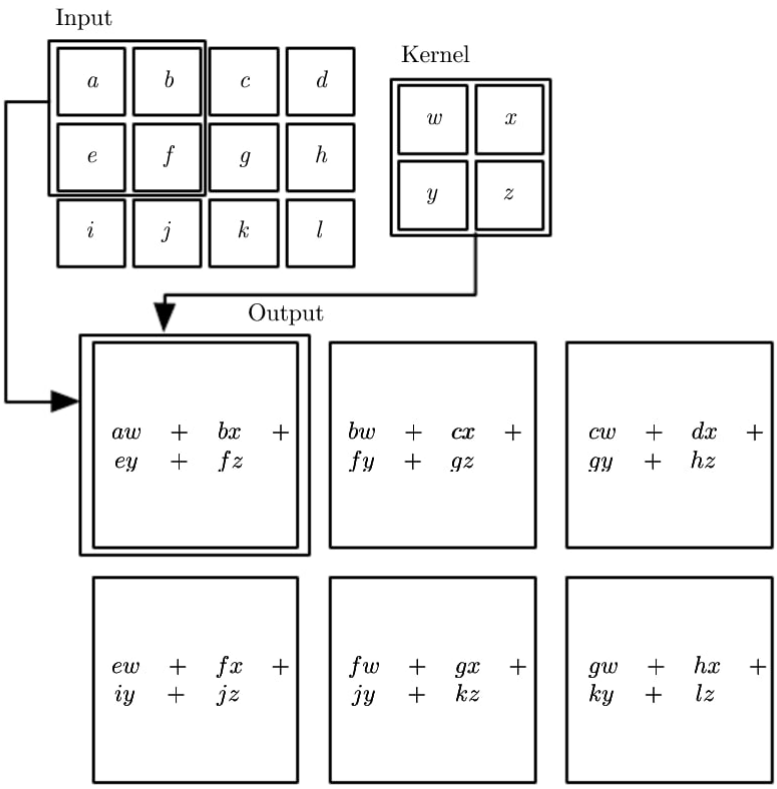

Figura 2.5: En esta imagen se puede ver un ejemplo de una convolución aplicada a una entrada de dimensiones 3×4 por un kernel de dimensiones 2×2. Se puede ver que la salida tiene tamaño 2×3. Notablemente se puede ver que al contrario que en la definición de la operación de convolución dada, aquí no se le da la vuelta al kernel, como suele ocurrir en la práctica. Imagen extraída de [GBC16].

Usualmente la entrada de las capas convolucionales tienen más de un canal (por ejemplo la capa inicial suele tener 3 canales, uno por cada color primario), en tal caso, el *kernel* tiene tantos canales como la entrada y aplica cada uno a su canal correspondiente, sumando la salida. Para un ejemplo ver la Figura 2.6. A la salida de una convolución se le llama **mapa de características**.

Por lo visto hasta ahora, el filtro se aplica posición a posición de izquierda a derecha y de arriba hacia abajo. Se denomina **stride** al parámetro que determina cada cuantas

posiciones se aplica el filtro. Por ejemplo un stride de 1 aplica el filtro de forma normal. Otro parámetro que se utiliza es el **padding** que indica si se debe expandir la imagen por los bordes y con qué valores expandirla en caso de hacerlo. Por ejemplo, en la figura 2.5 no se utiliza *padding* y por lo tanto la salida tiene una fila y una columna menos.

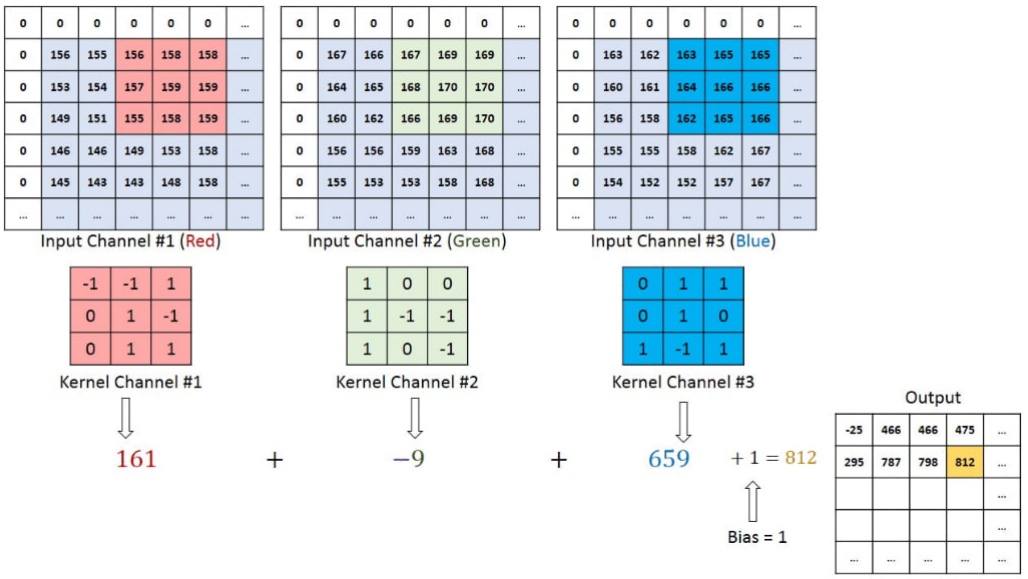

Figura 2.6: Aquí se puede ver un ejemplo de una convolución aplicada a una entrada con 3 canales, uno por cada color primario. Se ve cómo el kernel ahora tiene 3 canales también. Imagen extraída de https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/.

Ahora que ya está claro qué es una convolución, ya se puede definir una capa convolucional como una capa que aplica $n \in \mathbb{N}$ convoluciones a una entrada. Al número de convoluciones se le denomina **profundidad** y determina el número de canales que tendrá la salida de la capa. Las dimensiones espaciales de la salida dependerán de las dimensiones espaciales de la entrada y a las decisiones de *padding* y *stride* que se hagan. Además se aplica una función de activación, posición a posición, a la salida de la capa. Esta función habitualmente es la ReLU por las razones que ya se dieron en la subsección de funciones de activación.

El objetivo de las capas convolucionales es la detección de características locales de la entrada de la capa anterior ya que cada convolución aplica la misma operación

localmente a lo largo de toda la entrada. Además, este tipo de capas tienen un número reducido de parámetros a aprender por la red ya que sólo hace falta aprender los pesos de los filtros.

Capas de pooling

Las capas de *pooling* se encargan de reducir las dimensiones espaciales de la entrada manteniendo el número de canales. Esto es, disminuyen el número de filas y columnas. Para ello dividen cada mapa de características en ventanas disjuntas que cubren todas las filas y columnas, y "*resumen*" todos los valores en una ventana a uno sólo. Esto disminuye el coste computacional de la red. Este tipo de capas también busca juntar valores que son similares en uno sólo.

Los dos tipo típicos de *pooling* son el **max pooling** que toma el máximo de los valores de la ventana y el **average pooling** que toma la media aritmética de los valores de la ventana. Se cree que *max pooling* tiene mejor rendimiento y, de hecho, en la parte matemática de este TFG se pueden ver argumentos de por qué podría ser cierto. En la Figura 2.7 se puede ver un ejemplo de ambos tipos de *pooling*.

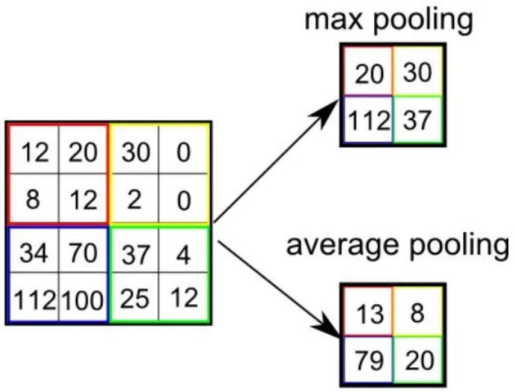

Figura 2.7: En la figura se ver cómo se aplica *max pooling* y *average pooling* a una entrada. Como las ventanas tienen tamaño 2×2 se dice que se aplica *pooling* 2×2. Imagen extraída de https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/.

Capas totalmente conectadas

Este tipo de capas no son más que una capa típica de MLP que conecta todos los valores de las neuronas de la capa anterior con todas las neuronas de esta capa. Se colocan después de todas las capas convolucionales de *pooling*. Por ejemplo, si se quisiera resolver un problema de clasificación de imágenes en cinco clases, se tendría una última capa totalmente conectada con cinco neuronas.

2.3. XAI

Los modelos de ML se pueden dividir en dos tipos: los **modelos de caja negra** (*black-box models*) y **modelos de caja blanca** (*white-box models*) [AAES+23, VL21].

Los modelos de caja blanca producen resultados que pueden ser interpretados por los usuarios o expertos que utilizan el modelo, normalmente mirando sus parámetros. Los modelos primitivos de ML, como la regresión lineal, regresión logística o árboles de decisión, suelen ser de este tipo.

Por otro lado los modelos de caja negra son modelos difíciles de interpretar y explicar debido a que tiene estructuras complejas, con una gran cantidad de parámetros y no lineales. Los modelos más importantes pertenecientes a esta clase son las ANNs.

Algunos trabajos incluyen una categoría conocida como **modelos de caja gris** (*gray-box models*), los cuales son modelos que se pueden interpretar, al menos parcialmente, si se diseñan adecuadamente.

La **Inteligencia Artificial eXplicable** (*eXplainable Artificial Inteligence, XAI*) [LBC+24, AAES+23, VL21, BDRD+20] es un campo de estudio que pretende desarrollar métodos para construir modelos de ML que aumenten la interpretabilidad de estos sin que esto afecte demasiado a su rendimiento. Para ello hay caminos:

- Construir modelos de caja blanca o gris con rendimientos altos.
- Dotar a los modelos de caja negra de un mayor nivel de interpretabilidad.

En XAI, la **Interpretabilidad** se refiere a la capacidad de comprender cómo los modelos de IA toman sus decisiones. Por otro lado la **explicabiliad** se refiere a la capacidad de hacer interpretaciones y describir el funcionamiento interno de un sistema de IA en términos humanos. Estos dos son los dos principales criterios a la hora de evaluar métodos de XAI, además de ellos existen otros como **transparencia**, **justicia** (*fairness*)

y **robustez**.

En general el objetivo de la explicabilidad es hacer algún aspecto del modelo entendible para el usuario. Si lo que se intenta hacer expllicable es una salida del modelo para una instancia, se habla de **explicabilidad local**. En cambio si es el dataset entero se habla de **explicabilidad global**.

Una distinción en los métodos de XAI es si el método es ***ante-hoc*** o ***post-hoc***. Los métodos *ante-hoc* son aquellos que entrenan un método explicable en primer lugar. Por otro lado los *post-hoc* pretenden explicar un modelo de caja negra después de ser explicado. A su vez, los métodos *post-hoc* se dividen en métodos **agnósticos al modelo** si funcionan para cualquier modelo, y métodos **específicos al modelo** si sólo funcionan para un tipo (o clase) de modelos.

Por último está la siguiente distinción de métodos según la metodología que utilicen: **basados en retropropagación** y **basados en perturbaciones**. Los que se basan en retropropagación propagan una señal de la salida del modelo hacia la entrada. Los basados en perturbaciones modifican la entrada del modelo utilizando distintas técnicas para investigar el impacto de estas modificaciones en la salida del modelo.

2.3.1. Explicaciones lineales locales y LIME

Las **Explicaciones lineales locales** (*Linear Local Explanation, LLE*) [SGLH23a, RSG16] son una clase de explicaciones para modelos de caja negra.

Sea $X \subset \mathbb{R}^F$ el datatset de entrada, siendo F el número de características por ejemplo, y sea $f : \mathbb{R}^F \to \mathbb{R}^C$ la función que implementa un modelo de caja negra, dónde C es el número de clases de un problema de clasificación al que se aplica. Sea $x \in X$ una entrada a ser explicada. Un explicador LLE de caja blanca es una función $g : \mathbb{R}^F \to \mathbb{R}^C$ definida como:

$$g(x) = Ax + B, \quad A \in \mathcal{M}_{F,C}, B \in \mathbb{R}^C$$

Dónde $\mathcal{M}_{F,C}$ es el conjunto de todas las matrices con F filas y C columnas. Es decir, g no es más que una función lineal que lleva una entrada del modelo al espacio de salida del modelo. Este modelo se puede interpretar pues intuitivamente para cada peso $a_{i,j}$ de A denota la importancia que tiene la característica de entrada i con la predicción de la salida j. Por otro lado cada valor b_j del vector B está conectado con la importancia en general de la salida j.

Para obtener la función g anterior los métodos LLE utilizan regresión lineal sobre una vecindad del ejemplo x. El error que minimizan es el siguiente:

$$\mathcal{L}(f, g, \pi_x) = \sum_{z \in N(x)} \pi_x(z)(f(z) - g(z))^2$$

Nótese que esto no es más que el error cuadrático medio ponderando cada vecino por una función π_x que mide la proximidad a x para cada vecino generado z, que dependerá del LLE utilizado. Aquí $N(x)$ representa el vecindario de x para el número de vecinos generados es un parámetro del LLE.

El método de **explicaciones locales interpretables agnósticas al modelo** (*Local Interpretable Model-agnostic Explanations, LIME*) [SGLH23a, RSG16] es una técnica LLE considerada como estado del arte. En LIME se realiza una regresión de cresta (*Ridge regresion*) para obtener g, definiendo la $\pi_x(z)$ como sigue:

$$\pi_x(z) = e^{\frac{-d(x,y)^2}{\sigma^2}}$$

dónde $d(\cdot, \cdot)$ es la distancia euclídea (en el caso de imágenes) y σ es un factor de regularización.

Para la generación de vecinos en $N(x)$ se muestrea un valor v' de la distribución exponencial $Exp(\lambda)$ con $\lambda = \frac{1}{\sigma}$. Después se toma $v = \min\{\lfloor v' \rfloor, F\}$. El valor v determina cuantas características de x se van a ocluir. Estas se escogen de forma aleatoria (uniformemente).

La elección de lo que es una característica depende del tipo de dato con el que se trate. En el caso de este TFM se trata con imágenes. Para las imágenes se puede tratar cada pixel individualmente como una característica. Sin embargo esto hace que haya muchas características lo que hace que generar este tipo de explicaciones sea caro. Además desde un punto de vista humano un pixel individual no suele tener un significado específico en una imagen, es más natural que un conjunto de pixeles tengan un significado. Por ello consideramos una división de la imagen en cuadrados del mismo tamaño y cada uno de ello conlleva una característica.

2.3.2. Métricas ReVEL

El framework **Evaluación robusta vectorizada de explicaciones-lineales-locales** (*Robust Evaluation VEctorized Loca-linear-explanation, REVEL*) [SGLH23a] ofrece un análisis robusto y consistente de explicaciones LLEs generadas para modelos de caja negra. En este se presentan cinco métricas que miden distintos aspectos cualitativos de la explicación.

Antes de introducir estás métricas introduzco algunas matrices antes. Nos situamos en un problema de clasificación en el que hemos entrenado un modelo de caja negra, sea g una explicación generada por un LLE, tal que $g : F \rightarrow Y_l$ dónde Y_l es el espacio *logit*. Este espacio no es más que la salida del modelo de caja negra antes de aplicar la función *softmax* que transforma la salida del modelo a probabilidades de cada clase. Recordar que $g(x) = Ax + B$.

Se define la **matriz de importancia con signo**, A^l como la matriz derivada sobre el espacio *logit*. Como es evidente al ser lineal $A^l = A$. Por otro lado, para la obtención del vector de probabilidades p se necesita aplicar la función *softmax* a $g(x)$ luego $p = softmax(g(x)) = softmax(Ax + B)$. Se define la matriz $A^p = D(softmax(g(x)))(x)$ siendo $D(\cdot)$ el operador derivada. La función $softmax$ es la función que para clase c se define como:

$$softmax(y_1, y_2, ..., y_C)_c = \frac{e^{y_c}}{\sum_{i=1}^{C} e^{y_i}}$$

La forma de interpretar estas dos matrices es teniendo en cuenta que el elemento en la fila i y columna j de cada matriz representa la importancia de la característica i sobre la clase j sobre los espacios *logit* y *probabilidad*. Por un lado A^l da información sobre como afecta, positiva o negativamente dependiendo del signo, cada característica a los distintos *logits* de las clases. Por otro lado, A^P proporciona información sobre las clases en las que el ejemplo es más probable de ser clasificado.

Como ambas matrices presentan información relevante se define la **matriz de importancia** como la matriz que combina la información de A^l y A^p como sigue:

$$A_{i,j} = sign(A^l_{i,j}) \sqrt{|A^l_{i,j}| \cdot |A^p_{i,j}|}$$

A partir de esta matriz se define la **matriz de importancia normalizada** A_r =

$\frac{A}{max_{A_{i,j}}\{|A_{i,j}|\}}$ que lleva los valores de A al rango $[-1, 1]$ manteniendo su signo. Finalmente se define la **matriz de importancia absoluta** como la matriz $|A|$ que simplemente es la matriz A_r con todos sus elementos en positivo.

Leyendo tu artículo no me queda claro si la matriz de importancia absoluta es $|A_z|$ o $|A|$

Ya estamos en condiciones de introducir las cinco métricas introducidas en REVEL. De aquí en adelante $g(x)$ se toma en el espacio de probabilidades, no en el de *logit*.

Local concordance

Se define como:

$$Local\_Concordance(g) = 1 - \frac{||f(x) - g(x)||}{C'}$$

Dónde $||\cdot||$ es una norma arbitraria (por ejemplo la norma euclídiana $||\cdot||_2$) y C' es la distancia máxima posible entre dos vectores de probabilidad (por ejemplo en el caso de $||\cdot||_2$ se cumple que $C' = \sqrt{2}$).

Notemos que esta métrica está definida en el intervalo $[0, 1]$ dónde el valor máximo se alcanza cuando $f(x) = g(x)$ y la mínima ocurre cuando ambas funciones clasifican x en clases distintas con probabilidad 1.

Esta métrica mide cómo de similar es la explicación al ejemplo original de la caja negra. Es importante que esté cercana a 1 ya que en otro caso la explicación propuesta no se corresponde con el ejemplo escogido.

Local fidelity

Se define como:

$$Local\_Fidelity(g) = \frac{1}{|N(x)|} \sum_{z \in N(x)} 1 - \frac{||f(z) - g(z)||}{C'}$$

Dónde $|N(x)|$ es el número de vecinos generados para la explicación.

Esta métrica se puede ver como la media de la métrica anterior sobre los vecinos de x, por lo tanto también está definida en el intervalo $[0, 1]$. Esta métrica mide cómo de cercan están las probabilidades calculadas por el modelo de caja blanca de las que calcula el modelo de caja negra. Luego mide la similitud entre la explicación y el modelo f en la vecindad de x. Al igual que pasaba con la *local concordance*, debe ser cercano a 1 para obtener buenas explicaciones.

Prescriptivity

Se define como:

$$Presciptivity(g) = 1 - \frac{||f(x+h) - g(x+h)||}{C'}$$

Dónde h es una perturbación sobre x que quita la presencia de las características más importantes (de forma positiva) de la clase asignada por g al ejemplo x. De esta forma, para encontrar h se van quitando dichas características hasta que se cumpla que $argmax(g(x)) \neq argmax(g(x+h))$.

Notemos que esta métrica también está contenida en $[0, 1]$ y obtiene el valor 1 cuando $f(x+h) = g(x+h)$ y obtiene el valor 0 cuando ambas funciones clasifican $x+h$ en clases distintas con probabilidad 1.

Esta métrica pretende estudiar si la explicación g es capaz de predecir correctamente los cambios necesarios en el ejemplo x para cambiar la clase original. Para ello busca un valor lo suficientemente alejado del ejemplo para que la explicación cambie la clase predicha y mida cómo cambia la predicción respecto del modelo de caja negra. Al contrario que las dos métricas anteriores un valor cercano a 1, aunque buscado, no es necesario para obtener buenas explicaciones.

Conciseness

Se define como:

Aquí he adaptado la fórmula de https://arxiv.org/abs/2211.06154 con la información de la documentación de tu código en github para que quede más claro (a mi me queda más claro).

$$Conciseness(g) = \frac{1}{f' - 1} \sum_{i=1}^{f'} 1 - I_i$$

Dónde f' es el número de filas de la matriz de importancia absoluta (equivalentemente, número de clases), sea v_i el vector fila i de la matriz $|A|$ entonces:

$$I_i = \frac{||v_i||}{max_{1 \leqslant j \leqslant f'}\{||v_j||\}}$$

Observemos que v_i es un vector en que cada elemento j representa la importancia de la característica i para la clase j luego se puede interpretar que I_i es la importancia de dicha característica (normalizada a $[0, 1]$). Si analizamos la expresión de la métrica tenemos que esta está comprendida entre $[0, 1]$ y alcanza 1 si una métrica tiene importancia 1 y el resto tiene importancia 0. Si todas las características tuvieran la misma importancia, por estar I normalizada serían todas 1 y se obtendría el menor valor posible en 0.

Esta métrica busca medir la brevedad de la explicación, es decir, cuantas menos características sean relevantes para la explicación, más concisa se considera la explicación. Luego mide la habilidad de la explicación de concentrarse en las características importantes. Al contrario que las métricas anteriores, dependiendo de la tarea se podría querer más o menos *conciseness* ya que valores demasiados bajos podrían indicar que el modelo se centra en cosas irrelevantes (por ejemplo en imágenes que se centre en pocos píxeles podría ser malo).

Robustness over explanations

Se define como:

$$Robustness(G) = \mathbb{E}[similarity(g, g')]$$

Dónde G representa el conjunto de todas las explicaciones que podría generar el LLE, $\mathbb{E}$ es la esperanza y *similiarity* es una función que mide cuanto difieren dos explicaciones distintas y se define como:

$$similarity(g, g') = \left(\frac{A_r \cdot A'_r}{||A_r||||A'_r||}\right)\left(1 - \frac{|\ \ ||A_r|| - ||A'_r||\ \ |}{max\{||A_r||, ||A'_r||\}}\right)$$

Observemos que la componente de la derecha mide cómo de similares son las magnitudes de las las matrices de importancia absoluta normalizadas de las explicaciones mientras que el valor de la izquierda se corresponde con la similaridad coseno entre A_r y A'_r (matrices correspondientes con g y g' correspondientemente), esta mide como distan las direcciones de las explicaciones. El producto de matrices de esta función es el elemento a elemento. El mejor caso, cuando ambas explicaciones son iguales, se tiene que la similaridad es 1 y el peor caso ocurre son "perpendiculares" y la similaridad es 0.

En la práctica, para aproximar la *Robustness* se hace la media de la similaridad generando muchas otra explicaciones g' para x. Esto únicamente se puede hacer para LLEs que no son deterministas, pues en otro caso siempre se tendría la misma explicación y obtendrían siempre 1. Por ejemplo con LIME sí se puede calcular esta métrica pues tiene no determinismo en la generación de la vecindad.

Esta no es una métrica sobre la explicación en específico sino sobre el método que las genera. Cuanta más cercana a 1 menos varían las explicaciones generadas por el método estudiado. Esta métrica al contrario que todas las anteriores puede obtener valores negativos, lo que indicaría explicaciones que se "contradicen".

2.3.3. Regularización X-Shield

La **regularización** [SGLH23b, MBM20] es la modificación de la arquitectura del modelo, del proceso de aprendizaje o inferencia para imponer algún criterio sobre este. Por ejemplo, dropout, que ya se vió previamente, es una regularización.

En particular nos interesan las regularizaciones que implican añadir a la función de coste un nuevo término que indica el criterio que queremos minimizar en el modelo. Algunos ejemplos típicos son la regularización L1 o L2. En general se tiene que una regularización de este tipo se expresa como:

$$coste\_regularizado(\Theta, X, Y) = coste(f_\Theta(X), Y) + regularizacion(X, \Theta)$$

Dónde se tiene que Θ son el conjunto de parámetros del modelo, X son las entradas del conjunto de datos y Y las salidas. Por lo tanto la regularización es una función

que puede ser de los parámetros y de las datos de entrada.

En primer lugar presentamos la familia de regularizaciones ***Transformation-Selective Hidden Input Evaluation for Learning* ,T-SHIELD** [SGLH23b]. Esta esta diseñada para hacer que el modelo aprenda con menos features. Para ello se ocluye parte de la entrada.

Para ello se utiliza una transformación $T : \mathbb{R}^F \to \mathbb{R}^F$ tal que $T(x)_i = x_0$ si i es una característica a esconder y $T(x)_i = x_i$ si no lo es. Es decir está transformación cambia todas las características de una entrada x por un valor constante x_0 y deja el resto intactos.

Para una transformación T como la anterior descrita, se define la regularización T-SHIELD como sigue:

$$T - SHIELD(x, \Theta) = KL(f_\Theta(T(x)), f_\Theta(x)) + KL(f_\Theta(x), f_\Theta(T(x)))$$

Dónde $KL(\cdot, \cdot)$ denota la distancia de Kullback-Leibler, que se trata de un tipo de distancia que mide como difieren dos distribuciones distintas. Por lo tanto esta regularización no es más que la distancia de Kullback-Leibler simétrica. Según como se construya la transformación T esta regularización penalizará ciertos aspectos del modelo. En la Figura 2.8 se puede ver cómo funcionaría una regularización de esta familia de forma general.

Se denomina **Random-SHIELD (R-SHIELD)** a la regularización de la familia T-SHIELD que para un parámetro $\lambda \in [0, 100]$ define la transformación T como aquella que esconde un λ % de las características de forma aleatoria (misma probabilidad para cada una).

Se denomina **XAI-SHIELD (X-SHIELD)** a otra regularización de la familia T-SHIELD. Comparte el parámetro λ de R-SHIELD pero en vez de ocultar el λ % de características aleatorias, va a ocultar las λ % menos importantes. Para ello, a cada característica le otorga la importancia $||v_i||$ donde v_i es la fila i de la matriz de importancia A introducida en la subsección 2.3.2, pero para el modelo que se está entrenando.

De manera más formal se describe $T_{xai}(x; \lambda)$ como la transformación:

$$T_{xai}(x_1, ..., x_F; \lambda) = (\underbrace{x_0, x_0 ..., x_0, x_0}_{0 \leqslant i \leqslant F: \quad \lambda < \frac{i}{n} \cdot 100\,\%}, x_i, x_{i+1}, ..., x_F)$$

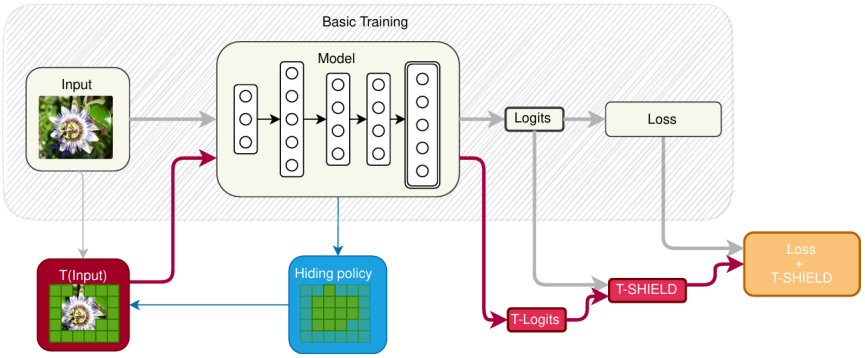

Figura 2.8: En esta figura se puede ver cómo funciona en general la familia de regularizaciones T-SHIELD durante el entrenamiento. En gris se puede ver el flujo normal de la imagen en el modelo. En azul se encontraría la forma en la que la regularización decide qué características ocluir. Finalmente en rojo se encontraría el flujo de la imagen con las características ya ocluidas y cómo se calcula la función de regularización. Imagen extraída de [SGLH23a].

Dónde se ha supuesto los atributos x_i ordenados de menor a mayor por importancia y tenemos que i es el menor valor tal que se ocluyen al menos el λ % de características.

En la Figura 2.9 se puede apreciar la diferencia entre los comportamientos de los dos tipos de regularización T-SHIELD vistos.

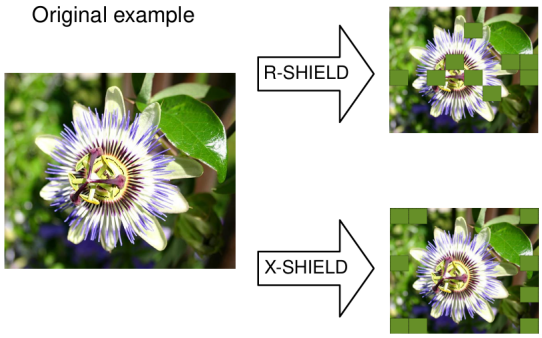

Figura 2.9: Comparación entre el comportamiento de T en R-SHIELD y X-SHIELD. La imagen está sacada del dataset *Flowers* de clasificación de imágenes de flores. Se puede ver que R-SHIELD ocluye partes de la imagen aleatorias, incluyendo partes de la flor a clasificar. Por otro lado X-SHIELD ocluye partes del fondo de la imagen que intuitivamente no parecen útiles para clasificar dicha flor. Imagen extraída de [SGLH23a].

3 Estado del arte

Este Trabajo de Fin de Máster trata de resolver el problema de construir un modelo de IA de clasificación para lesiones cancerosas en biposias de próstata y de la mejora en interpretabilidad del modelo mediante técnicas de XAI. Por lo tanto vamos a hacer una revisión del estado del arte para los cuatro siguientes puntos que son relevantes:

- Redes convolucionales para clasificación de imágenes.
- Métodos de XAI para producción de explicaciones, en particular, LLEs.
- Métodos de XAI de regularización para mejoras las explicaciones.
- Métricas de XAI de calidad de las explicaciones producidas para poder comparar los distintos métodos utilizados.

Modelo de clasificación

Efficientnet [TL19] es una red introducida en 2019 que se sigue considerando estado del arte en clasificación de imágenes cuando se mide eficiencia computacional, es decir, rendimiento obtenido número de parámetros. En la Figura 3.1 se puede apreciar como es más eficiente en cómputo respecto otros modelos de clasificación de imágenes ResNet [HZRS15] o DenseNet [HLvdMW18] para el dataset Imagenet.

Existen otros modelos más modernos como CoCa [YWV+22], DaViT [DXC+22] o ConvNeXt V2 [WDH+23], que utilizan transformadores en el caso de Coca y DaViT, o autoencoders en el caso de ConvNeXt V2. Estos modelos alcanzan precisiones mayores que EfficientNet sin embargo aumentan demasiado el número de parámetros utilizado para que sea una propuesta viable para nuestro trabajo.

Por ello hemos decidido utilizar el modelo EfficientNet en este trabajo ya que ofrece un balance muy bueno entre número de parámetros y precisión. En la Tabla 3.1 se puede apreciar las diferencias entre el modelo que utilizaremos EfficientNet-B2 y las versiones más pequeñas de los modelos mentados. Se aprecia que EfficientNet es más eficiente en el número de parámetros.

Modelo	Arquitectura Base	Parámetros (M)	Precisión Top-1 (%)
EfficientNet-B2	CNN	9.1	80.1
DaViT-Small	CNN + Transformer	49.7	84.2
ConvNeXt V2-Tiny	CNN moderna (con Masked Autoencoder)	28.6	85.1
CoCa-Base	Transformer	383	91.0

Tabla 3.1: Comparación de modelos en términos de precisión Top-1 en ImageNet y número de parámetros.

Métodos de XAI para producción de explicaciones

Una revisión de todos los métodos de explicabilidad *post-hoc*, que son los que se aplican a modelos de caja negra, está fuera de el alcance de este trabajo pues sería inabarcable. Sin embargo resumimos algunos métodos populares (esquema en la Tabla 3.2):

- **LIME** [RSG16]: Modelo explicado con detalle en la subsección 2.3.1.
- **SHAP** (*SHapley Additive exPlanation*) [LL17]: Basado en valores de Shapley de teoría de juegos. Atribuye importancia a cada variable de forma consistente; permite explicaciones locales y globales.
- **LRP** (*Layer-wise Relevance Propagation*) [MBL+19]: Propaga hacia atrás la relevancia de la predicción a través de cada capa de la red. Muy usado en visión y señales neuronales.
- **CRP** (*Concept Relevance Propagation*) [ADE+23]: Extiende LRP incorporando información contextual o conceptual. Promete explicaciones más interpretables en términos humanos.
- **Grad-CAM** (*Gradient-weighted Class Activation Mapping*) [SCD+17]: Técnica visual para CNNs que genera mapas de calor mostrando qué regiones de una imagen influyen en la clasificación.
- **Integrated Gradients** [STY17]: Método axiomatizado de atribución que compara la entrada con una baseline. Proporciona asignaciones más estables que gradientes simples.

Técnica	Tipo	Enfoque
LIME	Modelo-agnóstico	Local
SHAP	Modelo-agnóstico	Local y Global

Técnica	Tipo	Enfoque
LRP	Modelo-específico (DNN)	Local
CRP / Concept R.P.	Modelo-específico (DNN)	Local
Grad-CAM (+ variantes)	Modelo-específico (CNNs)	Visual
Integrated Gradients	Modelo-específico (DNN)	Local (atribución)

Tabla 3.2: En esta tabla se muestran algunas técnicas clave de XAI.

En este estudio nos hemos dejado muchos métodos como explicaciones por ejemplos (como generación de *counterfactuals* [LKLH19]) o las muchas extensiones de los métodos mentados (por ejemplo para LIME en [KMSB25] hay una taxonomía de estos).

Métricas para explicaciones

Debido a que hay muchos tipos de explicaciones en la literatura (LLEs, mapas de atribución, reglas...) no hay manera de comparar estas directamente. Debido a esto, se existen distintas métricas para medir de forma cuantitativa la calidad de las explicaciones.

En [Ros21] se proponen cuatro métricas para medir la calidad de explicaciones. Sin embargo estas son específicas para explicaciones basadas en reglas. Las métricas que representan son:

- D: la diferencia en rendimiento entre el modelo a explicar y la explicación basada en reglas. Cuantifica el cambio entre el modelo de caja negra y el mejor modelo de caja blanca observado.
- R: El número de reglas de la explicación. Busca cuantificar lo simple que es la explicación.
- F: El número de características que se han utilizado para construir la explicación. Es similar a R, cuanto menor es mejor se considera la explicación.
- S: La estabilidad de la explicación. Cuantifica la habilidad de la explicación de

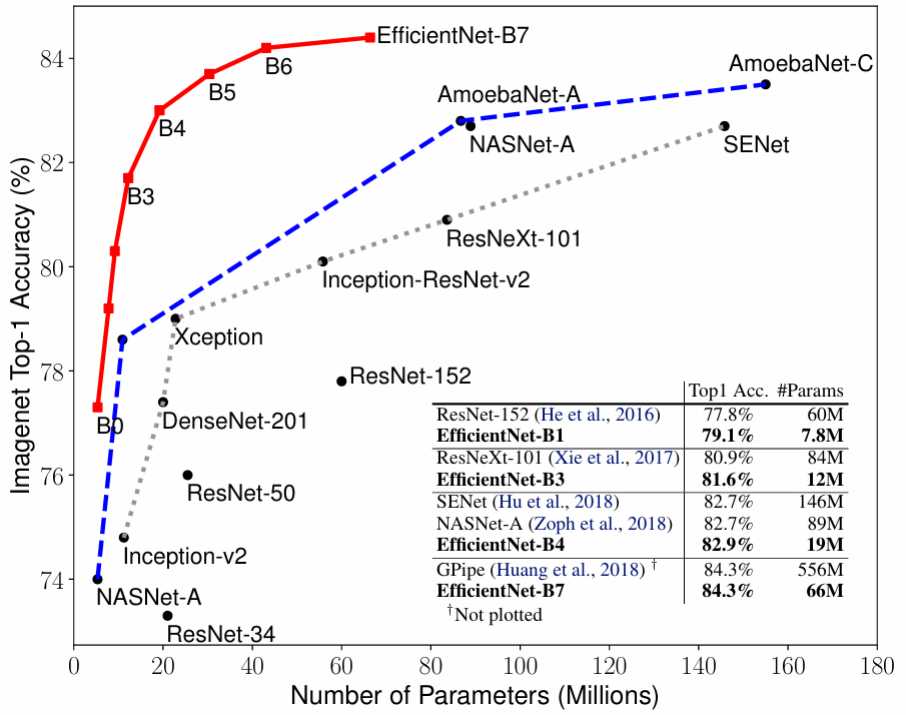

Figura 3.1: En esta gráfica se compara las distintas versiones de Efficientnet, B0-7, contra otros modelos de redes convolucionales. Se puede apreciar que es la más eficiente. Imagen extraída de [TL19].

manejar robustamente perturbaciones pequeñas.

En [APB21] se propone el framework *LEAF*. En este se introducen 5 métricas para evaluar de forma clara las explicaciones de LLEs. Las métricas son las siguientes:

- *Conciseness:* se interpreta como hasta qué punto el modelo es entendible por un humano. Se define como el número de pesos de la explicación (es una LLE) son no nulos.
- *Local Fidelity:* mide el modelo de caja blanca g (la explicación) aproxima el comportamiento del modelo de caja negra f para un ejemplo x y su vecindad $N(x)$. Este se calcula con el *F1-score* sobre dicha vecindad y dependerá por lo tanto de cómo se muestree esta.
- *Local Concordance:* mide como g aproxima a f justo en el ejemplo x. Se mide como $max\{0, 1 - |f(x) - g(x)|\}$.

- *Reiteration Similarity:* muchos métodos de LLEs utilizan aleatoriedad para generar sus explicaciones. Debido a esto aplicar muchas veces un algoritmo para generar una explicación para un mismo ejemplo x puede generar explicaciones distintas cada vez. Se propone esta métrica para medir la similaridad del conjunto de explicaciones generadas. Esta se calcula como la similaridad de *Jaccard* J esperada entre dos explicaciones g y g'. Es decir $\mathbb{E}[J(g, g')]$.

- *Prescriptivity:* mide en qué medida una explicación local puede usarse para modificar una instancia y cambiar su clasificación. Se calcula proyectando x sobre la frontera de decisión del modelo explicativo g, y evaluando qué tan cerca está esa proyección del límite. Sea x' dicha proyección entonces se calcula como la pérdida *hinge* normalizada de $f(x') - g(x')$.

Finalmente, en [SGLH23a] se introducen las métricas REVEL, que ya hemos introducido en 2.3.2. Este framework propone cinco métricas al igual que LEAF. REVEL está compuesto de las mismas 5 métricas, cambiando *Reiteration Similarity* por *Robustness* que busca medir lo mismo. Sin embargo REVEL cambia cómo se calcula cada una de las métricas introduciendo mejoras en todas ellas. Es por esto por lo vamos a utilizar en este trabajo estas métricas para medir la calidad de nuestras explicaciones.

¿Debería ir una por una explicando las mejoras de las métricas REVEL respecto a LEAF?

Métodos de regularización XAI

Los métodos de regularización en XAI [SGLH23b, WLBS23], que añaden un término regularizador a la función de pérdida, plantean una manera sencilla para modificar el comportamiento del modelo, favoreciendo criterios buscados. Estos son por construcción agnósticos al modelo, aunque algunos requieren que este sea diferenciable, y sencillos de implementar. Por ello son uno de los métodos más estudiados en la técnicas de XAI.

En [RHDV17] se introduce *Right for the Right Reasons* (RRR) que busca optimizar el razonamiento del modelo. En este trabajo el conjunto de datos X, a parte es estar etiquetado, debe proveer una máscara en la que se indique si cada dimensión de la entrada debe ser relevante o irrelevante para la decisión. En este trabajo las explicaciones se hacen mediante el gradiente respecto el logaritmo de la salida del modelo.

En [RDV18] se realiza un método similar al anterior, pero solucionan el problema

de la anotación manual de los datos para determinar qué parte de las entradas son irrelevantes. En este se generan perturbaciones de la entrada y se calcula sus mapas de atribución. El componente regularizador penaliza cambios significativos en las explicaciones. Este método busca que las explicaciones sean más estables y robustas.

En [RSMY20] se introduce *Contextual Decomposition Explanation Penalization* (CDEP). Este utiliza la diferencia absoluta entre la explicación producida y la explicación dada por un experto. Para ello, se emplea el método *Contextual Decomposition* (CD) como técnica de atribución, que permite calcular explicaciones diferenciables durante el entrenamiento. De esta forma, se penaliza directamente en la función de pérdida cualquier discrepancia entre el razonamiento del modelo y el conocimiento previo.

Finalmente, en [SGLH23b] se introduce X-SHIELD el método del que partimos. Este método no tiene restricciones teóricas sobre la entrada, salida o tarea que realiza el modelo, sólo que el modelo sea diferenciable. Está probado que este mejora el rendimiento en las métricas REVEL, que son las utilizadas en este trabajo, lo que nos proporcionará un método base con el que comparar nuestras propuestas. XSHIELD también ha demostrado que es capaz de aumentar no sólo las métricas REVEL sino que también mejora en muchos casos el rendimiento en clasificación del modelo.

4 Métodos

4.1. EfficientNet

El modelo de clasificación de imágenes que vamos a entrenar sobre nuestro conjunto de datos y sobre el que aplicaremos las distintas regularizaciones XAI se trata de **EfficientNet** [TL19]. EfficientNet es una familia de CNNs muy eficiente en rendimiento, obteniendo buenos resultados en accuracy versus número de parámetros presentes en la red (Figura 3.1). Esto hace que sea muy versátil y se adapte al poder de computo del que disponemos. Además la precisión final obtenida por la red, aunque es relevante, no es el objetivo central que ocupa este trabajo ya que también buscamos mejorar la interpretabilidad del modelo.

De entre la familia de estas redes hemos elegido **EfficientNet-B2** para realizar nuestros experimentos pues es el que mejor se adapta a nuestra potencia de cómputo (en la Tabla 4.1 se pueden ver los detalles).

Modelo	**Parámetros (M)**	**FLOPs (B)**
EfficientNet-B0	5.3	0.39
EfficientNet-B1	7.8	0.7
EfficientNet-B2	9.2	1
EfficientNet-B3	12	1.8
EfficientNet-B4	19	4.2
EfficientNet-B5	30	9.9
EfficientNet-B6	43	19
EfficientNet-B7	66	37

Tabla 4.1: Modelos EfficientNet B0–B7 con número de parámetros y FLOPs (*Floating Point OPerations per second*). Datos extraídos de [TL19]

En la Tabla 4.2 muestro la arquitectura de EfficientNet-B2. Los bloques **MBConv1** y **MBconv6** son los bloques *inverted bottleneck* introducidos en [SHZ+18] combinados con bloques *Squeeze-and-excitation (SE)* introducidos en [HSA+20].

Estapa	Operador	Resolución	#Canales de entrada	#Capas
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	48	2
5	MBConv6, k3x3	28 × 28	88	3
6	MBConv6, k5x5	14 × 14	120	3
7	MBConv6, k5x5	14 × 14	208	4
8	MBConv6, k3x3	7 × 7	352	1
9	Conv1x1 + Pooling + FC	7 × 7	1408	1

Tabla 4.2: Arquitectura de EfficientNet-B2 con entrada de 224 × 224.

4.2. XSHIELD

X-SHIELD [SGLH23b] es el método de regularización XAI con el que vamos a comparar los métodos propuestos a continuación. Además nuestros métodos están basados en los mismos principios que este método, esconder al modelo las características menos relevantes y penalizar la diferencia entre la predicción original del modelo y la predicción sin dichas características. Todos los detalles acera de XSHIELD están en la subsección 2.3.3.

4.3. Mis métodos propuestos

En las siguientes tres subsecciones voy a introducir mis propuestas de métodos de regularización.

4.3.1. FXShield

En **FX-SHIELD** (*Feature X-SHIELD*) proponemos una regularización que ocluye la distintas características extraídas por la parte de red convolucional del modelo. Sea $x \in \mathbb{R}^F$ un dato de entrada, luego F es el número de características de la entrada. Sea $f : \mathbb{R}^F \to \mathbb{R}^C$ la función que implementa el modelo de caja negra siendo C el número de clases del problema. En el caso de una red convolucional se tiene que $f(x) = l(c(x))$ dónde $c : \mathbb{R}^F \to \mathbb{R}^{F_c}$ es la función que implmentan las capas convolucionales de la red y $l : \mathbb{R}^{F_c} \to \mathbb{R}^C$ la función que implementan las capas no convolucionales (por ejemplo una o varias capas totalmente conectadas). En las expresiones anteriores F_c representa el número de características que extrae la red

convolucional de las entradas (ver Figura 4.1). Por ejemplo, en la Tabla 4.2 podemos ver que para EfficientNet-B2 se tiene que $F_c = 1408$.

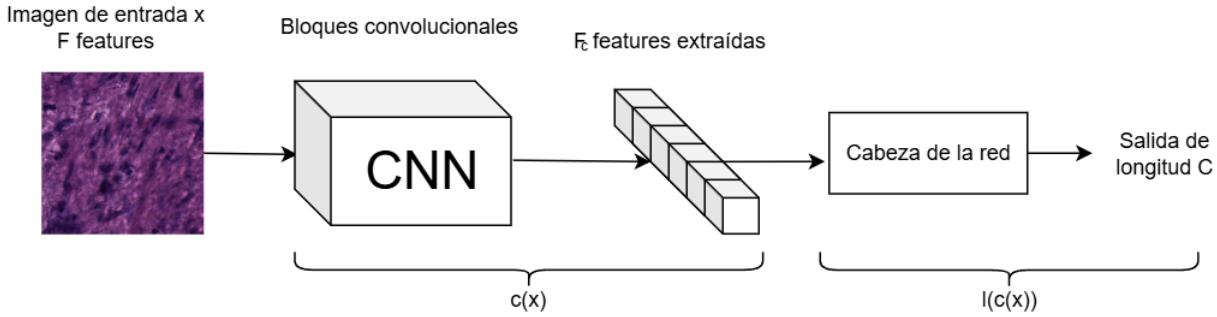

Figura 4.1: Esquema de la red completa. En este se diferencian las características extraídas por las capas convolucionales de la red de las características de la entrada.

Para X-SHIELD las características de las entradas, si son imágenes, dependen de como se quieran definir. Hay varias posibilidades como tomar que cada píxel se corresponda a una características, dividir la imagen en de entrada en cuadrados del mismo tamaño o utilizar métodos para dividir la imagen de entrada en superpíxeles. En FX-SHIELD la toma de esta decisión no es necesaria porque independientemente de la imagen de entrada (incluso para distintos tamaños de esta) siempre se va a obtener el mismo número F_c de características extraídas por los bloques convolucionales.

Notemos como $z = c(x)$ las características intermedias. Ahora buscamos ocluir las características menos importantes de z. La importancia que se le da a las distintas características se puede definir con la matriz de importancia $\mathcal{A} = \frac{\delta f}{\delta Z}(x) = \frac{\delta l}{\delta Z}(z)$. Para cada características $i \in \{1, 2, ..., F_c\}$ vamos a definir la importancia de esta como:

$$I_i(z) = \frac{||\mathcal{A}_i||_1}{C} = \frac{1}{C}\sum_{j=1}^{C} |\mathcal{A}_{i,j}|$$

Por lo tanto esta es la media aritmética de la suma absoluta de la derivada de cada valor de salida sobre dicha características. Volviendo al ejemplo de la arquitectura de EfficientNet-B2 4.2 tenemos que la cabeza de la red consta de una única capa totalmente conectada, es decir en este caso se tiene que:

$$l(z) = Az + B$$

Dónde A es una matriz real de dimensión $C \times F_c$ y B es un vector de C valores. Como l es una aplicación lineal se sigue que $\mathcal{A} = A$ y la importancia no es más que la suma para cada característica de los valores de A para esta.

Sea $\lambda \in [0, 100]$ un parámetro que determina que porcentaje de las características se va a ocultar, definimos la siguiente transformación $T_{fxshield} : \mathbb{R}^{F_c} \to \mathbb{R}^{F_c}$ que enmascara el λ % de elementos menos importantes. En la práctica se escoge el menor m tal que $\lambda < \frac{m}{F_c} \cdot 100$ y se enmascaran las m características menos importantes. En X-SHIELD, como se enmascaran características de la entrada que son imágenes, lo que se hace es sustituir estas por el color medio de la entrada. Para FX-SHIELD estamos en el espacio de features, dónde los valores que van de $-\infty$ a ∞. Por lo tanto hemos decidido sustituir dichos valores por 0. Suponiendo, sin pérdida de generalidad, que las características en z están ordenadas de menor importancia a mayor, se define $T_{fxshield}$ como:

$$z' = T_{fxshield}(z) = (0, 0, ..., 0, z_{m+1}, ..., z_{F_c})$$

Ahora podemos medir la predicción del modelo para este ejemplo modificado simplemente pasándolo por la cabeza de la red.

Ya estamos en condiciones de definir el factor de regularización que se utiliza en este método. Para un ejemplo x y unos parámetros Θ del modelo específicos, sean $y = f_\Theta(x) = l(z)$ la predicción original del modelo y $y' = l(T_{fxshield}(c_\Theta(x))) = l_\Theta(z')$ la predicción para el ejemplo con las características intermedias modificadas, entonces se tiene que:

$$FX - SHIELD(x, \Theta) = KL(y, y') + KL(y', y)$$

Dónde KL es la divergencia de Kullback-Leibler que se utiliza para medir la distancia estadística entre las distribuciones de probabilidad entre los modelos.

Es decir, FX-SHIELD utiliza este valor regularizador en el entrenamiento por lo que en este método se minimiza la siguiente función:

$$coste\_fxshield(\Theta, X, Y) = funcion\_perdida(f_\Theta(X), Y) + FX - SHIELD(X, \Theta)$$

Una ventaja que presenta este método sobre X-SHIELD es que no necesita que el modelo de caja negra sea diferenciable, aunque sigue necesitando que la cabeza del modelo lo sea (para el cálculo de la importancia de las características). A cambio, FX-SHIELD no es totalmente agnóstico al modelo, pues es necesario poder extraer unas características intermedias.

Otra ventaja es que es más rápido que X-SHIELD ya que para el cálculo de la regularización no hace falta pasar las características enmascaradas por toda la red sino por las capas posteriores a las convolucionales, que suelen ser mucho más rápidas y con menos parámetros.

En X-SHIELD se esconden las características menos relevantes de la entrada a la red, que es una imagen, con la idea de eliminar partes de imagen que no son relevantes para la tarea de clasificación. En FX-SHIELD se esconden las características menos relevantes de las extraídas por las capas convolucionales. Esto puede ser interesante pues varios trabajos anteriores como [BZK+17, KWG+17, OMS17] muestran que las capas intermedias en CNNs representan conceptos de alto nivel. Por lo tanto tiene sentido trabajar con estas características intermedias que codifican conceptos que tienen sentido para los humanos. Esto se puede ver como una ventaja de FXSHIELD sobre X-SHIELD (cuando se toman características de la imagen de entrada como una división de la imagen en cuadrados) ya que estas features intermedias pueden tener un mayor significado.

4.3.2. FRShield

FRSHIELD (*Random FX-SHIELD*) es el mismo método que FX-SHIELD pero cambia en que las características que oculta se escogen de forma aleatoria sin tener en cuenta la importancia de estas. Este método nos permite comparar FX-SHIELD con un método no informado. Además el hecho de que se oculten las características de forma aleatoria puede ayudar al modelo a no depender únicamente de las características más importantes y que las explicaciones estén apoyadas en más características. Finalmente este método se parece a *Dropout*, lo que también lleva a pensar que podría hacer que la red no sobreentrene.

4.3.3. HShield

Finalmente introducimos **H-SHIELD** (*Hybrid* X-SHIELD) un método que combina X-SHIELD con FX-SHIELD. Para ello modificamos el valor regulador para tener en cuenta ambos de la siguiente forma:

$$\begin{aligned} coste\_hshield(\Theta, X, Y) = & funcion\_perdida(f_\Theta(X), Y) + \\ & \alpha_1 \cdot X - SHIELD(X, \Theta) + \\ & \alpha_2 \cdot FX - SHIELD(X, \Theta) \end{aligned}$$

Dónde α_1, α_2 son dos parámetros positivos que utilizaremos para darle cambiar el peso que le damos a cada una de las regularizaciones por separado.

Con esta propuesta intentamos combinar las bondades de ambos métodos. El principal inconveniente de este método es su mayor coste computacional a los métodos por separado. La determinación de los parámetros α_1 y α_2 se hará empíricamente comparando los valores de regularización individuales que se obtienen para cada método por separado. El porcentaje de elementos que se ocultan por cada método, que denotabamos como λ, ahora son dos parámetros distintos λ_1 y λ_2, que serán dicho parámetro para X-SHIELD y FX-SHIELD respectivamente.

4.4. Implementación

Para la realización de este TFM se ha partido del código correspondiente al proyecto de X-SHIELD original [SGLH23b]. Este está escrito en *Python*. Se ha realizado un estudio de todo el código de el proyecto original para entender correctamente como funciona y así poder implementar los nuevos métodos que proponemos. Además se han escrito varios *scripts* en *bash* para la ejecución de los entrenamientos de los modelos aplicando los distintos métodos de regularización en los servidores de GPUs de la UGR.

A continuación hacemos un resumen de todos los módulos del proyecto que se han utilizado, sin contar los *scripts* de *bash* de , y mencionamos cuando se han modificado para el TFM:

- **SHIELD/shield.py:** este es el módulo más importante de todo. En él vienen las funciones para calcular las importancias de las características y las implementaciones de los distintos métodos. En particular se ha añadido a este archivo los métodos X-SHIELD y R-SHIELD.
- **procedures/cargar_datos.py:** este módulo es completamente nuevo y es necesario para cargar los datos del TFM en un *dataset* de *pytorch*. Este módulo se basa

en la manera en la que se cargan los *datasets* utilizados en el trabajo original de X-SHIELD, cuya implmentación se puede ver en el GitHub de las métricas REVEL.

- **procedures/procedures.py:** aquí vienen implementadas las funciones para cargar el modelo clasificador (*EfficientNet* en nuestro caso) y las funciones que realizan un paso en entrenamiento y validación.
- **tests/test.py y tests/train.py:** scripts para el entrenamiento de la red con X-SHIELD y para obtención de *accuracy* en test, respectivamente.
- **tests/mitrain.py, tests/mitest.py, tests/mitrain_hibrido.py:** los dos primeros son scripts para el entrenamiento de la red con X-SHIELD o FX-SHIELD y para obtención de *accuracy* en test, respectivamente. El tercero es una adaptación para entrenar H-SHIELD. Estos son muy parecidos a los anteriores pero están adaptados para utilizar nuestro conjunto de datos y los nuevos métodos.
- **tests/REVEL_metrics.py y tests/mi_REVEL_metrics.py:** *script* original para calcular las métricas REVEL para un modelo y la versión modificada para utilizar el conjunto de datos de este TFM.

Además durante la ejecución de los experimentos se disponía de una carpeta llamada **datosTFM** dónde había dos carpetas, **train** y **val** que contenían los datos de entrenamiento y test respectivamente.

Para la gestión de paquetes de *Python* se ha utilizado la herramienta *Conda* que permite crearentornos con los paquetes de *Python* necesarios para el proyecto. Se ha utilizado la versión de Conda 4.10.1 y se ha creado un entorno con Python 3.9.23. Se han instalado los siguiente paquetes paquetes de *Python*:

- torch (versión 2.0.1): con CUDA versión 11.7
- torchvision
- tensorboard
- efficientnet_pytorch
- torchsummary
- numpy
- pandas

- scipy
- scikit-learn
- scikit-image
- matplotlib
- seaborn
- plotly
- tqdm
- opencv-python
- ipywidgets
- ipython
- wget
- captum
- revel-xai: paquete con la implementación de las métricas REVEL.

Todo el código desarrollado para el TFM está disponible en el siguiente enlace.

5 Experimentos

5.1. Datos empleados

El conjunto de datos empleado está desarrollado por la iniciativa *aI4HealthyAging*. En particular se creó para el paquete de trabajo 7: cribado y vigilancia activa en cánceres prevalente de la 3ª edad.

LA CITA BIBLIOGRÁFICA SIGUIENTE ([AJD24] la he creado yo. Se corresponde al archivo que me mandaste con información del dataset, pero no he podido encontrarlo online.

Metodología de obtención y etiquetado de imágenes Para la obtención de las imágenes siguieron la metodología que se puede consultar en [AJD24]. En esta siguen el procedimiento estándar de procesamiento hispatológico. Partían de muestras de prostatectomía radical (la extirpación completa de la próstata). A continuación la fijan en formalina. Tras esto un patólogo o residente de patología realiza un examen macróscopico para extraer muestras para estudiar con microscopio. Estas muestras de tejido se procesan deshidratándolos e impregnándolos en parafina. Luego se cortan secciones del tejido de entre 3 μm y 5 μm. Después se colocan en láminas de vidrio y se tiñen con hemaxtoxilina-ecosina. Finalmente se cubren con otra lámina de vidrio para ser examinados con un microscopio.

A continuación las muestras se escanearon en un escáner de láminas, lo que produjo imágenes en formato *TIFF*. Dichas imágenes fueron segmentadas manualmente por patólogos profesionales del servicio de anatomía patológica del Hospital Miguel Servet de Zaragoza (ver Figura 5.1). Los diferentes segmentos fueron anotados como pertenecientes a una de las 14 clases posibles, presentes en la Tabla 5.1. Para llevar a cabo el etiquetado hicieron uso de la plataforma *Cytomine*.

Finalmente las imágenes se fragmentan en parches de tamaño 224×224 píxeles. Cuando cuando un parche contiene una proporción de píxeles perteneciente a una clase superando el umbral $h = 0.75$ este parche se anota con dicha clase, si esto no

ocurre para ninguna clase se descarta el parche.

Esta es la versión del conjunto de datos con el que partimos nosotros, tenemos los parches que han sido anotados con una de las 14 clases.

Nombre	**Descripción**	**Tipo**
Tejido sano diferente a estroma (HT)	Tejido sano sin estructuras relevantes asociadas a malignidad	Tejido no maligno
Estroma sano (HS)	Estroma sin estructuras relevantes asociadas a malignidad	Tejido no maligno
Glándulas no neoplásicas (NG)	Glándulas con morfología no neoplásica	Tejido no maligno
Tejido inflamatorio (IT)	Tejido con signos de inflamación	Tejido no maligno
Gleason 3 (G3)	Glándulas individuales, bien formadas, de tamaño variable, separadas por estroma	Tejido maligno
Gleason 4, patrón cribiforme (G4c)	Glándulas compuestas por células tumorales dispuestas en trabéculas redondeadas o irregulares, cohesivas, con perforaciones o luces perforadas	Tejido maligno
Gleason 4, patrón glándulas fusionadas (G4f)	Glándulas sin estroma entre ellas, con luz glandular conservada	Tejido maligno
Gleason 4, patrón glándulas pobremente formadas (G4p)	Glándulas pequeñas con luces desproporcionadas	Tejido maligno
Gleason 4, patrón glomeruloide (G4g)	Glándulas con un nido de células tumorales que ocupa la luz y está unida a la glándula por un polo	Tejido maligno
Gleason 5, patrón comedonecrosis (G5c)	Nidos tumorales redondeados con necrosis central de tipo comedo	Tejido maligno
Gleason 5, patrón sin glándulas definidas (G5w)	Células tumorales sueltas, en hilera o trabéculas. Ausencia de diferenciación glandular	Tejido maligno
Neoplasia intraepitelial prostática de alto grado (PIN)	Glándulas con capa basal y proliferación de células epiteliales con atipia y pérdida de mucosecreción; patrones micropapilar, cribiforme, plano, hiperplásico	Tejido maligno

Continúa en la siguiente página

Tabla 5.1 – continuación

Nombre	Descripción	Tipo
Adenocarcinoma ductal (DA)	Glándulas cribiformes y estructuras papilares y/o complejas revestidas por células altas columnares pseudoestratificadas	Tejido maligno
Carcinoma intraductal (IC)	Grandes ductos con capa basal y luz dilatada, distendida y ocupada por proliferación de células epiteliales con marcada atipia y crecimiento papilar, cribiforme laxo o cribiforme denso	Tejido maligno

Tabla 5.1: En esta tabla se muestra la clasificación de los tejidos, una breve descripción de qué significan y si el tejido es maligno. Tabla extraída y adaptada de [AJD24].

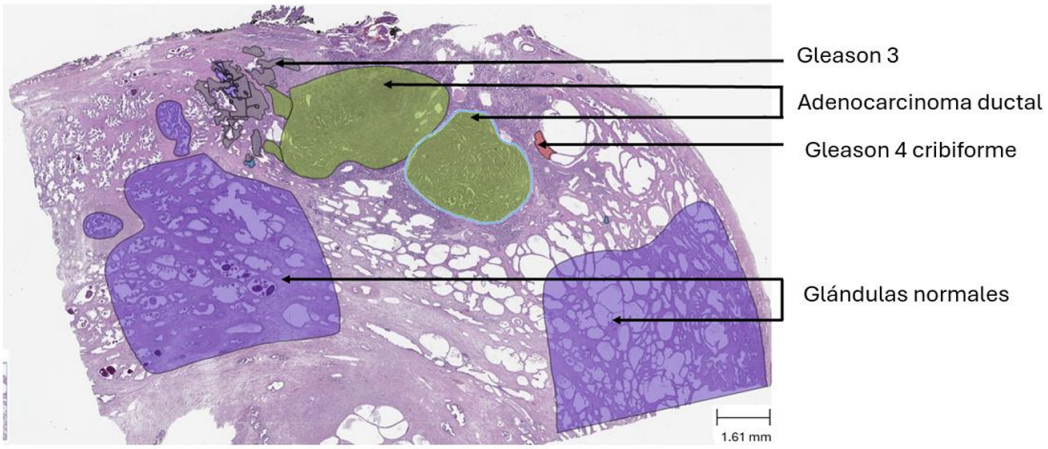

Figura 5.1: Imagen de tejido segmentado. Imagen extraída de [AJD24].

Distribución de los datos En total, el conjunto de datos contiene 126.109 imágenes anotadas. La distribución de estos entre las distintas clases se muestra en la Figura 5.2. Se puede apreciar un claro desbalanceo en los datos. La clase menos frequente es Gleason 4 glomeruloide (G4p) con 243 datos (0.19 % de los datos), mientras que la más frecuente es Glándulas no neoplásticas (NG) con 35.894 datos (28.46 % de los datos).

Además los datos están divididos en dos carpetas *train* y *val*. La carpeta *train* contiene 113.491 imágenes (90 % de los datos) mientras que *val* contiene las 12.618 imágenes

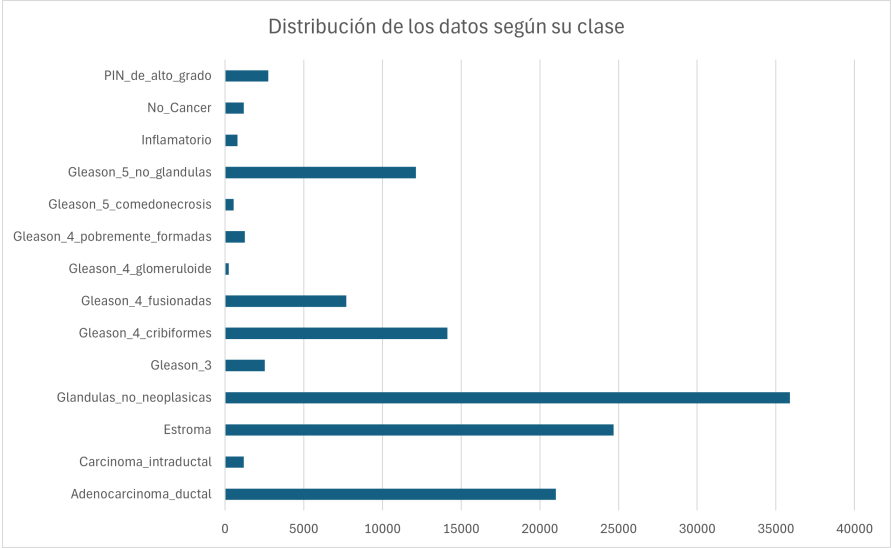

Figura 5.2: Distribución de las imágenes del conjunto de datos según su clase.

restantes (10 % de los datos). Está partición de los datos se realizó para entrenamiento/test de los modelos y vamos a respetarla. Se puede comprobar que esta partición de los datos mantiene la proporción de las clases como se puede ver en la Figura 5.3.

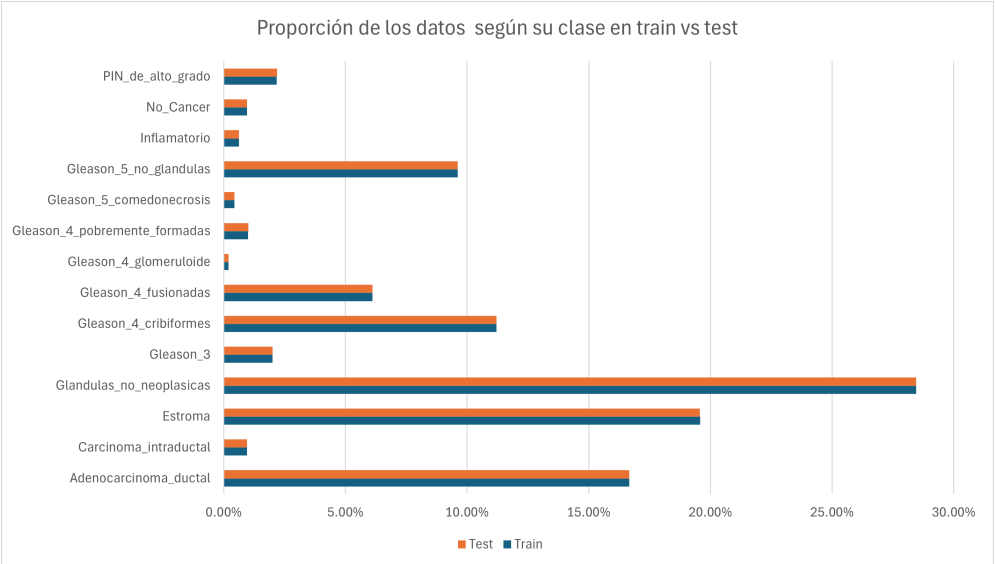

Figura 5.3: Distribución de las imágenes del conjunto de datos para entrenamiento y para test según su clase.

5.2. Experimentos realizados

Los experimentos han sido ejecutados en los servidores de cómputo Atenea y Hera, de los servidores NGPU de la Universidad de Granada. Las gráficas utilizadas son GTX Titan Xp con 128 Gb DDR4 y Titan RTX con 128 Gb DDR4, respectivamente.

5.2.1. Separación de datos

Cómo ya hemos comentado realizamos una división 90/10 de los datos totales entre entrenamiento/test. Los modelos se van a entrenar con el conjunto de entrenamiento y se evaluará con los datos del conjunto de test. Esta división de los datos se realiza para tener un conjunto, el de test, que no haya visto el modelo durante el entrenamiento para evaluar el rendimiento real del modelo para datos que aún no ha visto. Esta práctica es muy importante para evitar el sobreentrenamiento de la red. La división de 90/10 de los datos tiene sentido sobre otras típicas como 80/20 porque disponemos de una cantidad muy grande de datos (126.109 imágenes).

Validation Teniendo en cuenta el tamaño de nuestro conjunto de datos y de los recursos computacionales de los que disponemos hemos decidido utilizar un conjunto de validación fijo sobre otros métodos como *k-fold cross validation*.

Esta técnica consiste en separa un porcentaje del conjunto de entrenamiento para no entrenar con él. Sin embargo se va a utilizar para evaluar el modelo al final de cada época, calculando su función de error sobre este conjunto. Esta técnica es importante para el entrenamiento del modelo porque que cumple un rol distinto a los conjuntos de entrenamiento y test. Este nos ayudará a realizar un ajuste de hiperparámetros del modelo y ayuda a prevenir el sobreajuste, eligiendo el modelo que menor error tenga en validación, en vez de en entrenamiento.

Nosotros vamos a dividir el conjunto de entrenamiento en 90/10 para entrenamiento y para validación respectivamente. En la Figura 5.4 presentamos un esquema con la división del conjunto de datos seguida para los experimentos.

Optimizador y elección de hiperparámetros El **optimizador** elegido para entrenar el modelo es Adam [KB17], que es una modificación de SGD. Se ha elegido este optimizador porque es el que se utilizó originalmente en el artículo de X-SHIELD

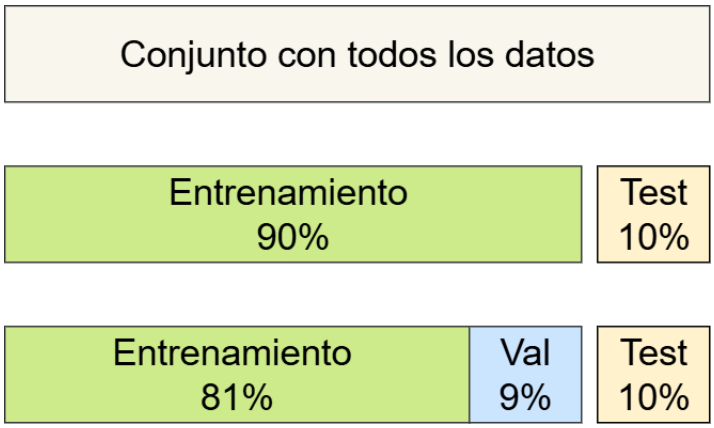

Figura 5.4: Distribución de los datos en los conjuntos finales de entrenamiento, validación y test.

[SGLH23b] y porque está probado empíricamente que tiene mejor rendimiento que el resto de optimizadores para una gran variedad de tareas.

Cómo ***learning rate*** inicial Adam se ha escogido el valor 0.0004. Un *learning rate* inicial es muy importante porque dicta como converge el modelo. Para la elección de este parámetro se han hecho pruebas iniciales sobre los modelos y hemos determinado que este valor es el que mejor convergencia daba al modelo.

El tamaño del ***minibatch*** escogido ha sido 32. Está demostrado que tamaños grandes de *minibatches* acelera el entrenamiento y estabiliza la convergencia del modelo. Por otro lado un tamaño pequeño puede mejorar la capaz de generalizar del modelo [SKYL18]. El tamaño suele escogerse como una potencia de 2. Hemos escogido tamaño 32 ya que para 64 la gráfica dónde ejecutamos los experimentos se quedaba sin memoria y empíricamente se comprobó que 32 daba mejores resultados que tamaños menores.

Para el número de **épocas** hemos escogido un número no muy alto ya por los recursos computacionales de los que disponemos. Por ello se hicieron pruebas experimentales previas para buscar un valor que garantizaba la convergencia de estos pero no se excedía en tiempo de entrenamiento. El número de épocas es por lo tanto 24. Con este número de épocas los modelos entrenados con la regularización X-SHIELD (la más costosa) tardan alrededor de 1 día (normalmente un poco más) en entrenarse.

Cómo ya sabemos el modelo que vamos a utilizar en los experimentos es EfficientNet-B2 con entradas de tamaño 224×224. Este modelo lo cargamos de la librería de

pytorch. Además lo vamos a cargar con los pesos preentrenados de esta red sobre el conjunto de datos IMAGENET-1K lo que se conoce como *transfer learning* y nos permite no tener que entrenar el modelo desde cero.

En primer lugar vamos a entrenar el modelo EfficientNet-B2 base con los parámetros que hemos especificado. Después se entrenará esta red aplicando los distintos métodos de regularización explicados, estos son: el método original con el que comparamos los propuestos (X-SHIELD) y los métodos nuevos que proponemos en este trabajo (FX-SHIELD, FR-SHIELD y H-SHIELD).

Los hiperparámetros a los que se va a probar a cambiar sus valores son los correspondientes a los métodos de regularización de XAI. Estos son:

- λ: En el caso de X-SHIELD, FX-SHIELD y FR-SHIELD este parámetro determina que porcentaje de las caracterísitcas (de las entrada en el caso de X-SHIELD y los intermedios en el caso de los nuestros) se van a ocultar para el cálculo de los valores de regularización. Para X-SHIELD se ha probado con los parámetros 3, 6, 9 y 12, ya que en [SGLH23b] se demostró que funcionaba mejor para valores similares a estos. Para los métodos nuevos, como no tenemos esta información hemos probado con más variedad de valores. Para FX-SHIELD hemos probado con los parámetros a 3, 6, 9, 12, 25, 35, 45, 55, 65, 75 y 85, como funcionaba mejor para 75 también probamos con los valores vecinos 70, 73, 77 y 80. Para FR-SHIELD hemos probado con 3, 45, 65, 75 y 77.

- α_1 y α_2: Estos son los parámetros para H-SHIELD. Son los pesos que se aplican a la regularización X-SHIELD y FX-SHIELD respectivamente. Para los hiperparámetros λ para cada uno de estos hemos escogido los que mejores resultados daban por individual que son 12 y 77 respectivamente. Para (α_1, α_2) se han probado $(1,1)$, $(1,2)$ y $(1,3)$

Además para las características de X-SHIELD se divide la imagen de entrada en 64 cuadrados del mismo tamaño para cada imagen como se hace en [SGLH23b].

De esta manera, el modelo se entrena aplicando cada método con la distintas elecciones de hiperparámetros. Para evitar el sobreentrenamiento, durante el entrenamiento de los modelos en cada época se calcula la función de pérdida sobre el conjunto de validación y el modelo final del entrenamiento es el que menor pérdida en validación haya obtenido, no necesariamente el que minimice el error en entrenamiento.

Para elegir los mejores hiperparámetros para cada método se escoge modelo entrenado que minimicen la *accuracy* en validación para dichos hiperparámetros.

5.3. Métricas

Para el entrenamiento del modelo se utiliza como función de pérdida el **error de entropía cruzada** que es la función de pérdida que se utiliza habitualmente para entrenar redes en clasificación.

Para evaluar el rendimiento del modelo en la tarea de clasificación utilizamos la ***accuracy*** que mide la proporción de ejemplos correctamente clasificados. Esta toma valores en $[0, 1]$ y cuanto mayor es, mejor es el rendimiento del modelo.

Por otro lado para evaluar el rendimiento utilizamos la métricas **REVEL**. Estas fueron formalmente introducidas en 2.3.2. Recordamos que estas son: *local concordance*, *local fidelity*, *presciptivity*, *conciseness* y *robustness*. Para cada método de regularización las calcularemos únicamente para el modelo final que ha sido calculado para los mejores parámetros para dicho método.

Para el cálculo de estas métricas se deben crear explicaciones con un LLE. Elegimos **LIME**. Utilizamos las mismas características que X-SHIELD para LIME, es decir 64 cuadrados del mismo tamaño sobre la imagen de entrada. Para los cálculos de la explicación de un ejemplo generamos 1000 vecinos. El parámetro σ (explicado en la subsección 2.3.1) lo ponemos a 12.

Para cada ejemplo generamos 10 explicaciones para el cálculo de las cuatro primeras métricas y calculamos la media aritmética de cada una para obtener su valor final para ese ejemplo. En total utilizamos 100 ejemplos. Para el cálculo de la *robustness* utilizan las 10 explicaciones generadas para las otras métricas para calcular esta métrica. Hemos elegido estos parámetros para calcular estas métricas por límites de coste computacional. El cálculo de estas métricas para un modelo ya entrenado tarda alrededor de 1 día.

Test estadísticos bayesianos Una vez tenemos los 100 valores de las métricas REVEL para cada método utilizamos la librería de *Python* baycomp. Esta biblioteca introduce los tests bayesianos para comparación de método. Estos tests permiten comparar dos métodos de manera probabilística, estimando directamente la probabilidad de que uno sea mejor que otro, o que la diferencia entre ellos sea insignificante.

Los tests de baycomp se basan en la estimación de la distribución posterior de la diferencia de desempeño entre dos métodos. Se calculan las tres probabilidades siguientes:

- Método A es mejor que método B.
- Método B es mejor que método A.
- Diferencia es irrelevante (*Region of Practical Equivalence*, ROPE)

ROPE define un umbral de diferencia prácticamente insignificante. En la página del propio método recomienda ROPE = 0.01 para métricas en el rango $[0, 1]$ que es el caso de las métricas REVEL (excepto técnicamente *robustness* que podría ser negativa, en la práctica suele estar en el rango).

Diremos que un método es mejor que otro para una métrica cuando la probabilidad de que sea mejor es $> 95\,\%$.

5.4. Resultados

Aquí muestro mis resultados + curvas de aprendizaje y comento brevemente estos.

5.5. Discusión

Aquí es donde discuto mis resultados obtenidos.

6 Conclusiones

Conclusiones del TFM y si he conseguido los objetivos que me propuse al inicio.

6.1. Trabajos futuros

Aquí comento trabajos futuros. Cómo probar con otros datsets o lo que comentamos de sacar las métricas REVEL sobre features (en nuestra primera charla).

Bibliografía

[AAES+23] Sajid Ali, Tamer Abuhmed, Shaker El-Sappagh, Khan Muhammad, Jose M. Alonso-Moral, Roberto Confalonieri, Riccardo Guidotti, Javier Del Ser, Natalia Díaz-Rodríguez, y Francisco Herrera. Explainable artificial intelligence (xai): What we know and what is left to attain trustworthy artificial intelligence. *Information Fusion*, 99:101805, 2023.

[ADE+23] Reduan Achtibat, Maximilian Dreyer, Ilona Eisenbraun, Sebastian Bosse, Thomas Wiegand, Wojciech Samek, y Sebastian Lapuschkin. From attribution maps to human-understandable explanations through concept relevance propagation. *Nature Machine Intelligence*, 5(9):1006–1019, 2023.

[AJD24] Jacobo Ayensa Jiménez y Manuel Doblaré. Paquete de trabajo 7. cribado y vigilancia activa en cánceres prevalentes de la 3ª edad: Informe sobre la tarea a69. diagnóstico automatizado del grado gleason en biopsias de próstata mediante inteligencia artificial. Informe técnico Tarea A69, Iniciativa AI4HealthyAging, 2024.

[Alp20] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, 2020.

[AMMIL12] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, y Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012.

[APB21] Elvio Gilberto Amparore, Alan Perotti, y Paolo Bajardi. To trust or not to trust an explanation: using leaf to evaluate local linear xai methods. *PeerJ Computer Science*, 7, 2021.

[BB23] Christopher M. Bishop y Hugh Bishop. *Deep learning: Foundations and concepts*. Springer Nature, 2023.

[BDRD+20] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, y Francisco Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020.

[Bis95] Christopher Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[Biso6] Christopher Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[BZK+17] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, y Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. *CoRR*, abs/1704.05796, 2017.

[DXC+22] Mingyu Ding, Bin Xiao, Noel Codella, Ping Luo, Jingdong Wang, y Lu Yuan. Davit: Dual attention vision transformers. En Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, y Tal Hassner, editores, *Computer Vision – ECCV 2022*, páginas 74–92, Cham, 2022. Springer Nature Switzerland.

[EARH17] Jonathan I Epstein, Mahul B. MD Amin, Victor E. Reuter, y Peter A. Humphrey. Contemporary gleason grading of prostatic carcinoma: An update with discussion on practical issues to implement the 2014 international society of urological pathology (isup) consensus conference on gleason grading of prostatic carcinoma. *The American Journal of Surgical Pathology*, 2017.

[GBC16] Ian Goodfellow, Yoshua Bengio, y Aaron Courville. *Deep Learning*. MIT Press, 2016.

[Hay98] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, USA, segunda edición, 1998.

[HJR+21] Sung H, Ferlay J, Siegel RL, Laversanne M, Soerjomataram I, Jemal A, y Bray F. Global cancer statistics 2020: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA: A Cancer Journal for Clinicians*, 2021.

[HLvdMW18] Gao Huang, Zhuang Liu, Laurens van der Maaten, y Kilian Q. Weinberger. Densely connected convolutional networks. *arXiv*, 2018.

[HSA+20] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, y Enhua Wu. Squeeze-and-excitation networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(8):2011–2023, 2020.

[HSK+12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, y Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv*, 2012.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, y Jian Sun. Deep residual learning for image recognition. *arXiv*, 2015.

[IS15] Sergey Ioffe y Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[KB17] Diederik P. Kingma y Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2017.

[KMSB25] Patrick Knab, Sascha Marton, Udo Schlegel, y Christian Bartelt. Which lime should i trust? concepts, challenges, and solutions, 2025.

[KWG+17] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie J. Cai, James Wexler, Fernanda B. Viégas, y Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). En *International Conference on Machine Learning*, 2017.

[LBC+24] Luca Longo, Mario Brcic, Federico Cabitza, Jaesik Choi, Roberto Confalonieri, Javier Del Ser, Riccardo Guidotti, Yoichi Hayashi, Francisco Herrera, Andreas Holzinger, Richard Jiang, Hassan Khosravi, Freddy Lecue, Gianclaudio Malgieri, Andrés Páez, Wojciech Samek, Johannes Schneider, Timo Speith, y Simone Stumpf. Explainable artificial intelligence (xai) 2.0: A manifesto of open challenges and interdisciplinary research directions. *Information Fusion*, 106:102301, 2024.

[LBH15] Yann LeCun, Y. Bengio, y Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 2015.

[LKLH19] Shusen Liu, Bhavya Kailkhura, Donald Loveland, y Yong Han. Generative Counterfactual Introspection for Explainable Deep Learning. *arXiv e-prints*, página arXiv:1907.03077, July 2019.

[LL17] Scott M. Lundberg y Su-In Lee. A unified approach to interpreting model predictions. En *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, página 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc.

[LLY+22] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, y Jun Zhou. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019, 2022.

[MBL+19] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, y Klaus-Robert Müller. *Layer-Wise Relevance Propagation: An Overview*, páginas 193–209. 09 2019.

[MBM20] Reza Moradi, Reza Berangi, y Behrouz Minaei. A survey of regularization strategies for deep models. *Artif. Intell. Rev.*, 53(6), 2020.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, New York, 1997.

[MOM12] Grégoire Montavon, Geneviève B. Orr, y Klaus-Robert Müller. *Neural networks: tricks of the trade*, volumen 7700. Springer, 2012.

[Mur22] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[OMS17] Chris Olah, Alexander Mordvintsev, y Ludwig Schubert. Feature visualization. *Distill*, 2, 11 2017.

[Pri23] Simon JD Prince. *Understanding deep learning*. MIT press, 2023.

[RDV18] Andrew Ross y Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

[RHDV17] Andrew Slavin Ross, Michael C. Hughes, y Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. En *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, páginas 2662–2670, 2017.

[Ros21] Avi Rosenfeld. Better metrics for evaluating explainable artificial intelligence. En *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, página 45–50, Richland, SC, 2021. International Foundation for Autonomous Agents and Multiagent Systems.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, y Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. En *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, página 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.

[RSMY20] Laura Rieger, Chandan Singh, William Murdoch, y Bin Yu. Interpretations are useful: Penalizing explanations to align neural networks with prior knowledge. En Hal Daumé III y Aarti Singh, editores, *Proceedings of the 37th International Conference on Machine Learning*, volumen 119 de *Proceedings of Machine Learning Research*, páginas 8116–8126. PMLR, 13–18 Jul 2020.

[SCD+17] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, y Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. En *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[SGLH23a] Iván Sevillano-García, Julián Luengo, y Francisco Herrera. Revel framework

to measure local linear explanations for black-box models: Deep learning image classification case study. *International Journal of Intelligent Systems*, 2023(1):8068569, 2023.

[SGLH23b] Iván Sevillano-García, Julián Luengo, y Francisco Herrera. X-shield: Regularization for explainable artificial intelligence. *ArXiv*, 2023.

[Sha24] Sovrin M. Shah. Gleason grading system [internet]. 2024. Revisado el 17 de Mayo, 2024; accedido el 25 de Agosto, 2025. Revisado por: VeriMed Healthcare Network, David C. Dugdale, Brenda Conaway y A.D.A.M. Inc.

[SHZ+18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, y Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. En *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, páginas 4510–4520, 2018.

[SKYL18] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, y Quoc V. Le. Don't decay the learning rate, increase the batch size. *International Conference on Learning Representations*, 2018.

[SSA20] Siddharth Sharma, Simone Sharma, y Anidhya Athaiya. Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, 04:310–316, 2020.

[STY17] Mukund Sundararajan, Ankur Taly, y Qiqi Yan. Axiomatic attribution for deep networks. En Doina Precup y Yee Whye Teh, editores, *Proceedings of the 34th International Conference on Machine Learning*, volumen 70 de *Proceedings of Machine Learning Research*, páginas 3319–3328. PMLR, 06–11 Aug 2017.

[TL19] Mingxing Tan y Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *International Conference on Machine Learning*, 2019.

[VL21] Giulia Vilone y Luca Longo. Notions of explainability and evaluation approaches for explainable artificial intelligence. *Information Fusion*, 76:89–106, 2021.

[WDH+23] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, y Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. En *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, páginas 16133–16142, 2023.

[WLBS23] Leander Weber, Sebastian Lapuschkin, Alexander Binder, y Wojciech Samek. Beyond explaining: Opportunities and challenges of xai-based model improvement. *Information Fusion*, 92:154–176, 2023.

[YWV+22] Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, y

Yonghui Wu. Coca: Contrastive captioners are image-text foundation models. *Transactions on Machine Learning Research*, 2022.