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 27 de agosto de 2025

Fdo: Carlos Lara Casanova

Índice general

Agradecimientos v

Summary vii

Resumen ix

1 Introducción 1
1.1 Objetivos . . . 2
1.2 Planificación . . . 3

2 Fundamentos teóricos 5
2.1 Aprendizaje automático . . . 5
2.1.1 Problema de clasificación . . . 6
2.1.2 Problema de regresión . . . 6
2.1.3 Optimización . . . 6
2.2 Deep Learning . . . 8
2.2.1 Redes convolucionales . . . 13
2.3 XAI . . . 16
2.3.1 Explicaciones lineales locales y LIME . . . 17
2.3.2 Métricas ReVEL . . . 19
2.3.3 Regularización X-Shield . . . 23

3 Estado del arte 27

4 Métodos propuestos 29
4.0.1 FXShield . . . 30
4.0.2 FRShield . . . 30
4.0.3 HShield . . . 30
4.1 Implementación . . . 31

5 Experimentos 33
5.1 Datos empleados . . . 33
5.2 Experimentos realizados . . . 33
5.2.1 Separación de datos . . . 33
5.3 Métricas . . . 33
5.4 Resultados . . . 33
5.5 Discusión . . . 34

6 Conclusiones **35**
6.1 Trabajos futuros 35

Bibliografía **37**

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.

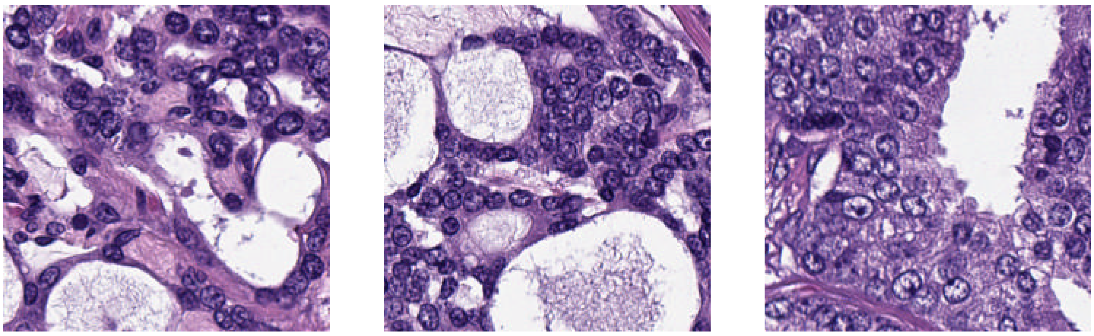

Figura 1.1: Tres imágenes correspondientes a tres biopsias distintas.

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 Cuál?
- 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 ReVEL/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 $\mathbf{w}$ 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.

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.

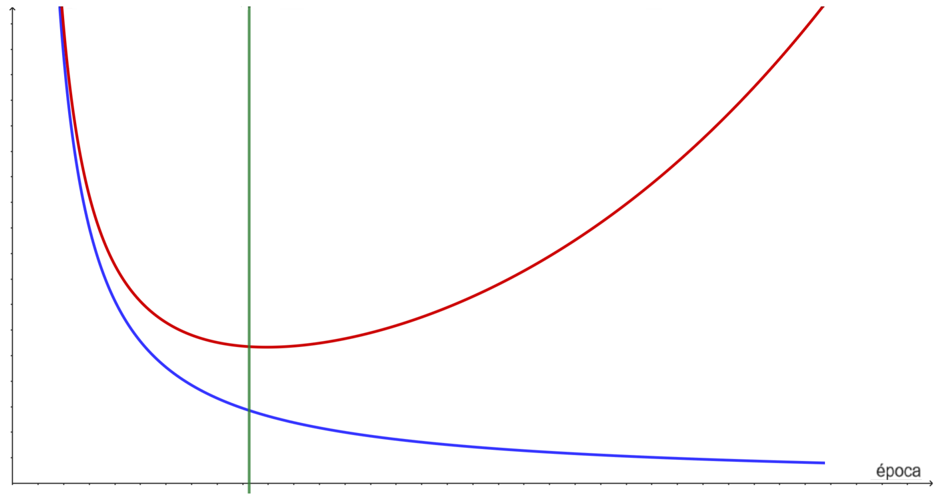

Figura 2.1: Esta figura muestra el error en entrenamiento en azul y el error en los datos con los que no se entrena el modelo en rojo a lo largo del entrenamiento. Se puede ver como a partir de la línea verde el modelo sigue aprendiendo de los datos de entrada pero empeora el poder de generalización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

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.

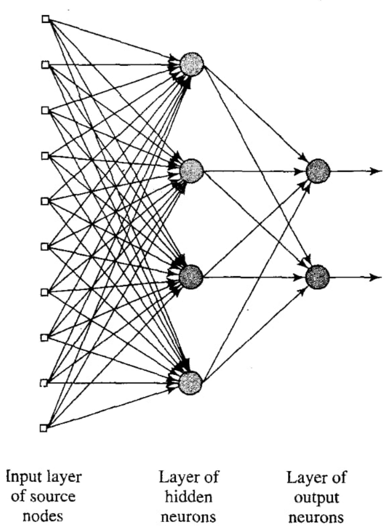

Figura 2.2: Ejemplo de ANN con una única capa oculta y con dos salidas. Imagen extraída de [Hay98].

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.

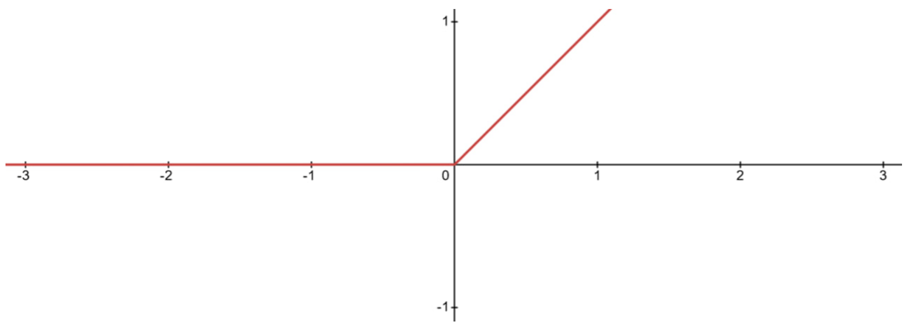

Figura 2.3: Función de activación ReLU.

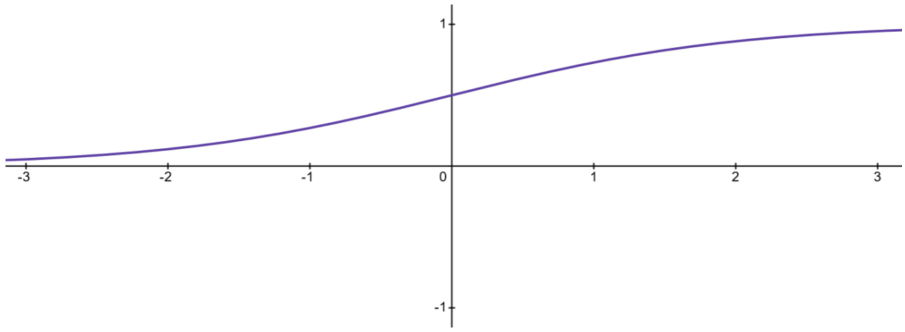

Figura 2.4: Función de activación sigmoidal.

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.

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

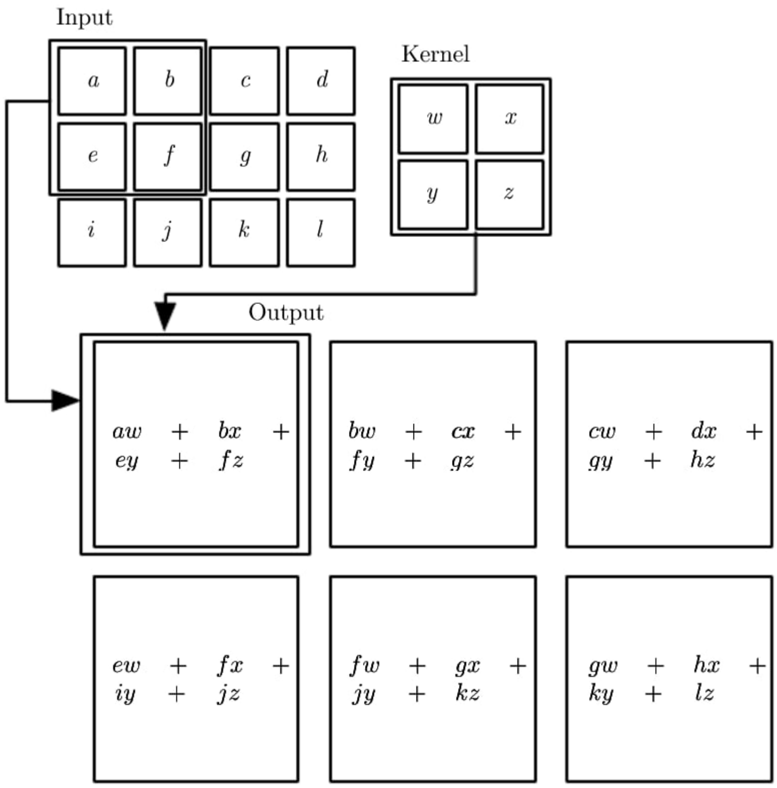

Figura 2.5: En esta imagen se puede ver un ejemplo de una convolución aplicada a una entrada de dimensiones 3×4 por un kernel de dimensiones 2×2. Se puede ver que la salida tiene tamaño 2×3. Notablemente se puede ver que al contrario que en la definición de la operación de convolución dada, aquí no se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

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.

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

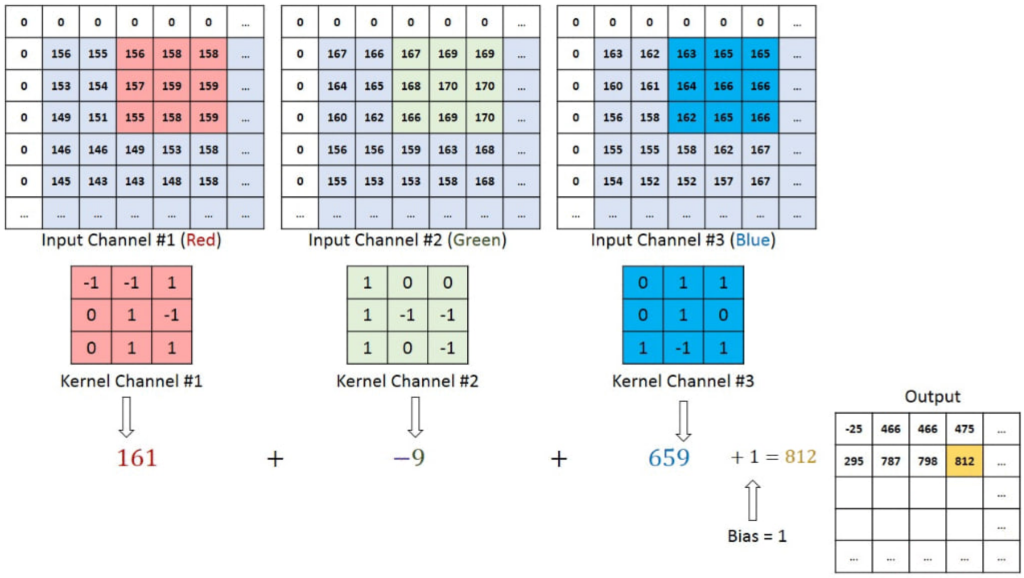

Figura 2.6: Aquí se puede ver un ejemplo de una convolución aplicada a una entrada con 3 canales, uno por cada color primario. Se ve cómo el kernel ahora tiene 3 canales también. Imagen extraída de `https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/`.

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*.

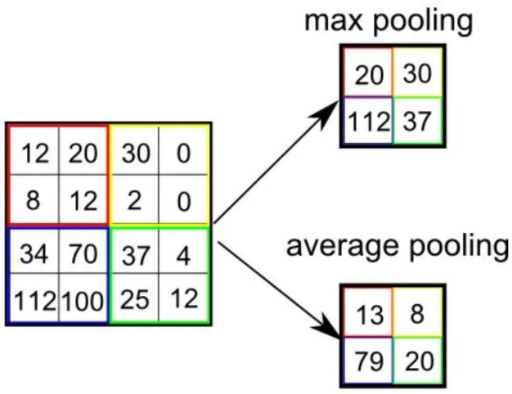

Figura 2.7: En la figura se ver cómo se aplica *max pooling* y *average pooling* a una entrada. Como las ventanas tienen tamaño 2×2 se dice que se aplica *pooling* 2×2. Imagen extraída de https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/.

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 $g : F \to Y_l$ 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 $\lambda\,\%$ 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 $\lambda\,\%$ de características aleatorias, va a ocultar las $\lambda\,\%$ menos importantes. Para ello, a cada característica le otorga la importan-

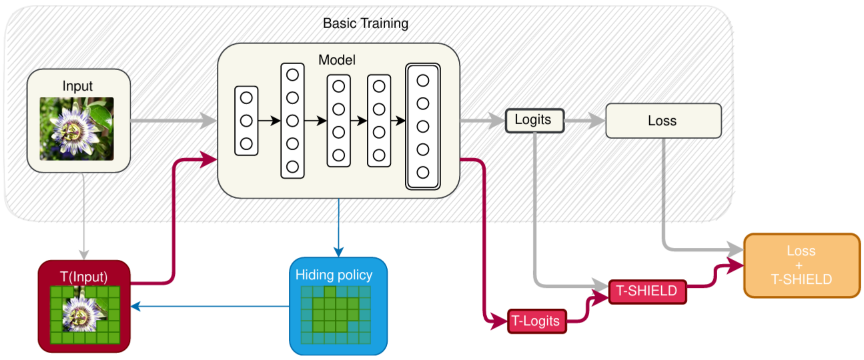

Figura 2.8: En esta figura se puede ver cómo funciona en general la familia de regularizaciones T-SHIELD durante el entrenamiento. En gris se puede ver el flujo normal de la imagen en el modelo. En azul se encontraría la forma en la que la regularización decide qué características ocluir. Finalmente 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].

cia $||v_i||$ 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:

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

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

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

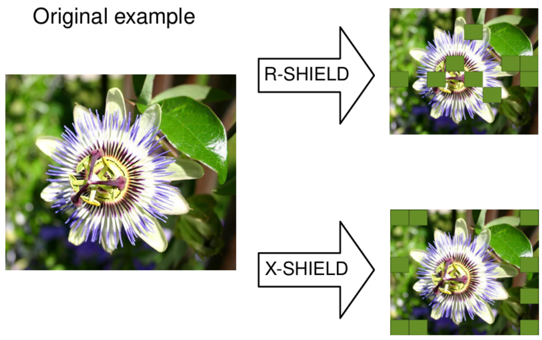

Figura 2.9: Comparación entre el comportamiento de T en R-SHIELD y X-SHIELD. La imagen está sacada del dataset *Flowers* de clasificación de imágenes de flores. Se puede ver que R-SHIELD ocluye partes de la imagen aleatorias, incluyendo partes de la flor a clasificar. Por otro lado X-SHIELD ocluye partes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

3 Estado del arte

En esta sección hago búsquedas en Scopus para ver el estado del arte y analizarlo. Para seros sinceros no tengo muy claro que debería buscar ya que dudo que encuentre artículos que apliquen explicabilidad a detección de cánceres de próstata (aunque está bien hacer la búsqueda para tenerlo claro).

Por otro lado quizás debería buscar trabajos que traten la detección automática de cánceres de próstata o que apliquen explicabilidad a otras tareas médicas a ver que encuentro. Estoy interesado en ver cómo debería enfocar el estado del arte.

4 Métodos propuestos

Antes de introducir mis propuestas haría un introducción sobre EfficientNet (en particular efficientnet b2) e introduciría su arquitectura y por qué la he elegido.

EfficientNet

4.0.1. FXShield

4.0.2. FRShield

4.0.3. HShield

4.1. Implementación

Aquí explico de qué código he partido (+ GitHub), breve introducción de cómo está organizado el proyecto (en carpetas/archivos), los datos, cómo me he organizado el proyecto en conda (versión de python + modulos) finalmente pondría un link al proyecto en mi GitHub). Por cierto me interesa saber si prefieres que cree un fork de alguno de tus proyectos en github (Iván) o si me creo uno a parte en el mío (obviamente referenciando tu github como partida), a mi me da igual.

5 Experimentos

5.1. Datos empleados

En esta sección explico el conjunto de datos del que parto. Explico como está organizado, cómo están balanceadas las clases, qué clases hay, qué significan, puedo mostrar algunos datos para mostrar ejemplos (si tengo permiso, claro). Me interesa saber de dónde provienen los datos también para ponerlo. En cuanto a estudio cualitativo de estos no creo que yo pueda aportar mucho así que no creo que haga (además de que hay demasiados datos).

5.2. Experimentos realizados

5.2.1. Separación de datos

Cómo están divididos los datos. He utilizado un conjunto de validación ya que validación cruzada es demasiado costoso en tiempo.

Validation

Optimizador y elección de hiperparámetros

5.3. Métricas

Explico que he utilizado Accurcy, cross entropy error y recuerdo el uso de la métricas revel. También hablo de los test estadísticos que he utilizado y los parámetros escogidos.

Test estadísticos bayesianos

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.

[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.

[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.

[Bis06] Christopher Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[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.

[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.

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

[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.

[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.

[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.

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

[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.

[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.

[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.

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