TRABAJOS FIN DE GRADO



Universidad de Jaén

Escuela Politécnica Superior de Jaén

Plataforma para la determinación de la explicabilidad de modelos de aprendizaje automático

Autor: Carlos Requena Doña

Grado: Ingeniería en Informática

Directores: Antonio Jesús Rivera Rivas y María Dolores Pérez Godoy
Departamento del director: Informática

Fecha: 31/7/2026

Licencia CC



CREA



UNIVERSIDAD DE JAÉN

D./Dª Antonio Jesús Rivera Rivas y D./Dª María Dolores Pérez Godoy, tutor(es) del Trabajo Fin de Grado titulado: **Plataforma para la determinación de la explicabilidad de modelos de aprendizaje automático**, que presenta Carlos Requena Doña, autoriza(n) su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, 31/7/2026

Agradecimientos

En primer lugar, quiero dar las gracias a mis tutores, Antonio Jesús Rivera Rivas y María Dolores Pérez Godoy, por su orientación, su disponibilidad y su paciencia a lo largo de todo el desarrollo de este trabajo, así como al grupo de investigación SIMIDAT de la Universidad de Jaén por darme la oportunidad de contribuir a la plataforma *Nets4Learning*.

Gracias también a mi familia, a mi madre y a mi padre por el apoyo constante e incondicional durante estos años de carrera. En especial, gracias a Martín por estar ahí de forma incondicional en todo momento y ser mi mejor amigo, el mejor que uno puede pedir.

A Fran y Diego por estar ahí para hablar y echarnos unas risas tanto en Discord como cuando estamos juntos en persona.

Y a Mar, por ser mi refugio seguro constante. Por las risas compartidas cuando más lo necesitaba y por darme, sin saberlo, en uno de mis peores momentos la fuerza y la motivación para cruzar la línea de meta.

Resumen

Aunque los modelos de aprendizaje automático destacan a la hora de realizar predicciones, su naturaleza de “caja negra” limita de forma notable su utilidad en muchos escenarios, destacando aquellos de alto riesgo como la medicina. En dichos casos es fundamental entender el comportamiento del modelo con el fin de confiar en este. Además, en campos como la enseñanza, entender el proceso es tan importante como el propio resultado. Este Trabajo Fin de Grado aborda el problema de opacidad diseñando e implementando una capa de Inteligencia Artificial Explicable (XAI) integrada de forma nativa en *Nets4Learning*, una plataforma educativa para el diseño y ejecución de modelos de *machine learning* y *deep learning*, que se ejecuta en el navegador y está construida con React y TensorFlow.js.

En lugar de desarrollar nuevos algoritmos de predicción, el proyecto se centra por completo en hacer que los modelos que la plataforma ya ofrece resulten interpretables para el usuario final. La aportación principal es un sistema de explicabilidad que se ejecuta íntegramente en el lado del cliente y que combina dos enfoques complementarios, un método post-hoc agnóstico al modelo que calcula la contribución de cada característica a partir de la perturbación de la entrada mediante WebSHAP y un método específico (LRP) que propaga la relevancia capa a capa por el interior de la red. Para las tareas basadas en imágenes se apoya en algoritmos de segmentación en superpíxeles (SLIC) y de segmentación facial, con el fin de agrupar la entrada en regiones con significado, generando mapas de calor y gráficos de importancia que explican el comportamiento de la red.

Esta arquitectura de XAI se ha integrado con éxito en la detección de objetos, la clasificación de imágenes (incluidos los conjuntos MNIST y KMNIST) y los modelos tabulares. La implementación final respeta la restricción de la plataforma de funcionar sin servidor, de modo que los datos del usuario no se envían fuera de su equipo, y se adapta a su interfaz multilingüe, que además hemos ampliado con el japonés, dando como resultado una herramienta práctica que acorta la distancia entre las predicciones en bruto de los modelos y la comprensión humana.

Palabras clave: Inteligencia Artificial Explicable, XAI, SHAP, LRP, aprendizaje automático, aprendizaje profundo, TensorFlow.js, explicabilidad en el navegador, superpíxeles.

Abstract

While machine learning models excel at making predictions, their “black-box” nature significantly limits their usefulness in many scenarios, focusing on high-stakes ones such as medicine. In those scenarios, understanding the model’s behaviour is essential in order to trust it. Moreover, in fields such as education, understanding the process is as important as the result itself. This thesis tackles the problem of opacity by designing and implementing an Explainable AI (XAI) layer natively into *Nets4Learning*, an educational platform for designing and running *machine learning* and *deep learning* models, which runs in the browser and is built with React and TensorFlow.js.

Rather than developing new predictive algorithms, this project focuses entirely on making the platform’s existing models interpretable for the end user. The core contribution is an explainability system running purely client-side that combines two complementary approaches: a post-hoc, model-agnostic method that, by integrating WebSHAP, computes feature attributions via input perturbation without any server dependencies, and a model-specific method (LRP) that propagates relevance layer by layer through the network. For image-based tasks, it utilizes SLIC superpixel and facial segmentation algorithms to group inputs into meaningful semantic regions, generating visual heatmaps and importance charts that explain the network’s behavior.

This XAI architecture was successfully integrated across object detection, image classification (including MNIST/KMNIST datasets), and tabular models. The final implementation respects the platform’s zero-server constraints, so the user’s data never leaves their device, and adapts to its multilingual interface, which we have also extended with Japanese, resulting in a practical tool that bridges the gap between raw model predictions and human comprehension.

Keywords: Explainable Artificial Intelligence, XAI, SHAP, LRP, machine learning, deep learning, TensorFlow.js, in-browser explainability, superpixels.

Tabla de contenidos

1. Introducción **1**

1.1. Contexto y motivación 1

1.2. Objetivos 2

1.3. Estructura de la memoria 3

2. Especificación del trabajo **5**

2.1. Descripción de la situación de partida 5

2.1.1. Descripción del entorno actual: *Nets4Learning* 5

2.1.2. Resumen de las deficiencias y carencias identificadas 10

2.2. Antecedentes 10

2.2.1. Conceptos básicos 10

2.2.2. La Inteligencia Artificial Explicable (XAI) 11

2.2.3. Técnicas de explicabilidad 12

2.3. Restricciones 13

2.3.1. Restricciones temporales 13

2.3.2. Restricción de ejecución en el cliente 14

2.3.3. Restricciones de integración 14

2.4. Metodología de desarrollo software 14

2.5. Estimación del tamaño y el esfuerzo 16
2.5.1. Estimación del tamaño: líneas de código (LOC) 16
2.5.2. Estimación del esfuerzo: modelo COCOMO 17
2.6. Tecnologías y herramientas utilizadas 18
2.6.1. Tecnologías 18
2.6.2. Herramientas de desarrollo 19
2.6.3. Ejecución de modelos en el navegador 19
2.7. Planificación 20
2.8. Presupuesto 22
2.8.1. Costes de personal 22
2.8.2. Costes de hardware y software 23
2.8.3. Coste total 23

3. Análisis de requisitos **25**
3.1. Requisitos funcionales 25
3.2. Requisitos no funcionales 25
3.3. Casos de uso 26

4. Diseño del sistema **35**
4.1. Arquitectura general 35
4.2. Integración con *Nets4Learning* 36
4.3. Diseño del módulo de explicabilidad 37
4.3.1. Explicabilidad agnóstica mediante SHAP 37
4.3.2. Explicabilidad específica mediante LRP 38
4.4. Diseño de la visualización 38

5. Desarrollo **41**

5.1. Fundamentos comunes 41

5.1.1. Bibliotecas utilizadas: WebSHAP y SLIC 41

5.1.2. Mecanismo general de SHAP 43

5.2. Incremento 1: explicabilidad en clasificación tabular 45

5.2.1. Objetivo del incremento 45

5.2.2. Diseño detallado 45

5.2.3. Implementación 47

5.2.4. Plan de pruebas 49

5.3. Incremento 2: explicabilidad en regresión 51

5.3.1. Objetivo del incremento 51

5.3.2. Diseño detallado 51

5.3.3. Implementación 53

5.3.4. Plan de pruebas 54

5.4. Incremento 3: explicabilidad en clasificación de imágenes 55

5.4.1. Objetivo del incremento 55

5.4.2. Diseño detallado 55

5.4.3. Implementación 57

5.4.4. Plan de pruebas 60

5.5. Incremento 4: explicabilidad en detección de objetos 62

5.5.1. Objetivo del incremento 62

5.5.2. Diseño detallado 62

5.5.3. Implementación 66

5.5.4. Plan de pruebas 69

5.6. Incremento 5: explicabilidad específica mediante LRP 71
5.6.1. Objetivo del incremento 71
5.6.2. Diseño detallado 71
5.6.3. Implementación 74
5.6.4. Plan de pruebas 78
5.7. Diagrama de clases completo 80
5.8. Patrones de diseño 82
5.8.1. Adapter 82
5.8.2. Factory 83

6. Conclusiones **85**
6.1. Conclusiones 85
6.2. Líneas de trabajo futuro 86

Bibliografía III

Lista de figuras

2.1. Pantalla de inicio de *Nets4Learning*, en la que el usuario elige una de las cuatro tareas disponibles. 7

2.2. Diseño y entrenamiento de un modelo propio en *Nets4Learning*: esquema de la red, editor de capas y editor de hiperparámetros. 8

2.3. Uso de un modelo ya entrenado sobre el conjunto Iris, con la descripción del modelo y los datos disponibles. 9

2.4. Manual de la plataforma, uno de los recursos de ayuda que ofrece *Nets4Learning* junto al glosario y los tutoriales guiados. 9

2.5. Diagrama de Gantt con la planificación temporal del proyecto a lo largo de los dos cuatrimestres. 22

3.1. Diagrama de casos de uso del módulo de explicabilidad, con un caso de uso por incremento. 27

4.1. Arquitectura general del sistema con la capa de explicabilidad integrada. 36

4.2. Diagrama de secuencia de la generación de una explicación mediante SHAP. 38

5.1. Diagrama de clases del algoritmo SLIC. 43

5.2. Diagrama de clases del incremento de clasificación tabular. 46

5.3. Diagrama de secuencia del caso de uso CU-1: explicación de una predicción de clasificación tabular. 46

5.4. *Storyboard* de la explicación tabular: predicción, explicación local y explicación global. 47

5.5. Predicción del modelo de clasificación tabular sobre una instancia del conjunto Iris. 48

5.6. Explicación local mediante SHAP de la predicción anterior: contribución de cada característica a la clase evaluada. 48

5.7. Importancia global mediante diagrama de enjambre (*beeswarm*), agregando cincuenta instancias del conjunto. 49

5.8. Diagrama de clases del incremento de regresión, en el que se reutilizan los elementos del incremento anterior. 51

5.9. Diagrama de secuencia del caso de uso CU-2: explicación de una predicción de regresión. 52

5.10. *Storyboard* de la explicación en regresión: predicción del valor, explicación local e importancia global. 52

5.11. Predicción y explicación local mediante SHAP sobre el modelo de regresión *Auto MPG*. 54

5.12. Diagrama de clases del incremento de clasificación de imágenes. . . . 56

5.13. Diagrama de secuencia del caso de uso CU-3: explicación de una clasificación de imagen mediante SHAP. 57

5.14. *Storyboard* de la explicación de imágenes: parámetros, cálculo con la galería de perturbaciones y mapas de calor por clase. 57

5.15. Predicción de partida del modelo de clasificación de imágenes (MobileNet) sobre la que se calculará la explicación. 58

5.16. Explicación mediante SHAP sobre MobileNet: galería de perturbaciones y mapa de calor por clase (rejilla de 6 y 75 muestras). 60

5.17. Misma explicación de la Figura 5.16 elevando el número de muestras de 75 a 300. 61

5.18. Diagrama de clases del incremento de detección de objetos. 63

5.19. Diagrama de secuencia del caso de uso CU-4: explicación de una detección de objetos. 64

5.20. Flujo de una explicación en detección de objetos, con la elección de la segmentación. 65

5.21. *Storyboard* de la explicación en detección de objetos: detección original con los controles de la máscara y mapa de calor resultante. 65

5.22. Explicación de una detección con COCO-SSD: un mapa de calor por cada etiqueta detectada. 67

5.23. Explicación de una detección facial con segmentación anatómica: mapas de calor de las clases de edad y emoción (arriba, primeras clases; abajo, las dos restantes de la misma tarjeta). 68

5.24. Caso límite en detección facial: al no ser un rostro real, el modelo no produce detecciones válidas y la explicación resultante no resalta ninguna región. 70

5.25. Ilustración del funcionamiento de LRP. 71

5.26. Diagrama de clases del incremento de explicabilidad mediante LRP. . . 72

5.27. Diagrama de secuencia del caso de uso CU-5: explicación de una clasificación de imagen mediante LRP. 73

5.28. Boceto del selector de método en la tarjeta de explicabilidad: con LRP disponible (a) y deshabilitado por no ser accesible la estructura interna del modelo (b). 74

5.29. Flujo de ejecución de una explicación mediante LRP. 78

5.30. Explicación mediante LRP sobre un modelo con estructura accesible entrenado en la plataforma: la relevancia sigue el trazo del carácter. . . 79

5.31. Diagrama de clases completo del módulo de explicabilidad, con las funcionalidades de todos los incrementos y los patrones de diseño usados en el desarrollo. 81

Lista de tablas

2.1. Tamaño aproximado del módulo de explicabilidad, en líneas de código. 17

2.2. Estimación del esfuerzo mediante COCOMO básico (modo orgánico). . 17

2.3. Planificación temporal del proyecto por fases e incrementos. 21

2.4. Estimación de costes de personal. 23

2.5. Estimación de costes de hardware y software. 23

2.6. Coste total estimado del proyecto. 23

3.1. Requisitos funcionales del sistema. 26

3.2. Requisitos no funcionales del sistema. 27

3.3. Trazabilidad entre casos de uso, incrementos y requisitos funcionales. . 28

3.4. Caso de uso CU-1: explicar una predicción de clasificación tabular. . . . 29

3.5. Caso de uso CU-2: explicar una predicción de regresión. 30

3.6. Caso de uso CU-3: explicar una clasificación de imagen mediante SHAP. 31

3.7. Caso de uso CU-4: explicar una detección de objetos. 32

3.8. Caso de uso CU-5: explicar una clasificación de imagen mediante LRP. 33

5.1. Pruebas unitarias sobre las utilidades de muestreo de SHAP. 50

5.2. Pruebas funcionales del incremento de clasificación tabular. 50

5.3. Pruebas funcionales del incremento de regresión. 55

5.4. Pruebas funcionales del incremento de clasificación de imágenes. . . . 62

5.5. Pruebas funcionales del incremento de detección de objetos. 69

5.6. Pruebas funcionales del incremento de explicabilidad mediante LRP. . . 79

5.7. Patrones de diseño aplicados en la solución. 82

Capítulo 1

Introducción

La Inteligencia Artificial (IA) y, más concretamente, el Aprendizaje Profundo (*Deep Learning*), ha experimentado un crecimiento exponencial en la última década, integrándose en sectores críticos como la medicina, la automoción y la educación. Sin embargo, el aumento en la complejidad de estos modelos ha traído consigo un problema fundamental, la falta de transparencia. Las redes neuronales profundas actúan a menudo como "cajas negras", ofreciendo predicciones precisas sin una justificación comprensible para el usuario humano.

En muchos ámbitos donde se aplican estos modelos, dicha opacidad presenta una barrera importante para su adopción y confianza por parte de los usuarios. Destacando especialmente áreas como la medicina donde acertar una predicción no es suficiente, sino que se requiere entender su comportamiento de modo que se pueda confiar en sus decisiones. De la misma forma se puede ver en la enseñanza, dado que para que un estudiante comprenda realmente cómo funciona una red neuronal no basta con ver si acierta o falla, sino que necesita entender qué características de los datos han motivado esa decisión.

1.1. Contexto y motivación

Este Trabajo Fin de Grado (TFG) se desarrolla en el contexto del grupo de investigación SIMIDAT de la Universidad de Jaén. El punto de partida es la plataforma *Nets4Learning* [1], una herramienta web diseñada para la enseñanza y experimentación

[1] Para facilitar la evaluación de este proyecto, se puede acceder al mismo a través de este enlace: https://simidat.ujaen.es/n4l/

con modelos de Deep Learning sin necesidad de hardware especializado ni conocimientos avanzados de programación.

La arquitectura de *Nets4Learning* es innovadora al ejecutar tanto el entrenamiento como la inferencia de los modelos en el lado del cliente (navegador) mediante TensorFlow.js, lo que democratiza el acceso a la IA. Al hacerse todo el cálculo en el equipo del usuario, los datos no se envían a ningún servidor, suponiendo una ventaja importante en cuanto a la privacidad se refiere.

Dicha plataforma hasta ahora carecía de un módulo que explicase las decisiones tomadas. La motivación principal de este proyecto es dotar a *Nets4Learning* de un módulo de Inteligencia Artificial Explicable (XAI). La condición principal es que dicho módulo debía respetar la filosofía de la plataforma, es decir, funcionar por completo en el navegador sin depender de ningún servidor que realizase los cálculos (*backend*). Esta restricción descarta las bibliotecas de explicabilidad más habituales, pensadas para ejecutarse en Python sobre un *backend*, y obliga a trabajar con las posibilidades que ofrece el cliente. Para abordarlo se han combinado dos enfoques complementarios, por un lado métodos agnósticos al modelo basados en la perturbación de la entrada, como SHAP, y por otro métodos específicos que analizan el interior de la red propagando la relevancia capa a capa, como LRP.

1.2. Objetivos

El objetivo general de este TFG es el desarrollo de los módulos para la implementación de la determinación de la explicabilidad de modelos de aprendizaje automático, integrándose con la plataforma *Nets4Learning*.

Para alcanzar esta meta, se han definido los siguientes objetivos específicos:

1. **Estudio del campo del aprendizaje automático, su definición y aspectos clave.** A la hora de explicar un modelo se debe entender como dicho modelo funciona de forma interna, además de comprender cómo este aprende y evoluciona. Es por ello que a la hora de realizar este proyecto se requiere un estudio de los conceptos básicos de inteligencia artificial, más concretamente los relacionados con aprendizaje automático, redes neuronales y *deep learning*.
2. **Estudio del concepto de inteligencia artificial explicable, ámbitos y metodologías.** Con el fin de elegir bien las técnicas a implementar, se deben analizar las

distintas familias de métodos de explicabilidad. Al analizarlas se puede comparar y entender tanto sus desventajas como sus ventajas respecto a otras técnicas.

3. **Análisis de tecnologías de desarrollo en aprendizaje automático.** Este proyecto se desarrolla en un entorno de ejecución de navegador por lo que es esencial estudiar las distintas tecnologías en la web que nos permiten entrenar y ejecutar modelos en el lado del cliente, así como la explicabilidad que es el enfoque principal en este proyecto.

4. **Desarrollo de la plataforma objetivo e integración con *Nets4Learning*.** Por último, diseñamos, implementamos y probamos los módulos de explicabilidad (SHAP y LRP) para distintas tareas de la plataforma. Dicha plataforma objetivo se debe de integrar con la plataforma ya existente de *Nets4Learning*, respetando su filosofía de ejecución en el lado del cliente y sin depender de ningún servidor.

1.3. Estructura de la memoria

Este documento se estructura siguiendo el ciclo de vida de un proyecto de ingeniería software:

- El **Capítulo 2** recoge la especificación del trabajo, desde la situación de partida y los antecedentes hasta las restricciones, la metodología, las tecnologías, la planificación y el presupuesto.
- El **Capítulo 3** especifica los requisitos funcionales y no funcionales del sistema, junto con sus casos de uso, uno por cada incremento.
- El **Capítulo 4** aborda la arquitectura del sistema y el diseño general de la solución.
- El **Capítulo 5** describe el desarrollo del módulo, organizado por incrementos, de forma que cada uno presenta un caso de uso con su diseño detallado junto a las pruebas sobre el mismo.
- Finalmente, el **Capítulo 6** expone las conclusiones y las líneas de trabajo futuro.

Capítulo 2

Especificación del trabajo

En este capítulo describimos el punto de partida del proyecto, es decir, el sistema sobre el que vamos a construir y las carencias que hemos detectado en él y que justifican el trabajo. Antes de plantear ningún requisito conviene entender bien qué ofrecía ya la plataforma *Nets4Learning* y en qué presentaba carencias, dado que esto limita el rango de las cosas que se pueden hacer en el proyecto.

2.1. Descripción de la situación de partida

El TFG parte de una herramienta desarrollada por el grupo de investigación SIMIDAT de la Universidad de Jaén. El proyecto a realizar consiste en la ampliación de la misma, heredando por consiguiente tanto sus posibilidades como sus limitaciones. En los siguientes apartados describimos el estado de la plataforma antes de la ampliación, junto a las carencias que se pueden observar en dicho estado.

2.1.1. Descripción del entorno actual: *Nets4Learning*

Nets4Learning es una plataforma web para el diseño, el entrenamiento y la prueba de modelos de redes neuronales artificiales y profundas [1], cuyo código está disponible de forma abierta [2] y cuya versión oficial se encuentra desplegada en los servidores del grupo SIMIDAT[1]. Está pensada para acercar el aprendizaje automático a un público menos especializado, de forma que cualquier persona, sea cual sea su disciplina, pueda entrenar modelos, hacer predicciones y ver cómo se comportan sin

[1] https://simidat.ujaen.es/n4l/

necesidad de tener hardware especializado ni conocimientos de programación, ya que basta con abrirla desde el navegador de cualquier dispositivo.

Como característica que define a *Nets4Learning* tenemos su arquitectura. Se trata de una aplicación web de página única (SPA, *Single Page Application*) construida con React, con la peculiaridad de que el entrenamiento y la inferencia de los modelos se hacen completamente en el lado del cliente mediante TensorFlow.js [3], de forma que se ejecutan aprovechando la GPU del dispositivo que renderiza la página. Esto permite que la herramienta funcione sin ningún tipo de infraestructura de servidor, respetando la privacidad de la información del usuario y permitiendo el acceso desde cualquier equipo moderno.

La plataforma plantea su contenido en torno a cuatro tareas clásicas de la ciencia de datos, accesibles por el usuario desde la pantalla de inicio, tal y como se ve en la Figura 2.1:

- **Clasificación tabular.** La clasificación tabular consiste en determinar la clase a la que pertenece un elemento a partir de las características, también conocidas como features, de dicho elemento. Incluye conjuntos de datos conocidos como Iris, Car o Lymphography.
- **Regresión.** En el caso de la regresión, no se determina una clase, sino que se predice un valor numérico continuo, sea el consumo de un vehículo o la calidad de un vino.
- **Clasificación de imágenes.** La clasificación de imágenes se divide en dos modelos principales que son el MNIST y modelos como MobileNet.
- **Identificación de objetos.** Reúne varios detectores ya entrenados, como COCO-SSD para objetos cotidianos, detectores y mallas faciales, estimación de la postura del cuerpo o reconocimiento de signos con la mano, que pueden trabajar sobre imágenes o sobre la cámara web en tiempo real.

Para cada una de estas tareas, *Nets4Learning* trae consigo las siguientes funcionalidades:

- **Diseñar y entrenar modelos propios.** Esta funcionalidad permite al usuario crear sus propios modelos de redes, con su editor de capas, neuronas, funciones de activación, editor de hiperparámetros, número de epochs, optimizador, función de pérdida, todo ello a elección del usuario. Una vez creado el modelo puede entrenarlo en la propia página y usarlo para predicciones.

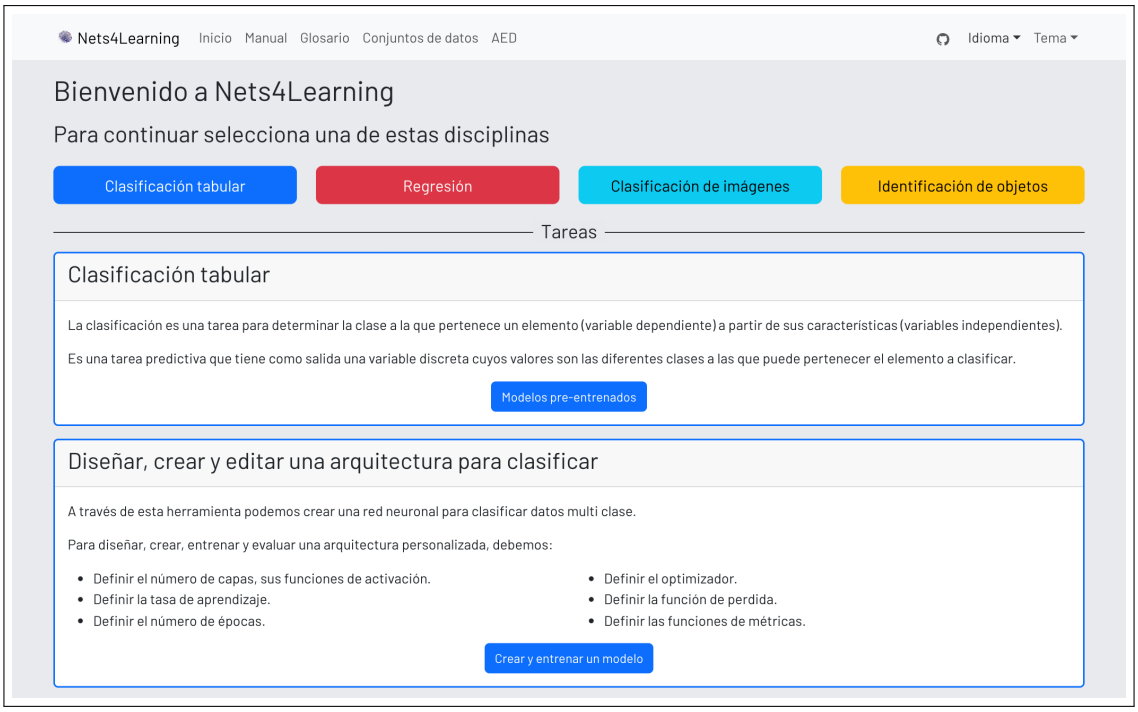


Figura 2.1: Pantalla de inicio de *Nets4Learning*, en la que el usuario elige una de las cuatro tareas disponibles.

- **Utilizar modelos ya entrenados.** En caso de querer predecir directamente con un modelo que esté entrenado de antemano, la plataforma tiene modelos preentrenados los cuales están explicados junto a su conjunto de datos, además de permitir la introducción de nuevas instancias tal como se ve en esta figura 2.3.
- **Ayuda al usuario.** Dado que esta herramienta esta orientada a la enseñanza, se cuentan con explicaciones variadas (Figura 2.4) de los distintos elementos que componen la aplicación.
- **Varios idiomas.** La interfaz está disponible en español y en inglés.

Además de la explicabilidad, como añadido extra de la ampliación hemos añadido el japonés como tercer idioma de la interfaz, traduciendo también todos los textos nuevos del módulo de explicabilidad.

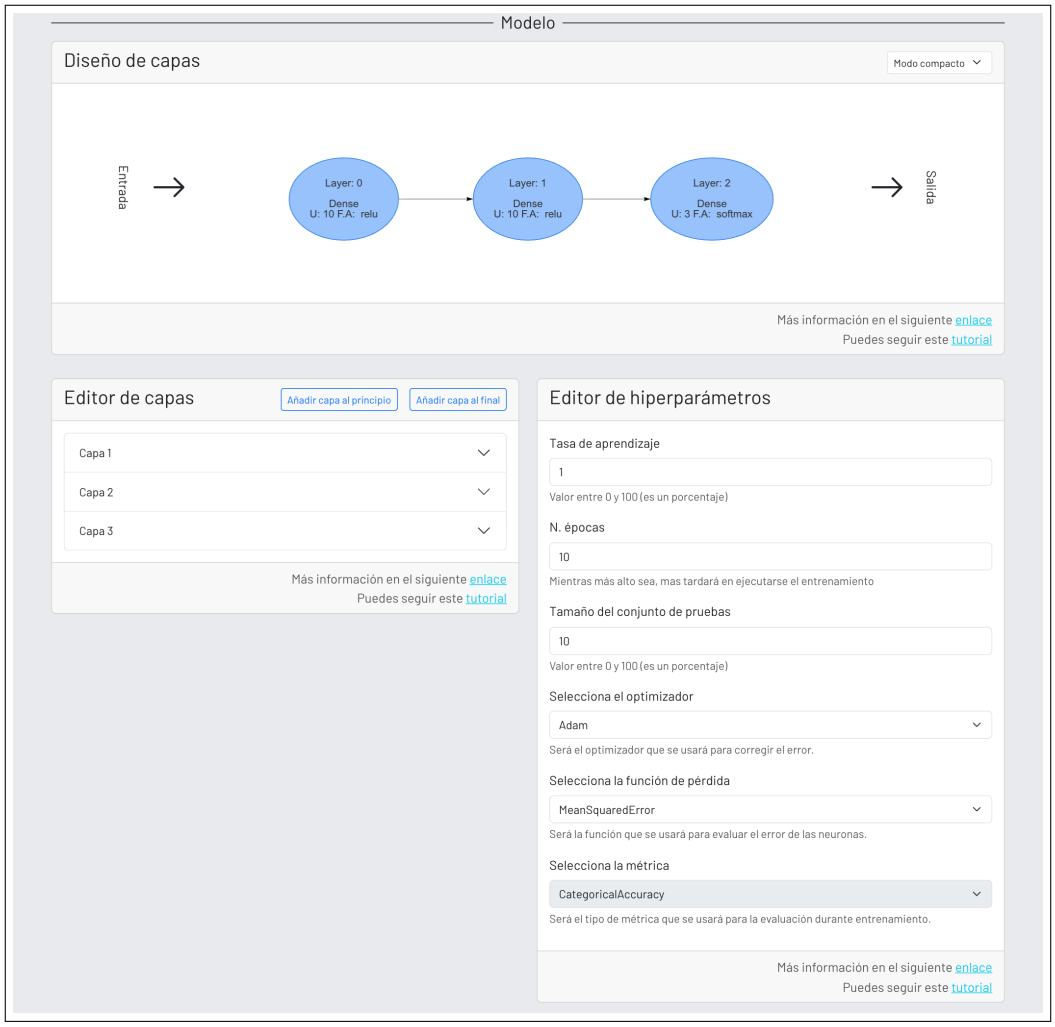


Figura 2.2: Diseño y entrenamiento de un modelo propio en *Nets4Learning*: esquema de la red, editor de capas y editor de hiperparámetros.

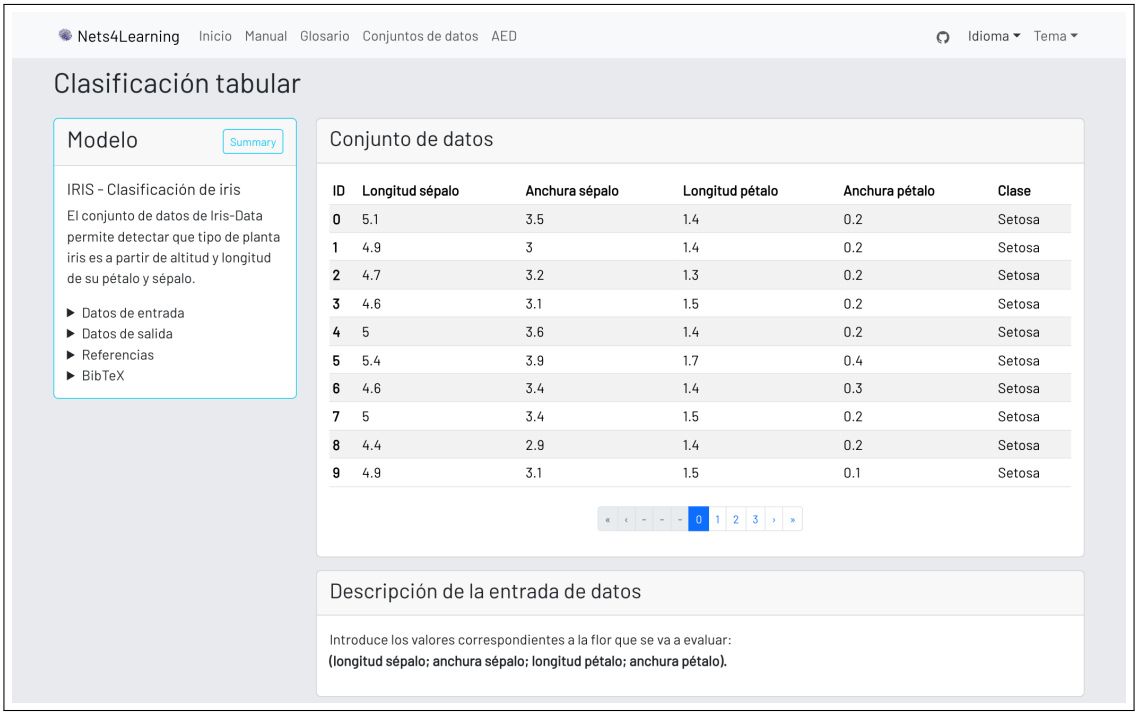


Figura 2.3: Uso de un modelo ya entrenado sobre el conjunto Iris, con la descripción del modelo y los datos disponibles.

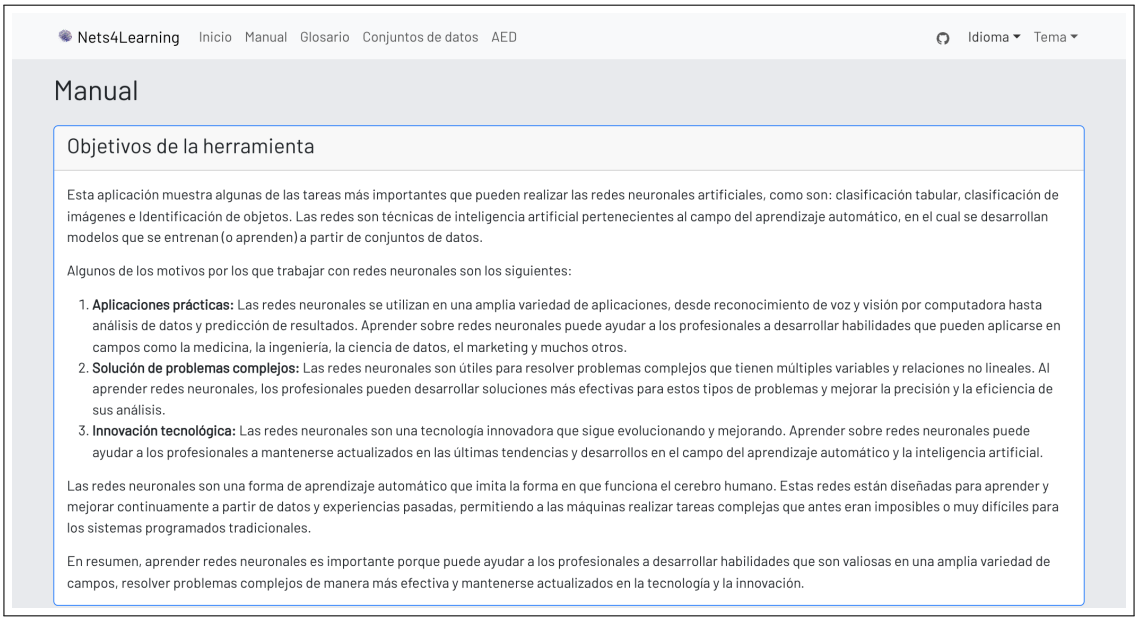


Figura 2.4: Manual de la plataforma, uno de los recursos de ayuda que ofrece *Nets4Learning* junto al glosario y los tutoriales guiados.

2.1.2. Resumen de las deficiencias y carencias identificadas

La plataforma *Nets4Learning* cubre de forma adecuada la parte de entrenamiento y predicción. Sin embargo, aun siendo el aprendizaje el objetivo principal de la plataforma, se podría considerar que carece de algún método para explicar el comportamiento de los modelos.

Especialmente en un contexto educativo, dicha carencia es realmente crítica dada la necesidad de que un estudiante comprenda de verdad cómo la red neuronal toma las decisiones. Conocer qué características o regiones de una imagen motivan la decisión de una red neuronal permite al estudiante la interpretación de los resultados. Veamos un ejemplo con datos tabulares. En el conjunto Iris saber que una flor se ha clasificado como Iris Versicolor no es suficiente, dado que lo interesante es descubrir que la longitud del pétalo es la característica que más ha contribuido a esa decisión.

De dicha carencia surge la necesidad de implementar un módulo de Inteligencia Artificial Explicable (XAI) de forma que cada predicción tenga una explicación visual interpretable por el usuario. A la hora de implementar el módulo XAI se debe mantener la estructura actual de *Nets4Learning*. Al ejecutarse en el dispositivo del usuario, no podemos hacer uso de las librerías de explicabilidad ya implementadas en Python y pensadas para ejecutarse en un servidor, lo que impone la principal restricción del proyecto y que detallamos más adelante, en la Sección 2.3.

2.2. Antecedentes

Este apartado recoge los fundamentos sobre los que se construye el proyecto. Los modelos de aprendizaje automático presentan un problema de opacidad, y para suplir esta carencia utilizamos la Inteligencia Artificial Explicable. Antes de entrar en ella definimos brevemente los conceptos básicos de los que partimos, y después presentamos las técnicas utilizadas en este proyecto además de analizar su rendimiento y ejecución en el navegador, entorno principal de este proyecto.

2.2.1. Conceptos básicos

Dado que a lo largo de la memoria se habla continuamente de inteligencia artificial, aprendizaje automático y de redes neurones, debemos definir dichos términos de

forma que quede claro qué entendemos por cada uno de estos.

- **Inteligencia Artificial (IA).** Es la disciplina que estudia el diseño de agentes capaces de percibir su entorno y de actuar sobre él de forma racional, es decir, tomando en cada momento la acción que mejor les permite alcanzar sus objetivos [4].
- **Aprendizaje automático** (*machine learning*). Es la rama de la IA que se ocupa de los programas que mejoran con la experiencia, de modo que en lugar de programar las reglas a mano, el sistema las aprende a partir de los datos. De forma más concreta, se dice que un programa aprende de una experiencia E respecto a una tarea T y una medida de rendimiento P si su rendimiento en T, medido por P, mejora con E [5].
- **Redes neuronales artificiales.** Son modelos de aprendizaje automático formados por unidades sencillas (neuronas) organizadas en capas y conectadas entre sí mediante pesos, de forma que cada neurona combina sus entradas y aplica una función de activación no lineal. Durante el entrenamiento, esos pesos se van ajustando para que la salida de la red se parezca cada vez más a la esperada [6].
- **Redes neuronales profundas y aprendizaje profundo** (*deep learning*). Las redes neuronales profundas con aprendizaje profundo permite la creación de modelos computacionales compuestos por múltiples capas que aprenden representaciones de datos con múltiples niveles de abstracción [7].

2.2.2. La Inteligencia Artificial Explicable (XAI)

El término “caja negra” se utiliza para describir aquellos modelos cuyo funcionamiento interno resulta inaccesible o incomprensible para un humano. Las redes neuronales profundas presentan millones de parámetros interconectados que hacen imposible el trabajo de análisis manual por parte de un humano.

La Inteligencia Artificial Explicable, o XAI (*Explainable Artificial Intelligence*), agrupa el conjunto de técnicas orientadas a hacer comprensibles las decisiones de estos modelos [8]. Dentro de este campo es habitual clasificar las técnicas según varios criterios. En función del alcance, se distinguen entre explicaciones *globales*, que describen el comportamiento del modelo en su conjunto, y *locales*, que se centran en una predicción concreta [9]. Según su relación con el modelo, tenemos tanto métodos

agnósticos como *específicos*. Los *agnósticos* tratan al modelo como una caja negra, centrándose en relacionar sus entradas con sus salidas e ignorando por completo su arquitectura interna, como ocurre con los métodos basados en los valores de Shapley [10, 11]. Por otro lado, los *específicos* funcionan con base en la arquitectura interna del modelo, por lo que solo sirven para un tipo concreto de modelos.

2.2.3. Técnicas de explicabilidad

Métodos agnósticos: LIME y SHAP

Los métodos agnósticos se basan en observar como reacciona el modelo ante entradas perturbadas, es decir, cambiar datos de entrada. De esta forma, se van sustituyendo las características permitiendo observar cómo cambia la predicción y deducir que características son las que más influyen en la decisión tomada. Dado que lo único necesario es obtener predicciones, son aplicables a cualquier tipo de modelo existente.

El primero de ellos sería LIME (*Local Interpretable Model-agnostic Explanations*) [12], que popularizó el enfoque agnóstico. LIME plantea que no importa si el modelo es muy complejo en general ya que, dada una instancia concreta, se puede aproximar por un modelo más sencillo como la regresión lineal. Dado que el resultado depende de como se generan esas muestras de una instancia concreta, dos ejecuciones pueden dar explicaciones distintas.

SHAP (*SHapley Additive exPlanations*) está basado en los valores de Shapley de la teoría de juegos cooperativos [10, 11], consistiendo en repartir de forma justa la contribución de cada característica a la predicción final, considerando todas las combinaciones posibles de características presentes y ausentes, es decir, comparamos excluyendo ciertas características con el fin de obtener sus contribuciones marginales. En particular, KernelSHAP, la variante que usamos en este proyecto, se puede ver como una versión de LIME en la que la ponderación de las muestras se elige de forma que los resultados coincidan con los valores de Shapley, lo que le da unas garantías teóricas de las que LIME carece [11]. Sin embargo, presenta un coste computacional elevado dado que para una sola predicción requerimos de la evaluación del modelo durante varias iteraciones. Dicho coste computacional será el principal obstáculo al implementarlo en un entorno donde el cliente debe aportar la potencia de computación.

Métodos específicos: LRP

Frente a los anteriores, LRP (*Layer-wise Relevance Propagation*) es un método específico que sí accede al interior de la red [13, 14]. LRP retropropaga la "relevancia" a través de cada una de las capas, hasta llegar a la entrada, permitiéndonos saber qué elementos aportaron más a la predicción. Es mucho más eficiente que SHAP dado que es una sola ejecución, pero a cambio está estrechamente relacionado con la arquitectura del modelo ya que cada tipo de capa requiere su propia regla de propagación, limitando el uso a modelos cuya estructura interna sea completamente accesible.

Segmentación en superpíxeles

Cuando la entrada es una imagen, atribuir importancia a cada píxel de forma individual resulta en explicaciones ruidosas y muy costosas. Una solución habitual es agrupar previamente los píxeles en regiones coherentes, comúnmente referidas como superpíxeles, y trabajar sobre esas regiones. Para poder llevar a cabo dicha agrupación, hacemos uso del algoritmo SLIC (*Simple Linear Iterative Clustering*), que segmenta la imagen en regiones de tamaño y forma regulares en función de la proximidad y la similitud de color [15]. De este modo, las explicaciones se calculan sobre un número manejable de regiones con significado visual, en lugar de sobre miles de píxeles aislados.

2.3. Restricciones

Este proyecto presenta límites marcados por el tiempo disponible para el desarrollo y la propia plataforma *Nets4Learning*. En este apartado se recogen las principales restricciones del proyecto de implementación del módulo de explicabilidad, las cuales explican decisiones tomadas durante el desarrollo del proyecto.

2.3.1. Restricciones temporales

El proyecto se desarrolla con una carga de trabajo previamente definida y un plazo de entrega fijo. Se ha desarrollado a lo largo de dos cuatrimestres. La limitación temporal ha obligado a priorizar la implementación del módulo de explicabilidad en las tareas más representativas.

2.3.2. Restricción de ejecución en el cliente

La restricción más determinante del proyecto no es temporal, sino técnica, y la heredamos directamente de *Nets4Learning*. Toda la plataforma funciona en el lado del cliente, sin ningún *backend* que procese los datos, y el módulo de explicabilidad debe respetar esa misma filosofía. Esto tiene las siguientes consecuencias:

- Quedan descartadas las bibliotecas de explicabilidad más habituales y potentes, como las de Python (SHAP, LIME, etc) que requieren de un servidor para funcionar en la Web. Nos obligamos, por tanto, a apoyarnos en soluciones nativas del navegador, como WebSHAP y TensorFlow.js.
- El cómputo depende de la GPU del propio equipo del usuario a través de WebGL, con unos límites de memoria y de rendimiento mucho más estrictos que los de un servidor.
- Si bien las restricciones presentan varios problemas, por otro lado nos presentan dos ventajas principales: la privacidad (los datos del usuario nunca salen de su equipo) y la accesibilidad (no hay que instalar nada ni disponer de hardware especializado).

2.3.3. Restricciones de integración

Al tratarse de una ampliación y no de un sistema nuevo, el módulo debe integrarse en la arquitectura existente de *Nets4Learning* respetando lo que ya funcionaba. Esto implica mantener las decisiones estructurales de la plataforma, además de adaptar todo al soporte multilingüe que ya incorporaba.

2.4. Metodología de desarrollo software

Para el desarrollo del proyecto se debe decidir qué metodología aplicar. De entre las varias metodologías de desarrollo software que se han planteado para el desarrollo del proyecto, expondremos las más habituales, contando sus inconvenientes y sus ventajas. Se justificará la elección final y que se ha sopesado en cada una de las metodologías contempladas.

- **Modelo en cascada.** Plantea el desarrollo como una sucesión de fases que se recorren una detrás de otra. El problema principal es que cualquier ajuste o cambio realizado obliga a rehacer una gran parte de lo anteriormente desarrollado.
- **Modelos iterativos e incrementales.** En lugar de entregarlo todo al final, se van completando incrementos, y cada uno aporta una porción funcional del sistema que ya se puede probar y validar por separado.
- **Metodologías ágiles.** Llevan esa idea un paso más allá con iteraciones cortas y una adaptación continua a los cambios y a la realimentación, siendo Scrum una de las más conocidas. Esta fue una candidata muy prometedora.

De entre todas, hemos optado por un **desarrollo incremental**. En este modelo el producto no se construye de una sola vez, este se hace de forma progresiva iterando versiones denominadas incrementos. Es importante entender que cada incremento debe representar una parte del sistema que sea funcional y que puede entregarse sin el resto de incrementos. De esta forma, el proyecto se reduce a piezas más pequeñas que se desarrollan en orden, y dentro de cada una se recorren las fases de análisis, diseño, implementación y pruebas.

En nuestro caso, dadas las siguientes razones hemos elegido hacer uso de la metodología incremental.

- **Descomposición natural.** El módulo de explicabilidad ya estaba formado, de por sí, por submódulos independientes —uno por cada tarea de la plataforma—, y encajan como incrementos sin tener que forzar ninguna división artificial.
- **Entrega y validación tempranas.** Como cada incremento es funcional por sí mismo, podemos probarlo y validarlo en cuanto está listo, sin necesidad de esperar a que el proyecto entero esté terminado.
- **Reducción de riesgos.** Al ir entregando en partes pequeñas, los problemas se detectan y se corrigen en etapas tempranas, cuando arreglarlos es mucho menos costoso que al final.
- **Flexibilidad ante los cambios.** Cualquier ajuste que surja se puede incorporar en los incrementos siguientes con poco coste, sin obligarnos a rehacer lo que ya estaba hecho.
- **Reflejo del desarrollo real.** Por último, esta forma de trabajar coincide con cómo se construyó de verdad el módulo, porque fuimos incorporando la explicabilidad

tarea a tarea a lo largo de los dos cuatrimestres, empezando por los datos tabulares y terminando con LRP.

Sobre esta base incremental nos hemos apoyado, además, en prácticas habituales del desarrollo ágil, como el control de versiones con Git y las revisiones periódicas con los tutores. Más adelante, en el capítulo de desarrollo, describimos cada uno de estos incrementos por separado, de forma que cada uno de ellos desarrolla uno de los casos de uso del sistema.

2.5. Estimación del tamaño y el esfuerzo

La estimación del tamaño y el esfuerzo de este proyecto se apoya en dos enfoques complementarios, las líneas de código (LOC) para el tamaño y el modelo COCOMO para el esfuerzo. Conviene tener presente que se trata de una arquitectura que funciona por completo en el lado del cliente, así que no hay componentes de servidor ni persistencia en base de datos que estimar.

2.5.1. Estimación del tamaño: líneas de código (LOC)

Para estimar el tamaño partimos del código realmente escrito para el módulo de explicabilidad, agrupado por bloques funcionales. La Tabla 2.1 recoge una medida aproximada de cada uno de ellos.

A esas cifras hay que sumar la integración del módulo en la interfaz ya existente de *Nets4Learning* (controles para lanzar las explicaciones, paneles de resultados y adaptación multilingüe), que está repartida por las distintas páginas de la plataforma, con lo que el tamaño total del módulo ronda las $3{,}5$ KLOC.

Ahora bien, para estimar el esfuerzo no tiene sentido tratar todo ese tamaño como código escrito desde cero, porque no lo es. Por un lado, las bibliotecas externas en las que nos apoyamos (WebSHAP y TensorFlow.js) son dependencias que no hemos programado nosotros y, por tanto, no cuentan. Por otro lado, la segmentación en superpíxeles (unas $1{,}1$ KLOC entre SLIC y SLIC0) parte de implementaciones previas que únicamente hemos adaptado. Descontando esa parte reutilizada, el código realmente desarrollado en este trabajo ronda las $2{,}1$ **KLOC**, que es la cifra sobre la que tiene sentido aplicar el modelo de esfuerzo.

Bloque funcional	LOC aprox.
Muestreo y preparación de datos para SHAP	100
Funciones predictoras y adaptadores de los modelos	420
Segmentación en superpíxeles (SLIC, SLIC0 y segmentación facial)	1230
Propagación de relevancia (LRP) y captura de activaciones	600
Coordinación del cálculo de la explicación en cada tarea	430
Visualización (mapas de calor, gráficos de importancia y diagrama de enjambre)	440
Total en módulos dedicados	$\approx$ **3220**

Tabla. 2.1: Tamaño aproximado del módulo de explicabilidad, en líneas de código.

2.5.2. Estimación del esfuerzo: modelo COCOMO

Para estimar el esfuerzo utilizamos el modelo COCOMO en su versión básica, que relaciona el tamaño del software con el esfuerzo y la duración mediante un par de expresiones. Dado que se trata de un proyecto pequeño, desarrollado por una sola persona y sin requisitos especialmente rígidos, encaja en el modo **orgánico**, cuyas fórmulas son:

$$E = 2{,}4 \cdot (\text{KLOC})^{1{,}05} \qquad T = 2{,}5 \cdot E^{0{,}38}$$

donde E es el esfuerzo en personas-mes, T la duración en meses y KLOC los miles de líneas de código. Sustituyendo el tamaño del código realmente desarrollado ($2{,}1$ KLOC) obtenemos los valores de la Tabla 2.2.

Magnitud	Valor
Esfuerzo estimado (E)	$\approx$ 5,2 personas-mes
Duración estimada (T)	$\approx$ 4,7 meses
Personal medio (E/T)	$\approx$ 1,1 personas

Tabla. 2.2: Estimación del esfuerzo mediante COCOMO básico (modo orgánico).

Estas cifras deben interpretarse con cierta cautela. Un esfuerzo de $\approx$ 5,2 personas-mes equivale, a razón de 152 horas por persona-mes, a unas 795 horas de trabajo, que siguen estando por encima de las aproximadamente 300 horas de dedicación que

corresponden a los 12 ECTS del TFG. Esta diferencia es esperable, porque COCOMO está calibrado para desarrollo profesional a tiempo completo y tiende a sobreestimar los proyectos pequeños llevados por una sola persona. Aun así, la estimación resulta útil, porque confirma que el código desarrollado tiene entidad suficiente para un Trabajo Fin de Grado, y el hecho de haberlo podido completar dentro del calendario de dos cuatrimestres se explica, además, por el apoyo en la base ya existente de *Nets4Learning* y en herramientas como WebSHAP o TensorFlow.js.

2.6. Tecnologías y herramientas utilizadas

En el desarrollo de este proyecto, a la hora de elegir tecnologías debíamos tener en cuenta las tecnologías ya implementadas. Dado que se trata de una ampliación de *Nets4Learning*, el trabajo se hace sobre las tecnologías ya existentes. Estas son tecnologías web que permiten ejecutar modelos directamente en el lado del cliente, y por supuesto, es tanto la filosofía del proyecto como el elemento más importante a la hora de estudiar las tecnologías a implementar por encima del proyecto.

2.6.1. Tecnologías

- **React.** Biblioteca de JavaScript sobre la que está construida la interfaz de *Nets4Learning*, organizada como una aplicación web de página única (SPA). El módulo de explicabilidad se implementa en React como componentes de interfaz.
- **JavaScript y TypeScript.** Estos lenguajes son los usados en el proyecto. Tras una actualización de la plataforma se hace el cambio a TypeScript donde se tiene un tipado especial que permite mantener código más robusto y de mayor calidad.
- **TensorFlow.js.** Es el elemento más importante de todo el proyecto, dado que permite la implementación de modelos de aprendizaje automático para uso en navegador sin necesidad de *backend* [3].
- **WebGL.** Tecnología web que TensorFlow.js utiliza por debajo para acelerar los cálculos en la GPU del cliente.
- **WebSHAP.** Implementación de SHAP escrita en JavaScript y TypeScript y pensada para ejecutarse por completo en el navegador [16]. Gracias a WebSHAP no requerimos de Python ni implementaciones complejas para poder usar los valores de Shapley en el módulo de explicabilidad.

2.6.2. Herramientas de desarrollo

Para la gestión y el desarrollo se ha hecho uso de únicamente herramientas de código abierto o gratuitas:

- **Git y GitHub.** Control de versiones del código, tanto de la plataforma como de esta memoria, lo que nos ha permitido trabajar por ramas en cada incremento y volver atrás cuando algo dejaba de funcionar.
- **SourceTree.** Cliente visual de Git, con el que resulta más cómodo revisar el historial y los cambios antes de confirmarlos.
- **Visual Studio Code.** Entorno de desarrollo utilizado para escribir y depurar todo el código.
- **Vitest.** Herramienta de pruebas unitarias que viene integrada con el entorno de construcción de la plataforma.
- **LaTeX.** Sistema de composición de textos utilizado, a través de Latex en Visual Studio Code, para la redacción de esta memoria.

2.6.3. Ejecución de modelos en el navegador

Como se menciona en numerosas ocasiones a lo largo de esta memoria, la ejecución de modelos en el navegador es la filosofía de este proyecto. Históricamente, el aprendizaje automático se ha asociado a lenguajes como R [17] y, sobre todo, Python, que sigue siendo a día de hoy el lenguaje más importante tanto para entrenar los modelos como para ejecutarlos, siendo PyTorch [18] y TensorFlow [19], ya sea solo o junto con Keras [20] las plataformas más utilizadas para ejecutarlos en un servidor. Gracias a la aparición de TensorFlow.js se puede desarrollar este proyecto permitiendo el entrenamiento y ejecución de modelos directamente en el navegador.

Las bibliotecas de XAI más extendidas están escritas en Python y dan por hecho la presencia de un *backend*, así que no se pueden utilizar directamente en el navegador. Esto es la mayor ventaja y limitación que tiene el proyecto actualmente. Sin embargo, dicha limitación puede superarse con el uso de WebSHAP, la cual es una implementación de SHAP hecha para ser ejecutada en el lado del cliente, asegurando la privacidad de los datos.

A pesar de todo, el panorama de los modelos en el navegador sigue cambiando, y recientemente se ha empezado a extender el uso de WebGPU, que permite trabajar con modelos mucho más grandes mediante nuevos motores como ONNX Runtime Web o Transformers.js.

2.7. Planificación

El proyecto se ha desarrollado a lo largo de dos cuatrimestres. El primero, más denso, concentró la investigación inicial y los primeros incrementos de explicabilidad basada en SHAP (datos tabulares, regresión, imágenes y detección de objetos), mientras que el segundo se dedicó sobre todo a la explicabilidad específica mediante LRP y a mejorar la aplicación base. La redacción de la memoria se ha llevado a cabo de forma transversal durante todo el periodo, apoyándonos en las herramientas descritas en la Sección 2.6.

La Tabla 2.3 resume las fases e incrementos en que se ha dividido el proyecto, con su duración aproximada, las actividades principales de cada uno y los resultados obtenidos.

Fase / Incremento	Duración	Actividades principales	Resultados
Investigación y formación	Meses 1–2	Estudio de la XAI y de las técnicas SHAP y LRP; familiarización con React y TensorFlow.js	Base teórica y dominio del entorno de trabajo
Análisis del sistema existente	Mes 2	Estudio del código de *Nets4Learning* y de cómo se utilizan sus modelos	Comprensión necesaria para integrar la explicabilidad
Incremento 1: clasificación tabular	Mes 3	Integración de WebSHAP y explicación por perturbaciones sobre datos tabulares	Explicabilidad SHAP en clasificación tabular
Incremento 2: regresión	Meses 3–4	Extensión de la explicabilidad tabular a los modelos de regresión	Explicabilidad en regresión
Incremento 3: clasificación de imágenes	Meses 4–5	Segmentación en superpíxeles (SLIC) y visualización mediante mapas de calor	Explicabilidad SHAP sobre imágenes
Incremento 4: detección de objetos	Meses 5–6	Segmentación facial y por rejilla (SLIC0) y enmascarado por desenfoque	Explicabilidad en detección de objetos
Incremento 5: LRP	Meses 6–8	Propagación de relevancia capa a capa y captura de activaciones	Explicabilidad específica sobre modelos con estructura accesible
Pruebas y validación	Meses 8–9	Pruebas unitarias y validación funcional de cada incremento	Módulo validado
Redacción de la memoria	Transversal	Documentación continua del trabajo	Memoria completa

Tabla. 2.3: Planificación temporal del proyecto por fases e incrementos.

Partiendo de esta tabla, y con una dedicación relativamente constante, se aprecia que cada mes se ha centrado en una fase o en un incremento concreto. En primer lugar se debía investigar y analizar el funcionamiento de sistema. A continuación tocaban los incrementos de SHAP, por el que comenzamos a desarrollarlo para la clasificación tabular siguiendo con las imágenes y la detección de objetos. El segundo cuatrimestre se centra de forma exclusiva en la implementación de LRP y su lógica subyacente dado que presenta una complejidad mayor y debe ser tratado como tal. Por otro lado, la memoria se escribe a lo largo de todo el tiempo de desarrollo. Esta distribución se resume en el diagrama de Gantt de la Figura 2.5.

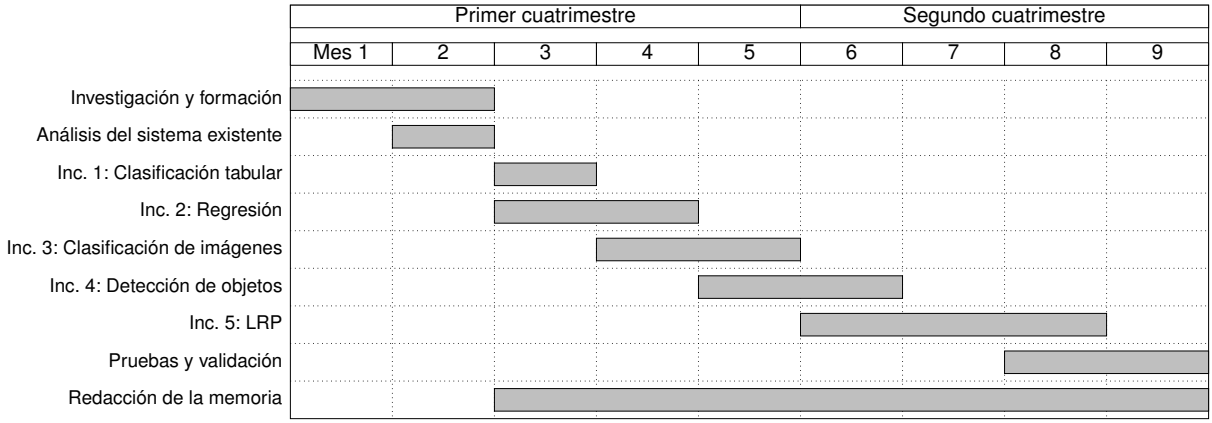


Figura 2.5: Diagrama de Gantt con la planificación temporal del proyecto a lo largo de los dos cuatrimestres.

2.8. Presupuesto

A lo largo del desarrollo del proyecto no se hace uso de herramientas de pago, sino únicamente de herramientas y bibliotecas de código abierto, con lo que el coste en licencias de software es nulo. El presupuesto se concentra, por tanto, en los costes de personal y en el equipo informático necesario para el desarrollo.

2.8.1. Costes de personal

Se estima la dedicación del proyecto en torno a las horas asociadas a la carga en créditos ECTS del TFG. Para fijar el coste por hora de cada rol hemos partido de referencias salariales reales y le hemos sumado la cotización a la Seguridad Social que corre a cargo del empleador, con el fin de obtener el coste real para la empresa y no solo el sueldo que percibe el trabajador:

- **Desarrollador.** El salario de un programador junior en España se sitúa entre los 22.000 y los 28.000 € brutos anuales, con una mediana en torno a los 25.000 € [21, 22]. Sumando aproximadamente un 30 % de cotizaciones, el coste anual para la empresa se acerca a los 32.400 € [21], de forma que al dividirlo entre unas 1.800 horas de trabajo al año dan un coste aproximado de 18 € por hora.
- **Tutores.** Partiendo de unos 40.300 € brutos anuales [23], y teniendo en cuenta un 30de la cotización a cargo de la universidad, el coste total se sitúa en unos 52.390 € anuales. Al dividirlo entre 1.800 horas, resulta un coste aproximado de 30 € por hora.

La Tabla 2.4 recoge el desglose de este coste.

Rol	Horas	Coste/hora	Total
Desarrollador (estudiante)	360	18,00 €	6.480,00 €
Supervisión (tutores)	30	30,00 €	900,00 €

Tabla. 2.4: Estimación de costes de personal.

2.8.2. Costes de hardware y software

Como recurso material se hace uso de un portátil donde se desarrolla el proyecto. La Tabla 2.5 recoge estos costes.

Concepto	Descripción	Total
Equipo informático	Amortización del portátil de desarrollo durante el periodo del proyecto	150,00 €
Software	Herramientas de código abierto y gratuitas	0,00 €

Tabla. 2.5: Estimación de costes de hardware y software.

2.8.3. Coste total

Sumando los conceptos anteriores, el coste total estimado del proyecto asciende a **7.600,00 €**, tal y como se refleja en la Tabla 2.6.

Concepto	Total
Costes de personal	7.450,00 €
Costes de hardware y software	150,00 €
Total	**7.600,00 €**

Tabla. 2.6: Coste total estimado del proyecto.

Capítulo 3

Análisis de requisitos

En este capítulo especificamos los requisitos del módulo de explicabilidad. Hay que tener en cuenta que nuestro sistema no es una aplicación independiente, sino una ampliación de la plataforma *Nets4Learning*, y por eso los requisitos los hemos definido pensando tanto en la funcionalidad que debe aportar el nuevo módulo como en las restricciones que impone integrarse en un sistema que ya existe. A continuación los recogemos primero como requisitos funcionales y no funcionales, y después los concretamos en sus correspondientes casos de uso, de forma que definimos un caso de uso por cada uno de los incrementos en los que se ha dividido el desarrollo y que describimos en el Capítulo 5.

3.1. Requisitos funcionales

Los requisitos funcionales recogen las capacidades concretas que el módulo tiene que ofrecer, y los resumimos en la Tabla 3.1.

3.2. Requisitos no funcionales

Los requisitos no funcionales, por su parte, establecen las condiciones de calidad y las restricciones bajo las que tiene que funcionar el sistema, y los recogemos en la Tabla 3.2.

ID	Descripción
RF-1	El usuario podrá solicitar una explicación de la predicción realizada por un modelo previamente entrenado o cargado.
RF-2	El sistema generará explicaciones mediante un método agnóstico al modelo (SHAP) para las tareas de clasificación y regresión de datos tabulares, clasificación de imágenes y detección de objetos.
RF-3	El sistema generará explicaciones mediante un método específico del modelo (LRP) para los modelos de imagen cuya arquitectura sea accesible.
RF-4	En las tareas de imagen, el sistema segmentará la imagen en superpíxeles antes de calcular la explicación.
RF-5	El sistema presentará las explicaciones de forma visual, mediante gráficos de importancia de características para datos tabulares y mapas de calor superpuestos para imágenes.
RF-6	El usuario podrá ajustar los parámetros de la explicación, como el número de muestras empleadas por SHAP o la granularidad de la segmentación.
RF-7	El sistema deshabilitará automáticamente los métodos de explicación no aplicables al modelo seleccionado.
RF-8	La explicación se calculará sobre la clase predicha o sobre las etiquetas seleccionadas por el usuario.

Tabla. 3.1: Requisitos funcionales del sistema.

3.3. Casos de uso

El actor principal del sistema es el **usuario** de la plataforma, que normalmente será un estudiante o cualquier persona que quiere entender cómo se comporta un modelo y por qué toma sus decisiones. A la hora de definir los casos de uso hemos seguido la misma división que en el desarrollo, de forma que cada caso de uso se corresponde con uno de los incrementos y contiene una funcionalidad completa que el usuario puede llevar a cabo por sí sola.

ID	Descripción
RNF-1	**Ejecución en cliente.** Todo el cálculo de las explicaciones deberá realizarse en el navegador, sin depender de ningún servidor que procese los datos.
RNF-2	**Integración.** El módulo deberá integrarse en la arquitectura existente de *Nets4Learning* (React y TensorFlow.js) sin alterar el funcionamiento de las funcionalidades previas.
RNF-3	**Usabilidad.** La interfaz deberá ser intuitiva y adecuada para un público educativo sin conocimientos avanzados de aprendizaje automático.
RNF-4	**Rendimiento.** Las explicaciones deberán generarse en un tiempo razonable dentro de las limitaciones del entorno del navegador, ofreciendo al usuario información sobre el progreso del cálculo.
RNF-5	**Extensibilidad.** El diseño del módulo deberá estar desacoplado de los modelos concretos de modo que puedan incorporarse nuevos modelos o técnicas con un impacto mínimo.
RNF-6	**Internacionalización.** El módulo deberá respetar el soporte multilingüe de la plataforma (español e inglés) y estar disponible también en japonés, idioma que se incorpora a la plataforma en este trabajo.

Tabla. 3.2: Requisitos no funcionales del sistema.

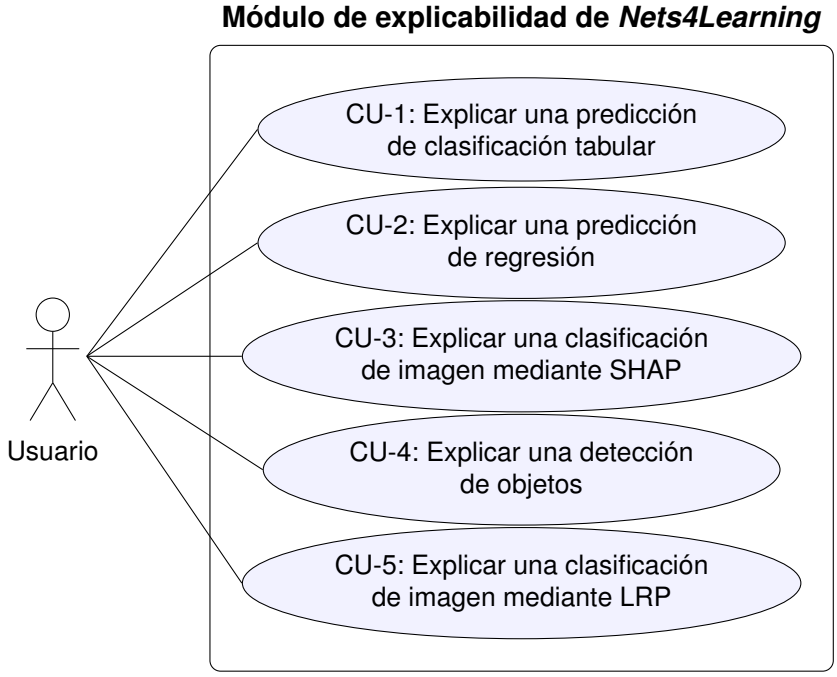


Figura 3.1: Diagrama de casos de uso del módulo de explicabilidad, con un caso de uso por incremento.

CU	Nombre	Incremento	Requisitos
CU-1	Explicar una predicción de clasificación tabular	1: clasificación tabular	RF-1, RF-2, RF-5, RF-6, RF-8
CU-2	Explicar una predicción de regresión	2: regresión	RF-1, RF-2, RF-5, RF-6
CU-3	Explicar una clasificación de imagen mediante SHAP	3: clasificación de imágenes	RF-1, RF-2, RF-4, RF-5, RF-6, RF-8
CU-4	Explicar una detección de objetos	4: detección de objetos	RF-1, RF-2, RF-4, RF-5, RF-6, RF-8
CU-5	Explicar una clasificación de imagen mediante LRP	5: LRP	RF-1, RF-3, RF-5, RF-7, RF-8

Tabla. 3.3: Trazabilidad entre casos de uso, incrementos y requisitos funcionales.

A continuación, en las Tablas 3.4 a 3.8, detallamos cada uno de los casos de uso, con su escenario principal paso a paso y los escenarios alternativos que pueden darse.

Campo	Descripción
Identificador	CU-1
Nombre	Explicar una predicción de clasificación tabular
Descripción	Permite al usuario conocer cuánto contribuye cada característica a la predicción de un modelo de clasificación tabular (explicación local) y qué características influyen más en el modelo en general (explicación global).
Actor	Usuario
Precondiciones	El usuario ha accedido a la tarea de clasificación tabular.
Escenario principal	1. El usuario selecciona un modelo de clasificación tabular. 2. El usuario introduce los valores de las características de la instancia y solicita la predicción. 3. El sistema muestra la clase predicha por el modelo. 4. El usuario selecciona la clase que quiere explicar pudiendo ajustar el número de muestras. 5. El usuario pulsa el botón para mostrar la explicación. 6. El sistema construye el conjunto de referencia muestreando filas del conjunto de datos. 7. El sistema calcula en el navegador la contribución de cada característica mediante KernelSHAP. 8. El sistema muestra un gráfico de barras con la contribución de cada característica a la clase seleccionada. 9. Fin del escenario con éxito.
Escenarios alternativos	5a. Si no se ha introducido ninguna instancia, el sistema avisa al usuario de que debe introducir una entrada y el caso de uso termina. 5b. Si el usuario pide la explicación global, indica el número de instancias, el tipo de gráfico (barras o enjambre) y el orden. El sistema calcula la explicación de una muestra de instancias del conjunto mostrando el progreso y mostrando la importancia global. Fin del escenario con éxito. 7a. En caso de producirse un error durante el cálculo, el sistema muestra un mensaje de error y el caso de uso termina sin explicación.
Postcondiciones	El usuario visualiza qué atributos han influido más en la predicción y en qué sentido.

Tabla. 3.4: Caso de uso CU-1: explicar una predicción de clasificación tabular.

Campo	Descripción
Identificador	CU-2
Nombre	Explicar una predicción de regresión
Descripción	Permite al usuario conocer cuánto contribuye cada característica al valor numérico que predice un modelo de regresión, tanto para una instancia concreta como de forma global.
Actor	Usuario
Precondiciones	El usuario ha accedido a la tarea de regresión.
Escenario principal	1. El usuario selecciona un modelo de regresión. 2. El usuario selecciona o introduce una instancia y solicita la predicción. 3. El sistema muestra el valor predicho por el modelo. 4. El usuario ajusta, si lo desea, el número de muestras. 5. El usuario pulsa el botón para mostrar la explicación. 6. El sistema construye el conjunto de referencia y toma la instancia con el mismo escalado que se aplicó en la predicción. 7. El sistema calcula en el navegador la contribución de cada característica mediante KernelSHAP. 8. El sistema muestra un gráfico de barras con la contribución de cada característica al valor predicho. 9. Fin del escenario con éxito.
Escenarios alternativos	5a. Si no se ha realizado ninguna predicción, el sistema avisa al usuario de que debe introducir una entrada y el caso de uso termina. 5b. Si el usuario pide la explicación global, indica el número de instancias y el tipo de gráfico; el sistema calcula la explicación de una muestra del conjunto informando del progreso y muestra la importancia global. Fin del escenario con éxito. 7a. Si se produce un error durante el cálculo, el sistema muestra un mensaje de error y el caso de uso termina sin explicación.
Postcondiciones	El usuario visualiza qué atributos han aumentado o reducido el valor predicho y en qué medida.

Tabla. 3.5: Caso de uso CU-2: explicar una predicción de regresión.

Campo	Descripción
Identificador	CU-3
Nombre	Explicar una clasificación de imagen mediante SHAP
Descripción	Permite al usuario ver sobre la propia imagen qué regiones han llevado al modelo a su predicción, mediante un mapa de calor calculado con el método agnóstico SHAP.
Actor	Usuario
Precondiciones	El usuario ha accedido a la tarea de clasificación de imágenes con un modelo cargado (por ejemplo, MobileNet).
Escenario principal	1. El usuario selecciona una imagen de ejemplo o carga una propia. 2. El usuario solicita la clasificación de la imagen. 3. El sistema muestra las clases más probables. 4. El usuario ajusta, si lo desea, la rejilla de segmentación, el número de muestras y la máscara. 5. El usuario pulsa el botón para explicar la predicción. 6. El sistema segmenta la imagen en superpíxeles. 7. El sistema fija utiliza una predicción inicial para determinar las clases a explicar. 8. El sistema evalúa el modelo sobre versiones perturbadas de la imagen y calcula la contribución de cada región mediante KernelSHAP. 9. El sistema muestra una galería con algunas de las imágenes perturbadas y un mapa de calor por cada clase. 10. Fin del escenario con éxito.
Escenarios alternativos	5a. Si no se selecciona ninguna imagen el sistema no permite la explicación. 8a. Si se produce un error durante el cálculo, el sistema muestra un mensaje de error y el caso de uso termina sin explicación.
Postcondiciones	El usuario visualiza qué regiones de la imagen han sido relevantes y cuales ha contribuido negativamente.

Tabla. 3.6: Caso de uso CU-3: explicar una clasificación de imagen mediante SHAP.

Campo	Descripción
Identificador	CU-4
Nombre	Explicar una detección de objetos
Descripción	Permite al usuario ver qué regiones de la imagen han motivado cada una de las detecciones de un detector de objetos, incluidos los modelos faciales.
Actor	Usuario
Precondiciones	El usuario ha accedido a la tarea de identificación de objetos con un detector cargado (por ejemplo, COCO-SSD o un modelo facial).
Escenario principal	1. El usuario sube una imagen. 2. El sistema procesa la imagen y muestra las detecciones del modelo. 3. El usuario ajusta, si lo desea, la rejilla de segmentación, el número de muestras y la máscara, eligiendo entre oscurecer o difuminar las regiones y, en este caso, el tamaño del desenfoque y el número de pasadas. 4. El usuario pulsa el botón para explicar la predicción. 5. El sistema fija las etiquetas a explicar a partir de las detecciones de la imagen original. 6. El sistema segmenta la imagen: en regiones anatómicas de la cara si el modelo es facial, o en una rejilla de superpíxeles en otro caso. 7. El sistema evalúa el detector sobre versiones perturbadas de la imagen, normalizando cada salida a un vector de longitud fija. 8. El sistema calcula la contribución de cada región mediante KernelSHAP. 9. El sistema muestra un mapa de calor por cada etiqueta detectada. 10. Fin del escenario con éxito.
Escenarios alternativos	4a. Mientras no se haya procesado ninguna imagen, el botón de explicar permanece deshabilitado. 6a. Si el modelo es facial pero no se detecta ninguna cara válida, el sistema segmenta la imagen con la rejilla de superpíxeles y continúa en el paso 7. 8a. Si se produce un error durante el cálculo, el sistema muestra un mensaje de error y el caso de uso termina sin explicación.
Postcondiciones	El usuario visualiza qué regiones de la imagen han sido relevantes y cuales ha contribuido negativamente.

Tabla. 3.7: Caso de uso CU-4: explicar una detección de objetos.

Campo	Descripción
Identificador	CU-5
Nombre	Explicar una clasificación de imagen mediante LRP
Descripción	Permite al usuario ver la relevancia que la red atribuye a cada píxel de la imagen, propagándola capa a capa desde la salida, en los modelos cuya arquitectura es accesible.
Actor	Usuario
Precondiciones	El usuario trabaja con un modelo de imagen cuya estructura interna es accesible, como el modelo de dígitos manuscritos (MNIST) o un modelo entrenado por él mismo en la plataforma.
Escenario principal	1. El usuario dibuja un dígito o selecciona una imagen de ejemplo. 2. El usuario solicita la clasificación de la imagen. 3. El sistema muestra la clase predicha. 4. El usuario pulsa el botón para explicar la predicción con el método LRP. 5. El sistema comprueba que el modelo permite obtener sus activaciones y propagar la relevancia. 6. El sistema ejecuta una inferencia capturando las activaciones de todas las capas. 7. El sistema inicializa la relevancia con la salida de la clase predicha. 8. El sistema propaga la relevancia capa a capa hasta la entrada, aplicando la regla que corresponde a cada tipo de capa. 9. El sistema muestra el mapa de relevancia superpuesto a la imagen. 10. Fin del escenario con éxito.
Escenarios alternativos	4a. Si la estructura interna del modelo no es accesible, la opción LRP del selector de método aparece deshabilitada y el usuario solo puede pedir la explicación mediante SHAP (CU-3). 4b. Si no se selecciona ninguna imagen el sistema no permite la explicación. 8a. Si se produce un error durante la propagación, el sistema muestra un mensaje de error y el caso de uso termina sin explicación.
Postcondiciones	El usuario visualiza qué trazos o zonas de la imagen sostienen la predicción y cuáles la contradicen.

Tabla. 3.8: Caso de uso CU-5: explicar una clasificación de imagen mediante LRP.

Capítulo 4

Diseño del sistema

En este capítulo describimos el diseño de la solución que hemos desarrollado. En primer lugar planteamos la arquitectura y de cómo el nuevo módulo de explicabilidad encaja dentro de la plataforma, para después ir bajando al diseño interno del propio módulo. A lo largo del capítulo iremos justificando las decisiones que hemos ido tomando, haciendo énfasis en las decisiones que se ven condicionadas por la restricción de tener que ejecutarlo todo en el lado del cliente (*client-side*), dado que ha sido lo que más ha condicionado el desarrollo y elección de herramientas.

4.1. Arquitectura general

Nets4Learning es una aplicación web de página única (*SPA*) construida con React, en la que la inferencia de los modelos se realiza en el lado del cliente haciendo uso de TensorFlow.js. El módulo de explicabilidad se integra como una capa más dentro de esa misma arquitectura y dado que va a formar parte de la experiencia final del usuario, le damos la misma importancia que al resto de la interfaz.

El sistema se organiza en tres niveles:

- **Capa de modelos:** los modelos de aprendizaje automático que ya ofrecía la plataforma, agrupados por tipo de tarea (clasificación y regresión sobre datos tabulares, clasificación de imágenes y detección de objetos).
- **Capa de explicabilidad:** el núcleo desarrollado en este trabajo, que se encarga de generar las explicaciones a partir de las predicciones de los modelos. Es independiente de los modelos concretos.

- **Capa de presentación:** ciertos componentes de la interfaz son añadidos para poder hacer funcional el módulo de explicabilidad.

La Figura 4.1 muestra esta organización y las relaciones entre las tres capas.

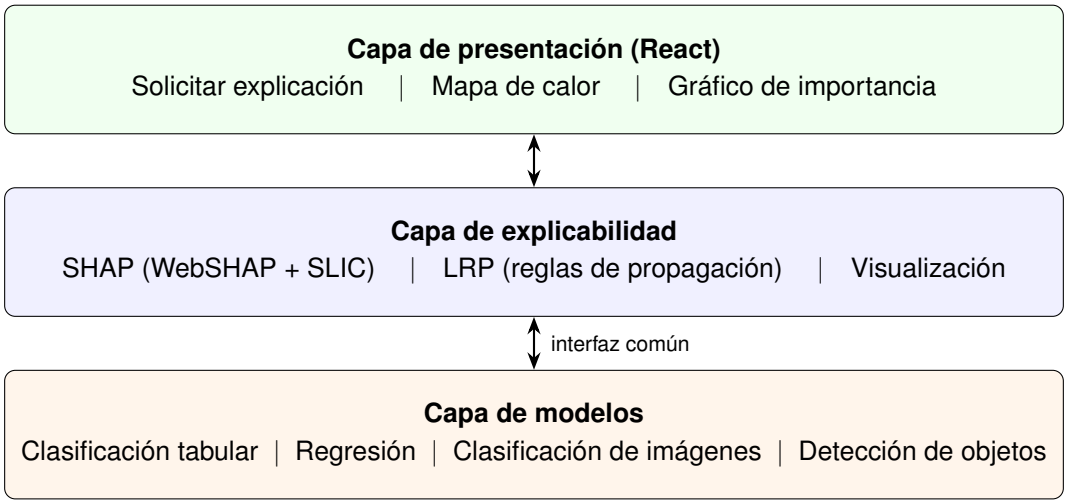


Figura 4.1: Arquitectura general del sistema con la capa de explicabilidad integrada.

4.2. Integración con *Nets4Learning*

Probablemente la decisión de diseño más importante de todo el proyecto es la forma en que la capa de explicabilidad se comunica con los modelos ya que de ella depende que el módulo se pueda reutilizar. Con el fin de poder aplicarlo tanto en la clasificación de imágenes como en los datos tabulares sin tener que conocer los detalles internos de cada modelo, hacemos uso de interfaces comunes de forma que la explicabilidad no trabaja contra un modelo concreto, sino contra una interfaz común que todos ellos implementan.

Lo que hacemos es que cada tipo de tarea cuente con una interfaz común que implementan todos los modelos de esa tarea y en la que se declaran, entre otras, las operaciones necesarias para habilitar un modelo y para obtener una predicción a partir de una entrada.

De esta forma, la explicabilidad trabaja siempre contra la misma abstracción mientras que cada modelo concreto aporta su propia implementación, dado que entre unos y otros hay bastantes diferencias. Gracias a ello cumplimos el requisito de extensibilidad (RNF-5), ya que añadir un nuevo modelo no requiere más cambios que la adaptación a la interfaz.

4.3. Diseño del módulo de explicabilidad

Dentro del módulo tenemos dos técnicas que, aunque las dos sirven para explicar, no se solapan entre sí dado que por un lado tenemos el método agnóstico de caja negra (SHAP) y, por otro, el método específico (LRP). Si bien son idénticos a nivel de interfaz, su estructura interna es completamente diferente, por lo que conviene tratarlas como tal.

4.3.1. Explicabilidad agnóstica mediante SHAP

Como el método agnóstico trata al modelo como una caja negra y lo único que necesita es poder evaluarlo una y otra vez, su diseño nos queda organizado en torno a tres elementos:

- **Función predictora.** Una función que, dada una o varias entradas, devuelve las predicciones del modelo invocando la operación de predicción de la interfaz. Actúa como adaptador entre el formato que espera el algoritmo de SHAP y el que ofrece el modelo, por lo que su diseño sigue el patrón *Adapter*.
- **Algoritmo de SHAP.** Se emplea una implementación de KernelSHAP capaz de ejecutarse en el navegador, que recibe la función predictora y un conjunto de datos de referencia, y devuelve la contribución de cada característica a la predicción.
- **Segmentación en superpíxeles.** En las tareas de imagen, antes de aplicar SHAP la imagen se divide en superpíxeles mediante el algoritmo SLIC. De este modo, las características sobre las que opera SHAP no son píxeles individuales, sino regiones con significado visual, lo que reduce drásticamente el número de evaluaciones necesarias y mejora la interpretabilidad del resultado.

El flujo de generación de una explicación mediante SHAP para una imagen se presenta en el diagrama de secuencia de la Figura 4.2, donde se segmenta la imagen, se generan versiones perturbadas activando y desactivando regiones, se evalúa el modelo sobre cada una y a partir de cómo cambia la predicción se estima la contribución de cada región.

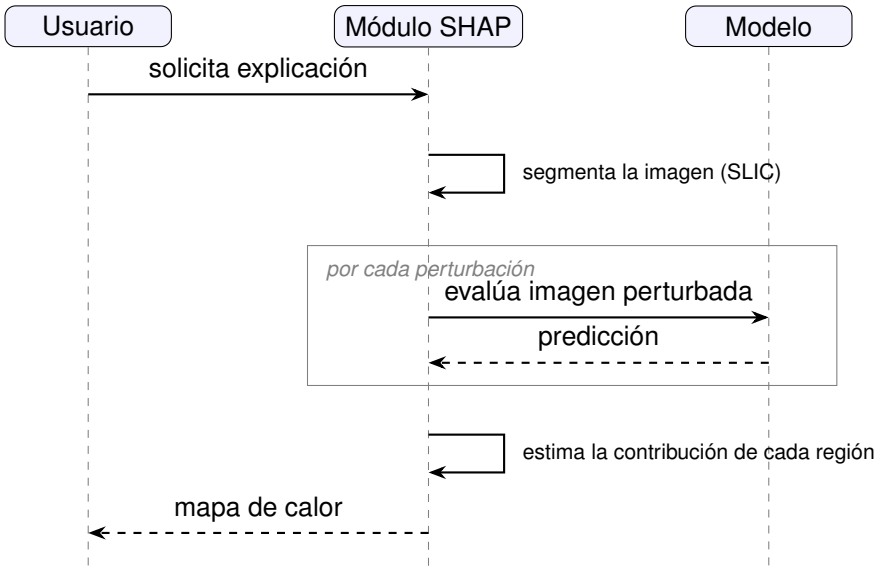


Figura 4.2: Diagrama de secuencia de la generación de una explicación mediante SHAP.

4.3.2. Explicabilidad específica mediante LRP

A diferencia de SHAP, el método LRP necesita acceder al interior de la red, por lo que su diseño nos obliga a ampliar la interfaz de los modelos cuya estructura interna es accesible con dos operaciones más, una para obtener las activaciones de las capas durante una inferencia y otra para llevar a cabo la propagación de la relevancia.

De forma práctica, la propagación funciona como un recorrido de la red en sentido inverso de forma que, partiendo de la salida, vamos repartiendo la relevancia hacia las capas anteriores hasta llegar a la entrada. Como cada tipo de capa necesita una regla de propagación distinta, el diseño contempla diferentes reglas que se van aplicando según el tipo de la capa que estemos procesando en cada paso.

Sin embargo, dado que LRP depende por completo de la arquitectura, solo lo ofrecemos para los modelos cuya estructura es accesible, es decir, para el resto la interfaz simplemente no implementa las operaciones de LRP y el sistema deshabilita esta opción de forma automática (RF-7).

4.4. Diseño de la visualización

En líneas generales, tanto una técnica como la otra nos devuelven básicamente lo mismo, es decir, un valor de importancia o de relevancia asociado a cada característica

o región de la entrada (en función de si se trata de una imagen o características). A partir de ahí, el componente de visualización se encarga de convertir esos valores en algo que el usuario pueda entender fácilmente:

- **Datos tabulares o Regresión:** la importancia de cada característica se representa mediante un gráfico de barras, que indica tanto la magnitud como el sentido de la contribución de cada atributo.
- **Imágenes:** los valores de importancia se proyectan sobre la imagen original en forma de mapa de calor, asociando a cada región un color en función de su contribución. De este modo el usuario percibe de un vistazo en qué zonas se ha fijado el modelo.

El diseño de este componente funciona de forma independiente del método que haya generado los valores de manera que tanto las explicaciones de SHAP como las de LRP comparten la misma lógica de visualización en el caso de las imágenes.

Capítulo 5

Desarrollo

En este capítulo describimos el desarrollo del módulo de explicabilidad. Siguiendo la metodología incremental que justificamos en la Sección 2.4, el trabajo se organiza en cinco incrementos, uno por cada tarea a la que damos explicabilidad. Antes de entrar en ellos recogemos los fundamentos técnicos comunes sobre los que se apoyan todos. Para cada incremento seguimos después la misma estructura, con su objetivo y los casos de uso, el diseño detallado, la implementación y el plan de pruebas.

Por último, al final del capítulo reunimos todas las clases en un único diagrama de clases completo y describimos los patrones de diseño que hemos empleado, indicando dónde y cómo los hemos utilizado.

5.1. Fundamentos comunes

Antes de entrar en los incrementos debemos hablar de las bibliotecas que usamos, dado que se reutilizan de forma continua en todos ellos, y del mecanismo general con el que calculamos una explicación mediante SHAP.

5.1.1. Bibliotecas utilizadas: WebSHAP y SLIC

A la hora de implementar una adaptación del algoritmo SHAP basado en la teoría de juegos, hacemos uso de WebSHAP, que fue publicado por Zijie J. Wang y Duen Horng Chau, ambos miembros del Instituto de Tecnología de Georgia [16]. WebSHAP hace uso de JavaScript y TypeScript, permitiendo su ejecución de forma local y sin

necesidad de *backend*. Por otro lado, también va acompañado de una implementación del algoritmo de segmentación SLIC para JavaScript, la cual hemos integrado y adaptado al proyecto.

SLIC funciona como una adaptación mejorada de K-Means, el algoritmo que divide un conjunto de datos en k grupos o *clusters* de forma que los puntos de un mismo grupo se parezcan más entre sí que a los de otros grupos. Partimos de la implementación incluida en WebSHAP [16, 24], cuya estructura se refleja en el diagrama de clases de la Figura 5.1, sobre la que realizamos los ajustes necesarios para integrarla en el flujo de TensorFlow.js de la plataforma. En concreto, en la plataforma utilizamos su variante SLICO (`computeSLICzeroMap`), que no necesita ajustar a mano la compacidad de las regiones y que solo recibe el lado de la rejilla inicial, de forma que el usuario controla la granularidad de la segmentación con un único parámetro.

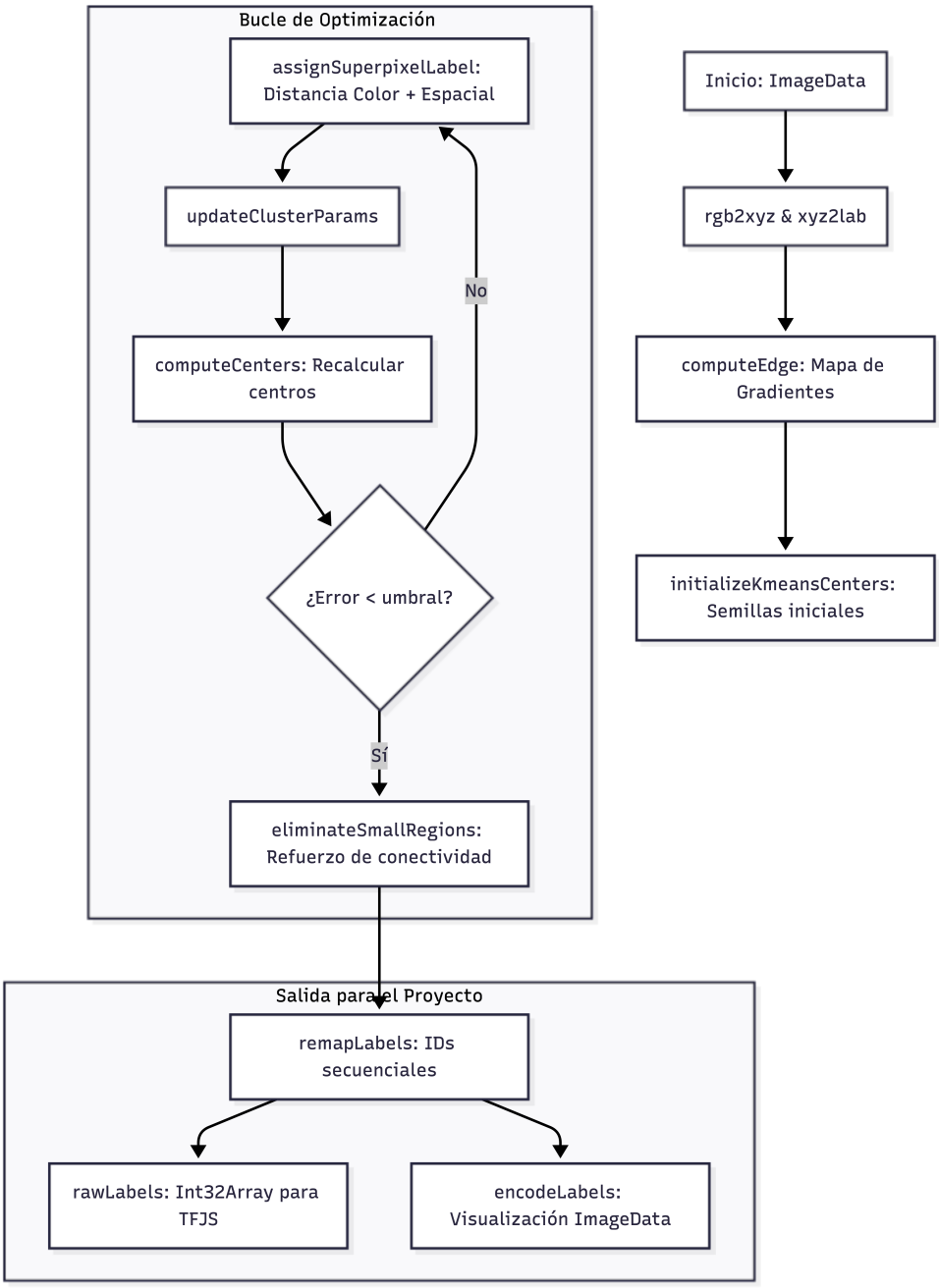


Figura 5.1: Diagrama de clases del algoritmo SLIC.

5.1.2. Mecanismo general de SHAP

La clave de SHAP es que es un método agnóstico, por lo que es aplicable a prácticamente todas las tareas que ofrece la plataforma. KernelSHAP trata el modelo como una caja negra y lo evalúa calculando la contribución marginal de cada una de las características.

Para el correcto funcionamiento de SHAP en todas las tareas, se crea una función

predictora, a la cual se le pasa un conjunto de entradas que debe adaptar para poder obtener predicciones en función del modelo a evaluar. Se invoca la predicción las veces necesarias hasta obtener el resultado, de esta forma podemos hacer que el algoritmo se adapte a cualquier modelo. Por dentro el cálculo es el mismo para todas las tareas, a excepción de los modelos que trabajan con imágenes, que requieren un tratamiento especial antes a pesar de mantener la misma estructura de procesamiento. El Código 5.1 muestra los tres pasos que se repiten siempre, tomando como ejemplo el caso tabular.

Código 5.1

```
// Datos de referencia muestreados del propio conjunto de datos
const backgroundData = buildShapBackground(backgroundPool,
    nFeatures);

// Función predictora que adapta el modelo a KernelSHAP
const predictor = myModelWrapper(model_ref);
const explainer = new KernelSHAP(predictor, backgroundData, 0.2022)
    ;

// Cálculo de las contribuciones para la instancia de entrada
const shapValues = await explainer.explainOneInstance(instance,
    nSamples);
```

Código 5.1: Cálculo de una explicación con KernelSHAP: datos de referencia, función predictora y explicador

El parámetro `nSamples` controla cuántas combinaciones de características evalúa el algoritmo. A mayor número de combinaciones, más precisa es la explicación, pero supone un alto coste de computación, así que es uno de los parámetros que el usuario puede ajustar (RF-6) para buscar el equilibrio entre precisión y velocidad más adecuado a su caso.

5.2. Incremento 1: explicabilidad en clasificación tabular

5.2.1. Objetivo del incremento

El objetivo de este primer incremento es desarrollar el caso de uso CU-1 (*Explicar una predicción de clasificación tabular*, Tabla 3.4), que va a permitir al usuario saber cuánto ha contribuido cada característica a la clase que predice un modelo tabular, tanto para una instancia concreta como para el modelo en su conjunto. Empezamos por esta tarea porque es la más sencilla de adaptar de todas, dado que las características que recibe SHAP son directamente las columnas del conjunto de datos, y nos sirve para dejar montada la base (función predictora, muestreo y visualización) que reutilizamos después.

5.2.2. Diseño detallado

La función predictora tabular es la más simple de todas. La Figura 5.2 recoge el diagrama de clases del incremento, con los nombres reales de la implementación. Para clasificar se crea y ejecuta el KernelSHAP que a través del adaptador `myModelWrapper` llama al modelo de TensorFlow.js, que devuelve las predicciones. Los resultados se muestran en un un gráfico de barras o en un diagrama de enjambre.

La Figura 5.3 muestra el diagrama de secuencia del caso de uso CU-1 desde que el usuario pulsa el botón de explicar hasta que ve el gráfico. Se debe destacar como se itera por lote de perturbaciones de modo que el panel manda a KernelSHAP el conjunto de entrada perturbado y se desentiende del proceso hasta recibir los valores SHAP finales.

A la hora de diseñar la interfaz se pretende integrarla con la existente en la plataforma, por lo que simplemente ponemos la sección de explicabilidad debajo de la de predicción, de modo que se entiende que la explicabilidad se utiliza tras la predicción. En la Figura 5.4 podemos ver el boceto con la idea principal de la interfaz. Como tenemos dos tipos de explicación posibles, la local y la global, en la local se muestra un gráfico de barras, mientras que a la hora de explicar globalmente el modelo el usuario puede elegir entre un gráfico de barras o un diagrama de enjambre. En ambos casos podemos ajustar el número de muestras (RF-6).

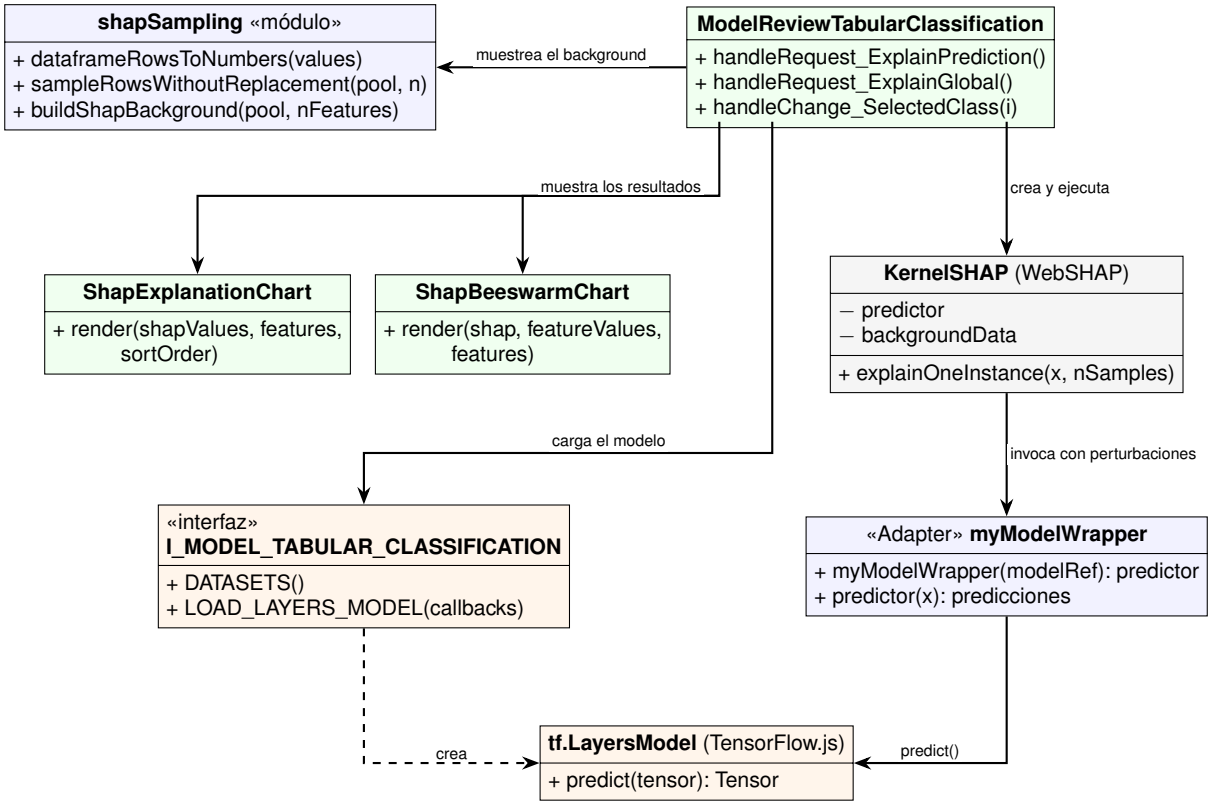


Figura 5.2: Diagrama de clases del incremento de clasificación tabular.

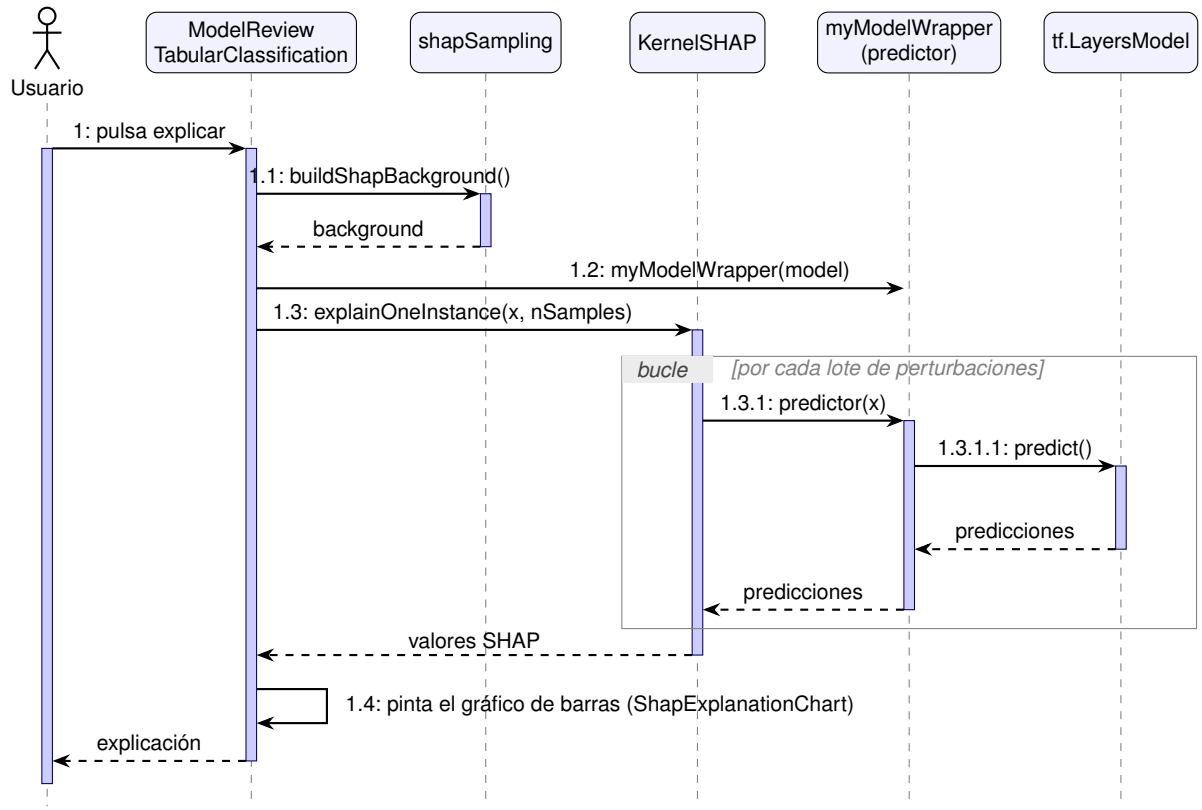


Figura 5.3: Diagrama de secuencia del caso de uso CU-1: explicación de una predicción de clasificación tabular.

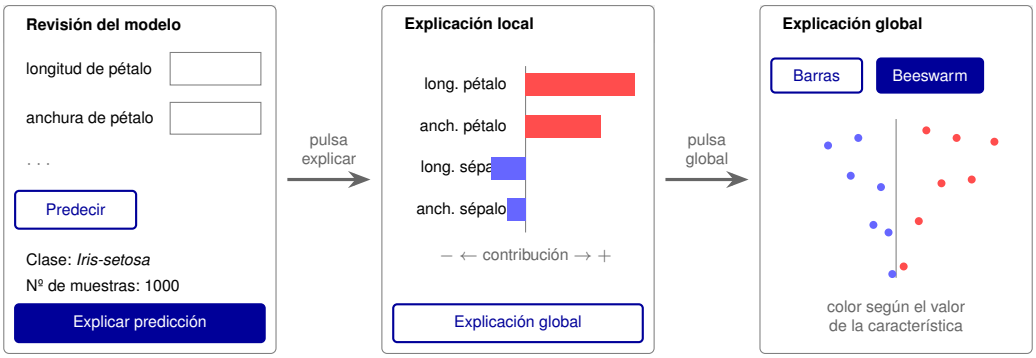


Figura 5.4: *Storyboard* de la explicación tabular: predicción, explicación local y explicación global.

5.2.3. Implementación

La función predictora construye un tensor con las instancias que recibe, evalúa el modelo y devuelve sus predicciones, tal y como muestra el Código 5.2.

Código 5.2

```
return async (x) => {
  const inputTensor = tfjs.tensor2d(x, [x.length, x[0].length]);
  const predictionTensor = model.predict(inputTensor);
  const predictionData = await predictionTensor.array();
  inputTensor.dispose();
  predictionTensor.dispose();
  return predictionData;
};
```

Código 5.2: Función predictora para modelos tabulares

El resultado es un valor de contribución por cada característica, que se presenta al usuario mediante un gráfico de barras con una barra por característica para la clase seleccionada, ordenadas de mayor a menor importancia (orden elegible por el usuario). La altura y el signo de cada barra indican cuánto empuja cada atributo a la predicción, de modo que a mayor barra, mayor contribución. La Figura 5.5 muestra la predicción del modelo sobre una instancia del conjunto Iris, clasificada como *Iris-versicolor*, y la Figura 5.6 muestra la explicación local de la predicción, en la que se aprecia que la longitud del pétalo es el atributo que más contribuye a la predicción, de forma muy superior a los demás.

Para la explicación global usamos un diagrama de enjambre (*beeswarm*) [25], donde cada punto es una instancia situada según su valor SHAP, las filas son las caracte-

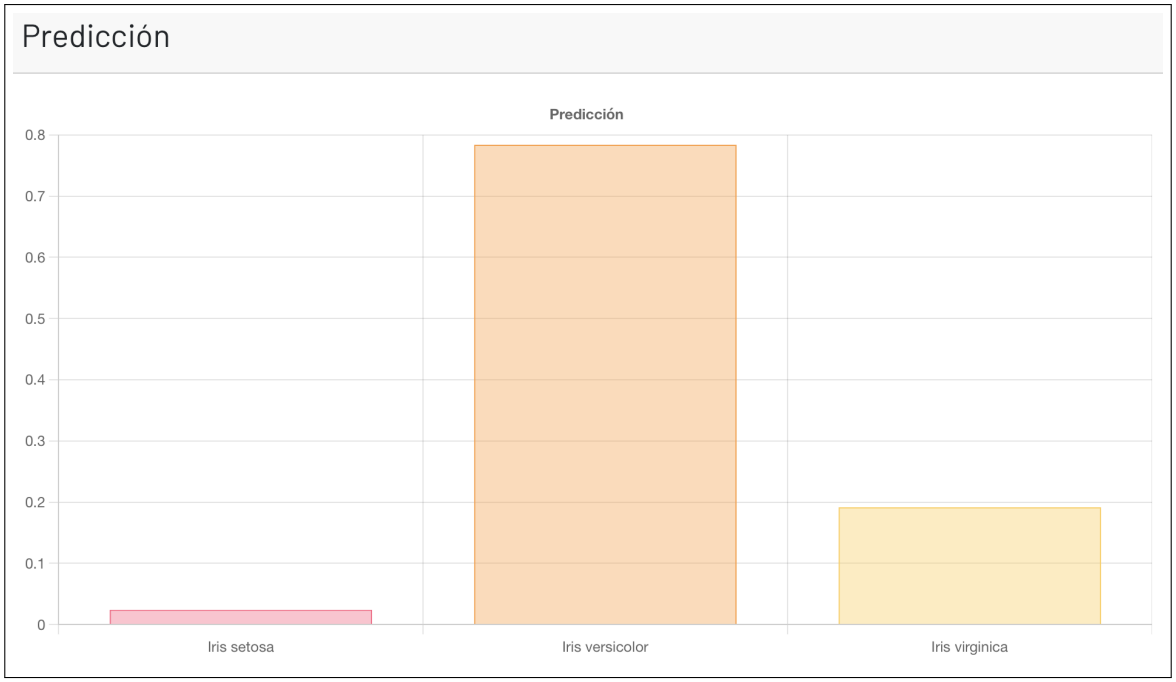


Figura 5.5: Predicción del modelo de clasificación tabular sobre una instancia del conjunto Iris.

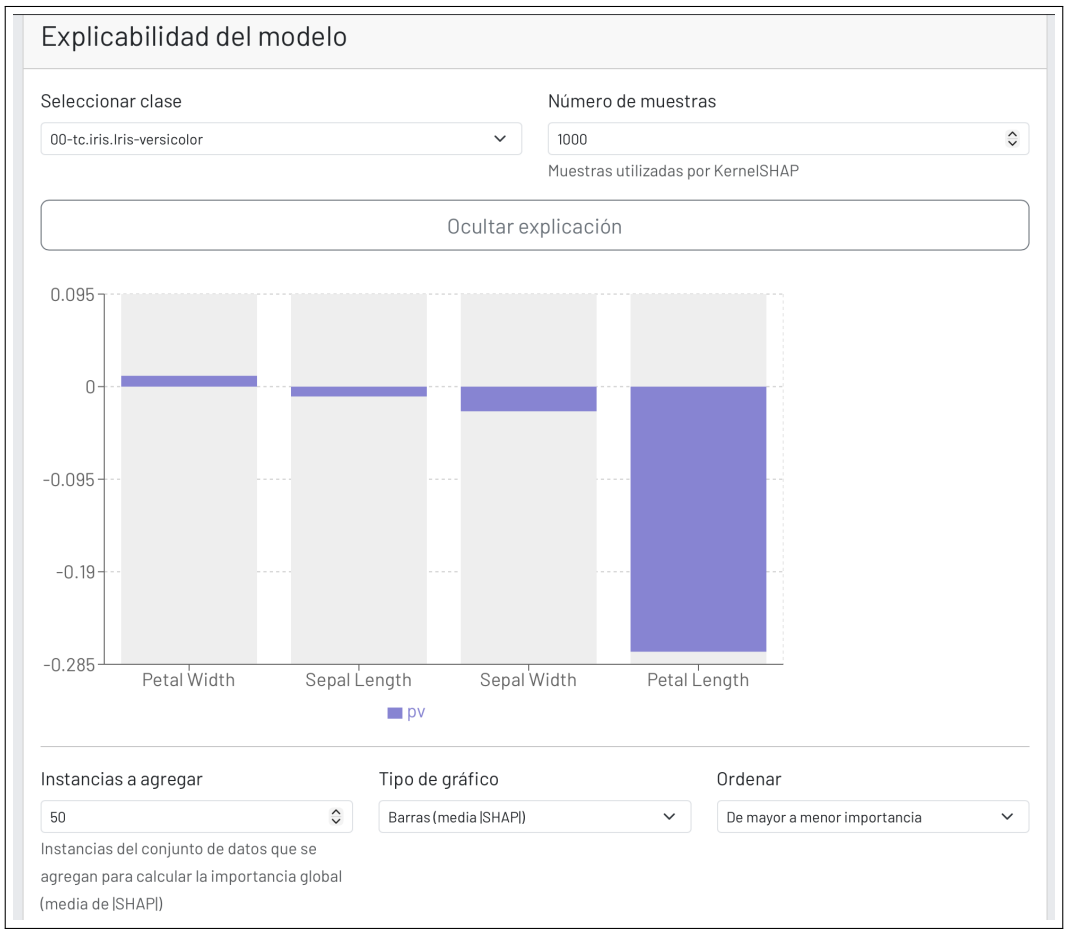


Figura 5.6: Explicación local mediante SHAP de la predicción anterior: contribución de cada característica a la clase evaluada.

rísticas ordenadas por importancia media y el color de cada punto representa el valor de la característica, de azul (bajo) a rojo (alto). La Figura 5.7 muestra el gráfico calculado sobre cincuenta instancias del conjunto, en el que podemos apreciar que las dos características del pétalo representan mucho mayor peso en el resultado de la predicción.

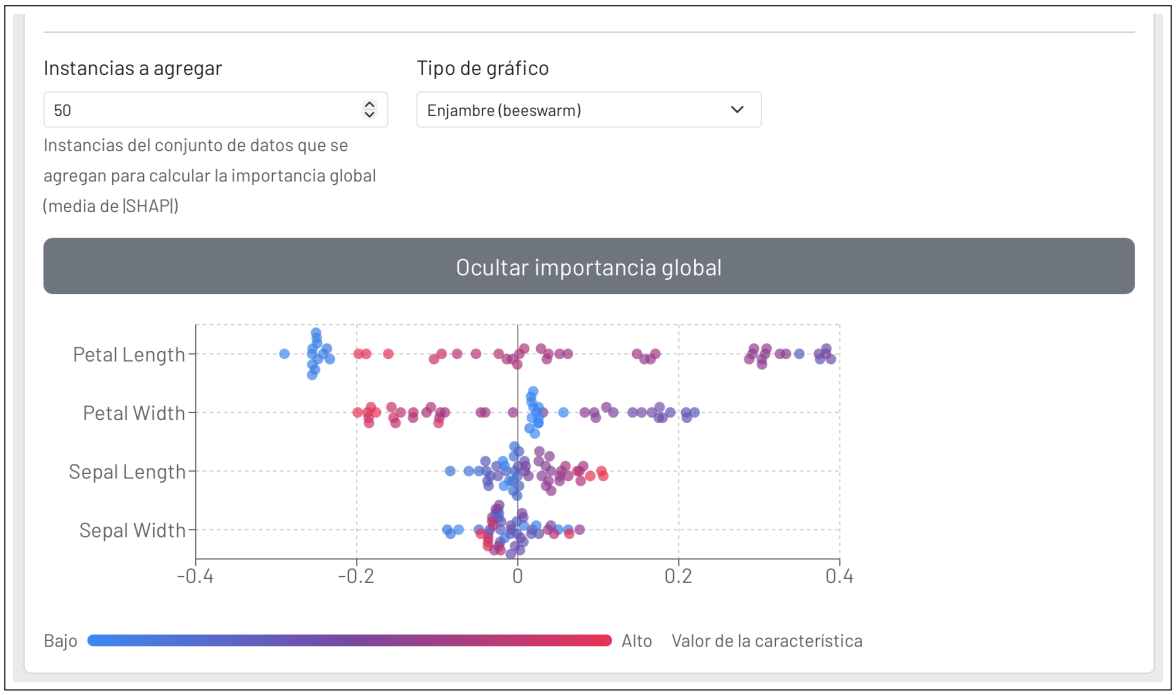


Figura 5.7: Importancia global mediante diagrama de enjambre (*beeswarm*), agregando cincuenta instancias del conjunto.

5.2.4. Plan de pruebas

En pos de comprobar que el muestreo funciona de forma correcta debemos asegurarnos de que los datos que se entregan a SHAP son correctos. Para ello usamos pruebas unitarias con Vitest. La Tabla 5.1 resume los casos comprobados, con un resultado de nueve pruebas superadas de nueve.

Función	Caso comprobado	Resultado
`dataframeRowsTo Numbers`	Convierte celdas de texto y numéricas a número.	OK
	Devuelve una lista vacía ante una entrada que no es un array.	OK
`sampleRowsWithout Replacement`	Devuelve exactamente n filas cuando $n \leq$ tamaño del conjunto.	OK
	No devuelve más filas que las disponibles.	OK
	Las filas devueltas son distintas y pertenecen al conjunto.	OK
`buildShap Background`	Muestrea filtrando las filas con el número correcto de características.	OK
	Genera un *background* de ceros si el conjunto está vacío.	OK
	Genera un *background* de ceros si ninguna fila tiene el número pedido.	OK
	Nunca muestrea más filas de las solicitadas.	OK

Tabla. 5.1: Pruebas unitarias sobre las utilidades de muestreo de SHAP.

A nivel funcional, hemos comprobado sobre la aplicación real que el gráfico de importancia da más peso a los atributos que son más determinantes y que la explicación global es coherente con las explicaciones locales individuales. Hemos comprobado los casos de prueba de la Tabla 5.2, con un resultado de cuatro pruebas superadas de cuatro.

ID	Caso de prueba	Resultado esperado	Resultado
PF-1.1	Explicación local de una instancia de Iris clasificada como *Iris-versicolor*.	Las características del pétalo reciben la mayor contribución.	OK
PF-1.2	Solicitar la explicación sin haber introducido ninguna instancia.	El sistema avisa al usuario y no lanza el cálculo.	OK
PF-1.3	Explicación global sobre cincuenta instancias con diagrama de enjambre.	La importancia global es coherente con las explicaciones locales individuales.	OK
PF-1.4	Cambiar la clase seleccionada después de calcular la importancia global.	Se descarta el resultado anterior y el usuario debe volver a calcularlo para la nueva clase.	OK

Tabla. 5.2: Pruebas funcionales del incremento de clasificación tabular.

5.3. Incremento 2: explicabilidad en regresión

5.3.1. Objetivo del incremento

Este incremento realmente consiste en extender lo realizado en clasificación tabular a Regresión realizando el caso de uso CU-2 (*Explicar una predicción de regresión*), Tabla 3.5),. Este es el más sencillo dado que a efectos prácticos el valor que tiene un Input para SHAP es realmente las características y sus valores, cosa que es igual tanto para clasificación tabular como para Regresión.

5.3.2. Diseño detallado

La función predictora `myModelWrapper`, KernelSHAP y los componentes de visualización se reutilizan sin cambios, de forma que la diferencia se puede encontrar en los datos que se entregan a SHAP dado que se da como resultado un valor único, mientras que en clasificación sería un vector de probabilidades por clase.

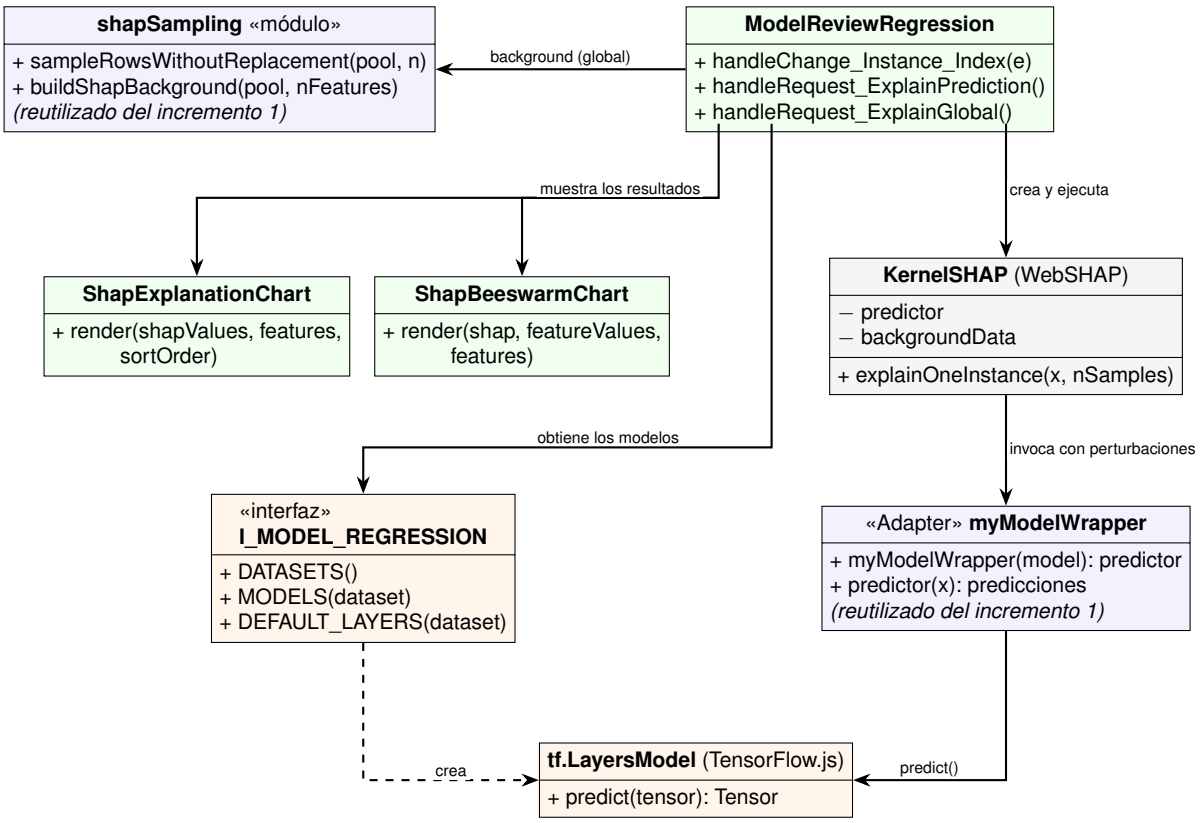


Figura 5.8: Diagrama de clases del incremento de regresión, en el que se reutilizan los elementos del incremento anterior.

He aquí el diagrama de secuencia del caso de uso CU-2 en la Figura 5.9.

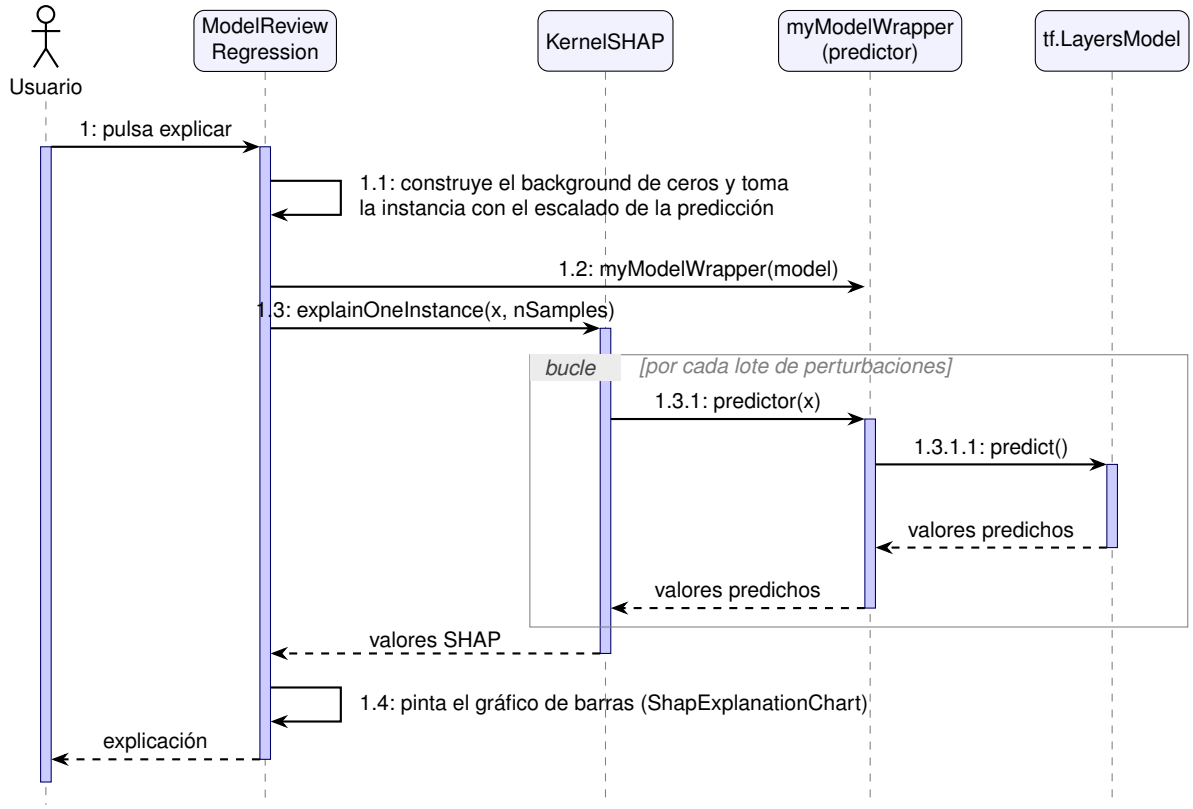


Figura 5.9: Diagrama de secuencia del caso de uso CU-2: explicación de una predicción de regresión.

La interfaz mantiene la misma idea y apenas cambia. La tarjeta de explicabilidad aparece justo debajo del resultado de la predicción. El boceto de la Figura 5.10 muestra la predicción del valor, la explicación local con el número de muestras a elegir y la importancia global.

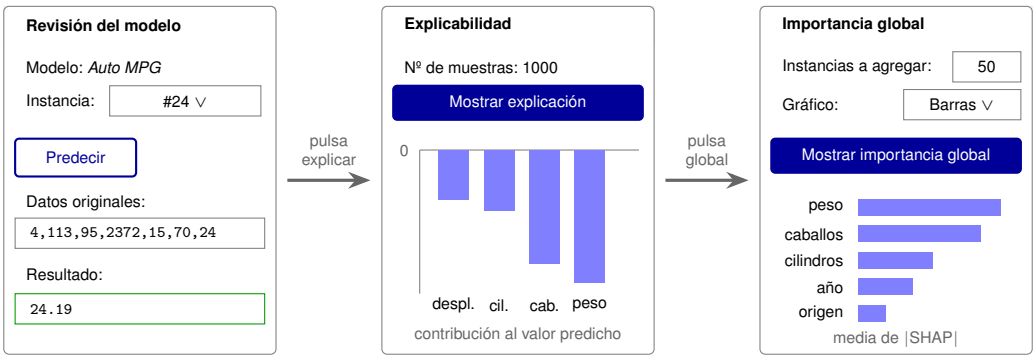


Figura 5.10: *Storyboard* de la explicación en regresión: predicción del valor, explicación local e importancia global.

5.3.3. Implementación

En este incremento se reutiliza la función predictora del incremento anterior de forma que solo cambia la forma de preparar los datos.

Código 5.3

```
// Background de ceros con tantas columnas como entradas tiene el modelo
const nFeatures = model.inputs[0].shape[1];
const backgroundData = Array(50).fill(null)
  .map(() => Array(nFeatures).fill(0));

// Se reutiliza la función predictora del incremento anterior
const predictor = myModelWrapper(model);
explainer.current = new KernelSHAP(predictor, backgroundData,
    0.2022);

// La instancia se toma ya escalada, igual que la recibe el modelo
const instance = prediction.input_3_dataframe_scaling.values[0];
const shapValues = await explainer.current.explainOneInstance(
    instance, nSamples);
```

Código 5.3: Preparación de la explicación en regresión: referencia de ceros e instancia escalada

La Figura 5.11 nos muestra el resultado sobre el modelo de regresión *Auto MPG*, donde se predicen para el vehículo $24{,}19$ millas por galón. Dicha explicación nos muestra que lo que más influye negativamente serían el peso y los caballos, siendo un resultado razonable y esperado.

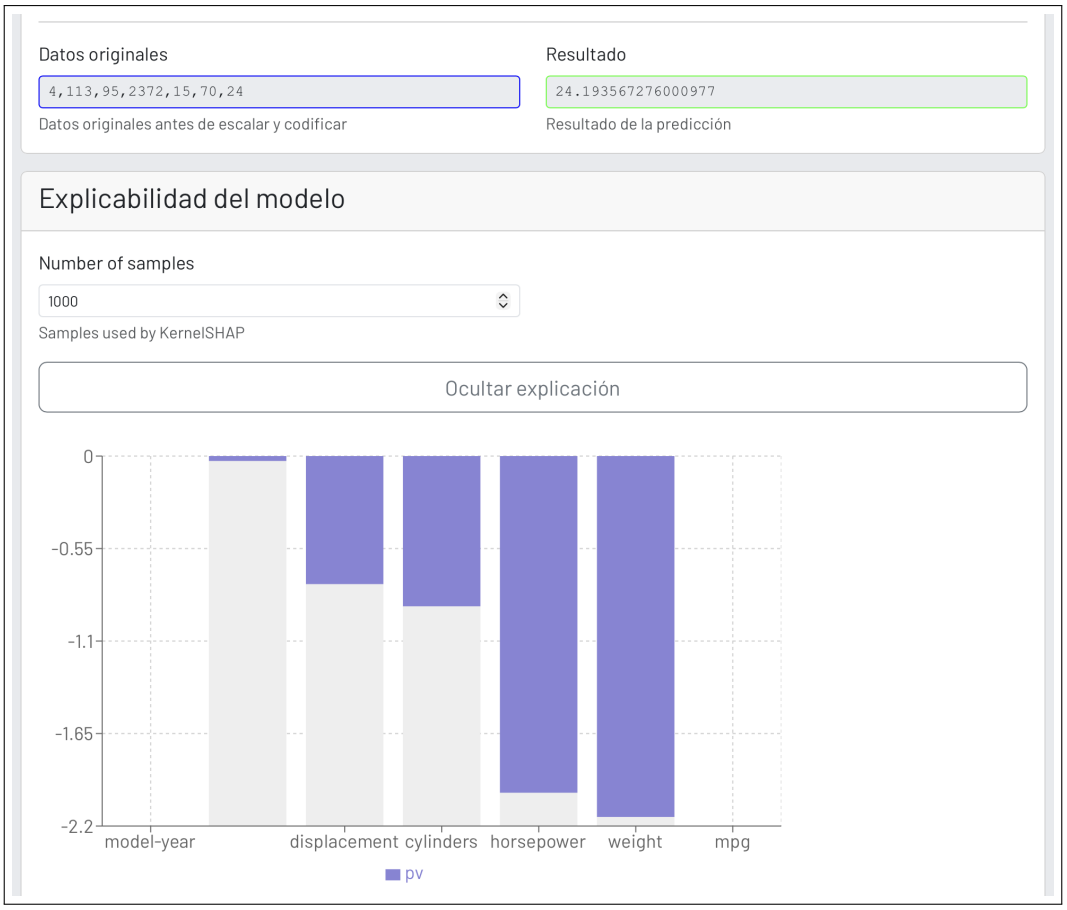


Figura 5.11: Predicción y explicación local mediante SHAP sobre el modelo de regresión *Auto MPG*.

5.3.4. Plan de pruebas

Dado que la implementación es idéntica al incremento anterior, las pruebas unitarias son las mismas. Finalmente, lo único a comprobar es la validación funcional, es decir, asegurarnos de que las contribuciones atribuidas por SHAP son coherentes con un modelo de regresión.

ID	Caso de prueba	Resultado esperado	Resultado
PF-2.1	Explicación local de un vehículo de *Auto MPG* con un valor predicho de $24{,}19$ millas por galón.	El peso y la potencia aparecen como los atributos que más reducen el valor predicho.	OK
PF-2.2	Solicitar la explicación sin haber realizado ninguna predicción.	El sistema avisa al usuario y no lanza el cálculo.	OK
PF-2.3	Explicación global sobre una muestra de instancias del conjunto.	Se muestra la importancia media de cada característica sobre el valor predicho.	OK

Tabla. 5.3: Pruebas funcionales del incremento de regresión.

5.4. Incremento 3: explicabilidad en clasificación de imágenes

5.4.1. Objetivo del incremento

El objetivo de este incremento es desarrollar el caso de uso CU-3 (*Explicar una clasificación de imagen mediante SHAP*, Tabla 3.6). Este incremento permite explicar las regiones de la imagen que más contribuyen a la predicción realizada por el modelo. Para esta sección se requiere de segmentación en superpíxeles y mapas de calor.

5.4.2. Diseño detallado

A diferencia del caso tabular, cuando se trata con imágenes no tiene sentido tratar cada píxel como una característica independiente, es por eso que debemos segmentar la imagen en superpíxeles/regiones con SLIC con el fin de poder tratar regiones como características, pasando de millones de características que puede tener una imagen a un número reducido alrededor de 7-10. Se presenta un mapa de calor superpuesto a la imagen para poder entender la contribución de cada región en la que se ha segmentado la imagen.

A continuación, la Figura 5.12 representa cómo funciona este incremento. Debemos tener en cuenta que tratamos con vectores binarios de regiones de la imagen por lo que va procesando la imagen perturbada al desactivar o activar diferentes regiones

de la imagen.

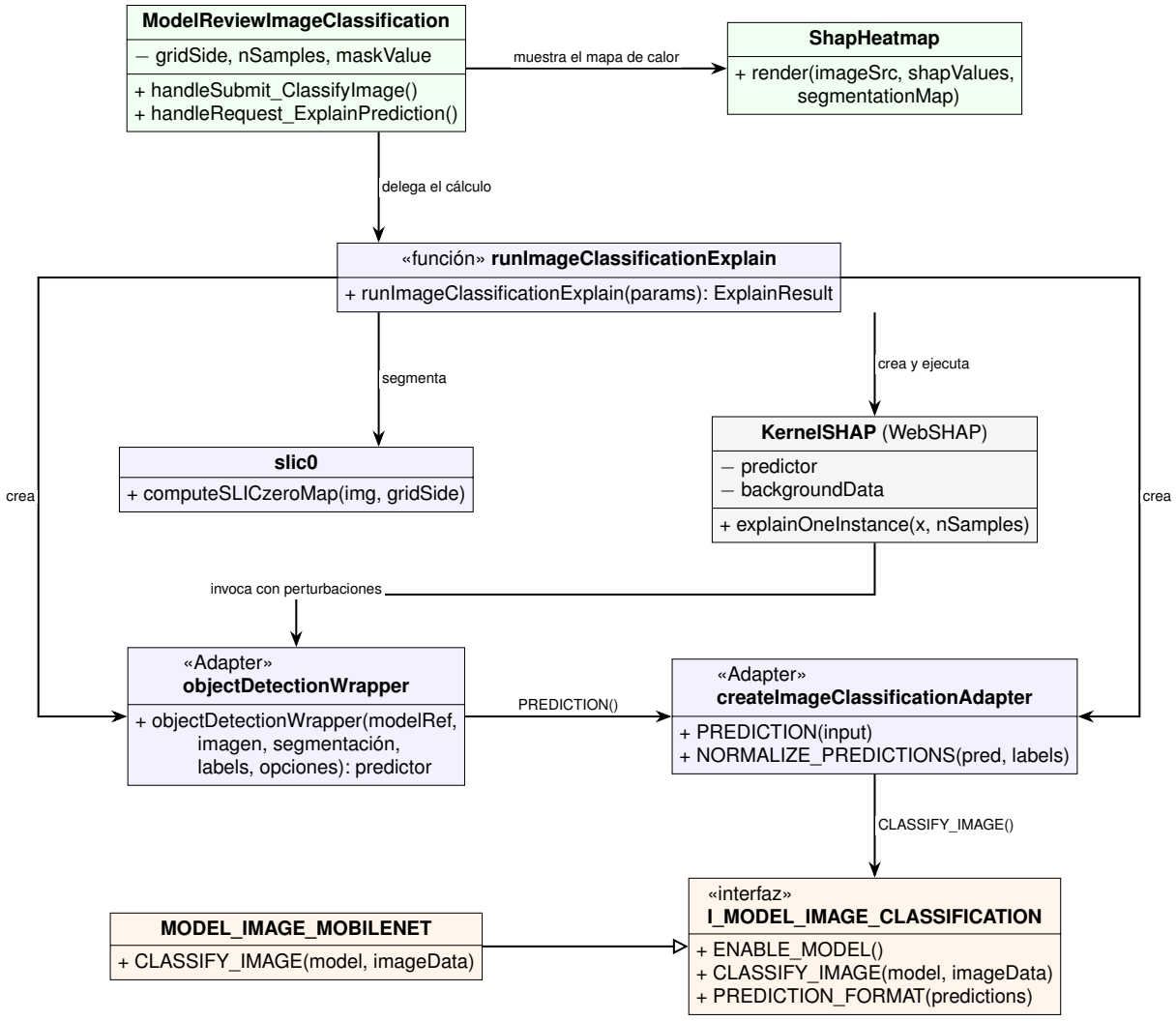


Figura 5.12: Diagrama de clases del incremento de clasificación de imágenes.

A continuación, la Figura 5.13 representa el diagrama de secuencia del caso de uso CU-3. Debemos tener en cuenta que tratamos con vectores binarios de regiones de la imagen, por lo que la función predictora va procesando la imagen perturbada al desactivar o activar diferentes regiones de la imagen.

La pantalla donde vive todo esto vuelve a ser la tarjeta de explicabilidad de la revisión del modelo, cuyo *storyboard* se recoge en la Figura 5.14. Además del botón para lanzar la explicación, la tarjeta incorpora los dos parámetros que controlan el equilibrio entre calidad y tiempo de cálculo (RF-6), que son el lado de la rejilla de segmentación y el número de muestras. Como apoyo didáctico, mientras se calcula la explicación vamos guardando algunas de las imágenes perturbadas que evalúa el modelo y las mostramos en una pequeña galería, de forma que el usuario puede ver con sus propios ojos qué le estamos enseñando al modelo, y el resultado final se presenta como un mapa de calor por cada clase considerada.

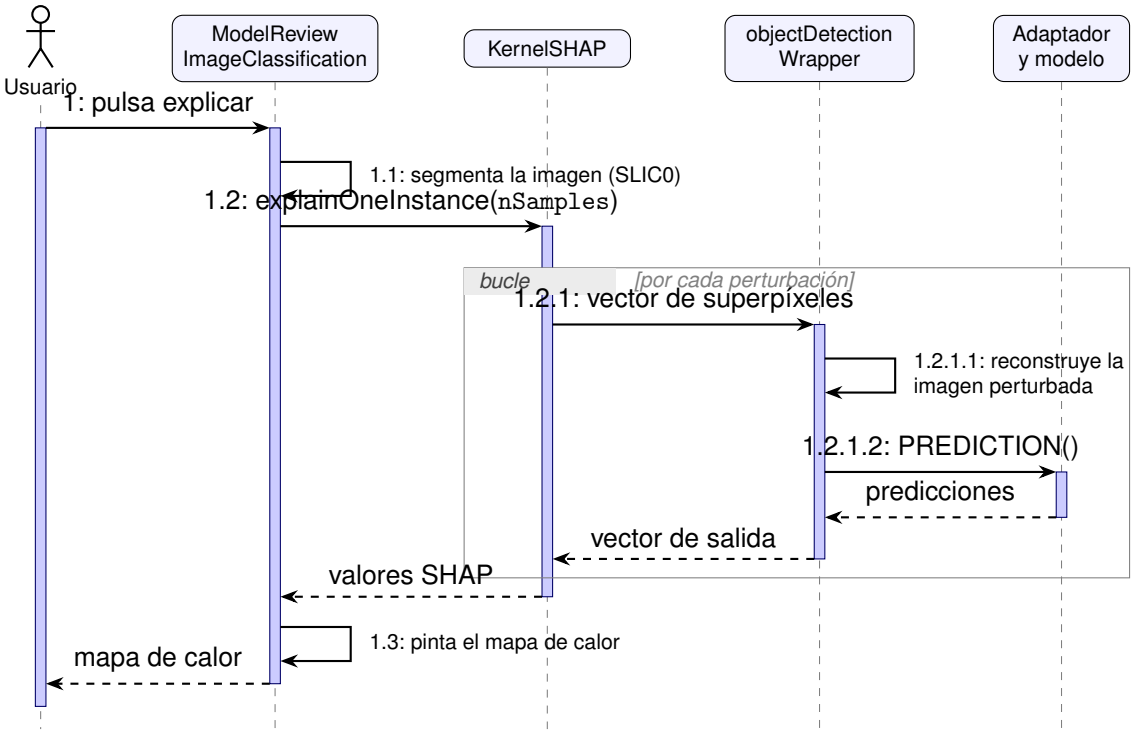


Figura 5.13: Diagrama de secuencia del caso de uso CU-3: explicación de una clasificación de imagen mediante SHAP.

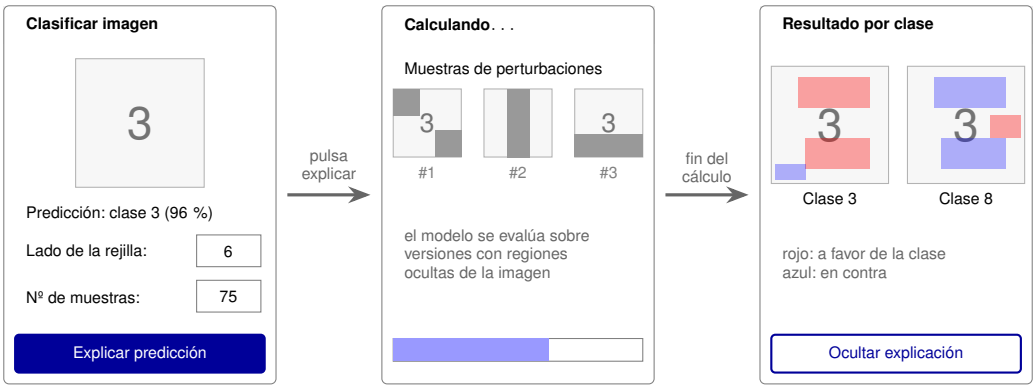


Figura 5.14: *Storyboard* de la explicación de imágenes: parámetros, cálculo con la galería de perturbaciones y mapas de calor por clase.

5.4.3. Implementación

Para seguir el proceso completo tomaremos un ejemplo real sobre el modelo de MobileNet. La Figura 5.15 recoge la predicción de partida, en la que el modelo clasifica la imagen como *cheeseburger* con una confianza altísima.

Las entradas que maneja KernelSHAP no son píxeles, sino vectores binarios que indican qué superpíxeles están activos. La función predictora reconstruye a partir de cada vector una versión perturbada de la imagen que mantiene visibles las regiones activas y enmascara el resto según la preferencia del usuario, y después consulta al

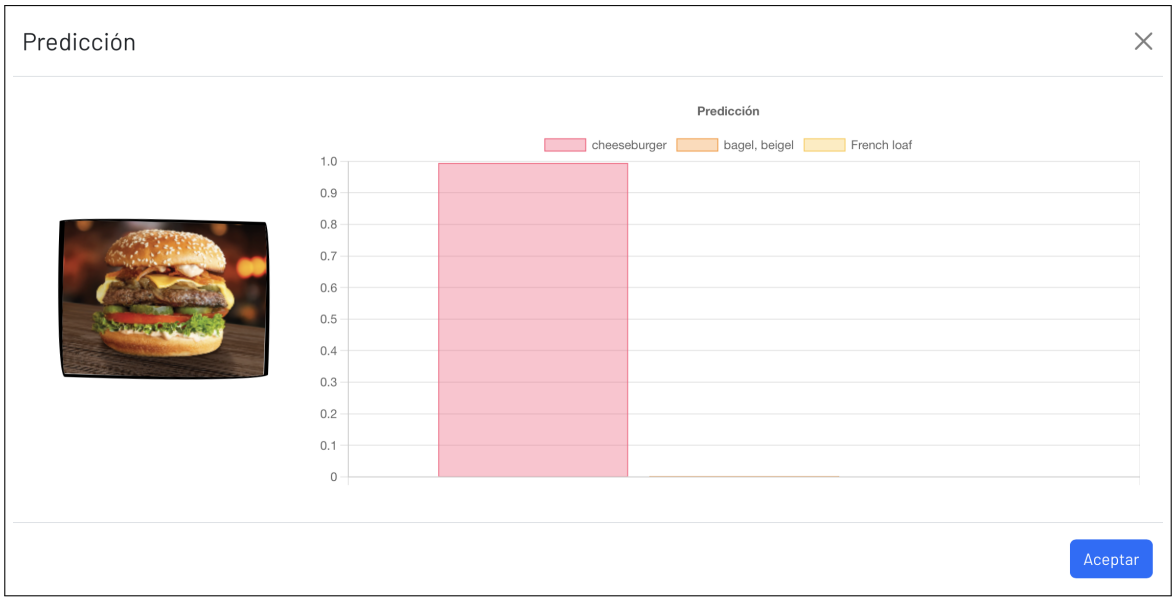


Figura 5.15: Predicción de partida del modelo de clasificación de imágenes (MobileNet) sobre la que se calculará la explicación.

modelo a través del adaptador, que traduce la llamada a la operación de clasificación del modelo y convierte su salida en un vector de longitud fija. El Código 5.4 muestra este proceso.

Código 5.4

```
return async (x) => {
  const batchVectors = [];
  for (let i = 0; i < x.length; i++) {
    // x[i] es el vector que indica qué superpíxeles están activos
    const inputTensor = tf.tidy(() => {
      const values = tf.tensor1d(Array.from(x[i]).flat());
      // Se proyecta el vector de superpíxeles sobre el mapa de segmentación
      const maskFlat = values.gather(segmentationTensor);
      const mask3d = maskFlat.reshape([height, width, 1]);
      const mask = mask3d.mul(maskMul).add(maskValue);
      return imgToTensor.mul(mask).toInt();
    });

    // El tensor perturbado se convierte de nuevo en imagen para el modelo
    await tf.browser.toPixels(inputTensor, reusableCanvas);

    // Se consulta al modelo a través del adaptador (interfaz común)
    const detections = await modelRef.PREDICTION(reusableCanvas,
    options);
    batchVectors.push(modelRef.NORMALIZE_PREDICTIONS(detections,
    labels));
```

```
20
21      inputTensor.dispose();
22    }
23    return batchVectors;
24  };
```

Código 5.4: Función predictora para imágenes: reconstrucción de la imagen perturbada y consulta al modelo

El uso de `tf.tidy` y de `dispose` es fundamental dado que cada explicación requiere de un número muy elevado de evaluaciones y sin estos la memoria de la GPU se agotaría rápidamente en el navegador.

Una vez obtenidas las contribuciones, la visualización se presenta dibujando sobre un *canvas* la imagen original y superponiéndole una capa de color. Para cada píxel miramos a qué superpíxel pertenece y recuperamos el valor de contribución de ese segmento. El color representa el signo (rojo para las contribuciones a favor de la clase, azul para las que van en contra) y la transparencia es proporcional a la magnitud, normalizada respecto al valor absoluto máximo, con una opacidad base en torno al 60 %. El Código 5.5 muestra el núcleo de ese pintado. El resultado de todo el proceso se recoge en la Figura 5.16. En la parte superior aparece la galería con algunas de las imágenes perturbadas que se han evaluado, y debajo un mapa de calor por cada una de las tres clases más probables. Para la clase predicha, el color rojo se concentra sobre la hamburguesa y el azul sobre el fondo, mientras que en las otras dos clases el reparto se invierte parcialmente.

Código 5.5

```
// Para cada píxel, buscamos su superpíxel y su contribución
const segmentId = map[i];
const shapVal = valuesToPaint[segmentId];
const absVal = Math.abs(shapVal);

// Ignoramos el ruido: por debajo del 5 por ciento del máximo no se pinta
if (absVal < maxAbsValue * 0.05) continue;

// Intensidad proporcional a la magnitud; color según el signo
const alpha = (absVal / maxAbsValue) * opacity;  // opacity  0.6
const r = shapVal > 0 ? 255 : 0;    // rojo -> contribución positiva
const b = shapVal > 0 ? 0 : 255;    // azul -> contribución negativa

// Mezcla alfa sobre el píxel original de la imagen
```

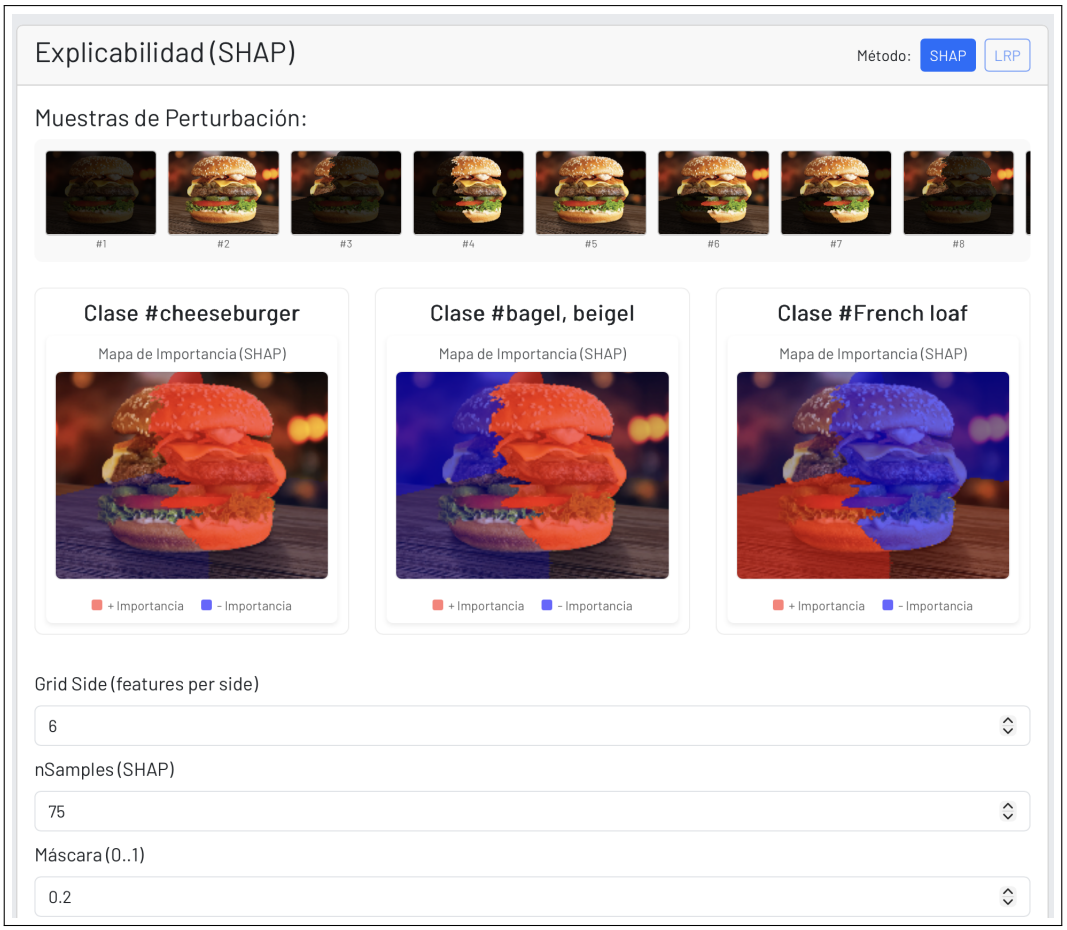


Figura 5.16: Explicación mediante SHAP sobre MobileNet: galería de perturbaciones y mapa de calor por clase (rejilla de 6 y 75 muestras).

```
15 data[idx]     = data[idx]     * (1 - alpha) + r * alpha;
16 data[idx + 1] = data[idx + 1] * (1 - alpha);
17 data[idx + 2] = data[idx + 2] * (1 - alpha) + b * alpha;
```

Código 5.5: Núcleo del mapa de calor: color según el signo e intensidad según la magnitud

5.4.4. Plan de pruebas

La validación funcional ha consistido en comprobar que el mapa de calor resalta las regiones de la imagen realmente asociadas a la clase predicha, y no el fondo ni otras zonas irrelevantes. También hemos verificado el efecto del parámetro `nSamples`, ya que al aumentarlo la explicación se estabiliza a costa de un mayor tiempo de cálculo. La Figura 5.17 repite la explicación de la Figura 5.16 sobre la misma imagen y la misma rejilla, elevando `nSamples` de 75 a 300, de modo que las regiones dominantes

se mantienen, pero los límites entre superpíxeles quedan mejor definidos y desaparecen algunas atribuciones aisladas del fondo. La Tabla 5.4 resume los casos de prueba del incremento.

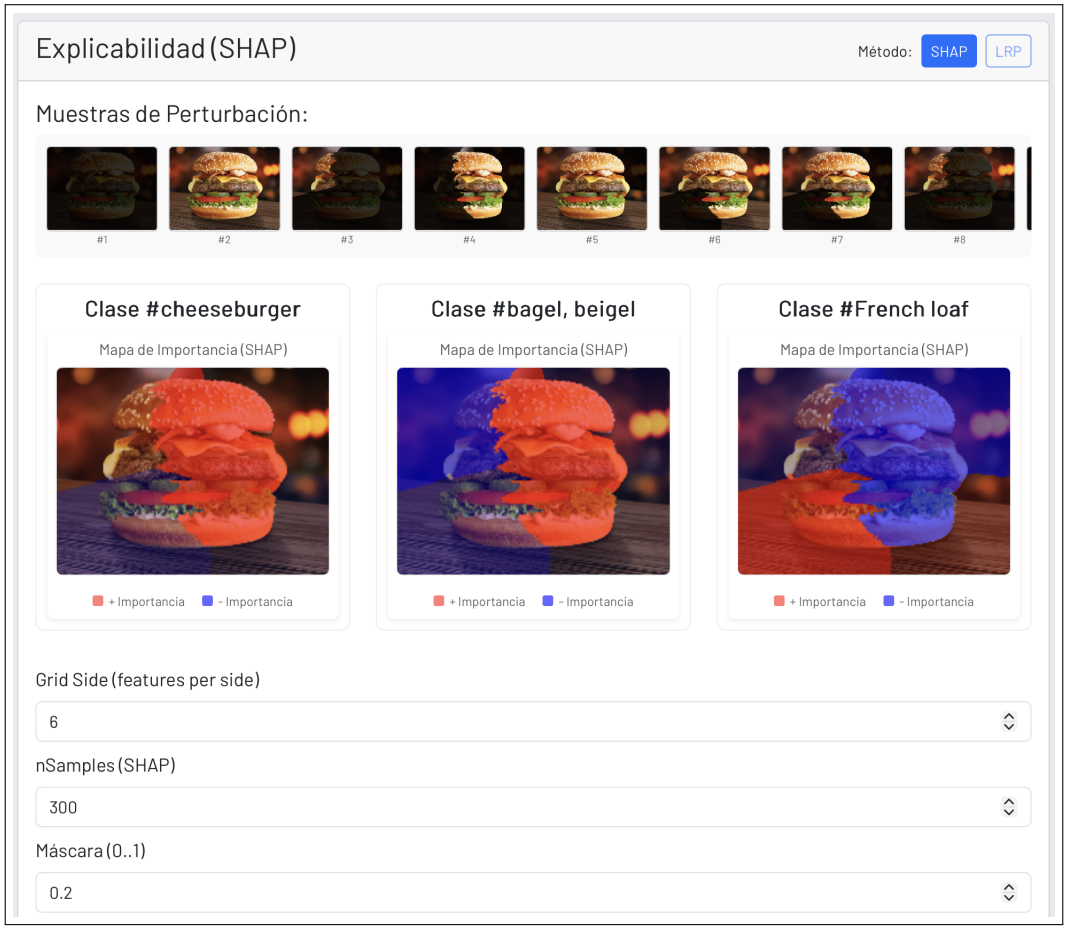


Figura 5.17: Misma explicación de la Figura 5.16 elevando el número de muestras de 75 a 300.

ID	Caso de prueba	Resultado esperado	Resultado
PF-3.1	Explicación con MobileNet de una imagen clasificada como *cheeseburger* (rejilla de 6 y 75 muestras).	Las regiones de la hamburguesa aparecen a favor de la clase y el fondo en contra.	OK
PF-3.2	Repetir la explicación elevando el número de muestras de 75 a 300.	Se mantienen las regiones dominantes y el mapa se estabiliza, a cambio de más tiempo de cálculo.	OK
PF-3.3	Observar la galería durante el cálculo.	Se muestran imágenes perturbadas con distintas regiones ocultas.	OK
PF-3.4	Solicitar la explicación sin haber clasificado ninguna imagen.	El sistema avisa al usuario y no lanza el cálculo.	OK

Tabla. 5.4: Pruebas funcionales del incremento de clasificación de imágenes.

5.5. Incremento 4: explicabilidad en detección de objetos

5.5.1. Objetivo del incremento

El objetivo de este incremento es desarrollar el caso de uso CU-4 (*Explicar una detección de objetos*, Tabla 3.7), que permita al usuario ver qué regiones de la imagen han contribuido más o menos a cada una de las detecciones. En este incremento se presenta el desafío de adaptarnos a cada modelo de detección ya que cada uno fuerza a adaptar tanto la función predictora como la segmentación.

5.5.2. Diseño detallado

La dificultad de esta tarea es que la salida de un detector no es un vector de probabilidades de tamaño fijo como en la clasificación, sino una lista variable de detecciones, cada una con su caja, su clase y su confianza de forma que de una imagen a otra el detector puede devolver un número distinto de objetos. Como KernelSHAP necesita comparar las predicciones de unas perturbaciones con otras y para ello requiere una salida de longitud constante, fijamos primero, a partir de las detecciones

de la imagen original, el conjunto de etiquetas que nos interesan, y para cada perturbación normalizamos la salida del detector a un vector de esa misma longitud (`NORMALIZE_PREDICTIONS`). Si en una perturbación el modelo deja de detectar algo, esa posición queda a cero.

La Figura 5.18 recoge el diagrama de clases del incremento. Para el detector de caras con el modelo FaceMesh obtenemos los puntos de la cara y se construye la segmentación facial. Para el resto de detectores utilizamos SLICO. La función predictora trabaja directamente con la interfaz de los detectores donde cada modelo implementa predicción y normalización de las detecciones.

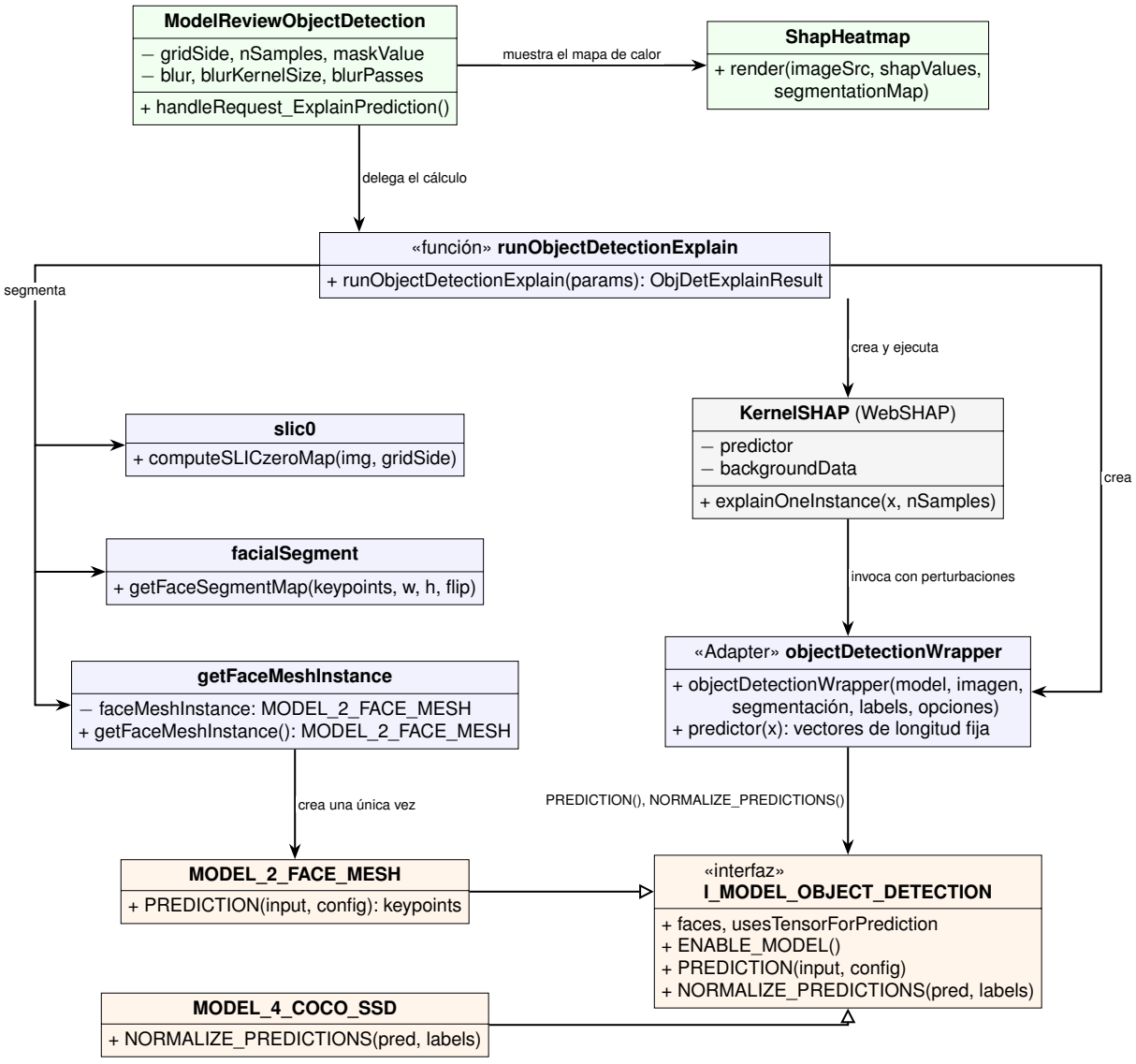


Figura 5.18: Diagrama de clases del incremento de detección de objetos.

La Figura 5.19 muestra el diagrama de secuencia del caso de uso CU-4, en el que a diferencia de los anteriores, se predice una vez sobre la imagen original para poder obtener las etiquetas correspondientes.

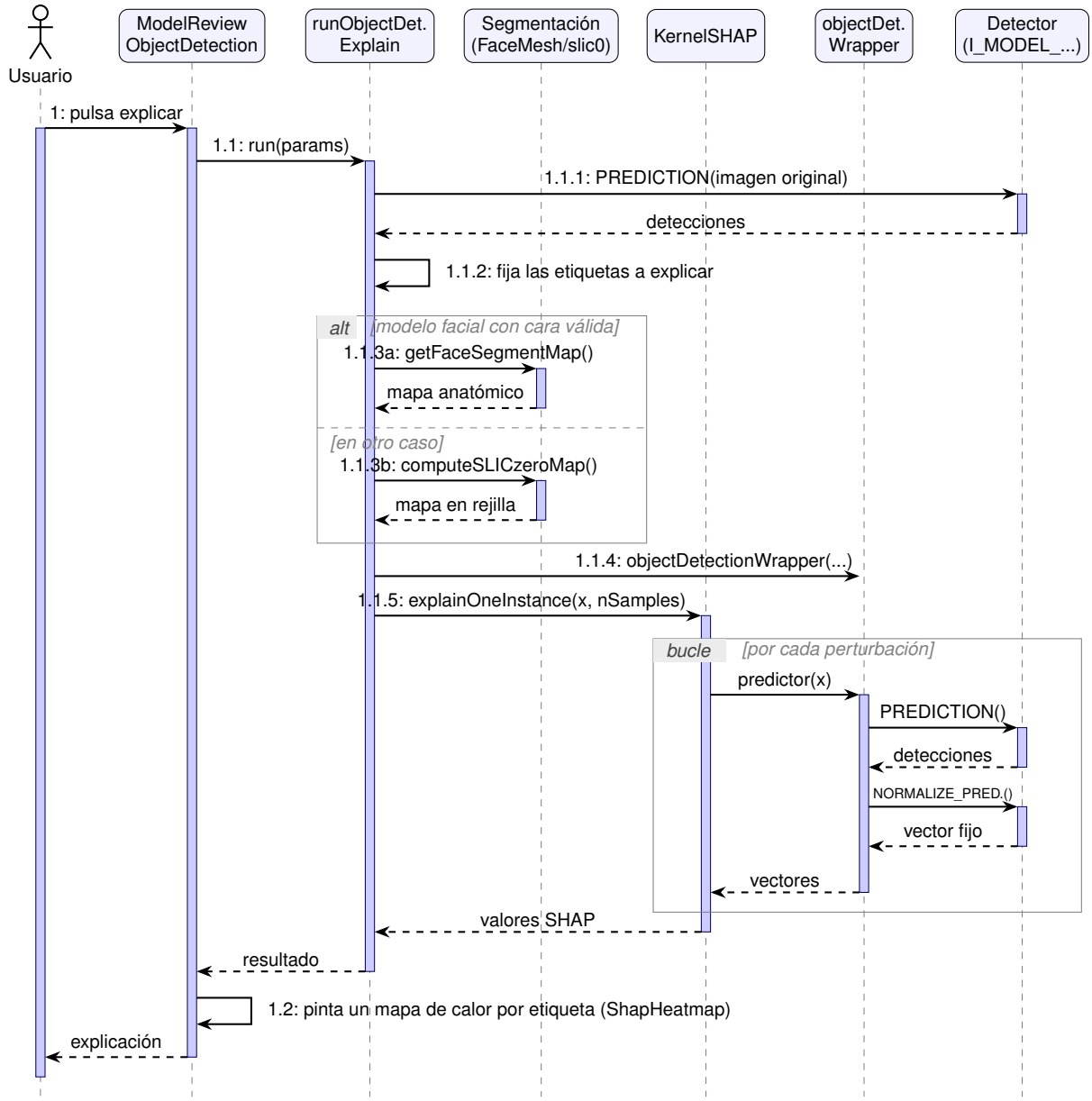


Figura 5.19: Diagrama de secuencia del caso de uso CU-4: explicación de una detección de objetos.

Como complemento, la Figura 5.20 resume el flujo completo del incremento en forma de diagrama de actividad, incluyendo la decisión de qué segmentación emplear en cada caso, que detallamos en el apartado de implementación.

Se puede añadir una máscara a la hora de ejecutar la explicabilidad, de forma que el usuario puede oscurecer o difuminar regiones desactivadas. Esto se puede ver reflejado en el storyboard. El diseño de la explicabilidad sigue en la línea de la clasificación de imágenes.

Todas las opciones de la máscara tienen su reflejo en la interfaz, cuyo *storyboard* se recoge en la Figura 5.21. La tarjeta de explicabilidad sigue el mismo diseño que

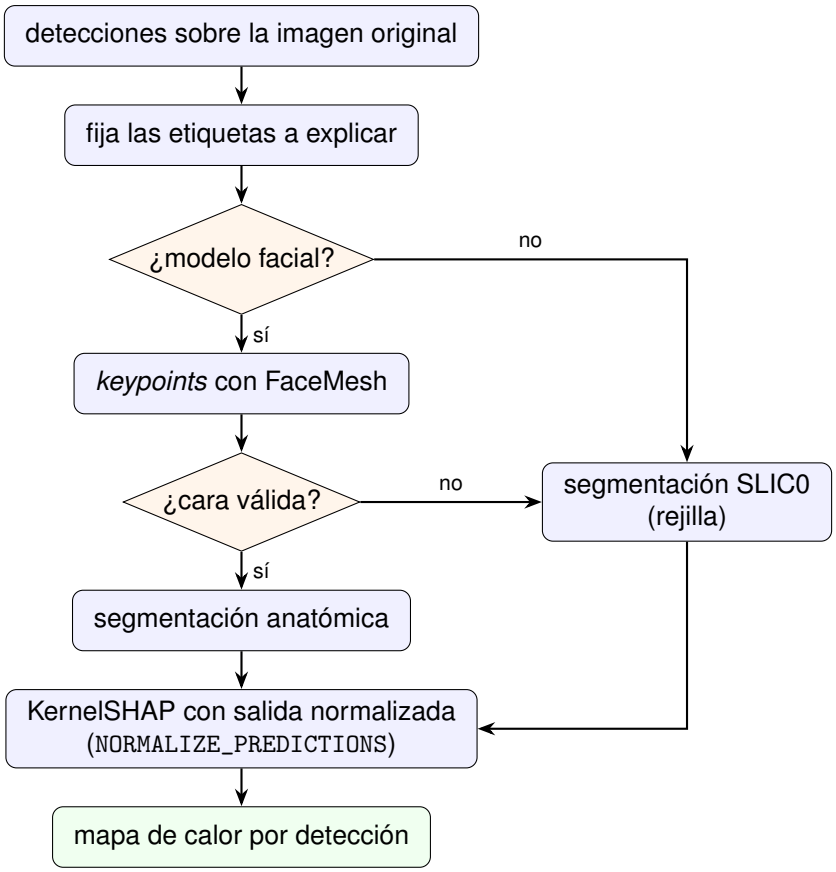


Figura 5.20: Flujo de una explicación en detección de objetos, con la elección de la segmentación.

en la clasificación de imágenes, pero añade los controles propios de la máscara de perturbación, de forma que el usuario puede elegir entre oscurecer o difuminar las regiones desactivadas y, en este último caso, ajustar el tamaño del desenfoque y el número de pasadas.

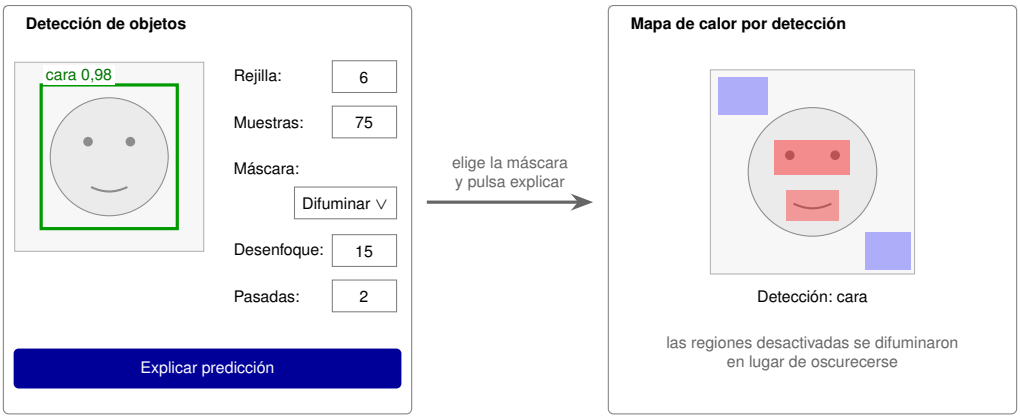


Figura 5.21: *Storyboard* de la explicación en detección de objetos: detección original con los controles de la máscara y mapa de calor resultante.

5.5.3. Implementación

La segmentación también es distinta en este incremento, porque no siempre usamos la rejilla de superpíxeles. Para los detectores generales volvemos a usar SLICO, igual que en la clasificación de imágenes, pero cuando el modelo es facial aprovechamos que disponemos de un modelo de malla facial (FaceMesh) para obtener los puntos característicos (*keypoints*) de la cara y construir con ellos una segmentación anatómica, de manera que los superpíxeles se corresponden con regiones reales de la cara en lugar de con cuadrados arbitrarios. Como respaldo, si no se detecta ninguna cara o la segmentación sale degenerada, volvemos automáticamente a SLICO. El Código 5.6 recoge esta elección.

Código 5.6

```
if (model.faces) {
  // Modelos faciales: keypoints de FaceMesh -> segmentación anatómica
  const faceMesh = await getFaceMeshInstance();
  const mesh = await faceMesh.PREDICTION(imageData, {
    flipHorizontal });
  ({ mapArray, numSegments } = getFaceSegmentMap(
    mesh[0].keypoints, imageData.width, imageData.height,
    flipHorizontal,
  ));

  // Respaldo: si no hay cara útil, caemos a una rejilla SLIC0
  if (!numSegments || numSegments <= 1) {
    ({ mapArray, numSegments } = computeSLICzeroMap(imageData,
    gridSide));
  }
} else {
  // Resto de detectores (COCO-SSD, etc.): rejilla SLIC0
  ({ mapArray, numSegments } = computeSLICzeroMap(imageData,
    gridSide));
}
```

Código 5.6: Elección de la segmentación en detección de objetos: *keypoints* faciales con respaldo en SLIC0

Por último, también cambia la manera de enmascarar las regiones. En la clasificación nos basta con oscurecer los superpíxeles desactivados, pero en la detección —y sobre todo en las caras— oscurecer del todo una región puede confundir al detector, porque una cara llena de parches negros deja de parecer una cara. Por eso, en lugar del oscurecido simple, damos la opción de mezclar la imagen original con una versión

desenfocada de sí misma, obtenida aplicando varias pasadas de *average pooling*, de forma que las regiones desactivadas se difuminan en vez de desaparecer y conseguimos perturbaciones mucho más naturales. El resto del proceso es idéntico al de la clasificación de imágenes y el resultado vuelve a mostrarse como un mapa de calor. La Figura 5.22 recoge el resultado sobre el detector general COCO-SSD, con un mapa de calor por cada una de las etiquetas detectadas en la imagen original.

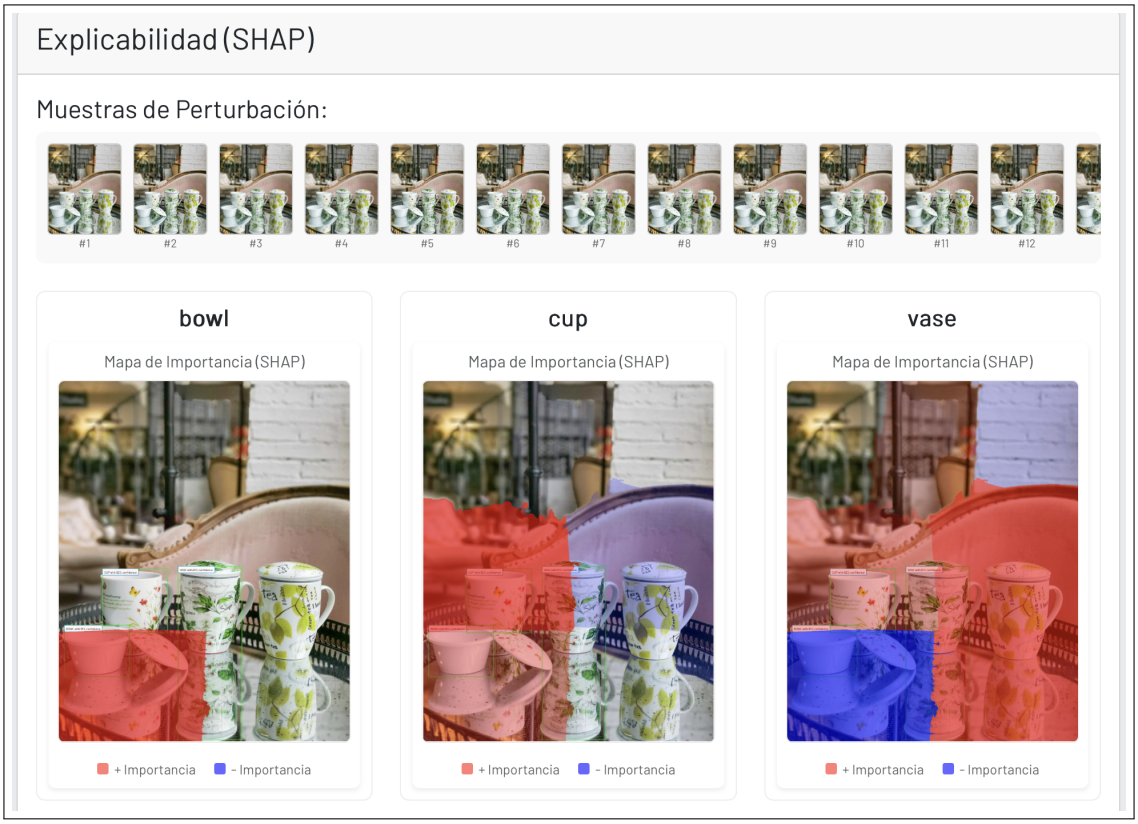


Figura 5.22: Explicación de una detección con COCO-SSD: un mapa de calor por cada etiqueta detectada.

La Figura 5.23 muestra el resultado en un modelo facial. Aquí los superpíxeles ya no son cuadrados de una rejilla, sino regiones anatómicas obtenidas a partir de los *keypoints* de FaceMesh, de forma que las contribuciones se reparten sobre partes reconocibles de la cara. En la clase *Enfadado*, por ejemplo, la boca aparece en azul, es decir, contradice esa etiqueta.

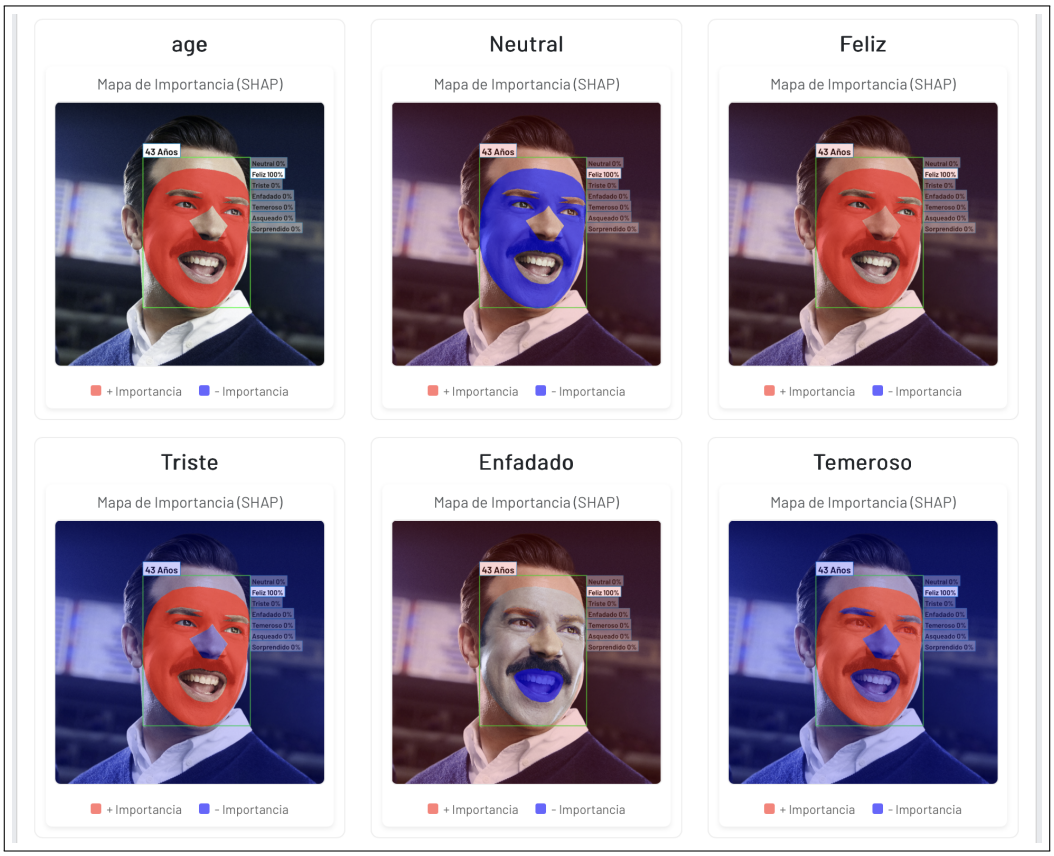




Figura 5.23: Explicación de una detección facial con segmentación anatómica: mapas de calor de las clases de edad y emoción (arriba, primeras clases; abajo, las dos restantes de la misma tarjeta).

5.5.4. Plan de pruebas

La validación funcional ha comprobado que, tanto en los detectores generales como en los faciales, el mapa de calor resalta las regiones que realmente motivan cada detección, y que el respaldo a SLICO entra correctamente cuando la segmentación facial no es viable.

Como caso límite hemos probado además con una imagen de estilo ilustrado, recogida en la Figura 5.24. Al no tratarse de un rostro real, el detector no llega a producir detecciones válidas sobre las que trabajar, así que no hay contribuciones que repartir y el mapa de calor se muestra vacío. Lo relevante de esta prueba no es el resultado en sí, sino que el módulo no falla ni bloquea la interfaz ante una entrada fuera del dominio del modelo, sino que completa el proceso y devuelve una explicación sin regiones resaltadas. La Tabla 5.5 resume todos los casos de prueba del incremento.

ID	Caso de prueba	Resultado esperado	Resultado
PF-4.1	Explicación de una imagen con COCO-SSD.	Se obtiene un mapa de calor por cada etiqueta detectada que resalta las regiones del objeto.	OK
PF-4.2	Explicación de un rostro real con un modelo facial.	La segmentación es anatómica y las contribuciones se reparten sobre partes de la cara.	OK
PF-4.3	Imagen de estilo ilustrado, fuera del dominio del modelo.	El proceso termina sin errores ni bloqueos, con un mapa sin regiones resaltadas.	OK

Tabla. 5.5: Pruebas funcionales del incremento de detección de objetos.

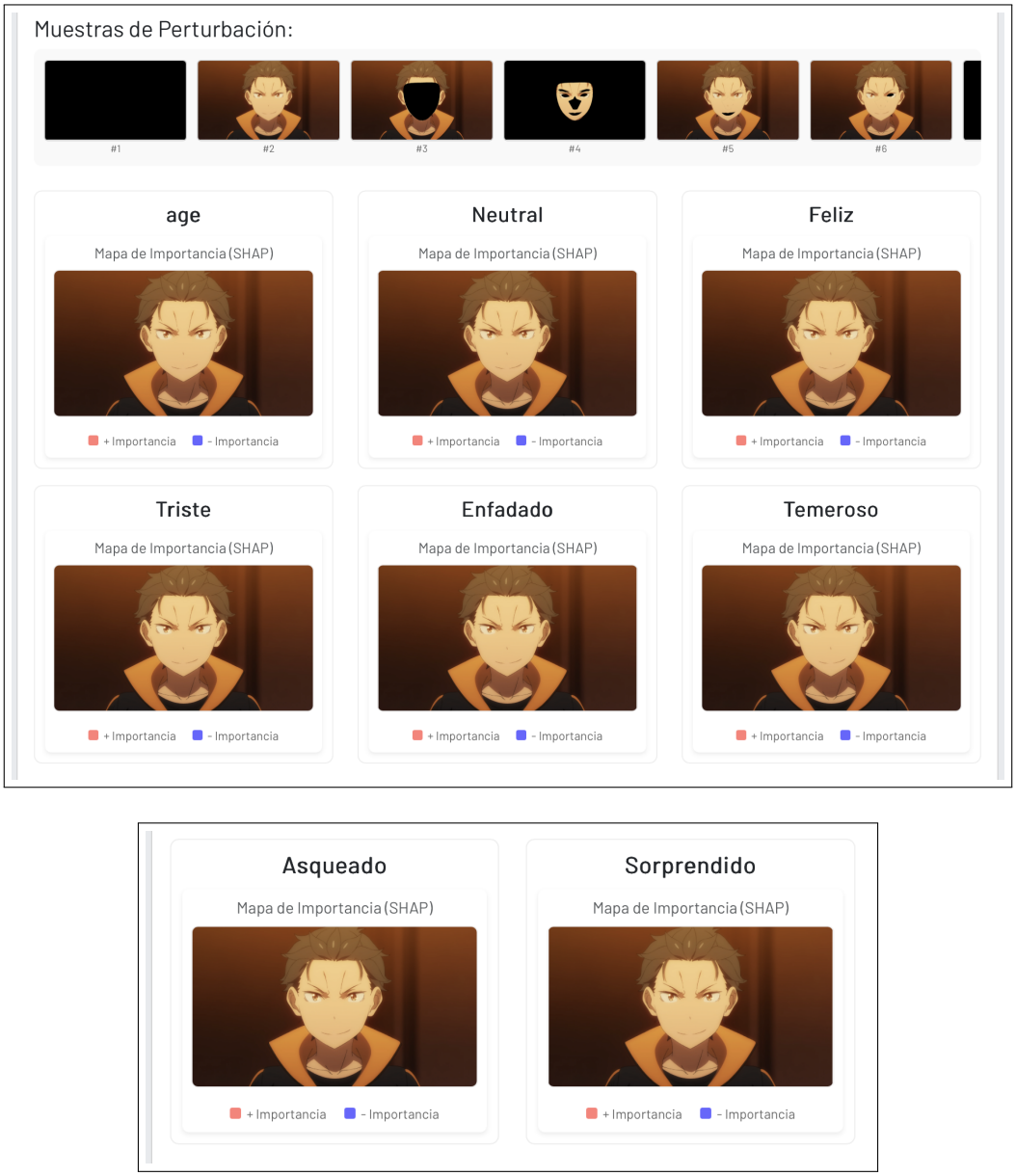


Figura 5.24: Caso límite en detección facial: al no ser un rostro real, el modelo no produce detecciones válidas y la explicación resultante no resalta ninguna región.

5.6. Incremento 5: explicabilidad específica mediante LRP

5.6.1. Objetivo del incremento

El objetivo de este último incremento es desarrollar el caso de uso CU-5 (*Explicar una clasificación de imagen mediante LRP*, Tabla 3.8), el cual permite al usuario ver la relevancia a los píxeles que conforman la imagen de entrada que en este caso sería un número. El resto de modelos carece de accesibilidad al interior del mismo. Al incorporar un método completamente distinto del resto permite comparar la explicabilidad agnóstica de SHAP con la específica que se tratan en el resto de modelos.

Cuando hablamos de los métodos más importantes dentro de la Inteligencia Artificial Explicable, LRP (*Layer-wise Relevance Propagation*) es sin duda esencial [13]. Se basa en descomponer la salida del clasificador propagándola hacia atrás por toda la red de manera que, empezando por la predicción, vamos repartiendo una puntuación de relevancia entre las características de entrada (los píxeles) hasta saber cuánto ha contribuido cada una a la decisión final. La Figura 5.25 ilustra esta idea.

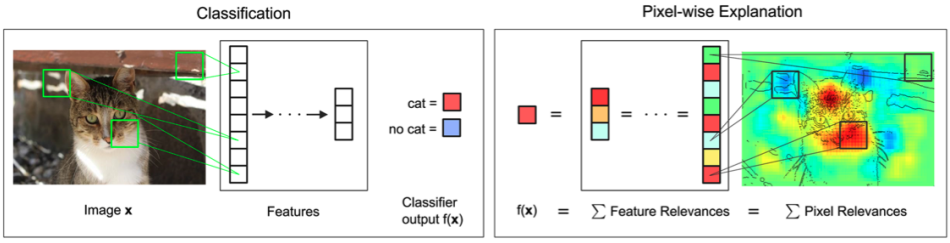


Figura 5.25: Ilustración del funcionamiento de LRP.

5.6.2. Diseño detallado

A diferencia de los métodos basados en el gradiente, que solo tienen en cuenta el entorno inmediato del punto de predicción y pueden quedarse atascados en óptimos locales poco representativos, LRP se apoya en un principio estricto de conservación de la relevancia de manera que la evidencia que sostiene una predicción no aparece de la nada ni se pierde por el camino. Este principio nos dice que, en cada capa, la suma de las relevancias que entran tiene que coincidir con la suma de las que salen hacia la capa anterior:

$$\sum_i R_{i \leftarrow j}^{(l,l+1)} = R_j^{(l+1)} \tag{5.1}$$

Así conseguimos que toda la relevancia asociada a la predicción final se vaya repartiendo hacia atrás hasta llegar a los píxeles de entrada, sin que se genere ni se disipe a medida que la señal recorre la red [13].

El diagrama que representa el incremento se recoge en la Figura 5.26. Se puede apreciar que a la hora de explicar la predicción se requiere de poder mirar dentro del modelo, es por ello que hacemos uso de dos operaciones que permiten capturar las activaciones de las capas y otra para propagar la relevancia. El modelo se almacena en caché para poder reutilizarlo sin tener que cargarlo nuevamente. Dado que diferentes capas requieren distintas reglas, `applyLRP` se encarga de elegir la regla adecuada según el tipo de cada capa.

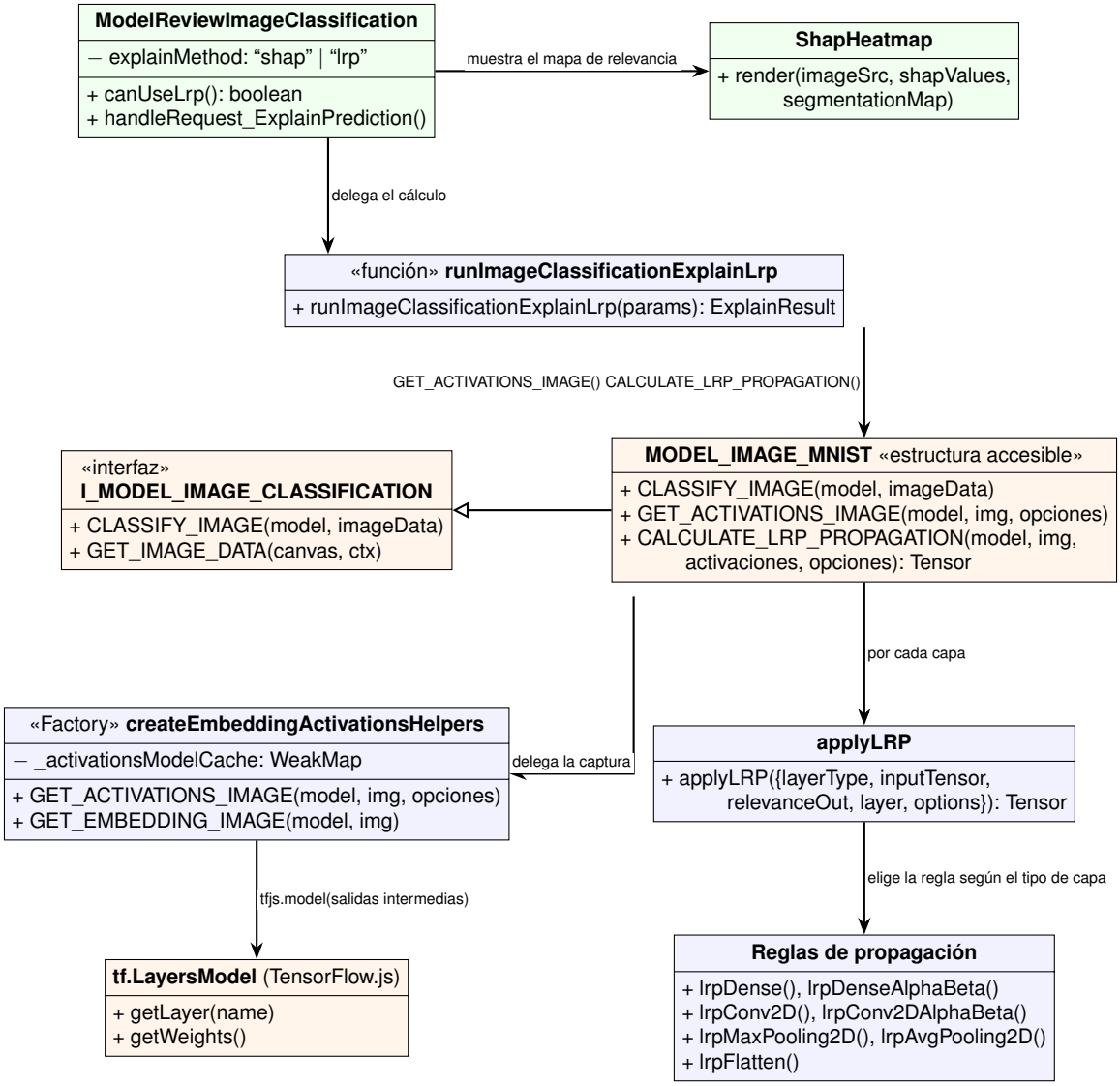


Figura 5.26: Diagrama de clases del incremento de explicabilidad mediante LRP.

La Figura 5.27 muestra el diagrama de secuencia del caso de uso CU-5. En este caso vemos que el bucle de perturbaciones no es necesario dado que obtenemos los

datos a partir de las activaciones de la red.

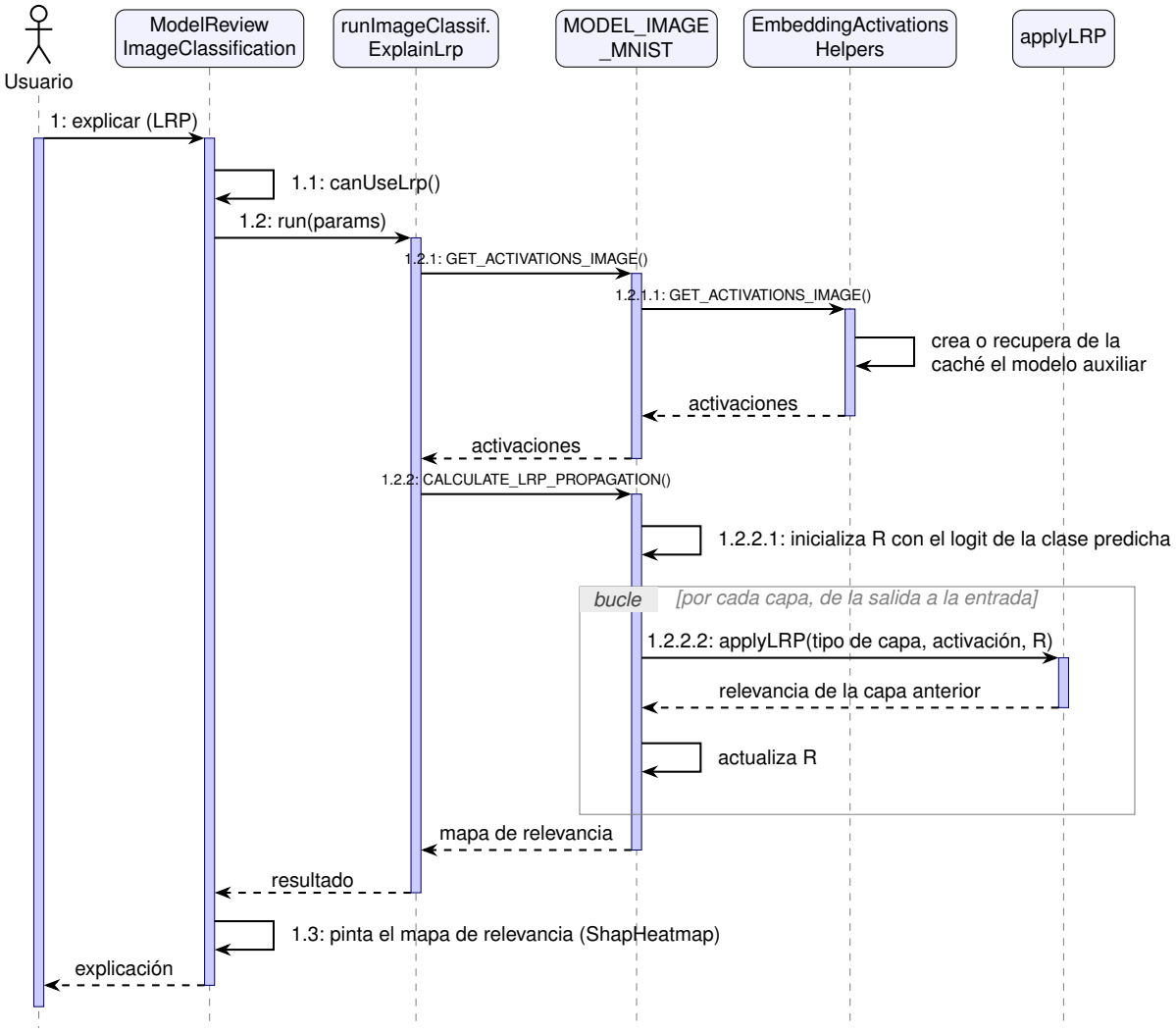
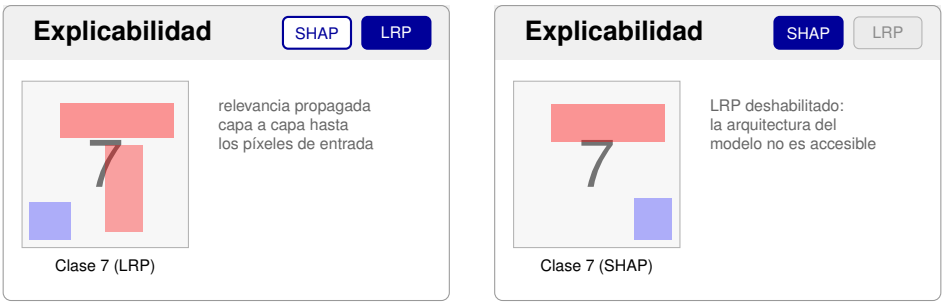


Figura 5.27: Diagrama de secuencia del caso de uso CU-5: explicación de una clasificación de imagen mediante LRP.

La interfaz de usuario es prácticamente idéntica a la de SHAP. Este método es exclusivo del modelo de detección de números. En un prototipo inicial 5.28 se había planteado un selector de método pero dado que este modelo no funciona bien para SHAP se ha decidido que sea LRP el único método disponible para este modelo.



(a) estructura accesible (b) estructura no accesible

Figura 5.28: Boceto del selector de método en la tarjeta de explicabilidad: con LRP disponible (a) y deshabilitado por no ser accesible la estructura interna del modelo (b).

5.6.3. Implementación

Al implementar la propagación hacia atrás no usamos una única fórmula, sino varias, porque cada tipo de capa tiene sus particularidades y no todas se comportan igual de bien numéricamente.

Regla épsilon (ϵ)

En las capas densas se realizan muchísimos cálculos y es fácil que aparezca ruido numérico, sobre todo cuando el denominador se acerca a cero. Para evitar esos valores problemáticos, en este tipo de capas hacemos uso de la regla épsilon [14], que introduce un pequeño término estabilizador ϵ en el denominador al calcular la relevancia:

$$R_i = \sum_j \frac{x_i \cdot w_{ij}}{z_j + \epsilon \cdot \text{sign}(z_j)} R_j \tag{5.2}$$

Código 5.7

```
// Pase hacia adelante matricial: z = x * W + b
let z = inputTensor.matMul(weights);
if (bias) {
  z = z.add(bias);
}

// Inyección del estabilizador epsilon respetando el signo del tensor original
const stabilizer = tfjs.where(
  z.greaterEqual(tfjs.scalar(0)),
  tfjs.scalar(epsilon),
  tfjs.scalar(-epsilon)
```

```
);
z = z.add(stabilizer);

// Retropropagación matemática estabilizada (s = Rout / z)
const s = relevanceOut.div(z);
const c = s.matMul(weights, false, true);

// Cálculo final de la relevancia de entrada (Rin)
const relevanceIn = inputTensor.mul(c);
```

Código 5.7: Implementación de la regla épsilon (ϵ) en capas densas

Regla alfa-beta ($\alpha\beta$)

En las capas convolucionales, más cercanas a la imagen original, es donde se detectan los bordes, las texturas y las formas, así que aquí no solo nos interesa saber qué apoya la predicción (excitación), sino también qué la contradice (inhibición). Para tratar por separado ambas cosas hacemos uso de la regla $\alpha\beta$ [13], que mantiene diferenciadas las activaciones positivas y negativas y las pondera mediante dos parámetros, α y β:

$$R_{i\leftarrow j}^{(l,l+1)} = R_j^{(l+1)} \cdot \left(\alpha \cdot \frac{z_{ij}^{+}}{z_j^{+}} + \beta \cdot \frac{z_{ij}^{-}}{z_j^{-}} \right) \tag{5.3}$$

En el código separamos los pesos en su parte positiva (w^+) y su parte negativa (w^-) y propagamos cada una por su lado, para que las contribuciones que suman y las que restan no se mezclen. Para que se siga cumpliendo la conservación de la relevancia trabajamos siempre bajo la restricción $\alpha + \beta = 1$ [13]. El Código 5.8 muestra la regla para las capas densas; en las convolucionales el procedimiento es el mismo, sin más que sustituir el producto matricial por la operación de convolución.

Código 5.8

```
// Bifurcación del tensor de pesos en excitaciones (wPos) e inhibiciones (wNeg)
const wPos = tfjs.maximum(weights, 0);
const wNeg = tfjs.minimum(weights, 0);

// Pase hacia adelante asimétrico y paralelo
let zPos = inputTensor.matMul(wPos).add(epsilon);
let zNeg = inputTensor.matMul(wNeg).sub(epsilon);

// Ponderación de la relevancia mediante los hiperparámetros de control
```

```
const sPos = relevanceOut.div(zPos).mul(alpha);
const sNeg = relevanceOut.div(zNeg).mul(beta);

// Proyección hacia la capa inferior (matrices traspuestas)
const cPos = sPos.matMul(wPos, false, true);
const cNeg = sNeg.matMul(wNeg, false, true);

// Fusión de las evidencias para obtener la relevancia conservada
const relevanceIn = inputTensor.mul(cPos.add(cNeg));
```

Código 5.8: Aplicación de la regla alfa-beta ($\alpha\beta$) en capas densas

Capas de reducción espacial (*pooling*)

Durante la propagación hacia atrás, LRP también tiene que deshacer los cambios de forma que introducen algunas capas de una red convolucional, como las capas *flatten* o las de *pooling*, porque de lo contrario no podríamos devolver la relevancia a su espacio original. Para el *max pooling*, la formulación estricta de LRP dice que la relevancia debería viajar únicamente hacia la posición de mayor activación, siguiendo un criterio de "el ganador se lo lleva todo" (*winner-takes-all*) [26]. El problema es que recuperar esos índices exactos (*argmax*) resulta muy costoso para la memoria de la GPU (WebGL) dentro del navegador y penalizaría demasiado la experiencia de usuario.

Por ese motivo optamos por una aproximación que se comporta como un agrupamiento promedio, es decir, expandimos el tensor hasta su tamaño original mediante interpolación bilineal (`resizeBilinear`) y repartimos la relevancia a partes iguales entre todos los píxeles que cubre cada región, dividiéndola por el área de la ventana. De este modo seguimos respetando la conservación de la relevancia, pero con un coste asumible en el cliente. El Código 5.9 recoge esta lógica.

Código 5.9

```
// 1. Deshacer la reducción espacial extrayendo la geometría original
const [poolH, poolW] = Array.isArray(poolSize) ? poolSize : [
    poolSize, poolSize];

// 2. Expansión geométrica mediante interpolación bilineal nativa
const resized = tfjs.image.resizeBilinear(relevanceOut, [inH, inW])
    ;
```

```
6
7  // 3. Normalización de la relevancia desplegada (conservación)
8  const poolArea = poolH * poolW;
9  const result = resized.div(poolArea);
10
11 return result;
```

Código 5.9: Despliegue espacial y normalización de la relevancia como aproximación al max pooling

Captura de las activaciones

Como se ve en las reglas anteriores, casi todas necesitan, además de la relevancia que les llega de la capa superior, las activaciones originales de esa capa. Para obtenerlas ejecutamos una inferencia hacia delante capturando el estado interno de la red. El problema es que TensorFlow.js, por su orientación al *frontend*, no nos deja acceder directamente a las activaciones intermedias, así que creamos modelos auxiliares cuya salida son precisamente esos tensores intermedios. Para no crearlos una y otra vez para las mismas capas los guardamos en una caché, tal y como muestra el Código 5.10.

Código 5.10

```
1  function _getOrCreateActivationsModel(model, layerNames) {
2    const key = layerNames.join('|');
3    let perModel = _activationsModelCache.get(model);
4
5    if (!perModel) {
6      perModel = new Map();
7      _activationsModelCache.set(model, perModel);
8    }
9
10   let cached = perModel.get(key);
11   if (cached) return cached;
12
13   const outputs = layerNames.map((name) => model.getLayer(name).
       output);
14   const actModel = tfjs.model({ inputs: model.inputs, outputs });
15
16   cached = { layerNames: [...layerNames], actModel };
17   perModel.set(key, cached);
```

```
18
19    return cached;
20  }
```

Código 5.10: Función de caché para optimizar la extracción de modelos de activación

Una vez se han guardado las activaciones y las reglas han sido implementadas debemos recorrer las capas en sentido inverso, mirando que tipo de capa es para aplicar la regla a usar y para cada una ellas. En este recorrido inverso se propaga la relevancia de la capa superior a la inferior hasta llegar a la imagen original. En la Figura 5.29 se puede apreciar de forma resumida el recorrido completo.

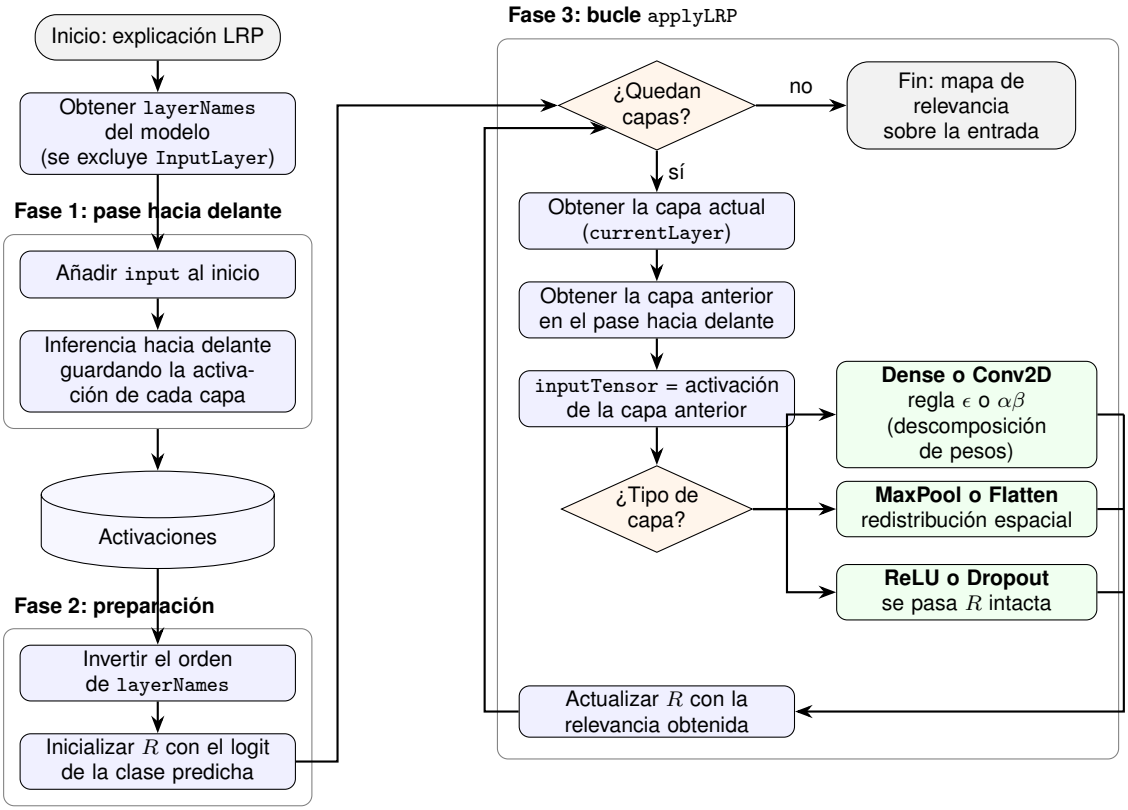


Figura 5.29: Flujo de ejecución de una explicación mediante LRP.

5.6.4. Plan de pruebas

Para validar este incremento hemos comprobado que la propagación de relevancia produce mapas coherentes con la predicción sobre los modelos con estructura accesible entrenados en la propia plataforma. Como comprobación adicional, hemos verificado que se cumple de forma aproximada el principio de conservación de la relevancia entre capas, lo que confirma que las reglas de propagación están bien implementadas.

Por último, hemos comprobado que el método LRP se deshabilita automáticamente cuando la estructura interna del modelo seleccionado no es accesible (RF-7).

La Figura 5.30 recoge una de estas explicaciones sobre un dígito dibujado a mano en el modelo de MNIST. A diferencia de los mapas obtenidos con SHAP, la relevancia no se reparte por regiones de una rejilla, sino que sigue el propio trazo del dígito, marcando en rojo los tramos que sostienen la clase predicha y en azul los que la contradicen. La Tabla 5.6 resume los casos de prueba del incremento.



Figura 5.30: Explicación mediante LRP sobre un modelo con estructura accesible entrenado en la plataforma: la relevancia sigue el trazo del carácter.

ID	Caso de prueba	Resultado esperado	Resultado
PF-5.1	Explicación LRP de un dígito dibujado a mano en el modelo de MNIST.	La relevancia sigue el trazo del dígito, a favor y en contra de la clase predicha.	OK
PF-5.2	Comprobar la suma de la relevancia entre capas consecutivas.	La relevancia total se conserva de forma aproximada.	OK
PF-5.3	Explicación LRP sobre un modelo entrenado por el usuario en la plataforma.	El método está disponible y genera el mapa de relevancia.	OK
PF-5.4	Seleccionar un modelo sin estructura accesible, como MobileNet.	La opción LRP aparece deshabilitada y solo se puede usar SHAP.	OK

Tabla. 5.6: Pruebas funcionales del incremento de explicabilidad mediante LRP.

5.7. Diagrama de clases completo

Para representar el módulo en su forma final con todas las implementaciones realizadas por incrementos se crea la Figura 5.31 que contiene todas las clases y funciones del módulo. Dado que es un diagrama muy grande se separa en columnas por tareas y se debe visualizar en horizontal dado que no hay forma de que sea legible en vertical. Los módulos de explicabilidad se representan en azul, los de presentación en verde y por último los modelos que irían en color naranja. Cabe destacar que el módulo de explicabilidad se implementa mayormente como funciones de TypeScript por lo que no son clases realmente, sin embargo, en el diagrama se representan como clases al tener sus métodos y responsabilidades de forma que a efectos prácticos se comportan como clases.

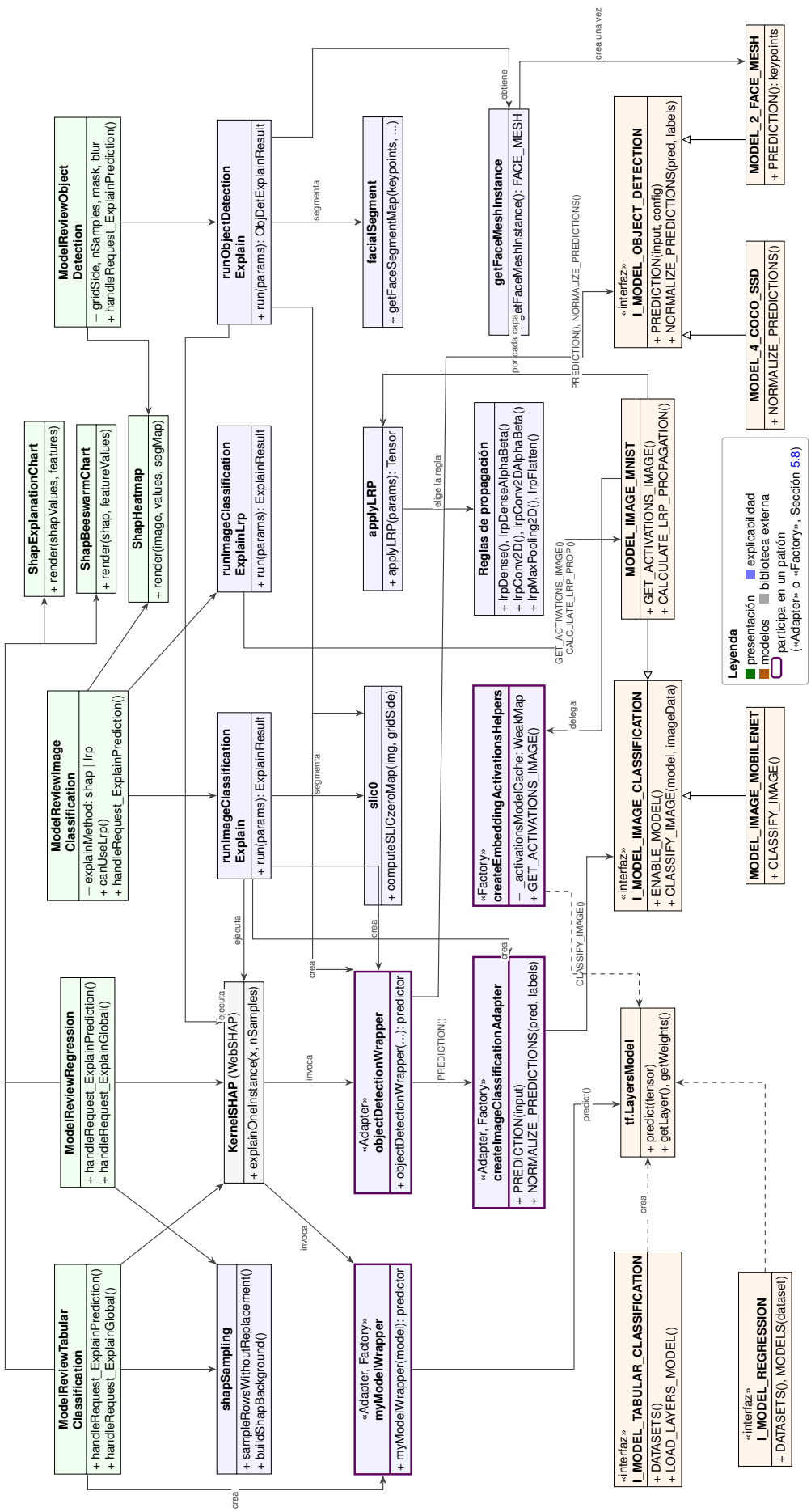


Figura 5.31: Diagrama de clases completo del módulo de explicabilidad, con las funcionalidades de todos los incrementos y los patrones de diseño usados en el desarrollo.

5.8. Patrones de diseño

Los patrones de diseño aparecen como soluciones a problemas a la hora de desarrollar el proyecto, los expongo en la siguiente tabla 5.7.

Patrón	Problema que resuelve	Dónde se aplica
Adapter	Los modelos no ofrecen la interfaz que espera KernelSHAP ni la que espera la función predictora de imágenes.	`myModelWrapper`, `objectDetectionWrapper` y `createImageClassificationAdapter`.
Factory	Crear los objetos de apoyo ya configurados sin que quien los usa conozca cómo se construyen.	`myModelWrapper`, `createImageClassificationAdapter` y `createEmbeddingActivationsHelpers`.

Tabla. 5.7: Patrones de diseño aplicados en la solución.

5.8.1. Adapter

El patrón *Adapter* permite crear una clase intermedia que sirva de puente entre dos clases que no son compatibles. En este proyecto donde KernelSHAP opera con una matriz de números y devuelve una matriz de predicciones, es fundamental utilizar este enfoque con el fin de superar la limitación dada por la forma que tienen los modelos de predecir tran variada.

- `myModelWrapper` El wrapper es fundamental para que KernelSHAP pueda trabajar sin conocer los detalles internos de los modelos con los que trabaja.
- `objectDetectionWrapper` Este wrapper actua de forma parecida a `myModelWrapper` manejando un vector de superpíxeles activos que se traduce en una imagen perturbada y detecciones en un vector de longitud fija.
- `createImageClassificationAdapter` A la hora de realizar predicciones de imagen cada modelo tiene su forma de hacerlo. Por ello, este adaptador normaliza la entrada y salida de los modelos de clasificación de forma que el wrapper de detección pueda trabajar con ellos.

Código 5.11

```
return {
  // Traduce PREDICTION a la operación de clasificación del modelo
  async PREDICTION(input) {
    const imageData = getImageDataFromInput(input);
    const result = await iModel.CLASSIFY_IMAGE(modelInstance,
     imageData);
    return result?.predictions ?? [];
  },

  // Convierte la lista de clases en un vector de longitud fija
  NORMALIZE_PREDICTIONS(predictions, labels) {
    const output = new Array(labels.length).fill(0);
    const indexByLabel = new Map(labels.map((l, i) => [String(l), i
    ]));
    for (const p of predictions) {
      const idx = indexByLabel.get(String(p?.className));
      if (idx !== undefined) output[idx] = Number(p?.probability);
    }
    return output;
  },
};
```

Código 5.11: Adaptador de los modelos de clasificación a la interfaz de los detectores

5.8.2. Factory

El patrón *Factory* es un patrón de diseño de creación que proporciona una interfaz para crear objetos en una superclase, permitiendo que sean las subclases las que decidan el tipo exacto de objeto a instanciar.

- `myModelWrapper` Esta factoría crea un adaptador para los modelos de clasificación tabular.
- `createImageClassificationAdapter` Esta factoría crea un adaptador para los modelos de clasificación de imagen.
- `createEmbeddingActivationsHelpers` Esta factoría crea un conjunto de funciones auxiliares para obtener las activaciones intermedias de los modelos de clasificación de imagen.

Capítulo 6

Conclusiones

En este último capítulo recogemos las conclusiones del trabajo, contrastando lo que hemos conseguido con los objetivos que nos marcamos al principio y proyectamos una visión a futuro del proyecto.

6.1. Conclusiones

El objetivo general de este Trabajo Fin de Grado era dotar a la plataforma *Nets4Learning* de un módulo de Inteligencia Artificial Explicable que funcionase por completo en el navegador sin depender de ningún servidor, y podemos decir que ese objetivo lo hemos alcanzado ya que la plataforma es ahora capaz de acompañar cada predicción con una explicación visual calculada íntegramente en el lado del cliente.

Si repasamos uno a uno los objetivos específicos que definimos en el Capítulo 1, podemos concluir:

- Se ha **estudiado el campo del aprendizaje automático** permitiendo entender por qué estos modelos se comportan como cajas negras. También se ha estudiado como explicar dichas cajas negras.
- Se ha **estudiado el concepto de inteligencia artificial explicable** permitiendo conocer los diferentes métodos de inteligencia artificial explicable. De esta forma se pudo elegir las técnicas de explicabilidad más adecuadas para el proyecto y las diferentes tareas a las que se aplican.
- Se han **analizado las tecnologías de desarrollo en aprendizaje automático**

pudiendo aprender las plataformas y librerías más utilizadas en el desarrollo de modelos de aprendizaje automático en navegador capaces de tratar datos en plataformas de cliente.

- Se ha **desarrollado la plataforma objetivo y se ha integrado con *Nets4Learning*** a lo largo de cinco incrementos, consiguiendo una plataforma educativa completa y funcional acerca de la inteligencia artificial, modelos de aprendizaje automático y explicabilidad.

El mayor reto ha sido la restricción de implementación en cliente de forma segura y privada ya que ha condicionado la elección de las técnicas y ha forzado a tomar decisiones comprometidas, como la aproximación que empleamos en las capas de *pooling* de LRP.

En el plano más personal, el proyecto nos ha permitido profundizar en el área de la explicabilidad que apenas se toca durante la carrera, cuya relevancia está todavía por explotar y desarrollarse en diferentes ámbitos como en el de la salud.

6.2. Líneas de trabajo futuro

El módulo si bien es completo, presenta vías de progreso a futuro que mejorarían enormemente el producto.

- **Incorporar nuevas técnicas de explicabilidad**, como Grad-CAM para los modelos convolucionales o LIME. (RNF-5).
- **Ampliar la cobertura de LRP** a una mayor cantidad de arquitecturas y capas complejas.
- **Aprovechar WebGPU** y los nuevos motores de inferencia en cliente con el fin de disipar la limitación de rendimiento de modelos actuales de TensorFlow.
- **Mover el cálculo a Web Workers** de manera que el grueso del trabajo de SHAP se ejecute por completo en un hilo aparte y la interfaz quede aún más liberada durante las explicaciones más largas.

Bibliografía

[1] Antonio Mudarra, David Valdivia, Pietro Ducange, Manuel Germán, Antonio J. Rivera, and María Dolores Pérez-Godoy. Nets4Learning: A web platform for designing and testing ANN/DNN models. *Electronics*, 13(22):4378, 2024. doi: 10.3390/electronics13224378.

[2] Grupo de investigación SIMIDAT. Nets4learning: Web platform for the design and execution of deep learning models. https://github.com/SIMIDAT/nets4learning, 2023. Repositorio de GitHub. Fecha de consulta: 15 de enero de 2026.

[3] Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Ann Yuan, Nick Kreeger, Ping Yu, Kangyi Zhang, Shanqing Cai, Eric Nielsen, David Soergel, et al. TensorFlow.js: Machine learning for the web and beyond. *Proceedings of Machine Learning and Systems (MLSys)*, 1:309–321, 2019.

[4] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2021.

[5] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

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

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 2015. doi: 10.1038/nature14539.

[8] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020. doi: 10.1016/j.inffus.2019.12.012.

[9] Christoph Molnar. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. 2022. URL https://christophm.github.io/interpretable-ml-book/. Fecha de consulta: 12 de febrero de 2026.

[10] Lloyd S. Shapley. A value for n-person games. In *Contributions to the Theory of Games II*, volume 28 of *Annals of Mathematics Studies*, pages 307–317. Princeton University Press, 1953.

[11] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 4765–4774, 2017.

[12] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016. doi: 10.1145/2939672.2939778.

[13] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7): e0130140, 2015.

[14] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. Layer-wise relevance propagation: an overview. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 193–209. Springer, 2019.

[15] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurélien Lucchi, Pascal Fua, and Sabine Süsstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34 (11):2274–2282, 2012. doi: 10.1109/TPAMI.2012.120.

[16] Zijie J. Wang and Duen Horng Chau. WebSHAP: Towards explaining any machine learning models anywhere. In *Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion)*, pages 262–266. ACM, 2023. doi: 10.1145/3543873.3587362.

[17] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Viena, Austria, 2024. URL https://www.R-project.org/. Fecha de consulta: 2 de febrero de 2026.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al.

PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

[19] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[20] François Chollet et al. Keras. https://keras.io, 2015. Fecha de consulta: 2 de febrero de 2026.

[21] Globental. Sueldo programador junior españa 2026: punto de partida real. https://globental.com/blog/sueldo-programador-junior-espana/, 2026. Fecha de consulta: 6 de febrero de 2026.

[22] Indeed. Salario de programador/a junior en españa. https://es.indeed.com/career/programador-junior/salaries, 2026. Fecha de consulta: 6 de febrero de 2026.

[23] Universidad de Málaga. Retribuciones PDI funcionario 2026. https://www.uma.es/habilitacion-seguridadsocial/navegador_de_ficheros/TABLAS_RETRIBUTIVAS/descargar/RETRIBUCIONES%20PDI.pdf, 2026. Fecha de consulta: 6 de febrero de 2026.

[24] Mingyu Kim. superpixel: implement superpixel with JS & TS. https://github.com/ysjk2003/superpixel, 2021. Repositorio de GitHub. Fecha de consulta: 15 de abril de 2026.

[25] Aidan Cooper. Explaining machine learning models: A non-technical guide to interpreting shap analyses. https://www.aidancooper.co.uk/a-non-technical-guide-to-interpreting-shap-analyses/, 2021. Fecha de consulta: 4 de febrero de 2026.

[26] Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, Klaus-Robert Müller, and Wojciech Samek. Layer-wise relevance propagation for neural networks with local renormalization layers. In *Artificial Neural Networks and Machine Learning–ICANN 2016*, pages 63–71. Springer, 2016.

Generado con la plantilla LaTeX *TFG Dpto. Informática - EPSJ - v1.3*

Licencia CC0 1.0 Universal (CC0 1.0) - Francisco Charte Ojeda