



ugr | Universidad
de **Granada**

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA

Análisis del lenguaje de diseño de vehículos mediante Deep Learning

Autor

Andrés Arias Navarro

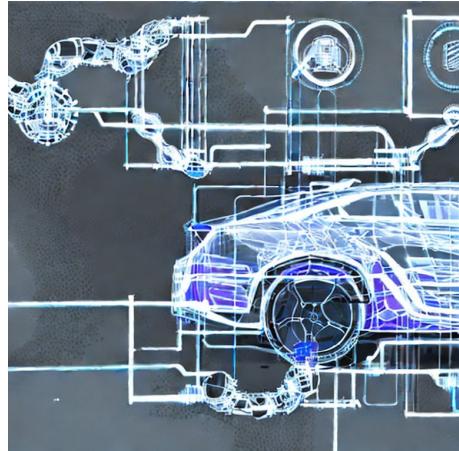
Director

Julián Luengo Martín



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, 12 de Julio de 2023



Análisis del lenguaje de diseño de vehículos mediante Deep Learning



<https://github.com/anariasnav/Car-design-analisis>

Autor

Andrés Arias Navarro

Director

Julián Luengo Martín

Análisis del lenguaje de diseño de vehículos mediante Deep Learning

Andrés Arias Navarro

Palabras clave: Deep Learning, CNN, Pytorch, GAN, Explainer, Shap, Stylegan, clasificación de imágenes, diseño de vehículos, visión por computador

Resumen

En un mercado tan competitivo como el actual, la diferenciación entre marcas es primordial por lo que el diseño de la línea de los vehículos desempeña un papel importante a la hora de conseguir características distintivas de las cuales aquellas exitosas son susceptibles de ser ‘copiadas’ o servir de inspiración a los competidores.

Por ello este TFG busca abordar la identificación de aspectos distintivos del lenguaje de diseño particular de las diversas marcas y modelos, haciendo uso de técnicas de Deep Learning tales como Lime, Shap y las imágenes generadas en las primeras épocas de entrenamiento por el generador de la GAN.

De forma análoga se proponen modelos de clasificación para las imágenes de vehículos de cara a un posible uso en identificación de marcas y modelos de vehículos para fines de investigación criminal.

Se plantea también la posible aplicación de esta capacidad de clasificación e identificación de características de los vehículos para la generación de prototipos de diseños futuros ayudando a inspirar a los diseñadores. Estos diseños pueden ser generados basándose en características propias de la marca o remodelando la línea de diseño por medio de la combinación de características de interés. Para alcanzar este objetivo se investigará el uso y rendimiento de diversas técnicas como GAN’s y Stylegan3.

Analysis of vehicle design language through Deep Learning

First name, Family name (student)

Keywords: Deep Learning, CNN, Pytorch, GAN, Explainer, Shap, Stylegan, image classification, vehicle design, computer vision

Abstract

In today's highly competitive market, brand differentiation is paramount and vehicle line design plays an important role in achieving distinctive features from which successful brands are likely to be 'copied' or serve as inspiration to competitors.

Therefore this TFG seeks to address the identification of distinctive aspects of the particular design language of the various brands and models, making use of Deep Learning techniques such as Lime, Shap and the images generated in the early training epochs by the GAN generator.

Similarly, classification models for vehicle images are proposed for possible use in vehicle make and model identification for criminal investigation purposes.

The possible application of this ability to classify and identify vehicle characteristics for the generation of prototypes of future designs is also proposed, helping to inspire designers. These designs can be generated based on the brand's own characteristics or remodeling the design line by combining characteristics of interest. To achieve this objective the use and performance of various techniques such as GAN's and Stylegan3 will be investigated.

Yo, **Andrés Arias Navarro**, alumno de la titulación en Ingeniería informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77553082A, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Andrés Arias Navarro

Granada a 12 de Julio de 2023 .

D. Julián Luengo Martín, Profesor titular del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *análisis del lenguaje de diseño de vehículos mediante Deep Learning*, ha sido realizado bajo su supervisión por **Andrés Arias Navarro**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 13 de Julio de 2023 .

El director:

Julián Luengo Martín

Agradecimientos

A mi familia por su incansable paciencia y por haberme inculcado los valores de esfuerzo, pasión y trabajo que hoy dan sus frutos.

Índice general

1. Introducción	1
1.1. Motivación	2
1.2. Objetivos	3
1.3. Presupuesto	4
1.4. Planificación	4
2. Estado del arte y conceptos	7
2.1. Concepto de aprendizaje, machine Learning y tipos de aprendizaje	8
2.2. Visión artificial	9
2.2.1. Clasificación de imágenes	10
2.2.2. IA generativa y generación de imágenes	11
2.3. Redes neuronales y clasificación de imágenes	11
2.3.1. Concepto de Neurona y red neuronal	11
2.3.2. Redes neuronales convolucionales (CNN)	13
2.3.3. Función de pérdida y algoritmos de optimización	19
2.3.4. Arquitecturas evaluadas	21
2.4. Generative Adversarial Networks(GAN)	24
2.4.1. Descripción formal	25
2.4.2. DCGAN	27
2.4.3. StyleGAN	28
2.5. Técnicas de explicabilidad de los modelos de clasificación	29
2.5.1. LIME (Local Interpretable Model-agnostic Explanations)	30
2.5.2. SHAP (SHapley Additive exPlanations)	34
3. Marco experimental	39
3.1. Entorno de experimentación	39
3.2. Dataset	41
3.3. Preprocesado	42
3.4. Métricas de evaluación	43

4. Clasificación e identificación de patrones de interés en imágenes de vehículos	47
4.1. Análisis del desempeño de modelos ante transformaciones y preprocesamiento	48
4.2. Estimación de hiperparámetros	49
4.3. Clasificación de imágenes	50
4.3.1. Especificidad MM	50
4.3.2. Especificidad MMA	53
4.3.3. Especificidad M	54
5. Generación Automática de imágenes para asistencia al diseño de vehículos	57
5.0.1. Implementaciones necesarias	58
5.1. GANs para la generación automática de imágenes de vehículos	59
5.2. Transferencia de estilos con Stylegan	66
6. Casos prácticos	69
6.1. Aplicación práctica de explainers para análisis del lenguaje de diseño de las marcas	69
6.1.1. Detección de características de interés	70
6.1.2. Detección de plagio entre marcas	72
6.2. Uso de redes de generación automática de imágenes para la asistencia al diseño automotriz	75
6.2.1. Aplicación de arquitecturas GAN al diseño de vehículos	75
6.2.2. Obtención de diseños por transferencia de estilos	76
7. Conclusiones y trabajos Futuros	81
7.1. Conclusiones	81
7.2. Trabajo futuro	82
Bibliografía	87
Apéndices	89
A. Manual de usuario	91
A.1. modCarpetas.py - reestructuración de la base de datos.	91
A.2. Denoising.py - reducción de ruido en imágenes generadas	92
A.3. Split.py - división de rejillas generadas a imágenes individuales	92
A.4. PlagiarismDetector.py - detección de similitudes entre marcas	93
B. Resultados obtenidos de los modelos generativos	95
B.1. Resultados GAN	96
B.2. Resultados StyleGAN	99

Índice de figuras

2.1.	Estructura de la neurona	12
2.2.	Estructura de las redes neuronales convolucionales	14
2.3.	Operador de convolución de una imagen con un kernel	14
2.4.	Funciones de activación no lineal.	17
2.5.	Bloque residual	22
2.6.	Modelos basados en ResNet	23
2.7.	Estructura de DenseNet-121	24
2.8.	Proceso de entrenamiento de los modelos de GAN	26
2.9.	Estructura y retropropagación de las GAN	27
2.10.	Importancia explicabilidad	30
2.11.	Idea base de LIME	32
2.12.	Aplicación de LIME sobre una imagen de interés	32
2.13.	Características positivas vs negativas en explicabilidad	33
2.14.	Ejemplo de uso de SHAP para explicabilidad de modelos de clasificación de imágenes	35
3.1.	Ejemplos de instancias de la base de datos utilizada	41
3.2.	Matriz de confusión para clasificación multiclas	44
4.1.	Detección de errores en modelos por explicabilidad.	49
4.2.	Estimación learning rate	50
4.3.	Análisis explicabilidad modelo MM	53
4.4.	Análisis explicabilidad modelo MMA	54
4.5.	Análisis explicabilidad modelo M	55
5.1.	Denoising	59
5.2.	Resultados iniciales de la GAN donde se puede ver que las primeras características que aprende a imitar son aquellas de interés para el clasificador, tal y como se ve en la explicabilidad del modelo de clasificación con LIME.	60
5.3.	Comparativa de las imágenes generadas en presencia o no de data augmentation	61
5.4.	Imágenes generadas para el tamaño de Batch estimado	62

5.5. Imágenes generadas por la GAN para el conjunto de datos completo	63
5.6. Gráfica pérdida GAN	63
5.7. Análisis de imágenes generadas por la GAN	64
5.8. Clasificación imágenes generadas	64
5.9. Imágenes GAN con base de datos reducida a BMW y Mercedes	65
5.10. Imágenes erróneas generadas por Stylegan	66
5.11. Evolución de diseño en Stylegan con distintas características .	67
6.1. Análisis de características de interés de Audi	71
6.2. Análisis de plagio para Audi	74
6.3. Modelos generados para BMW y Mercedes	76
6.4. Diseños generados con interés en el color	77
6.5. Imágenes distorsionadas	78
6.6. Diseños realistas generados por Stylegan	79
6.7. Análisis del modelo Mercedes de interés usando explainers .	80
B.1. Imágenes generadas por la GAN para el conjunto completo de datos de entrenamiento	96
B.2. Imágenes generadas por la GAN para el conjunto reducido de imágenes de Bmw	97
B.3. Posibles diseños generados a partir de una misma imagen .	99
B.4. Diseños con color de interés	100
B.6. Modelos de interés	102

Índice de cuadros

1.1. Presupuesto proyecto	4
1.2. Planificación inicial del proyecto. En rojo se indica el tiempo de estudio y familiarización con los conceptos de las diversas técnicas aplicadas. En naranja el tiempo de implementación y experimentación. Y finalmente en azul el tiempo dedicado a la memoria del proyecto.	5
1.3. Planificación temporal seguida finalmente para el desarrollo del proyecto donde podemos observar como el desarrollo y experimentación no ha sido lineal sino que ha sido necesario mejorar ciertas implementaciones a lo largo del tiempo.	5
3.1. Resumen de tecnologías/bibliotecas/hardware usadas en el proyecto	41
4.1. Desempeño de los modelos ante la presencia o ausencia de ciertas transformaciones en el conjunto de datos	49
4.2. Comparativa entre las distintas arquitecturas para especificidad MM.	51
4.3. Comparativa de modificaciones en el modelo	52
4.4. Rendimiento de ADAM VS SGD	52
4.5. Evaluación de modelos para especificidad MMA	53
4.6. Evaluación de modelos para especificidad M	55
6.1. Probabilidades Audi Vs Hyundai	73
6.2. Tabla de resultados del script PlagiarismDetector.py sobre Audi	73
6.3. Probabilidad de un diseño de interés para diversas marcas . .	79

Capítulo 1

Introducción

El diseño de la línea de los vehículos desempeña un papel fundamental en la industria de la automoción. Las marcas de fabricantes se esfuerzan por mantener una identidad única en sus modelos, ya que esto les permite diferenciarse en un mercado altamente competitivo. Esta búsqueda de la singularidad en el diseño, con unas características rompedoras que cautiven al comprador, puede llevar a la imitación o inspiración por parte de los competidores. A lo largo de los años hemos visto en repetidas ocasiones como modelos rompedores con el diseño de mercado han acabado marcando una nueva tendencia y siendo imitados en mayor o menor medida por los competidores (ejemplo renault Kangoo).

En este proyecto se plantea el uso de técnicas de Deep Learning [15] para abordar este desafío. El objetivo principal es desarrollar un sistema capaz de identificar aspectos susceptibles de plagio en el diseño de vehículos, así como aquellos que definen un lenguaje de diseño particular. Este enfoque analítico de características distintivas nos permitirá analizar y comprender las sutilezas que caracterizan el diseño de cada marca en lugar de limitarnos a detectar copias directas o influencias evidentes abriendo nuevas posibilidades en el ámbito del diseño automotriz permitiéndonos generar de forma automática nuevos diseños basados en los patrones característicos de cada marca.

En el pasado, el diseño se consideraba como una actividad puramente humana, impulsada por la creatividad y la intuición de los diseñadores. Sin embargo, con el auge de los modelos generativos basados en inteligencia artificial (IA generativa [5]), se han abierto nuevas posibilidades que desafían esa concepción tradicional. Los modelos generativos, como las Redes Generativas Antagónicas (GAN)[4] y los modelos basados en difusión [3](como Stable Diffusion), han demostrado la capacidad de generar contenido visual original y creativo de gran calidad. La potencia de los modelos generativos combinada con la intuición humana abre un espacio para la generación de

diseños innovadores y sorprendentes.

Para lograr el objetivo de la generación de diseños, se plantea la posibilidad de aplicar la capacidad de clasificación e identificación de características de vehículos en la generación de prototipos de diseños futuros. Estos prototipos pueden ser generados partiendo de características propias de la marca permitiendo generar diseños que sigan la tendencia de la misma (GAN's), o remodelando la línea de diseño a través de la combinación de características de interés haciendo uso de una red para la transferencia de estilo como Stylegan3[13].

La capacidad de identificar patrones y tendencias en el diseño automotriz tiene implicaciones significativas. Por un lado, permitirá a las marcas proteger su identidad y propiedad intelectual, identificando rápidamente casos de plagio. Por otro lado el análisis de características presentes en los diseños de vehículos con gran aceptación en el mercado puede ser de gran interés para las marcas al permitirles ver que patrones de diseño serán más rentables a la hora de lanzar al mercado un nuevo modelo.

En este TFG, se explorará cómo el uso de técnicas de Deep Learning, como la clasificación y la generación automática de diseños, puede contribuir a la evolución del diseño automotriz. Al hacerlo, se pretende fomentar la creatividad y la innovación en un campo donde la originalidad es fundamental para mantenerse a la vanguardia.

Además se buscará abordar la identificación de aspectos distintivos del lenguaje de diseño propio de diferentes marcas utilizando técnicas de expli- cabilidad de modelos como LIME[20] y SHAP[16] y su comparación con las imágenes iniciales obtenidas con el generador de la GAN.

1.1. Motivación

El diferenciación y la novedad son 2 aspectos cruciales en el marketing actual. Provocar el interés en el comprador y la necesidad de adquirir un producto en una sociedad con un ritmo de vida frenético no es tarea fácil, por lo que es necesario mantener en constante cambio los modelos y actualizaciones de diseño en los productos.

Esta necesidad, junto con el incipiente auge de técnicas para la generación automática de datos, supuso una gran oportunidad para la investigación de las posibles aplicaciones de dichas técnicas para la generación automática de diseños de vehículos. Por otra parte, el lenguaje de diseño de una marca es algo abstracto que, a pesar de ser identificado por todos, no tiene un fácil análisis y comprensión por parte de los humanos. ¿Como reconocemos una marca?, ¿Cuales son las características que la diferencian?... son preguntas que a pesar de ser de vital importancia puede ser demasiado tedioso respon-

der así que ¿por qué no automatizar este razonamiento?. Con este objetivo, se extendió el enfoque de este proyecto a un conjunto más amplio de técnicas de Inteligencia Artificial como el uso de explicabilidad [8] para los modelos de clasificación como identificador de características distintivas.

1.2. Objetivos

El presente proyecto plantea la aplicación de diferentes algoritmos y técnicas de Inteligencia Artificial[23] para tareas relacionadas con la industria automotriz. Con el desarrollo del mismo se pretenden alcanzar los siguientes objetivos:

1. En primer lugar, se plantea el entrenamiento de diversas redes neuronales para la clasificación de imágenes de vehículos, con el propósito de utilizarlas en la identificación de vehículos en fotografías o videos relevantes para investigaciones criminales.
2. Otro objetivo del proyecto es explorar y evaluar las posibles aplicaciones de diferentes técnicas de aprendizaje automático para tareas relacionadas con el diseño automotriz y sentar las bases para futuras investigaciones y aplicaciones en la industria. Esto incluye el análisis de características distintivas de vehículos mediante técnicas de explicabilidad de modelos, con el fin de proporcionar a los diseñadores una herramienta para extraer información visual de imágenes de diseños previos.
3. Además, se planteará como objetivo el análisis del rendimiento de diversas técnicas de generación automática de imágenes para la creación de diseños de vehículos que puedan inspirar a los diseñadores. Se explorará el estado del arte en estas técnicas, buscando obtener resultados realistas y con cierto grado de control.
4. Finalmente, dada la preocupación creciente sobre el impacto de la Inteligencia Artificial en el ámbito laboral, surge la pregunta fundamental: ¿Puede la IA, en su nivel actual, llegar a sustituir a un diseñador humano? Este proyecto también se propone abordar esta interrogante, analizando los avances y limitaciones actuales en términos de generación automática de diseños.

1.3. Presupuesto

En esta sección se analizaremos los costes. En el Cuadro 1.1 se puede ver el reparto de costes realizado para poder disponer de los recursos necesarios para la realización del proyecto actual.

En concreto la información referente al precio de la mano de obra ha sido extraido de la plataforma glassdoor¹. Por otra parte se considera el alquiler de GPU por su menor costo que la adquisición de una con suficiente potencia. La GPU utilizada es la NVIDIA V100² y se arrendará a través de la plataforma Google Cloud con un costo de 0.7€ por GPU cada hora.³.

Concepto	Cantidad	Precio	Subtotal
Mano de obra	3 meses x 20 días x 8 horas=480 horas	20,44€/hora	9811.2€
GPU (NVIDIA V100 8 GPU Google Cloud)	350 horas de entrenamiento efectivo	5.61€/hora	1963.5€
Portátil del alumno	3 meses de trabajo en un portátil con un costo de 1000€ y una vida útil de 4 años	20€/mes	83.3€
Total			11.646€

Cuadro 1.1: Presupuesto proyecto

1.4. Planificación

Inicialmente se propuso una gestión temporal como se ve en el Cuadro 1.2, basada en un primer periodo de familiarización con el estado del arte en las diversas técnicas a usar y planteamiento del proyecto, seguido de un la implementación y experimentación con los diversos scripts. Y finalmente un margen de 2 semanas para poder escribir la presente memoria.

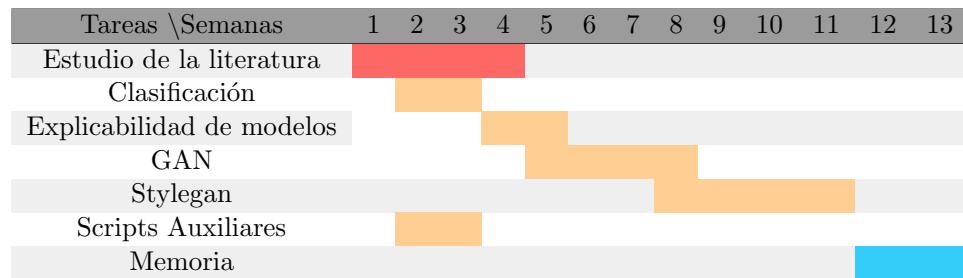
Sin embargo, tal y como se puede ver en el Cuadro 1.3 la planificación temporal que finalmente se siguió durante el desarrollo de este proyecto es algo distinta.

Inicialmente se comenzó la experimentación en mi ordenador personal con 6GB de memoria de GPU, sin embargo tal y como se verá más adelante los requisitos Hardware de los algoritmos y técnicas a evaluar nos obligaron a cambiar la ejecución de los mismos a los servidores de GPU de la UGR lo cual supuso una pausa en la experimentación. Por otra parte, nos dimos cuenta de que el planteamiento secuencial de la implementación de scripts auxiliares no era realista ya que fueron necesarios scripts que no habían sido planteados en primera instancia.

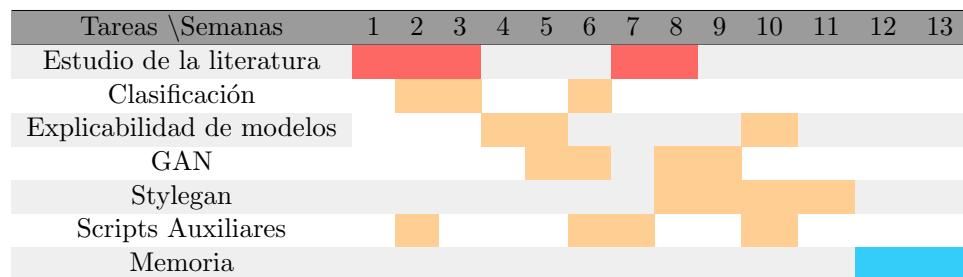
¹https://www.glassdoor.es/Sueldos/data-scientist-sueldo-SRCH_K00,14.htm

²NVIDIA V100 - <https://www.nvidia.com/en-us/data-center/v100/>

³Precios GPU - <https://cloud.google.com/compute/gpus-pricing?hl=es>



Cuadro 1.2: Planificación inicial del proyecto. En rojo se indica el tiempo de estudio y familiarización con los conceptos de las diversas técnicas aplicadas. En naranja el tiempo de implementación y experimentación. Y finalmente en azul el tiempo dedicado a la memoria del proyecto.



Cuadro 1.3: Planificación temporal seguida finalmente para el desarrollo del proyecto donde podemos observar como el desarrollo y experimentación no ha sido lineal sino que ha sido necesario mejorar ciertas implementaciones a lo largo del tiempo.

Capítulo 2

Estado del arte y conceptos

En este capítulo se exponen los conceptos fundamentales utilizados en este proyecto, junto con un resumen del estado del arte en técnicas de generación automática de imágenes y explicabilidad de modelos. También se investigarán sus aplicaciones en la identificación de patrones distintivos y la generación automatizada de diseños de vehículos.

En las Secciones 2.1, 2.2 y 2.3, se abordarán los conceptos fundamentales del aprendizaje automático, visión artificial y las redes neuronales, así como el concepto de neurona en el contexto de estas redes. También se proporcionará una descripción de los conceptos utilizados en la clasificación y una introducción a las arquitecturas que se evaluarán posteriormente.

A continuación, en la Sección 2.4 se explicarán en detalle las redes generativas adversariales (GAN) y los diversos algoritmos que se probarán para la generación automática de imágenes. Se explorarán los principios y el funcionamiento de las GAN, así como los enfoques utilizados para entrenar y mejorar la calidad de las imágenes generadas.

Finalmente, en la Sección 2.5 se realizará un análisis exhaustivo de las técnicas de explicabilidad, destacando tanto su concepto e importancia en el contexto actual como la descripción formal de las dos técnicas que se utilizarán en este proyecto. Se abordará la necesidad de comprender y explicar las decisiones tomadas por los modelos de aprendizaje automático, y se presentarán las técnicas específicas que se utilizarán para lograr una mayor transparencia y comprensión de los resultados obtenidos.

2.1. Concepto de aprendizaje, machine Learning y tipos de aprendizaje

En el contexto de la Inteligencia Artificial, el concepto aprender se refiere a la capacidad de un sistema o algoritmo de adquirir conocimiento o habilidades a través de la experiencia o de los datos permitiéndole identificar patrones, relaciones y regularidades en los mismos y utilizar esa información para realizar tareas o tomar decisiones. El objetivo principal es que el sistema pueda mejorar su rendimiento o tomar decisiones más precisas a medida que se le presentan más datos o situaciones.

Dentro de la Inteligencia Artificial encontramos el Machine learning [29] que implica la construcción de modelos o algoritmos que pueden aprender de los datos disponibles. En lugar de escribir reglas o instrucciones específicas, se utilizan métodos matemáticos y estadísticos para entrenar modelos con los que identificar patrones y relaciones en los datos y utilizar esa información para hacer predicciones o tomar decisiones.

Dentro del aprendizaje automático encontramos 3 tipos distintos de aprendizaje en función de como se relaciona el modelo con los datos:

- Aprendizaje supervisado: El modelo tiene acceso tanto a las instancias de entrada como a las salidas esperadas consiguiendo así que aprenda la relación entre las entradas y las salidas de cara a predecir nuevas entradas que no se encuentren etiquetadas. Para esto se entrena el modelo con un conjunto de datos etiquetados.
- Aprendizaje no supervisado: Es aquel en el que a partir de un conjunto de datos sin etiquetar para que se detecten patrones, agrupamientos, relaciones entre las variables etc
- Aprendizaje por refuerzo: Este modelo es un punto intermedio entre los 2 anteriores. El modelo toma decisiones y recibe 'castigos' o 'recompensas' en función de su rendimiento en un entorno. De tal forma que el modelo aprende a maximizar las recompensas.

En este TFG se hará uso de 2 de estos tipos para las distintas tareas a tratar. Para el problema de clasificación de imágenes se hará uso de aprendizaje supervisado mientras que la generación automática de imágenes utiliza un enfoque mixto en el que el discriminador realiza un tipo de aprendizaje supervisado (clasifica las instancias como reales o sintéticas) y por su parte el generador lleva a cabo una estrategia de aprendizaje por refuerzo en la que genera imágenes sintéticas y obtiene una retroalimentación en forma de castigo-recompensa por parte del discriminador.

Más concretamente, para el desarrollo de este proyecto se hace uso de una subrama del aprendizaje automático que se conoce como Deep Learning. El Deep Learning[15] se basa en el concepto de modelos para reconocer patrones y hacer predicciones usando los datos de entrenamiento al igual que el aprendizaje automático genérico. Sin embargo, este enfoque nos permite aprender representaciones más complejas y una extracción automática de las características basándose en el concepto de redes neuronales profundas.

2.2. Visión artificial

La visión artificial [25] es la disciplina de la Inteligencia Artificial que a través del desarrollo de sistemas y algoritmos trata de dotar a las máquinas de la capacidad de obtener, procesar y analizar imágenes o conjuntos de las mismas obteniendo información visual de forma similar a los seres humanos.

Las tareas típicas en visión artificial incluyen:

- Reconocimiento y clasificación de objetos: Se trata de identificar y asignar etiquetas o categorías a los objetos presentes en una imagen o video. Puede involucrar la clasificación de imágenes en categorías predefinidas o la identificación de objetos específicos en una imagen. Por ejemplo, el reconocimiento de caras en fotografías, clasificación de objetos en imágenes médicas, identificación de señales de tráfico, entre otros.
- Generación automática: consiste en generar contenido visual sintético a partir de datos o características existentes. Esta tarea implica crear imágenes o videos que sean perceptualmente similares a las imágenes reales, pero que no existen en el mundo real y son generadas por un modelo de visión artificial.
- Detección de objetos: Consiste en identificar y localizar la presencia de objetos específicos en una imagen o en un video. Puede involucrar la detección de rostros, detección de peatones, detección de vehículos, etc. Esta tarea es esencial en aplicaciones como sistemas de vigilancia, conducción autónoma, reconocimiento facial, entre otros.
- Seguimiento de objetos: Implica rastrear y seguir la trayectoria de un objeto específico a lo largo del tiempo en secuencias de imágenes o videos. Esto es útil en aplicaciones como el seguimiento de objetos en movimiento en videos de vigilancia, seguimiento de vehículos en un entorno de conducción autónoma, seguimiento de jugadores en deportes, entre otros.

- Segmentation de imágenes: Implica la asignación de etiquetas a píxeles individuales en una imagen para diferenciar diferentes regiones o objetos. Por ejemplo, la segmentación semántica se utiliza para distinguir diferentes categorías de objetos en una imagen, mientras que la segmentación de instancias se utiliza para identificar y separar cada objeto individualmente.

Dentro de esta disciplina encontramos las 2 tareas principales de este proyecto la clasificación y generación automática de imágenes.

2.2.1. Clasificación de imágenes

En el campo de la visión artificial la clasificación de imágenes consiste en asignar una etiqueta a una imagen de entrada haciendo uso de métodos que permiten identificar las características más importantes de dicha instancia de entrada.

La clasificación puede ser llevada a cabo con diferentes métodos como máquinas de soporte vectorial, árboles de decisión etc En concreto para este proyecto se estudiará y propondrá el uso de redes neuronales convolucionales para el clasificación de imágenes debido al buen rendimiento que han demostrado en tareas de análisis de imágenes gracias a su complejidad.

El proceso de clasificación de imágenes con CNN es una tarea de aprendizaje supervisado ya que implica entrenar con imágenes etiquetadas previamente y estas etiquetas serán las que 'supervisen' a la red indicando la calidad de sus predicciones y permitiéndole aprender los patrones y características específicos de las imágenes de cada etiqueta de tal forma que pueda diferenciar a que clase pertenece una instancia desconocida.

Para poder entrenar una CNN para la clasificación de imágenes es necesario preparar con anterioridad la base de datos de entrenamiento de tal forma que se dividan en conjuntos de entrenamiento, validación y test en los que cada foto se encuentre etiquetada correctamente. En esta fase de preprocesado de los datos es donde estableceremos unas dimensiones comunes para todas las imágenes y se definirán normalizaciones y data augmentation con transformaciones afines, filtros etc para mejorar el rendimiento del modelo.

Tras esto procederemos a diseñar la arquitectura de la red, como combinación de las distintas capas anteriormente descritas, en función de la complejidad del problema. En nuestro caso estudiaremos el desempeño de diversas arquitecturas preentrenadas para otros conjuntos de datos, ajustando la cabeza de clasificación del modelo a nuestro problema concreto.

Una vez definida la arquitectura de la CNN procederemos al entrenamiento de la misma pasándole de forma iterativa el conjunto de datos de

entrenamiento y ajustando los peso de la red con backpropagation haciendo uso del valor de la función de perdida seleccionada mediante el algoritmo de optimización.

2.2.2. IA generativa y generación de imágenes

La inteligencia artificial generativa[5] es una rama de la inteligencia artificial que se enfoca en la capacidad de generar contenido original y creativo, como imágenes, texto o música, a partir de modelos de aprendizaje automático.

En el contexto de la generación de imágenes, la IA generativa utiliza técnicas y algoritmos para crear imágenes sintéticas que son perceptualmente similares a las imágenes reales. Esto se logra mediante el entrenamiento de modelos generativos, como las Redes Generativas Adversarias (GAN, por sus siglas en inglés) y las Redes Neuronales Generativas de Flujo Variacional (VAE, por sus siglas en inglés), utilizando grandes conjuntos de datos de imágenes reales.

La generación de imágenes con IA generativa se basa en un proceso iterativo de aprendizaje y refinamiento. El modelo generativo aprende a través de la observación de imágenes reales y trata de capturar las características y patrones clave presentes en esos datos. Luego, el modelo utiliza esta información para generar imágenes que sean perceptualmente similares a las imágenes reales, pero que no existen en el conjunto de datos de entrenamiento.

2.3. Redes neuronales y clasificación de imágenes

En el campo del aprendizaje automático y la visión por computadora, las redes neuronales han demostrado ser una herramienta poderosa para abordar la tarea de clasificación de imágenes. Las redes neuronales están inspiradas en el funcionamiento del cerebro humano y busca alcanzar a través de la extracción automática de características de los datos de entrenamiento y la ponderación de las mismas, la capacidad de aprender representaciones de datos a diferentes niveles de abstracción.

2.3.1. Concepto de Neurona y red neuronal

Las redes neuronales tratan de simular el procesamiento de la información por parte del cerebro humano basándose en el concepto de neurona que se puede apreciar en la Figura 2.1, fue descrito por McCulloch y Pitts [26]

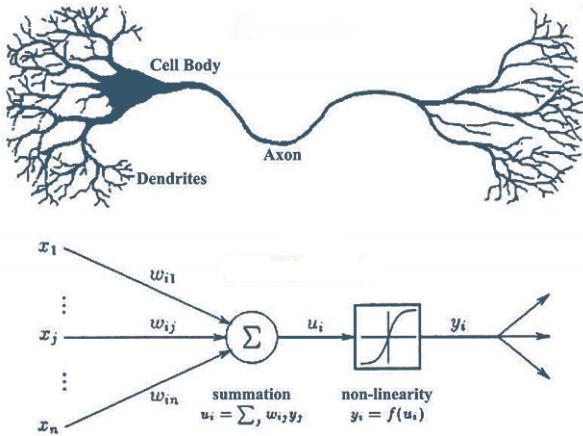


Figura 2.1: Estructura de la neurona

a modo de abstracción de las propiedades de las neuronas y las conexiones entre ellas con un conjunto primitivo de elementos de computo.

El concepto de neurona que propusieron se centraba en las capacidades computacionales de las mismas, describiéndola como un dispositivo binario en el que la salida depende de si las entradas recibidas por las sinapsis excitadoras son suficiente fuertes para activar la neurona y propagar el impulso a la siguiente en función de un umbral.

Se conoce como red neuronal al conjunto de neuronas artificiales conectadas entre sí formando capas permitiéndonos abordar problemas que no son lineales. Cuando se incrementa la profundidad de la red, es decir mayor número de capas, y de neuronas por capa se aumentará la complejidad del modelo permitiéndonos generalizar y ajustar conjuntos de datos con una mayor complejidad.

El proceso de entrenamiento de las redes neuronales consiste en asignar y ajustar el peso de todas las entradas x_i de las neuronas del modelo de tal forma que las estimaciones obtenidas en la capa de salida se ajusten al máximo a los datos que conocemos. Este proceso se lleva a cabo haciendo uso de backpropagation[22] que es un método de cálculo basado en el ajuste de los pesos de las neuronas para reducir el error obtenido en la salida del modelo (la diferencia entre la salida producida y la deseada) por medio de propagar este error desde la capa de salida hasta la capa inicial pasando por todas las neuronas y recalculando los pesos de sus entradas.

En este punto también cabe resaltar que no siempre un mejor ajuste de los datos de entrenamiento proporcionará un modelo más válido para el problema a ajustar. Esto se debe al sobreajuste (overfitting) del modelo sobre el conjunto de entrenamiento obteniendo predicciones precisas en train pero

no generalizando bien de cara a instancias de datos nuevas ya que el modelo aprende características literales de las instancias de entrenamiento (Por ejemplo para nuestro caso de clasificación de imágenes de vehículos el color de un coche o del fondo de la imagen) que no tienen relación con la clase a predecir. Por ello es importante lograr un equilibrio entre la complejidad del modelo, cantidad de instancias de entrenamiento etc de cara a la obtención de un modelo correcto.

2.3.2. Redes neuronales convolucionales (CNN)

En concreto nosotrosaremos uso de las redes neuronales convolucionales [7], que son aquellas diseñadas para procesar los datos en forma matricial lo que las hace ideales para las tareas de visión por computador debido a la forma de representación interna de las imágenes. Estas son representadas en el interior del ordenador como una matriz de 3 dimensiones (anchura X altura x canales) que contiene los píxeles de la imagen con 3 canales para el caso de imágenes a color y 1 canal para escala de grises, cada uno de los píxeles será codificado con un valor de intensidad de color entre 0 y 255.

Lo que caracteriza a estas redes es su capacidad de aprender en las primeras capas una serie de características relevantes básicas de la imagen (bordes, texturas, formas simples etc) extraídas de forma automática mediante técnicas de procesamiento de imágenes para permitir la interpretación de información visual. En la Figura 2.2 se puede observar la estructura genérica de las CNNs que está formada por varios tipos de capas entre las que destacan:

- Capas convolucionales
- Capas no lineales
- Capas de pooling
- Capa fully connected
- Softmax

Capas convolucionales

Primeramente y como pieza central en el ámbito del procesamiento de imágenes en redes neuronales convolucionales encontramos la convolución, que es una operación matemática lineal que nos permite aplicar un filtro, definido como una pequeña matriz de pesos o filtros, sobre la imagen original

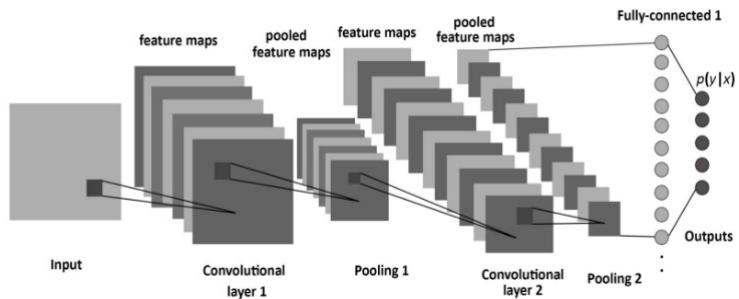


Figura 2.2: Estructura de las redes neuronales convolucionales

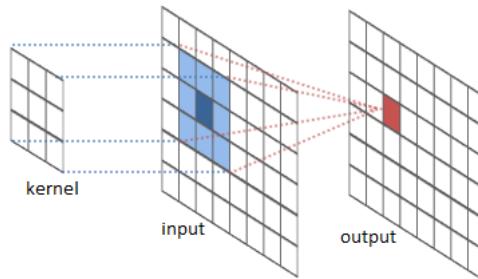


Figura 2.3: Operador de convolución de una imagen con un kernel

de manera que se combinan para producir una nueva imagen tal y como se aprecia en la Figura 2.3.

La convolución consiste en ir desplazando iterativamente el kernel sobre la imagen, calculando el valor de cada píxel de una nueva matriz llamada 'feature map' como la suma de los valores obtenidos al multiplicar los valores correspondientes del núcleo con los de la imagen en cada posición.

En este punto, es importante introducir dos conceptos fundamentales que se utilizan en las capas de convolución y pooling, y que definen la forma en que se aplican sobre la imagen. Estos conceptos son *Stride* y *Padding*.

Stride se refiere al número de píxeles que se desplaza el kernel entre las iteraciones de la convolución. Es decir, determina el salto o la desplazamiento aplicado al filtro a medida que se mueve por la imagen de entrada. Un *stride* mayor significa que el filtro o la ventana se mueven en saltos más grandes, lo que resulta en una reducción de la resolución espacial de la salida.

Por otro lado, tenemos el concepto de *padding*, que consiste en agregar píxeles nulos simétricamente alrededor de la matriz de entrada antes de aplicar la operación de convolución. El *padding* se usa para que la dimensión de la salida sea la misma que la de la entrada y evitar una reducción excesiva de la dimensión espacial, permitiendo que los bordes de la imagen de entrada se tengan en cuenta adecuadamente durante la operación y ayudando a

preservar la información espacial en la salida.

Dadas 2 funciones $f, h : \mathbb{R}^* \rightarrow \mathbb{R}$ se define la convolución entre ellas denotada como $f * h$ como una nueva función g definida por:

$$g(x) = (f * h)(x) = \int_{-\infty}^{+\infty} f(x-u)h(u)du \quad (\textbf{1D}) \quad (2.1)$$

$$g(x, y) = (f * h)(x, y) = \int \int_{-\infty}^{+\infty} f(x-u, y-v)h(u, v)dudv \quad (\textbf{2D}) \quad (2.2)$$

La convolución cumple las siguientes propiedades:

- **Commutativa:** $f * h = h * f$
- **Asociativa:** $(f * g) * h = g * (g * h)$
- **Distributiva sobre la suma:** $f * (g + h) = (f * g) + (f * h)$
- **Factor escalar externo:** $kf * h = k(f * h) \forall k \in \mathbb{R}$
- **Identidad:** Existe un impulso unidad $e = [..., 0, 0, 1, 0, 0, ...]$ tal que $f * e = f$
- **Teorema de la convolución:** La transformada de fourier $F(f * h) = F(f)xF(h)$

En el caso de uso en redes neuronales convolucionales para el análisis y extracción de características, haremos uso de la convolución teniendo en cuenta H y F como dos matrices donde H se corresponde con la imagen original y F con el kernel a aplicar sobre la misma, definimos la convolución discreta en 2D $G = H * F$ como:

$$G[i, j] = \sum_{u,v=-\infty}^{\infty} F[i - u, j - v]H[u, v] \quad (2.3)$$

se da la vuelta al filtro tanto vertical como horizontalmente y se desliza por la imagen computando el valor de cada posición.

En caso de ser una máscara separable, las convoluciones 2D pueden ser reducidas a 2 convoluciones 1D (1 por filas y 1 por columnas) suponiendo una reducción en la complejidad del filtrado.

Considerando una imagen de $n \times n$ dimensiones y un kernel de $m \times m$, la complejidad en caso de usar convolución 2D sería de $-O(nm)$, con la separabilidad del kernel podemos reducirla a $O(nm)$

la convolución discreta en 1D se define como:

$$G[i] = \sum_{u=-\infty}^{\infty} F[i-u]H[u] \quad (2.4)$$

La convolución es especialmente útil en el procesamiento de imágenes porque permite extraer características locales y patrones relevantes de una imagen. Los filtros utilizados en la convolución pueden detectar bordes, texturas, formas u otras características visuales específicas.

En el contexto de las redes neuronales convolucionales, la convolución se utiliza en las capas convolucionales para aplicar múltiples filtros (tales como filtros de frecuencia, alisados por convolución con máscaras gaussianas, convolución con la laplaciana de la gaussiana para resaltado de bordes etc) a una imagen de entrada y generar múltiples mapas de características que capturan diferentes aspectos y niveles de abstracción de la imagen.

Capas no lineales

Tras las operaciones de convolución encontramos la capa de activación no lineal la cual aplica una función de activación para el mapa de características, introduciendo la capacidad de aprendizaje de características más complejas y representaciones no lineales de los datos de entrada.

A continuación se explican las funciones más comunes de activación no lineal junto con sus gráficas anexas en la Figura 2.4

- **ReLU (Rectified Linear Unit):** La función de activación ReLU es una de las capas no lineales más utilizadas en las CNN. La función ReLU mapea cualquier valor negativo a cero y mantiene los valores positivos sin cambios. La función matemática se define como $f(x) = \max(0, x)$, donde x es la entrada. La capa ReLU introduce no linealidad en la red al activar o desactivar ciertos valores de las características aprendidas.
- **Leaky ReLU:** Similar a ReLU, la función Leaky ReLU también mapea los valores negativos a cero, pero con una pendiente pequeña en lugar de ser completamente lineal. Esto se hace para evitar la 'muerte' de neuronas, donde la salida de una neurona se vuelve siempre cero. La función matemática se define como $f(x) = \max(ax, x)$, donde x es la entrada y a es un valor pequeño positivo que controla la pendiente para valores negativos.
- **Sigmoid:** La función de activación sigmoide aplica una transformación no lineal que comprime los valores de entrada en un rango entre 0 y 1.

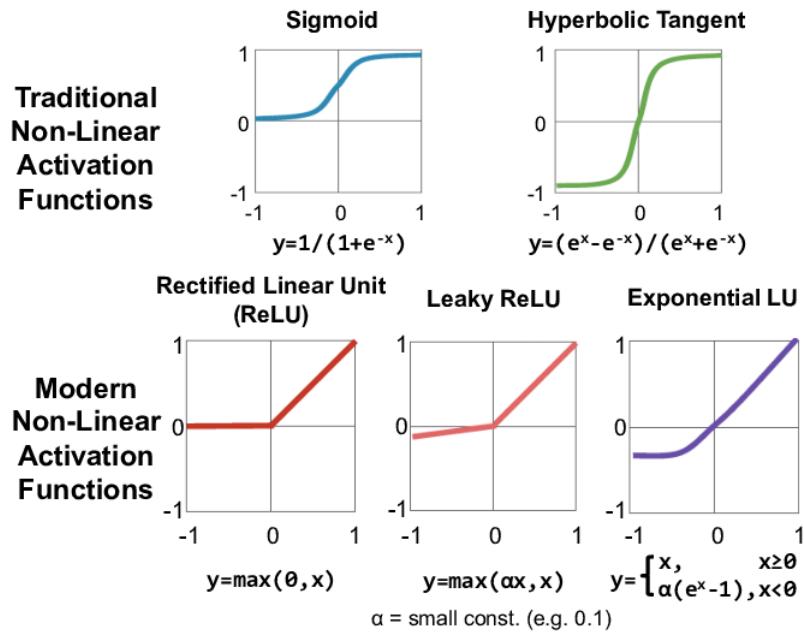


Figura 2.4: Funciones de activación no lineal.

La función matemática es $f(x) = 1/(1 + \exp(-x))$. Aunque la función sigmoide tiene ciertas limitaciones, como el problema de desvanecimiento del gradiente, aún se utiliza en ciertos casos, como en la capa de salida para clasificación binaria.

- **Tanh (Tangente hiperbólica):** La función tangente hiperbólica es similar a la función sigmoide, pero su rango se extiende de -1 a 1. La función matemática es $f(x) = (\exp(x) - \exp(-x))/(\exp(x) + \exp(-x))$. La función tanh también se utiliza como una función de activación no lineal en algunas redes convolucionales.

Capas de pooling

También conocidas como capas de agrupamiento, son introducidas entre capas de convolución y son usadas para reducir la dimensionalidad de los mapas de características extraídas por las capas convolucionales, por medio de un submuestreado de la entrada, combinando localmente las características agrupando las que son semánticamente similares pero preservando las características relevantes. Además las capas de pooling ayudan a reducir el overfitting.

Las capas de pooling consisten en la división de la entrada en regiones disjuntas y calcular para cada una un valor de la salida en función de

los valores de dicha región. Cabe destacar que en las capas de pooling no interviene la profundidad de la entrada, ya que se aplican de manera independiente a cada canal de representación de características permitiéndonos conservar las características e información espacial de cada canal.

Los dos tipos más comunes de capas de pooling para CNNs son:

- **Max Pooling:** Selecciona el valor máximo de los elementos de la región proporcionando cierta invarianza a pequeñas translaciones. Esta invarianza viene dada por la naturaleza de esta capa de pooling, al tomar el valor máximo de cada región se detectará la característica más prominente independientemente de su posición exacta (siempre y cuando la translación no modifique el máximo de la región barrida, es decir la característica de interés cambie de región).
- **Average Pooling:** En este caso se selecciona el promedio de los valores de la región ayudando a suavizar y generalizar las representaciones.

Capa Fully connected

Las capas fully connected es una capa en la que cada neurona está conectada a todas las de la capa anterior y se suelen encontrar al final de una CNN tras haber realizado la extracción de características.

La entrada a esta capa es un vector unidimensional para poder conectar cada valor de la entrada con cada neurona de la capa. Dicho vector es obtenido a partir del tensor tridimensional de características con una operación de aplanado o 'flatten', por lo que a diferencia del resto de capas vistas hasta ahora, como convolución y pooling, en este caso la capa no es espacialmente estructurada por lo que no se mantiene la relación espacial de las características.

Tal y como vimos anteriormente en la descripción de la neurona de McCulloch y Pitts en la Sección 2.3.1, cada neurona realiza de forma interna una ponderación de los valores de entrada en función de los pesos asociados a dichas conexiones y luego aplica una función de activación no lineal para ver si el impulso es retransmitido a la siguiente capa. En este caso, al tener cada neurona unida a todas las de la capa anterior, se puede aprender a combinar las características extraídas con anterioridad para realizar tareas de detección, segmentación, clasificación etc

Junto con esta capa suelen aparecer técnicas de regularización y dropout para reducir el sobreajuste y mejorar el rendimiento debido a que la gran cantidad de parámetros de estas capas pueden llevarnos a sobreentrenar el modelo.

Softmax

A continuación de la capa fully connected solemos encontrar como capa final de la red, una capa de activación llamada softmax que toma el vector de valores reales obtenido de la capa fully connected y lo transforma a una distribución de probabilidades donde $\sum_{x=0}^n p_x = 1$.

Para el caso de clasificación, la capa softmax es la que nos permite obtener salidas interpretables como probabilidades, lo que facilita la toma de decisiones y la comparación entre diferentes clases. El resultado será una distribución de probabilidad sobre las diferentes clases, donde cada valor representará la probabilidad de que una instancia de entrada pertenezca a una clase en concreto basándose en las características obtenidas en las capas anteriores. La clase con probabilidad más alta es la predicción final de la red.

Cabe mencionar que esta capa es idónea para problemas de clasificación, sin embargo no es adecuada para problemas de regresión o cuando las clases no son mutuamente excluyentes.

2.3.3. Función de pérdida y algoritmos de optimización

Para poder entrenar correctamente una CNN es necesario escoger bien la función de pérdida para evaluar el modelo y el algoritmo de optimización que se utilizará para ir actualizando los pesos de la red. En nuestro caso optamos por la entropía cruzada como función de pérdida aunque fueron evaluadas otras métricas como *NLL*(negative log likelihood) y *MSE*(Mean Square error) sin embargo los resultados obtenidos no podían competir contra entropía cruzada.

La función de pérdida de entropía cruzada mide la discrepancia entre las probabilidades predichas por el modelo y las etiquetas reales de los datos, penalizando con una mayor intensidad las predicciones que más se alejan del valor real.

La fórmula de la entropía cruzada para problemas de clasificación multiclase se puede expresar de la siguiente manera:

$$L(y, \hat{y}) = - \sum_{i=1}^C y_i \log(\hat{y}_i) \quad (2.5)$$

Siendo L la función de entropía cruzada, y es el vector de etiquetas reales codificado en 'one-hot', es decir, un vector de tamaño igual al número de clases siendo y_i la pertenencia (1) o no (0) de la instancia a la clase i. También se define \hat{y} como el vector de probabilidades predichas por el modelo para cada clase.

Por otra parte se toman en consideración *SGD* y *ADAM* como algoritmos de optimización.

SGD (*Stochastic gradient descent*)^[21] es un método iterativo basado en la aproximación estocástica del descenso de gradiente, que hace uso del gradiente de una función diferenciable para, siguiendo la dirección opuesta del mismo, alcanzar el mínimo de dicha función.

Se inicializan los parámetros del modelo, se calcula el gradiente de la función de pérdida (∇_w) para los parámetros actuales y se actualizan los parámetros en función del gradiente y el *learning rate* (η), se repite iterativamente el cálculo y actualización hasta alcanzar el número máximo de iteraciones o una convergencia.

SGD tiene la misma base y funcionamiento, solo que en lugar de calcular el gradiente de la función de pérdida en todo el conjunto de datos, lo calcula en cada iteración de forma estocástica, utilizando solo un subconjunto (*mini-batch*) aleatorio de los datos de entrenamiento. Esto hace que el *SGD* sea más eficiente en términos de uso de memoria y tiempo de cálculo en comparación con los métodos que utilizan el conjunto completo de datos.

Encontramos por tanto que el *learning rate* será una constante durante el entrenamiento que determinará el tamaño de los pasos en la actualización de pesos que quedaría como sigue:

$$w = w_0 - \eta * \nabla w_0 \quad (2.6)$$

ADAM [14] es una algoritmo de optimización basado en gradientes de primer orden en funciones estocásticas basado en la estimación adaptativa del *Momentum*. Frente a *SGD*, que mantiene un único *learning rate* invariante durante el entrenamiento para todas las actualizaciones de los pesos, *ADAM* mantiene el *learning rate* para cada peso de la red pero adaptándolos por separado durante el entrenamiento de la red a partir de estimaciones del primer y segundo momento de los gradientes.

La actualización del *learning rate* durante el entrenamiento sigue la siguiente fórmula:

$$\eta = \eta_0 * \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} \quad (2.7)$$

Donde β_1 y β_2 son los hiperparámetros que indican la importancia del primer y segundo orden del gradiente y t el número de iteraciones.

Básicamente *ADAM* incluye *Momentum* al algoritmo *RMSprop* (Propagación del error cuadrático medio), el cual adapta el *learning rate* de cada parámetro en función del promedio de las magnitudes recientes de los gradientes

para el peso, estableciendo el *Momentum* como un promedio móvil exponencial (En concreto el del gradiente cuadrado).

La actualización de los pesos por tanto quedaría como sigue:

$$w_t = w_{t-1} - \eta * \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad (2.8)$$

Donde \hat{m}_t y \hat{v}_t son las estimaciones del primer y segundo momento y ϵ es una constante pequeña que evita la división entre 0.

De forma empírica se estiman los hiperparámetros (*learning rate*, tamaño de *batch*, número de épocas etc) que consigan ajustar mejor el modelo al conjunto de datos produciendo unas mejores predicciones en validación (Nos aseguramos que los patrones que está aprendiendo el modelo generalizan bien y no estamos sobreentrenando).

2.3.4. Arquitecturas evaluadas

Durante el desarrollo de este proyecto han sido evaluadas diversas arquitecturas para la clasificación de imágenes entre las cuales destacamos ResNet y Densenet.

ResNet(Residual Network)

A medida que una red neuronal adquiere profundidad de cara a aumentar la complejidad de las funciones que puede ajustar, su entrenamiento se hace más difícil debido al problema de la pérdida de gradiente (*vanishing gradients*) en el proceso de backpropagation del valor de la función de pérdida para el ajuste de los pesos de la red.

Para paliar el efecto de esta pérdida de gradiente y conseguir así poder entrenar modelos más profundos sin llegar a un estancamiento del modelo, Kaiming He et al. propusieron ResNet [9] con la idea de un 'bloque residual' formado por 2 rutas de propagación del gradiente. Una primera ruta residual, que aprende la diferencia entre la entrada y la salida esperada, y una conexión de atajo que pasa la entrada por la función identidad (Sin cambios). De esta forma se permite que el gradiente de la función de perdida retroceda hacia la entrada sin sufrir tanta degradación a través de conexiones de atajo. Además al combinar la información directa y la que pasa por todas las neuronas intermedias, se consigue potenciar el rendimiento del modelo.

El bloque residual básico es aquel formado en la conexión residual por 2 capas de pesos separadas por una función de activación no lineal cuyo

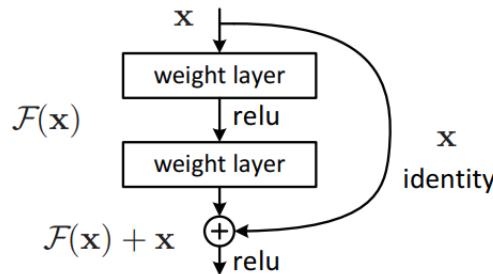


Figura 2.5: Bloque residual

resultado se suma al gradiente sin modificar. De esta forma permitimos que las características y el gradiente puedan tomar 2 caminos.

Se define formalmente la función subyacente al bloque residual como $H(x) = F(x) + x$ dejando así que las capas no lineales se ajusten a $F(x) = H(x) - x$. La hipótesis subyacente a esta asunción es que la optimización de la función residual es más sencilla que optimizar la origina.

ResNet es una arquitectura de redes neuronales que añaden conexiones residuales a una CNN convencional permitiendo así el aprendizaje efectivo de redes con mayor complejidad y profundidad sin el problema de la pérdida de gradiente.

Esta estrategia supuso un gran avance en el uso de redes neuronales al permitir un aumento significativo en la complejidad de los modelos. Así por ejemplo logró superar en rendimiento a VGG en el conjunto de test de IMAGENET¹.

En concreto se han estudiado el contenido de 3 arquitecturas basadas en ResNet con 18, 50 y 101 capas cuya estructura principal se define a continuación:

DenseNet(Densely Connected Convolutional Network)

Ante la observación de que redes con conexiones más cortas entre las primeras capas y las más cercanas a la salida podían ser sustancialmente más precisas y eficientes de entrenar, Gao Huang et al. propusieron la red convolucional densa DenseNet [11], que conecta cada capa a todas las siguientes de tal forma que el mapa de características producido como salida de una neurona pasará como entrada a todas las posteriores por lo que la entrada de cada neurona se corresponde con el conjunto de los mapas de activación de todas las neuronas anteriores.

De esta forma si una CNN convencional con N capas presenta N co-

¹Imagenet - <https://www.image-net.org/index.php>

layer name	output size	18-layer	50-layer	101-layer
conv1	112×112		7×7, 64, stride 2	
			3×3 max pool, stride 2	
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1		average pool, 1000-d fc, softmax	
FLOPs		1.8×10^9	3.8×10^9	7.6×10^9

Figura 2.6: Modelos basados en ResNet

nexiones, una entre cada 2 capas adyacentes, DenseNet presentará $\frac{N(N+1)}{2}$ conexiones directas. Podemos definir la salida de una capa t como

$$x_n = H_n([x_0, x_1, \dots, x_{n-1}]) \quad (2.9)$$

donde H_n es la función que representa las operaciones realizadas a nivel interno dentro del bloque denso y x_0, x_1, \dots, x_{n-1} son los mapas de características obtenidos de las capas anteriores.

DenseNet es una arquitectura con una estrategia de crecimiento aditivo que va incrementando el número de canales en cada capa por lo que se define el concepto de tasa de crecimiento (*Growth rate*) que determina cuantos canales se agregan en cada capa de un bloque denso en relación con los que ya existían. Si para una capa t H_n produce k mapas de características, la entrada de dicha capa serán $k_0 + k*(n-1)$ canales, donde k_0 son los canales de entrada al modelo.

Debido a este incremento intrínseco de parámetros y la complejidad computacional de los mismos que se sufre a causa de conectar todas las capas con las siguientes, se dividen en bloques densos con un número determinado de capas en su interior y entre ellos una capa de transición (*transitional layers*) que puede incluir capas de pooling, que ayudan a reducir la dimensionalidad espacial de las características, y convoluciones 1x1 para comprimir la información reduciendo la cantidad de canales.

Cada uno de estos bloques densos suele estar formada por múltiples bloques densos. Dentro de cada bloque, se suelen utilizar capas de convolución 3x3, seguidas de capas de normalización y funciones de activación (como ReLU) para aprender características no lineales. Además, en algunos blo-

Layers	Output Size	DenseNet-121
Convolution	112×112	
Pooling	56×56	
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	
	28×28	
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	
	14×14	
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Transition Layer (3)	14×14	
	7×7	
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$
Classification Layer	1×1	

Figura 2.7: Estructura de DenseNet-121

ques, se puede utilizar una capa de agrupamiento (pooling) para reducir la dimensionalidad y controlar el costo computacional

La arquitectura DenseNet se compone de bloques densos, donde cada bloque tiene múltiples capas y conexiones densas. Esta estructura altamente conectada facilita el flujo de información y mejora la capacidad de la red para capturar características relevantes en diferentes niveles de abstracción. El modelo concreto que utilizamos es DenseNet-121 que, con una tasa de crecimiento de $k = 32$, presenta la siguiente estructura donde cada capa marcada como 'conv' corresponde al esquema Batch normalization- ReLU - Convolución.

Encontramos que frente a ResNet que utilizaba la adición por elementos de un mapeo de identidad para promover la propagación del gradiente, DenseNet utiliza la concatenación de todos los mapas de características anteriores logrando un 'conocimiento colectivo' aunando los resultados obtenidos por capas anteriores.

2.4. Generative Adversarial Networks(GAN)

Las GAN son una arquitectura de Deep Learning para la generación de nuevos datos, en concreto imágenes, que sigan una distribución de datos conocida (muestras de entrenamiento). Se utilizan para el aprendizaje no supervisado y fueron inicialmente propuestas por Ian J. Goodfellow et al. en su paper Generative Adversarial Nets[4].

La idea principal de las mismas es la estimación de modelos generativos mediante un proceso adversarial en el que se entrena 2 modelos (G y D) de forma simultanea.

El modelo generador G, se encarga de crear falsificaciones de la muestra real capturando la distribución de datos intentando maximizar la probabilidad de que D cometa un error. La red ha de aprender las características comunes en las fotos de entrenamiento de tal forma que sea capaz de generar de forma autónoma una imagen que cumpla con las características del objeto a imitar pero sin copiar las imágenes de entrenamiento.

Por otra parte encontramos el discriminador D que se centra en estimar la probabilidad de que una muestra proceda de los datos de entrenamiento y no del modelo generador, descartando no solo las imágenes que se alejen de los datos de entrenamiento, es decir no se puedan identificar como los objetos que se intentan generar, sino también si la imitación es demasiado precisa es decir, una copia de los datos de entrenamiento.

El entrenamiento de las GAN se basa en el marco de un juego minimax de suma cero de dos jugadores en el que el generador trata de mejorar su capacidad de generar imágenes que logren engañar al discriminador y él por su parte tratará de conseguir una mejor distinción entre datos reales y generados.

Durante el proceso de mejora simultanea de los modelos mediante un aprendizaje adversarial ilustrado en la Figura 2.8, se tiende a un equilibrio en el que dentro del espacio de funciones arbitrarias G y D, G es capaz de estimar y reproducir la distribución de los datos de entrenamiento y D es $\frac{1}{2}$, es decir da una clasificación correcta para la mitad de las instancias de entrada.

Cabe destacar que para el correcto funcionamiento de este algoritmo es necesario que ambas redes tengan una complejidad similar de tal manera que se mejoren mutuamente sin que una red sea superior. En caso contrario el entrenamiento del modelo llegaría a un estancamiento ya que, en caso de ser más potente la generadora, D no podría distinguir los datos falsificados y no podría seguir aprendiendo y en caso contrario la red Discriminadora detectaría todos los falsos datos producidos por G impidiendo que está mejorara.

2.4.1. Descripción formal

En nuestro caso, G y D son ambos modelos de perceptrón multicapa permitiéndonos entrenar el modelo completo con retropropagación. Para que el generador aprenda la distribución sobre los datos de entrenamiento p_g , se define como entrada un vector de ruido obtenido aleatoriamente de una

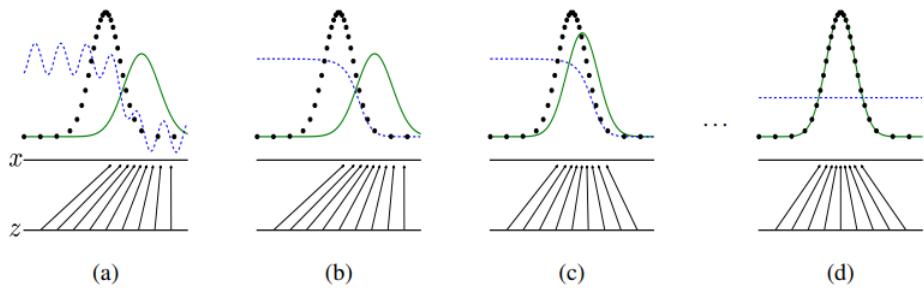


Figura 2.8: En estas gráficas podemos observar el entrenamiento simultáneo de las redes generativas antagónicas donde se va actualizando la distribución del discriminador (azul) para distinguir entre los datos de entrenamiento (gráfica negra) y aquellos producidos por la distribución del generador p_g (en verde). Tras varias épocas de entrenamiento (d) podemos ver como se alcanza un punto el que $p_g = p_x$ y el discriminador es incapaz de distinguir las imágenes generadas por G de las del conjunto de entrenamiento $D(x) = \frac{1}{2}$. Imagen obtenida de [4]

distribución gaussiana $p_z(z)$ que es mapeado al espacio de datos usando una función diferenciable G (que es un perceptrón multicapa) y tomando como parámetros θ_g como $G(z; \theta_g)$.

Por su parte se define el discriminador $D(x; \theta_d)$ donde tomando como entrada los parámetros θ_d y los datos de entrada x se produce como salida un escalar que representa la probabilidad $D(x)$ de que x sea una instancia de los datos en lugar de venir de p_g .

El método de actualización de pesos ilustrado en la Figura 2.9 consiste en que a medida que D intenta maximizar la probabilidad de obtener la etiqueta correcta, G trata de minimizar la cantidad de instancias producidas que son rechazadas por D, por lo que trata de minimizar $\log(1 - D(G(z)))$.

De esta forma quedaría definido el juego minimax entre D y G con función de pérdida $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_z(x)} [\log D(x)] + \mathbb{E}_{x \sim p_z(z)} [\log(1 - D(G(z)))] \quad (2.10)$$

Al principio del entrenamiento, los modelos generados por G son bastante pobres por lo que D puede rechazar las muestras fácilmente, por ello G puede entrenarse para maximizar $\log(D(G(z)))$ en lugar de minimizar $\log(1 - D(G(z)))$ que en este caso podría saturar.

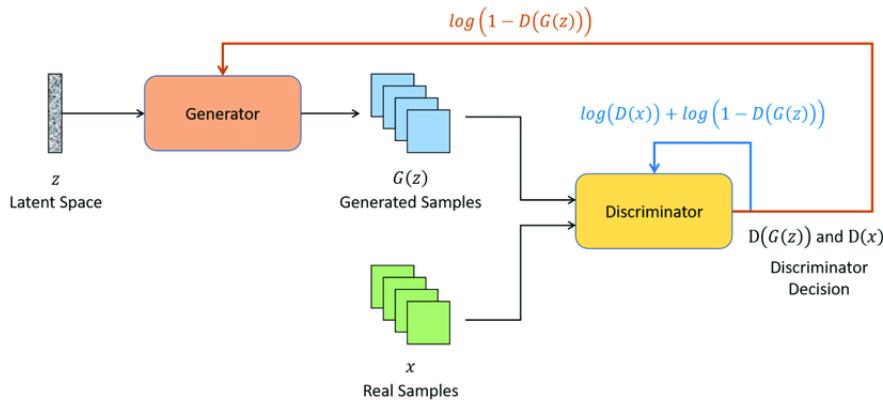


Figura 2.9: Estructura y retropropagación de las GAN

2.4.2. DCGAN

A pesar de que el generador G está diseñado para generar imágenes basándose en la distribución de las imágenes de entrenamiento en cooperación con el discriminador, se observó que las GAN tenían dificultades para converger durante el proceso de aprendizaje, lo que limita su capacidad para generar imágenes de alta calidad. Para abordar este problema, Radford et al. propusieron un modelo de GAN llamado Deep Convolutional GAN (DCGAN) [19], el cual utiliza capas de convolución y convoluciones transpuestas tanto en el discriminador como en el generador. La incorporación de dichas capas permite que el generador y el discriminador aprendan características espaciales más complejas y capturan detalles a diferentes escalas en las imágenes generadas, por lo que se puede alcanzar una mayor calidad. Dicha capacidad de generación de detalles es lo que nos llevó a decantarnos por usar la implementación de DCGAN en pytorch² como arquitectura para la generación automática de imágenes.

La arquitectura que se define para DCGAN especifique que tanto para G como D se hará uso de batch normalization y se eliminarán las capas ocultas de tipo fully connected para arquitecturas más profundas. Además en el discriminador se hará uso de LeakyReLU como función de activación lineal y las capas de pooling son sustituidas por capas de convolución con stride. De la misma forma para el generador se usará ReLU como función de activación y en este caso las capas de pooling serán sustituidas por convoluciones transpuestas.

²DCGAN

Pytorch https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

2.4.3. StyleGAN

Otro de los modelos que esta suponiendo el estado del arte en técnicas de generación automática de imágenes realistas en los últimos años es StyleGAN [12]. StyleGAN es una red generativa antagónica propuesta por NVIDIA a finales de 2018 que supuso un salto cualitativo sin igual permitiendo por ejemplo la creación del software 'This Person Does Not Exist' donde se muestra el gran potencial de esta red aplicado a la creación de rostros humanos hiperrealistas no pertenecientes a ninguna persona.

Este método esta basado en las GAN tradicionales pero, en lugar de aprender a mapear una distribución de ruido a la distribución de imágenes de entrenamiento, utiliza la técnica de 'decomposición de estilo' con la que se separa el estilo de una imagen para realizar transferencia de estilos lo cual nos permite un mayor control sobre los aspectos visuales de las imágenes generadas.

La transferencia de estilos nos permite combinar el contenido de una imagen de origen con el estilo de otra imagen de referencia. En este proceso, se extrae el contenido de la imagen de origen y se combina con los patrones y estilos aprendidos del modelo StyleGAN utilizando técnicas de procesamiento de imágenes. El resultado es una nueva imagen que conserva la estructura y el contenido de la imagen de origen, pero con el estilo visual de la imagen de referencia.

De cara a nuestro enfoque de generación de nuevos diseños de vehículos, nos permite fusionar características actuales de la marca o modelo con estilos ajenos de otras marcas o modelos de tal forma que sobre la línea de diseño actual supone una renovación del mismo.

Para utilizar esta aproximación a la generación automática de imágenes se ha hecho uso de la última implementación que ha lanzado NVIDIA llamada StyleGAN3³⁴ que mejora la consistencia entre los detalles finos y gruesos del generador.

El principal inconveniente de esta arquitectura es el elevado costo computacional y temporal que implica entrenar dicha red. Los resultados presentados en los capítulos 5 y 6 son muestras obtenidas después de un entrenamiento de dos semanas.

³Documentación StyleGAN <https://nvlabs.github.io/stylegan3/>

⁴Código stylegan3 <https://github.com/NVlabs/stylegan3>

2.5. Técnicas de explicabilidad de los modelos de clasificación

En el campo del aprendizaje automático y la Inteligencia Artificial, la explicabilidad (relación cualitativa entre la predicción del modelo y los componentes de los datos que el modelo usó para la predicción) de los modelos se ha vuelto cada vez más importante ante el auge de la complejidad de los modelos y la presencia de modelos caja negra tales como redes neuronales que no tienen un fácil análisis.

A medida que los modelos se vuelven más complejos y poderosos, también se vuelven más difíciles de entender y explicar pudiendo llevar en ocasiones a clasificadores que, a pesar de presentar un buen rendimiento para el conjunto de test, no realicen las predicciones basándose en información fiable. En la Figura 2.10 podemos apreciar un ejemplo de ello. Esta dificultad de comprensión en las antecedentes y características que llevan a un modelo a inferir una predicción plantea desafíos significativos, especialmente en aplicaciones críticas donde se requiere una justificación clara de las decisiones tomadas por el modelo.

En respuesta a esta necesidad surge XAI(eXplainable AI)[8] que es un conjunto de técnicas de explicabilidad de modelos que nos permiten una comprensión más profunda de la toma de decisiones de los mismos, proporcionando explicaciones intuitivas y basadas en la contribución de características individuales. Cabe destacar LIME (Local Interpretable Model-agnostic Explanations)[20] y SHAP (SHapley Additive exPlanations)[16] como técnicas que se centran en proporcionar explicaciones locales y comprensibles para las predicciones de los modelos de clasificación, lo que permite a los usuarios entender cómo y por qué se toman ciertas decisiones.

Ambas técnicas, LIME y SHAP, son métodos poderosos y ampliamente utilizados para mejorar la explicabilidad de los modelos de clasificación. Proporcionan insights valiosos sobre cómo funcionan los modelos y qué características son relevantes para las predicciones. Esto no solo brinda confianza en los resultados del modelo, sino que también ayuda a detectar posibles sesgos y mejorar la transparencia en las aplicaciones de aprendizaje automático.

En nuestro caso concreto aplicaremos ambas técnicas para comprender cuáles son las características de interés para que el clasificador pueda distinguir a qué marca pertenece el coche de una imagen y de esa forma, identificar patrones distintivos del lenguaje de diseño de las marcas, lo que les permitirá proteger su seña de identidad de cara al diseño de modelos futuros o la modificación de la misma para alcanzar características más atractivas para el consumidor.

30 2.5. Técnicas de explicabilidad de los modelos de clasificación

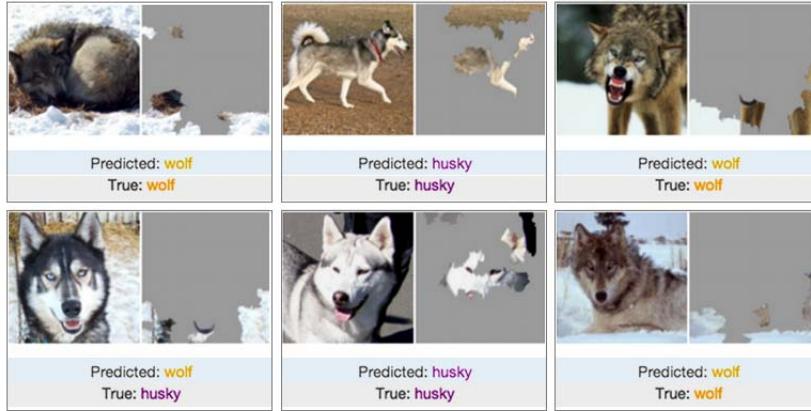


Figura 2.10: Análisis de modelo de clasificación donde vemos que no se fija en las características del individuo (lobo) para distinguir entre clases, sino que esta dándose cuenta que las fotos suelen tener nieve en el fondo. Clasificador incorrecto ya que no distingue al individuo, sino el fondo

2.5.1. LIME (Local Interpretable Model-agnostic Explanations)

LIME [20] es un algoritmo que nos permite explicar las predicciones de los modelos, aproximándolos localmente (dentro del entorno de la muestra a explicar) con un modelo interpretable. Proporciona una explicación intuitiva al identificar las características más influyentes para una predicción específica, lo que permite comprender qué atributos o variables han contribuido más a la toma de decisión del modelo.

Cuando tratamos de realizar explicaciones interpretables, se hace necesario el uso de representaciones comprensibles por los humanos independientemente de las características reales del modelo. Así para el caso que nos atañe (Clasificación de imágenes), una representación interpretable podría ser un vector binario que indique la 'presencia' o no de un conjunto de píxeles contiguos similares (super-píxel) mientras que el clasificador representa la imagen como un tensor con 3 canales de color por píxel.

Siendo la representación original de la instancia a explicar $x \in \mathbb{R}^d$, denotaremos como $x' \in \{0, 1\}^{d'}$ al vector binario para su representación interpretable, por lo que el dominio del modelo de explicabilidad será $\{0, 1\}^{d'}$.

Para el modelo a explicar $f : \mathbb{R}^d \rightarrow \mathbb{R}$, una explicación será un modelo g perteneciente a una clase de modelos potencialmente interpretables G , tal que no todo $g \in G$ será interpretable, por lo que se define $\Omega(g)$ como la medida de complejidad (inversamente proporcional a la interpretabilidad de los modelos).

Para clasificación $f(x)$ es la probabilidad de que x pertenezca a una clase determinada y $\pi_x(z)$ una medida de proximidad entre una instancia z y la instancia de referencia x , definiendo así el entorno vecino de x .

Con estos conceptos establecidos podemos pasar a explicar el sistema de explicabilidad de LIME, que garantiza la interpretabilidad y fidelidad local del modelo que explica una instancia por medio de la minimización de una medida $\mathcal{L}(f, g, \pi_x)$ que indica como de desleal es g a la hora de aproximarse al modelo original f en la localidad definida por π_x y garantizando que $\Omega(g)$ sea suficiente bajo como para ser interpretable, de tal forma que la explicación producida por LIME se obtendría como:

$$\xi(x) = \arg \min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g) \quad (2.11)$$

Como esta minimización ha de realizarse sin realizar ninguna suposición sobre f (para que el explicador sea independiente del modelo), se aproxima $\mathcal{L}(f, g, \pi_x)$ extrayendo muestras aleatoriamente de instancias tanto en las proximidades de x (que tienen una ponderación elevada π_x), como lejos de x (con un peso bajo debido a π_x).

LIME presenta una explicación, en forma de modelo lineal que es localmente fiel, donde la localidad es capturada por π_x tal y como se aprecia en la Figura 2.11. Se realiza un muestreo y obtención de un conjunto de datos en torno a la instancia a explicar utilizando perturbaciones en los mismos, de tal forma que a partir de una muestra perturbada $z' \in \{0, 1\}^{d'}$ (que contiene una fracción de elementos no nulos de x'), pasamos la muestra en la representación original $z \in \mathbb{R}^d$ y obtenemos la etiqueta $f(z)$ para el modelo de explicación. Obtenemos la explicación $\xi(x)$ optimizándola para el conjunto de muestras perturbadas.

De esta manera se puede explicar el modelo original localmente aunque sea demasiado complejo como para explicarlo globalmente. Finalmente se aplica una técnica de selección como Lasso para obtener las k características más importantes y luego se aprenden los pesos mediante mínimos cuadrados (este procedimiento se conoce como K-LASSO).

El proceso que lleva a cabo LIME en el caso de explicabilidad de modelos de clasificación de imágenes (Figura 2.12) consiste en primeramente se considera la representación de las imágenes como conjuntos de superpíxeles, que son conjuntos de píxeles contiguos similares en términos de textura, color etc se uso de técnicas de segmentación de imágenes para la identificación y delimitación de los mismos. Esto nos lleva a una reducción del ruido y redundancia de la explicación a la vez que se reduce la dimensionalidad de los datos.

Con esta transformación cada superpíxel se trata como una característica

32 2.5. Técnicas de explicabilidad de los modelos de clasificación

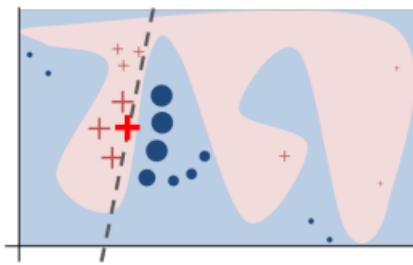


Figura 2.11: El modelo de caja negra esta representado en el fondo (Que no puede ser aproximado correctamente con un modelo lineal). LIME toma muestras y obtiene sus predicciones usando f y les otorga un peso en función de su distancia a la instancia a explicar (cruz resaltada). De esta manera se obtiene un modelo local (linea discontinua) que puede explicar el modelo de forma local a la instancia que esta siendo estudiada.

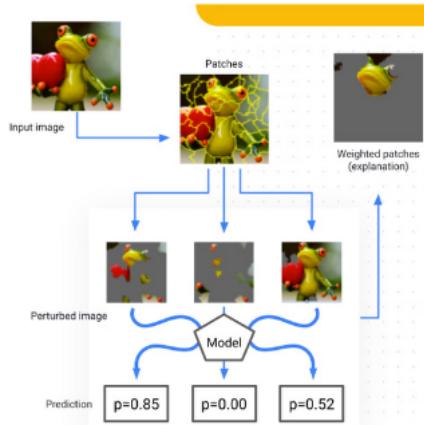


Figura 2.12: Aplicación de LIME sobre una imagen de interés

por lo que la representación interpretable de la imagen es un vector binario donde 1 indica el superpíxel original y 0 indica el superpíxel en gris (neutro).

El siguiente paso del procedimiento es la generación del conjunto de datos modificados introduciendo pequeñas perturbaciones en al imagen original estableciendo de forma aleatoria cada valor de característica como 0 o 1 (modificando el valor de presencia de píxeles o superpíxeles) y mapeando cada imagen generada al espacio original quedando ,tal y como se ha indicado anteriormente, imágenes en los que los superpíxeles marcados como 1 quedan intactos y los marcados como 0 son sustituidos por regiones en gris.

Tras esto pasamos las imágenes del conjunto de datos generado a f para obtener sus etiquetas y se construye el modelo explicativo con la información obtenida.



Figura 2.13: Características positivas (mostradas en verde) y negativas (en rojo) para la clasificación de una imagen como perteneciente a la clase gato. Podemos observar como el clasificador se centra en la forma del cuerpo para identificar el gato y por otra parte la cabeza del perro sería un identificador de la clase perro contrario a la predicción gato

Cuando utilizamos explicaciones lineales para clasificadores de imágenes, puede ser deseable resaltar únicamente los superpíxeles con peso positivo hacia una clase específica, ya que proporcionan intuición sobre por qué el modelo piensa que esa clase puede estar presente. Sin embargo, en el caso de LIME⁵, también tenemos la posibilidad de visualizar las regiones de la imagen que nos llevan a pensar que una instancia NO pertenece a la clase con mayor probabilidad tal y como se ve en la Figura 2.13. Esto nos permite no solo explicar que el modelo está realizando una clasificación basada en características relevantes de la imagen, evitando así predicciones que no se basen en elementos de interés, como el ejemplo del lobo, sino también nos permite identificar las características que están generando incertidumbre dentro de una instancia.

Esta capacidad de LIME de visualizar tanto las características positivas como las negativas en la clasificación de una imagen tiene varias ventajas. En primer lugar, nos permite comprender mejor las similitudes intraclasa y detectar posibles mejoras en el modelo mediante la segmentación de regiones de interés.

Además, al visualizar las regiones de la imagen que el modelo considera negativas, también obtenemos información valiosa sobre los errores potenciales del modelo y posibles casos de sesgo o falta de generalización. Esta información nos ayuda a identificar patrones problemáticos y a tomar medidas para corregirlos.

Al realizar explicaciones locales, se generan explicaciones adaptadas a instancias específicas en lugar de ofrecer una global del modelo completo. No obstante, LIME tiene como desventaja su sensibilidad a las perturbaciones en la instancia de entrada. Al modificar la instancia original para crear

⁵Código y documentación LIME <https://github.com/marcotcr/lime>

34 2.5. Técnicas de explicabilidad de los modelos de clasificación

el conjunto de datos locales, cualquier cambio, incluso pequeñas perturbaciones, puede dar lugar a explicaciones significativamente diferentes.

2.5.2. SHAP (SHapley Additive exPlanations)

Por otro lado, SHAP[16] es una de las librerías ⁶ de explicabilidad de modelos basada en la teoría de juegos que asigna valores de importancia a cada característica o atributo en función de su contribución a la predicción del modelo.

Utiliza el concepto de los valores de Shapley [27] que es un concepto matemático de la teoría de juegos definido como el beneficio total generado por la coalición de todos los jugadores en un juego cooperativo.

Para comprender que son los valores de Shapley, supongamos un juego cooperativo con un conjunto de n jugadores llamado F ($F = 1, 2 \dots n$), un subconjunto S de F llamado coalición ($S \subseteq F$) y una función v definida como $v : 2^n \rightarrow \mathbb{R}$ tal que $v(S)$ (valor de la coalición S) representa la ganancia total de los miembros de S por su cooperación.

Dado que \emptyset no tiene jugadores $v(\emptyset) = 0$, por ello partimos de dicho conjunto para calcular la aportación de cada jugador, de tal forma que vamos añadiendo jugadores y calculando su contribución a la coalición actual como $v(S') - v(S_0)$ donde S' es la coalición al añadir el jugador y S_0 es la coalición de la que partimos. Por ejemplo partiendo del conjunto vacío, la contribución del jugador 1 al conjunto de jugadores $\{1\}$ sería $v(\{1\}) - v(\{\})$

Dado que el orden de incorporación de los jugadores es de interés a la hora de calcular la contribución total de un jugador al juego, han de estudiarse todas las permutaciones posibles entre los jugadores teniendo en cuenta el orden de incorporación, por ejemplo para $N=3$ el conjunto de coaliciones posibles sería:

$$\{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 1\}, \{2, 3\}, \{3, 1\}, \{3, 2\}, \\ \{1, 2, 3\}, \{1, 3, 2\}, \{2, 1, 3\}, \{2, 3, 1\}, \{3, 1, 2\}, \{3, 2, 1\}\} \quad (2.12)$$

De esta forma, el valor de Shapley para cada característica X , es calculado como el promedio de la contribución de dicha característica a todas las coaliciones posibles en las que esté involucrada dicha variable. En nuestro caso las características se corresponden con píxeles o conjuntos de ellos dado que estamos trabajando sobre imágenes.

De esta forma SHAP utiliza el concepto de los valores de Shapley para calcular la contribución individual de cada característica, permitiendo obtener una explicación global y local sobre cómo se forman las predicciones del

⁶Documentación SHAP <https://shap.readthedocs.io/en/latest/>

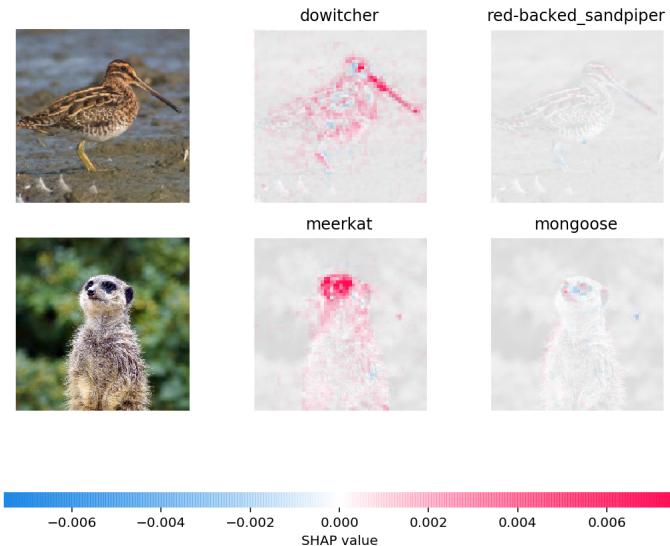


Figura 2.14: Ejemplo de uso de SHAP para explicabilidad de modelos de clasificación de imágenes

modelo de clasificación, permitiéndonos analizar cuales son las características distintivas en los que el modelo se está fijando para discernir entre clases. En concreto tal y como se ve en la Figura 2.14 se nos muestran resaltados en la imagen los píxeles con mayor contribución a la predicción de la instancia como perteneciente a una clase.

ϕ_j se corresponde con la contribución que aporta la característica j a la predicción de esta instancia particular en comparación con la predicción promedio para el conjunto de datos.

No se debe confundir la interpretación del valor de Shapley. El valor de Shapley es la contribución promedio de un valor de característica a la predicción en diferentes coaliciones, NO es la diferencia en la predicción si elimináramos la característica del modelo.

La fórmula para el calculo del valor de Shapley en un juego de coalición (v, N) quedaría como sigue:

$$\phi_i(v) = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} (v(S \cup \{i\}) - v(S)) \quad (2.13)$$

Por otra parte los valores de Shapley han de cumplir ciertas propiedades que nos permiten valorar las explicaciones tanto a nivel global como local, estas propiedades son:

- **Eficiencia:** La suma de las contribuciones de los N Jugadores equivale

36 2.5. Técnicas de explicabilidad de los modelos de clasificación

a la ganancia total, es decir, la ganancia total se distribuye.

$$\sum_{i=1}^{|F|} \phi_i = v(F) \quad (2.14)$$

- **Simetría:** Si 2 jugadores i y j son equivalentes, es decir:

$$v(S \cup \{i\}) = v(S \cup \{j\}) \forall S : \{i, j\} \cap S = \emptyset \implies \phi_i = \phi_j \quad (2.15)$$

Esto quiere decir que si tienen la misma ganancia para toda coalición entonces han de tener la misma contribución.

- **Dummy (jugador Nulo):** Un jugador es Nulo cuando:

$$v(S) = v(S \cup \{i\}) \forall S : i \notin S \implies \phi_i = 0 \quad (2.16)$$

lo cual quiere decir que si la ganancia aportada por i es nula para cualquier coalición, entonces su contribución será 0.

- **Aditividad:** Si tenemos 2 funciones de ganancia v y w para un mismo juego entonces las ganancias distribuidas han de corresponderse con las ganancias derivas de v y w, es decir

$$\phi_i(v + w) = \phi_i(v) + \phi_i(w) \forall i \in F \quad (2.17)$$

Además de forma análoga $\forall a \in \mathbb{R}$:

$$\phi_i(av) = a\phi_i(v) \quad (2.18)$$

SHAP es una librería encargada que nos proporciona diversos 'explainers' para el cálculo de dichos valores, en concreto en este proyecto se hará uso de DeepExplainer⁷ es un explainer agnóstico para un modelo diferenciable que realiza una aproximación a los valores Shapley para modelos de aprendizaje profundo mediante la integración de múltiples muestras de fondo.

Las estimaciones realizadas son la suma de la diferencia entre la salida del modelo esperado en las muestras de fondo pasadas y la salida actual del modelo ($f(x) - E[f(x)]$)

DeepExplainer es una versión mejorada de DeepLIFT(Deep Learning Important FeaTures[24]), un método de descomposición de la predicción de salida de una red neuronal realizando la retro-propagación de las contribuciones, calculadas acordes a la diferencia entre la activación de cada neurona y su 'activación de referencia', de todas las neuronas hasta las características de entrada.

⁷Documentación de DeepExplainer - <https://shap-lrjball.readthedocs.io/en/latest/generated/shap.DeepExplainer.html>

Cuando hablamos de 'activación de referencia' nos referimos a la activación de una neurona frente a una linea base, en el caso de imágenes se puede medir la activación de una neurona frente a diferentes estímulos 'visuales' en los píxeles, siendo la activación de referencia un estímulo neutro o de poco interés, identificando así que características provocan mayor activación en la neurona.

Capítulo 3

Marco experimental

En este capítulo describiremos inicialmente el entorno de experimentación en el que han sido realizadas las pruebas, métricas usadas para evaluar los modelos, arquitecturas estudiadas etc Posteriormente se mostrarán los diversos experimentos llevados a cabo durante la realización de este proyecto junto con los resultados obtenidos en los mismos. Concretamente se dividirá esta sección en los 3 enfoques de especificidad que se han tenido en cuenta para el estudio de esta base de datos:

- Marca-Modelo-Año (MMA): clasificación de imágenes.
- Marca-Modelo (MM): Clasificación de imágenes y aplicación de expli- cabilidad de modelos para la detección de características distintivas en modelos de interés.
- Marca (M): En este nivel de granularidad además de lo especificado anteriormente procederemos al uso de redes generativas para la ob- tención de diseños adaptados a las diversas marcas, tanto siguiendo la línea de diseño presente en la marca y que ha sido inferida por los modelos de clasificación, como la renovación de marcas y modelos haciendo uso de transferencia de estilos.

3.1. Entorno de experimentación

La complejidad de los modelos de visión por computador, si bien es lo que los dota del potencial para aprender las distribuciones de conjuntos de imágenes para tareas cada vez más complejas, es también la mayor dificul- tad, ya que son computacionalmente muy costosos y requieren de hardware específico basado en GPU(Unidades de procesamiento gráfico) para lograr entrenar e inferior los modelos en un tiempo asequible.

Las GPUs son componentes de hardware diseñados específicamente para realizar cálculos intensivos de manera paralela. Son capaces de procesar múltiples tareas a la vez, lo que acelera y optimiza significativamente la manipulación y cálculos matemáticos con imágenes.

Dada la complejidad de los modelos a utilizar en este proyecto como las GANs y la complejidad a la que asciende el problema de la clasificación con bases de datos del tamaño de la usada aquí, realizar la experimentación en mi ordenador personal con una tarjeta gráfica de 6GB de espacio se hizo inviable por lo que fue necesario el uso de un servidor con mayor capacidad de computación.

En concreto se ha hecho uso de los servidores GPU de la universidad de Granada localizados en el edificio CDP Santa Lucía a través de la cola de slurm dios. Se ha tenido acceso a diversos nodos de cómputo, sin embargo cabe destacar el uso de Dionisio debido a su mayor memoria CUDA lo cual me permitió el entrenamiento de modelos con mayor complejidad tales como Stylegan. Dicho nodo de cómputo cuenta con 2 GPUs Quadro RTX 8000

Sobre el servidor se montaron 2 entornos de anaconda¹ con el conjunto de paquetes necesarios para la correcta realización de los experimentos, 1 primer entorno para la ejecución de Stylegan3 basándose en las especificaciones indicadas por NVIDIA para el entrenamiento de dicha arquitectura y otro que contuviera los paquetes necesarios para el resto de experimentos tales como PyTorch, CUDA, SHAP, lime etc

Las arquitecturas de redes neuronales que han sido evaluadas como ResNet y DenseNet están implementadas utilizando Pytorch y están pre-entrenadas para conjuntos de imágenes estándar, en este caso fueron diseñadas y pre-entrenadas para Imagenet que es uno de los conjuntos de datos de referencia más populares para la clasificación de imágenes con más de 1.2 millones de imágenes etiquetadas en 1000 categorías distintas. Antes de proceder a entrenar con dichos modelos tuvieron que adaptarse las arquitecturas a nuestra base de datos de entrenamiento sustituyendo la cabeza de los modelos (Capas fully-connected, softmax etc) por una nueva cuyos pesos se entrenarán con nuestro conjunto de datos.

Inicialmente surgió un problema de cara a la utilización de dichos servidores para la ejecución de código que requeriera, como era el caso, el acceso a CUDA, ya que el entorno anaconda que se definía siguiendo las instrucciones del servidor no permitía la correcta instalación de CUDA debido a la opción -c al instalar Pytorch. Tras diversas pruebas y comunicaciones con los administradores de los servidores se logró detectar el error y solucionarlo, aunque esto supuso un retraso en el desarrollo del proyecto.

En el Cuadro 3.1 se pueden encontrar de forma resumida las distintas

¹Documentación anaconda - <https://www.anaconda.com/>

Frameworks	Anaconda	https://www.anaconda.com/
Lenguaje de programación	Python	https://www.python.org/
Bibliotecas	Pytorch SHAP LIME	https://pytorch.org/ https://shap.readthedocs.io/en/latest/index.html https://github.com/marcotcr/lime
Hardware	NVIDIA V100	https://www.nvidia.com/en-us/data-center/v100/

Cuadro 3.1: Resumen de tecnologías/bibliotecas/hardware usadas en el proyecto



Figura 3.1: Ejemplos de instancias de la base de datos utilizada

tecnologías, frameworks, bibliotecas y hardware utilizado para el desarrollo de este proyecto.

3.2. Dataset

Para el desarrollo de este proyecto se ha hecho uso de un conjunto de datos cedido por el Instituto Andaluz Interuniversitario en Data Science (DASCI²). Dicha base de datos será publicada en dicha web cuando terminen los NDA(Non-Disclosure Agreement) del proyecto que originó la recolección de los datos pero para la evaluación de este TFG será puesta a disposición del tribunal de forma temporal en el repositorio de GitHub del proyecto³. Dicho conjunto contiene 5712 imágenes de frontales de vehículos comprendiendo distintas marcas, modelos y remesas de un mismo modelo a lo largo de los años (Figura 3.1). Dichas imágenes no tienen un tamaño fijo por lo que será necesario redimensionarlas en el preprocesado para pasárselas a la red ya que esta necesita que las imágenes tengan todas iguales dimensiones. Las imágenes están repartidas en un conjunto de 461 clases distintas(especificidad marca-modelo-año) divididas en train y test.

Se hace necesario definir una estructura clara para la base de datos de cara a trabajar con ella posteriormente por lo que será necesario tener 3 sub-datasets³ cada uno con una especificidad de las expuestas anteriormente para ser utilizado para una tarea concreta. Por otra parte, si bien es cierto que supone un coste computacional mayor, se ha optado por una

²<https://dasci.es/es/>

³Base de datos utilizada - <https://github.com/anariasnav/Car-design-analisis/tree/ed5652e8cd2129760e041139f9b6ab87bb9e3c43/BD-461>

estrategia de validación cruzada que nos ofrecerá mejores resultados dado el bajo número de instancias en nuestro dataset. Esta estrategia consiste en la división del dataset en 3 conjuntos de datos train, test y val donde train será utilizado para entrenar el modelo para el aprendizaje de patrones por medio del ajuste de los parámetros. El conjunto de validación (val) se utiliza para realizar ajustes adicionales durante el entrenamiento y detectar posibles problemas de sobreajuste. Finalmente el conjunto de test nos servirá para evaluar el rendimiento generalizado del modelo final pasando como entrada datos nunca vistos por el modelo.

Para alcanzar esta estructura en el dataset se implementó un script Python *modCarpetas.py* (ver código en la carpeta '*Auxiliary scripts*' del repositorio del proyecto ⁴) que leyendo la carpeta con la base de datos, genera de forma automática 3 carpetas cada una de las cuales pasará a albergar el conjunto de imágenes con una granularidad distinta, así encontraremos 3 carpetas MMA, MM y M. Para ello se fusionan las clases pertenecientes al mismo modelo o a la misma marca respectivamente.

Además y de forma simultánea generará dentro de cada carpeta la estructura de validación cruzada definida anteriormente. De esta forma train quedará intacto y test será dividido entre validación y test.

3.3. Preprocesado

Una vez que hemos establecido la estructura del conjunto de datos y comprobado que todas las instancias están etiquetadas correctamente podemos pasar a cargar la base de datos en los programas para entrenar las distintas arquitecturas para nuestro conjunto de imágenes concreto. Haciendo uso de 2 herramientas proporcionadas por torchvision como son **ImageFolder** y **DataLoader** procederemos a cargar las imágenes indicando la ruta donde se encuentran los archivos y una serie de transformaciones que se realizarán sobre las imágenes de entrada para prepararlas de cara a entrenar con ellas.

Las transformaciones comunes a todos los subconjuntos de datos son:

- **Resize()** - redimensiona las imágenes por medio de una interpolación bilineal para que todas tengan el mismo tamaño, en concreto se establece sus dimensiones a 256x256x3.
- **ToTensor()** - transforma las imágenes de entrada a un tensor, escalando los valores del rango [0.0,255.0] a [0.0,1.0].

⁴<https://github.com/anariasnav/Car-design-analisis/tree/efeddef12fda221b1d43554a39fb4343fdee7e24/Auxiliary%20scripts>

- `Normalize()` - Normaliza el tensor de entrada a una media y desviación típica indicados.

Por otra parte para poder aumentar la cantidad y diversidad de datos de entrenamiento y mejorar así la capacidad de generalización del modelo reduciendo el *overfitting*, se incluye *data augmentation* que es una técnica consistente e la aplicación de transformaciones a las muestras de datos existentes de manera que al aplicar dichas transformaciones se generen nuevas muestras sintéticas.

A continuación se enumeran las distintas transformaciones que se tuvieron en cuenta para aplicarlas sobre el conjunto de datos de entrenamiento y la explicación de porque han sido o no aplicadas finalmente.

- `RandomHorizontalFlip` y `RandomRotation` - son 2 transformaciones afines consistentes en voltear horizontalmente o rotar un cierto ángulo algunas imágenes de entrada. Aportan diversidad al conjunto de datos dotando al modelo de una cierta independencia a las transformaciones afines y del punto de vista de la cámara.
- `ColorJitter` - Altera aleatoriamente el brillo, el contraste, la saturación y el tono de una imagen.
- `GaussianBlur` - Aplica un filtro de alisado gaussiano aleatorio sobre la imagen.
- `RandomEqualize` - Esta transformación mejora el contraste de la imagen la vez que resalta los detalles de la misma por medio de la ecualización del histograma de dicha imagen.
- `RandomAutocontrast` - Se estudia su uso con el mismo objetivo que `RandomEqualize` obtener un resaltado de los detalles en la imagen. Esta transformación contrasta automáticamente los píxeles de la imagen al azar con una probabilidad dada.

3.4. Métricas de evaluación

Finalmente concluimos la definición del marco experimental en el que trabajaremos hablando de las métricas de evaluación que serán utilizadas para, una vez concluido el entrenamiento del modelo, poder medir y comparar las prestaciones de los modelos entrenados.

Para los casos de clasificación binaria las ecuaciones se definen en función de los valores de la matriz de confusión donde:

	Clase	Etiquetas predichas				Total
		a	b	c	d	
Clase a la que pertenecen	a	6	0	1	2	9
	b	3	9	1	1	14
	c	1	0	10	2	13
	d	1	2	1	12	16
Total		11	11	13	17	52

Figura 3.2: Matriz de confusión para clasificación multiclas

- TP(True positive): representa las instancias que han sido correctamente clasificadas como pertenecientes a una clase por el modelo.
- TN(True negative): son aquellas instancias que se clasifica correctamente que no pertenece a una clase.
- FP(False positive): instancias que han sido clasificadas como pertenecientes a una clase pero sin embargo no pertenecen a la misma.
- FN(False negative): instancias que son clasificadas como no pertenecientes a una clase a la que en realidad si pertenecen.

Sin embargo en los casos de clasificación multiclas [6] como el nuestro, la matriz de confusión involucra a las N clases posibles. Por lo que la matriz de confusión es una tabla cruzada que registra el número de ocurrencias entre dos clasificadores, la clasificación verdadera/real y la clasificación predicha, como se muestra en la Figura 3.2 donde las columnas representan la predicción del modelo y las filas la etiqueta real para una instancia. Las clases se enumeran en el mismo orden en las filas que en las columnas, por lo que los elementos correctamente clasificados se encuentran en la diagonal principal, desde arriba hacia abajo, correspondiéndose los valores al número para el que las filas y las columnas coinciden.

Por lo que para las métricas de error habría de calcularse el resultado de enfrentar cada clase individual con el resto y realizar la media ponderada. Las ecuaciones que a continuación se muestran calculan las métricas para una clase en concreto.

- **Accuracy:** Mide la proporción de instancias para las cuales el modelo ha realizado una predicción correcta con respecto al total de predicciones realizadas. Esta medida puede presentar un pequeño inconveniente para el caso de conjuntos con clases desbalanceadas, como es el caso, ya que puede no quedar bien representado el error de clasificación para clases con pocas instancias.

$$Accuracy_k = \frac{TP_k + TN_k}{TP_k + TN_k + FP_k + FN_k} \quad (3.1)$$

- **Precisión Promedio:** indica la capacidad de realizar predicciones precisas para todas las clases. Se calcula tomando la media de las precisiones, muestras correctamente clasificadas de una clase entre todas las que han sido etiquetadas con dicha clase.

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \quad (3.2)$$

- **Recall:** Representa la tasa de verdaderos positivos, que representa la capacidad de identificar correctamente muestras de una clase.

$$Recall_k = \frac{TP_k}{TP_k + FN_k} \quad (3.3)$$

- **F1-score:** Es la media armónica de la precisión y el recall.

$$F1 - Score = 2 * \left(\frac{precision * recall}{precision + recall} \right) \quad (3.4)$$

Capítulo 4

Clasificación e identificación de patrones de interés en imágenes de vehículos

Una vez definidos el marco de trabajo y los conceptos generales que atañen a las implementaciones realizadas (Capítulos 2 y 3 respectivamente), procedemos en este capítulo a ilustrar el proceso de entrenamiento y evaluación de los distintos modelos de clasificación de imágenes propuestos, haciendo hincapié en las distintas propuestas de arquitecturas, valores de hiperparámetros, transformaciones evaluadas junto con los razonamientos y elecciones a los que nos han llevado así como los resultados experimentales obtenidos.

En este capítulo se abordará, en primer lugar, las transformaciones de preprocessamiento que se aplicarán y se evaluará su impacto en el desempeño de los modelos. En la Sección 4.1 se explorará el proceso de estimación de hiperparámetros llevado a cabo cuyo objetivo es obtener el mejor rendimiento de nuestros modelos. Posteriormente en la Sección 4.2, se presentarán las diferentes arquitecturas utilizadas, así como las modificaciones realizadas en ellas y los algoritmos de optimización empleados. En la Sección 4.3 se mostrarán los resultados experimentales obtenidos para todos los experimentos realizados realizando una distinción de los resultados para cada especificidad estudiada. Finalmente, se mostrarán los resultados obtenidos a partir de estos experimentos y se aplicarán técnicas de explicabilidad para una mejor comprensión del rendimiento de los distintos modelos, cerciorándonos así de que los resultados obtenidos son fiables y la red está tomando en consideración las características relevantes.

4.1. Análisis del desempeño de modelos ante transformaciones y preprocesamiento

Para realizar el entenamiento de nuestro modelo es necesario cargar nuestros datos en el script de cara a trabajar con ellos. Para lograrlo se utilizaran ImageFolder y DataLoader pasándoles como argumento la ruta al dataset y el conjunto de transformaciones a aplicar sobre las imágenes de entrada.

Las transformaciones para las que se ha estudiado su uso fueron explicadas en la Sección 3.3. A continuación procederemos a explicar, basándonos en los resultados empíricos obtenidos mostrados en el Cuadro 4.1, las conclusiones obtenidas y el porque de la presencia o ausencia de las distintas transformaciones.

Definimos un conjunto de transformaciones base para el modelo que comprende Resize, ToTensor, Normalize y las transformaciones geométricas afines RandomHorizontalFlip y Random Rotation. Sobre este conjunto se probó a añadir otras como:

- ColorJitter: Su uso no afecta apenas al rendimiento del modelo, llegando incluso a empeorar los resultados obtenidos por lo que no se usó finalmente.
- GaussianBlur: Se ha determinado experimentalmente que esta transformación empeora el rendimiento del modelo al alisar los detalles de los vehiculos que justamente son los que permiten a la red distinguir entre modelos. Por ello esta transformación no es coherente añadirla dado el dominio del problema (Si es útil en casos en los la variabilidad interclase son mayores o debidas a aspectos más generales como la forma, color etc)
- RandomEqualize - Al observar los resultados y conclusiones obtenidos de la aplicación del filtro de alisado gaussiano fue inmediato pensar en el resaltado de los detalles como una opción más que interesante. Sin embargo y a pesar de que las métricas de error para el conjunto de test suponían una mejora sustancial sobre el rendimiento, al aplicarle modelos de explicabilidad se obtuvieron los resultados mostrados en la Figura 4.1 con lo que se pudo comprobar que el clasificador estaba observando elementos del fondo en lugar de realizar una clasificación basada en las características del vehículo por lo que se descarto su uso de inmediato.
- RandomAutocontrast: se propuso como alternativa a RandomEqualize debido a que al observar algunas imágenes del conjunto de datos cargado con la transformación anterior, nos dimos cuenta de que el filtro

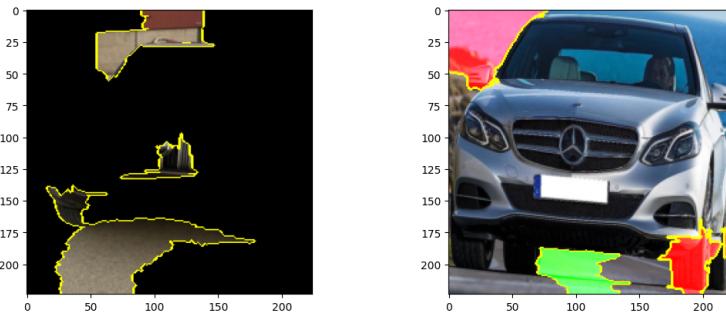


Figura 4.1: Podemos observar como el modelo esta utilizando características ajenas al diseño del vehículo, en concreto del fondo, para realizar la clasificación.

	Accuracy	Precisión promedio	Recall	F1
Base	91.21	90.39	91.21	90.19
ColorJitter	90.76	89.82	90.76	89.61
GaussianBlur	87.6	87.39	87.6	86.67
RandomEqualize	93.01	92.05	93.01	91.97
RandomAutocontrast	90.87	90.17	90.87	89.89

Cuadro 4.1: Desempeño de los modelos ante la presencia o ausencia de ciertas transformaciones en el conjunto de datos

estaba reduciendo casi por completo la información visual extraible de la imagen. Esta transformación se basa en el contraste de los pixeles de la imagen dada al azar con una probabilidad pasada como parámetro. A pesar de que en este caso las imágenes si presentaban un nivel de detalle mínimamente mayor, los resultados del modelo eran mejores sin incluir esta transformación.

4.2. Estimación de hiperparámetros

Una vez cargados los datos, podemos proceder a definir y ajustar todos los hiperparámetros que afectarán al rendimiento de nuestro modelo. Para comenzar se realizó una evaluación a priori con un tamaño pequeño de batch para realizar una aproximación del valor de learning rate que nos permitiera maximizar el rendimiento de los modelos para el problema concreto de clasificación. Tras estudiar los valores teóricos propuestos en los papers de los distintos algoritmos de optimización utilizados, se propuso una pequeña prueba que sobre, un mini-batch de menor tamaño, fuera probando distintos valores de learning rate y obteniendo el valor de la función de pérdida que se

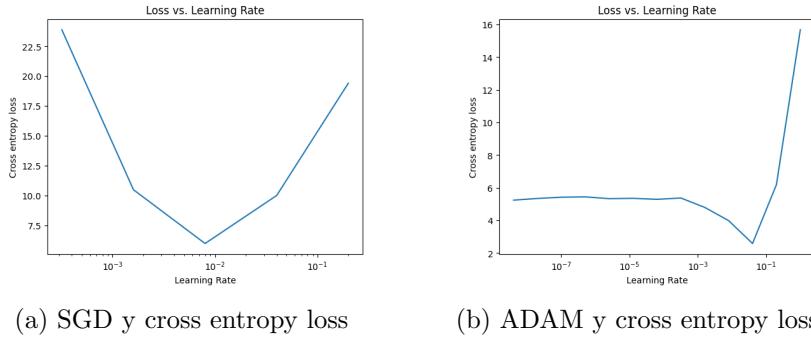


Figura 4.2: Gráficas que muestran la relación entre η y el valor de la función de pérdida de Entropía cruzada.

obtenía tras entrenar con ese conjunto de datos. En la Figura 4.2 se pueden ver las gráficas obtenidas del estudio de valores de η comprendidos en el rango $[10^{-7}, 10^{-1}]$ en relación al valor de pérdida que obtienen.

Así se demostró de forma empírica que el valor propuesto a nivel teórico $\eta = 10^{-2}$ es el más adecuado para el entrenamiento de nuestros modelos.

Por otra parte fue necesario estimar el número de épocas y el tamaño de batch para entrenamiento. En el primer caso no fue necesario más que la inclusión de early stopping en el entrenamiento, de tal forma que se ha podido indicar un número de épocas elevado por si fuera necesario pero el modelo detendrá el entrenamiento en caso de dejar de mejorar o comenzar a sufrir overfitting (Se empeora el rendimiento en validación). Por otra parte para el caso del tamaño de batch se observó de manera empírica como el aumento del mismo no siempre supone un mejor o más rápido ajuste del modelo, por lo que se concluyó que el valor óptimo sería de 64.

4.3. Clasificación de imágenes

Una vez realizadas las suposiciones iniciales y estimado los hiperparámetros, procederemos a evaluar y comparar los diferentes modelos entrenados. Analizaremos las diferencias en términos de arquitectura y sus modificaciones, los optimizadores utilizados y el nivel de granularidad al que se ha entrenado cada modelo.

4.3.1. Especificidad MM

Para la especificidad Marca-Modelo, tenemos 193 clases diferentes que representan los distintos modelos. En este contexto, se analizará el rendimiento de los diferentes modelos propuestos, así como las posibles modificaciones

en las últimas capas de clasificación de dichos modelos.

Los resultados obtenidos en este apartado nos dotan de información que luego será utilizada para el resto de especificidades como optimizador y función de pérdida con mayor rendimiento y valor de learning rate adecuado (mostrado anteriormente).

En la primera parte del estudio, se analizó cuál era la arquitectura que mejor se adaptaba a nuestro conjunto de entrenamiento. Se consideraron las arquitecturas ResNet18, ResNet50, ResNet121 y DenseNet101. Podría ser lógico pensar que el aumento en el número de capas para las arquitecturas basadas en ResNet nos proporcionaría un aumento en el rendimiento del modelo, puesto que a mayor profundidad mayor es la complejidad de la distribución de datos que puede ser ajustada. Sin embargo, a la luz de los resultados mostrados en el Cuadro 4.2 observamos como ResNet18 es la que mejor ajusta el conjunto de datos de las arquitecturas ResNet. Al usar modelos más complejos como ResNet50 y ResNet101 corremos el riesgo de estar sobreajustando, tal y como se ve en la disminución del rendimiento en test. Por otra parte vemos como el modelo DenseNet presenta un mejor rendimiento para nuestro conjunto de datos concreto.

	Accuracy	Precisión promedio	Recall	F1
ResNet18	90.98	89.56	90.98	89.82
ResNet50	84.33	83.95	84.33	82.99
ResNet101	86.13	86.40	86.13	84.92
DenseNet121	91.21	90.39	91.21	90.19

Cuadro 4.2: Comparativa entre las distintas arquitecturas para especificidad MM.

Una vez seleccionado el modelo con mejor rendimiento para nuestro conjunto de imágenes, intentamos adaptarlo de manera más efectiva a la base de datos. Para lograrlo, reemplazamos la cabeza simple, que consta de una única capa lineal que transforma las características de salida de la red a un conjunto de probabilidades del mismo tamaño que el número de clases diferenciadas en nuestro conjunto de datos, por 2 posibles configuraciones.

En primer lugar, se implementó una 'cabeza 1' basada en una capa lineal que reduce a la mitad el número de características, seguida de una función de activación no lineal y otra capa lineal que reduce los parámetros restantes al número de clases. Sin embargo, esta estructura mostró resultados considerablemente peores al ser evaluada en el conjunto de test, lo que llevó a la creación de otra cabeza con una complejidad ligeramente mayor. En este caso, se siguió la misma estructura que en el caso anterior, pero con 4 capas lineales en lugar de 2, entre las cuales se incluyeron funciones de activación ReLU. Además, se introdujeron capas de Dropout y Batch Normalization.

	Accuracy	Precisión promedio	Recall	F1
Cabeza simple	91.21	90.39	91.21	90.19
Cabeza 1	78.24	76.86	78.24	75.46
Cabeza 2	87.15	87.16	87.15	85.21

Cuadro 4.3: Comparativa de modificaciones en el modelo

	Accuracy	Precisión promedio	Recall	F1
SGD	91.21	90.39	91.21	90.19
ADAM	89.97	90.44	89.97	89.04

Cuadro 4.4: Rendimiento de ADAM VS SGD

A pesar de las diferentes estructuras que se estudiaron, se encontró que se obtuvieron mejores resultados al conservar la estructura de la cabeza simple para el modelo de clasificación, como se puede observar en el Cuadro 4.3.

Tras haber seleccionado empíricamente la arquitectura que obtiene un mejor desempeño en la tarea de clasificación de imágenes, nos queda determinar qué algoritmo de optimización, entre ADAM y SGD, proporcionará mejores resultados. En el Cuadro 4.4 se muestra el promedio de resultados de varias ejecuciones utilizando ambos algoritmos. Estos resultados demuestran que el rendimiento del modelo es muy similar para ambos algoritmos de optimización, lo que indica que cualquiera de los dos es igualmente válido para nuestro proyecto.

Durante todo el proceso de entrenamiento y selección del modelo, se consideró de manera fundamental la aplicación de técnicas de explicabilidad para garantizar que las predicciones del modelo tuvieran en cuenta el verdadero dominio del problema. Este enfoque nos permitió identificar problemas durante el entrenamiento de los modelos, como se describe en el preprocesado de datos (Sección 4.1 - filtro RandomEqualize). Un análisis rápido de los resultados obtenidos nos permitió darnos cuenta de que el modelo no era correcto.

De igual forma se aplicaron los explainers sobre el resto de modelos entrenados para obtener un análisis sobre el verdadero desempeño de los mismos. Esta información, junto con los resultados experimentales mostrados anteriormente, nos permitió concluir que el modelo con mejor rendimiento para el conjunto de test es DenseNet121 con tamaño de batch de 64, SGD como algoritmo de optimización (con el valor de learning rate establecido a $\eta = 10^{-2}$) y entropía cruzada como función de pérdida.

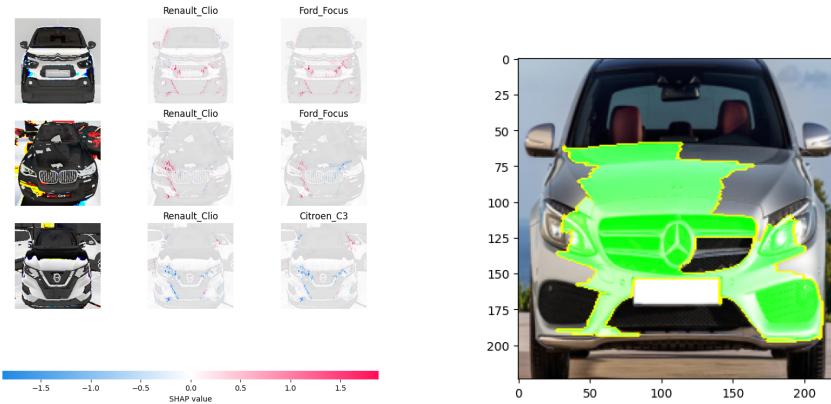


Figura 4.3: El modelo se centra en las características distintivas de los modelos, destacando como luego se verá los faros y parrilla

4.3.2. Especificidad MMA

La principal preocupación al entrenar un clasificador para la especificidad MMA es la poca variabilidad interclase que podemos ver en algunos casos debido a que las variaciones dentro de un mismo modelo entre los años son muy sutiles. Sin embargo tal y como se observa en el Cuadro 4.5 se obtiene un desempeño excelente para el conjunto de test con el modelo DenseNet101 entrenado. Además para un subconjunto de imágenes se estudiaron las 5 clases con mayor probabilidad predicha por el modelo en comparación con la etiqueta real de la instancia de entrada, y comprobamos como en los casos de error la predicción del modelo confundía dentro de un mismo modelo la pertenencia a distintas versiones o a lo sumo distintos modelos pero estimando de forma correcta la marca del mismo.

Accuracy	Precisión promedio	Recall	F1
92.36	93.20	92.36	91.70

Cuadro 4.5: Resultados del modelo DenseNet101 para especificidad MMA

Estos resultados comparados con los obtenidos para MM nos llevaron a pensar en un posible error en el modelo por el estuviera aprendiendo a diferenciar imágenes pero utilizando aspectos ajenos a las características del diseño concreto de cada modelo y año. Para comprobar si los resultados obtenidos del modelo eran reales o un mero espejismo se procedió a aplicar las técnicas de explicabilidad expuestas anteriormente. En la Figura 4.5 se puede ver el resultado obtenido que nos hace concluir que el modelo estaba aprendiendo las características del diseño y por lo tanto el clasificador es válido para nuestro problema de clasificación.

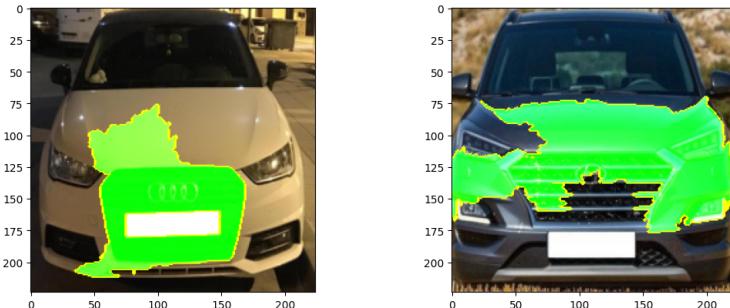


Figura 4.4: Podemos observar como el modelo se fija en los detalles de aspecto del vehículo garantizando así que las características de las que aprende son las correctas.

4.3.3. Especificidad M

Finalmente, para concluir la sección de clasificación de imágenes, se procedió a entrenar y comparar diferentes modelos de clasificación en nuestra base de datos con especificidad M. Entre los modelos utilizados se incluyeron ResNet18, ResNet50 y DenseNet121.

Al igual que en las secciones anteriores, cada uno de los modelos ha sido entrenado utilizando los mismos conjuntos de datos y parámetros de entrenamiento. Se han evaluado su rendimiento en el conjunto de test para las distintas métricas de rendimiento expuestas en la Sección 3.4 obteniéndose los resultados que se muestran en el Cuadro 4.6. Al analizar estos resultados, se puede concluir que la arquitectura DenseNet121 tiene un mejor rendimiento en términos de precisión de clasificación, al igual que para el resto de especificidades evaluadas.

Por otra parte, si bien es deseable lograr una mayor precisión en la clasificación, también es importante considerar si la relación rendimiento/coste de este modelo nos da la mayor eficiencia computacional sobre el conjunto de datos ya que tal y como se observa ResNet18 tiene un rendimiento similar.

La elección entre ambos modelos dependerá en gran medida del contexto y las aplicaciones futuras. Si se valora principalmente la eficiencia computacional, ResNet18 puede ser una opción más ligera y rápida, sin comprometer significativamente el rendimiento de clasificación. Sin embargo, considerando los recursos disponibles, tanto a nivel de hardware como temporal, y las tareas adicionales como el análisis de características distintivas de marcas y la detección de plagio (que se expondrán en la Sección 6.1), consideramos que DenseNet121 al ser una arquitectura más profunda y con mayor rendimiento brindará un mayor potencial para detección de detalles sutiles.

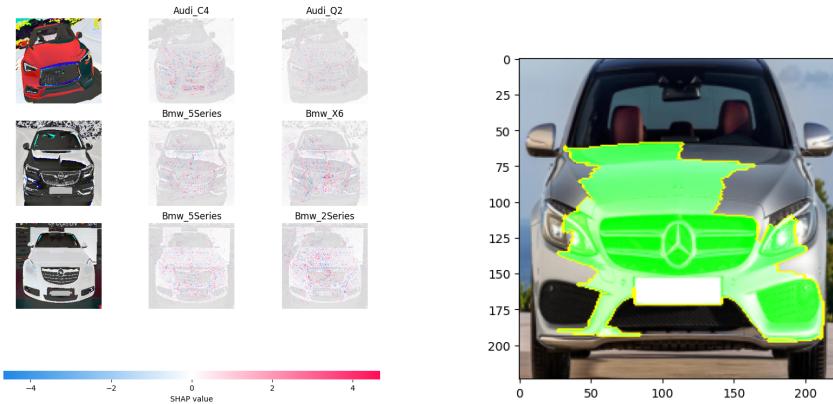


Figura 4.5: Podemos observar como el modelo se fija en los detalles de aspecto del vehículo garantizando así que las características de las que aprende son las correctas.

	Accuracy	Precisión Promedio	Recall	F1
ResNet18	97.63	97.71	97.63	97.60
ResNet50	84.55	84.19	84.55	83.36
DenseNet121	98.42	98.41	98.42	98.34

Cuadro 4.6: Métricas de error para los modelos entrenados para especificidad M

Capítulo 5

Generación Automática de imágenes para asistencia al diseño de vehículos

El otro pilar fundamental de este proyecto es la generación automática de imágenes de vehículos. Para lograr esto, utilizamos las características de vehículos de diversas marcas a través de una arquitectura GAN, lo que permite que el generador aprenda a imitar la distribución de las imágenes de entrenamiento creando imágenes sintéticas de vehículos.

En este capítulo, abordaremos la explicación del proceso para generar imágenes de vehículos de forma automática a partir de nuestro conjunto de datos inicial. Proponemos tres enfoques principales para lograr este objetivo. En la Sección 5.1, trataremos el proceso de entrenamiento de GANs básicas utilizando nuestro conjunto de datos completo y un conjunto reducido del mismo. Tras esto, exploraremos el uso de transferencia de estilos con Stylegan 3 para combinar características de distintas marcas y modelos en los vehículos generados.

Para cada uno de estos enfoques, no solo se describirá el proceso de entrenamiento de la red y los problemas que enfrentamos durante su desarrollo, sino que también se mostrarán los resultados finales e intermedios obtenidos con cada arquitectura. Además, se presentarán los resultados menos satisfactorios que demuestran las posibles imprecisiones de los distintos modelos en algunos casos. En el Anexo B se puede consultar una batería de imágenes generadas por los distintos modelos en diferentes partes de su entrenamiento.

5.0.1. Implementaciones necesarias

Debido a la naturaleza inherente de las GAN, las imágenes generadas a menudo pueden contener ruido no deseado. Esto se debe a la complejidad del proceso de generación, la falta de control directo sobre los resultados y la influencia del ruido introducido para generar variabilidad en las imágenes generadas.

Para abordar este problema y mejorar la calidad de las imágenes generadas, se implementó un script en Python que aplicaba técnicas de reducción de ruido, específicamente utilizando el filtro bilateral proporcionado por la biblioteca OpenCV¹.

El filtro bilateral es una técnica de suavizado que preserva los bordes y detalles de la imagen mientras reduce el ruido. A diferencia de otros métodos de suavizado, como el filtro gaussiano, el filtro bilateral tiene en cuenta tanto la proximidad espacial como la similitud de los valores de píxeles. Esto permite preservar los detalles importantes de la imagen mientras se suavizan las áreas con ruido.

Tal y como se puede ver en la Figura 5.1 al aplicar el filtro bilateral a las imágenes generadas, se logró reducir el ruido, mejorando significativamente la precisión y el realismo de los resultados. Las características y detalles de los vehículos generados se mantuvieron intactos, mientras que el ruido se eliminó de manera efectiva.

La implementación de este script de reducción de ruido (*Denoising.py*) puede encontrarse en la carpeta '*Auxiliary scripts*' del repositorio del proyecto².

Además, durante el proceso de entrenamiento de la arquitectura GAN, se van almacenando imágenes generadas en intervalos regulares para poder monitorizar el progreso en el entrenamiento y visualizar las mejoras en la calidad de las instancias generadas. Estas imágenes se almacenan en forma de una cuadrícula de tamaño 5x5, pudiendo mostrar un total de 25 imágenes en cada una. Esta agrupación se realiza tanto para ahorrar memoria como para facilitar la visualización de un conjunto de imágenes generadas en una misma época del entrenamiento. Al finalizar el proceso de entrenamiento, podemos analizar de manera clara si los resultados promedio obtenidos son satisfactorios desde un punto de vista cualitativo.

¹cv2.bilateralFilter - https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga9d7064d478c95d60003cf839430737ed

²<https://github.com/anariasnav/Car-design-analisis/tree/efeddef12fda221b1d43554a39fb4343fdee7e24/Auxiliary%20scripts>



(a) Imagen generada

(b) Imagen tras denoising

Figura 5.1: Imagen producida por el generador antes y después de pasar por el filtro de reducción de ruido.

Sin embargo, con el propósito de utilizar las imágenes generadas en escenarios futuros, como la clasificación de instancias generadas para determinar si pertenecen a marcas específicas y con qué grado de precisión, o para su inclusión en un nuevo conjunto de datos, es necesario dividirlas en imágenes individuales. Por lo tanto, se ha desarrollado un script *Split.py* (consultar Código fuente en la carpeta '*Auxiliary scripts*' del repositorio del proyecto³) que toma como entrada una carpeta que contiene las cuadrículas de imágenes generadas y genera como resultado un conjunto de imágenes individuales correspondientes a dichas cuadrículas.

5.1. GANs para la generación automática de imágenes de vehículos

Inicialmente, se llevaron a cabo pruebas iniciales con un tamaño de lote (batch size) de 16 y 200 épocas para verificar el correcto funcionamiento de la arquitectura. Al comparar los resultados obtenidos en estos casos de uso inicial con el análisis de explicabilidad previamente realizado para los modelos de clasificación (Figura 5.2), se puede observar cómo, a medida que el generador comienza a aprender la distribución de la muestra de entrada, adquiere conocimiento sobre características como la forma de las parrillas, los faros y la silueta de los vehículos, indicando así que estos son los elementos más distintivos entre las marcas.

³<https://github.com/anariasnav/Car-design-analisis/tree/efeddef12fda221b1d43554a39fb4343fdee7e24/Auxiliary%20scripts>

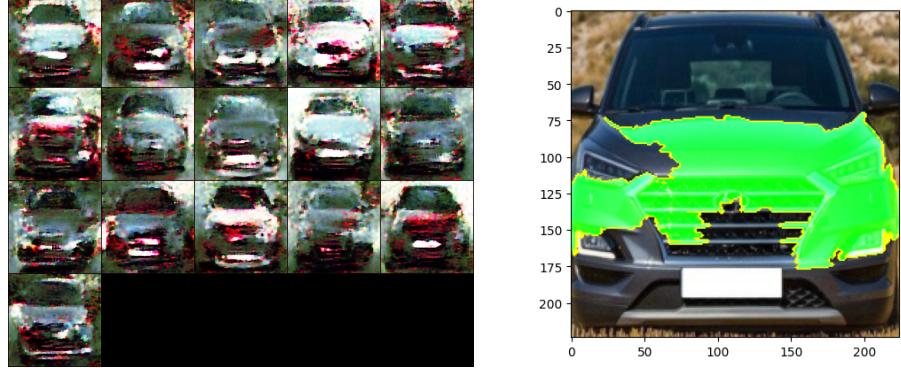


Figura 5.2: Resultados iniciales de la GAN donde se puede ver que las primeras características que aprende a imitar son aquellas de interés para el clasificador, tal y como se ve en la explicabilidad del modelo de clasificación con LIME.

Esto implica que a medida que el generador se entrena y aprende de las imágenes de entrenamiento, logra capturar y reproducir las características distintivas de los vehículos, lo que conduce a la generación de imágenes que comparten similitudes visuales con las instancias de entrada. Esta capacidad de aprendizaje del generador es crucial para lograr resultados realistas y convincentes en la generación automática de imágenes de vehículos.

Las pruebas iniciales proporcionaron una confirmación del funcionamiento adecuado de la arquitectura y sentaron las bases para continuar con el proceso de entrenamiento y mejora de la generación de imágenes de vehículos. A medida que se aumenta el tamaño del lote y se incrementan las épocas de entrenamiento, se espera que el generador continúe mejorando su capacidad para capturar aún más características relevantes y generar imágenes de mayor calidad.

A continuación, se exploró la posibilidad de utilizar técnicas de transformación en el preprocesamiento de datos, para obtener un aumento de dimensionalidad y la introducción de variedad en el conjunto de entrenamiento mediante el data augmentation. Sin embargo, tal y como se puede ver en la Figura 5.3 los resultados obtenidos utilizando estas técnicas eran cualitativamente peores en comparación con el entrenamiento sin data augmentation para el mismo número de épocas.

Teniendo en cuenta esto, se tomó la decisión de no utilizar el data augmentation en el proyecto. Si bien el data augmentation puede ser beneficioso en muchos casos para mejorar la capacidad de generalización y el rendimiento de los modelos de aprendizaje automático, en este caso particular no se consideró adecuado debido a los resultados cualitativamente peores y al aumento en el costo computacional y temporal asociado.



(a) Imágenes generadas con transformaciones incluidas en el conjunto de entrenamiento
(b) Imágenes generadas sin Data Augmentation

Figura 5.3: Comparativa de las imágenes generadas en presencia o no de data augmentation

Es importante recordar que cada proyecto y conjunto de datos son únicos, y las decisiones relacionadas con el uso de técnicas como el data augmentation deben evaluarse cuidadosamente según las necesidades y restricciones específicas de cada caso.

La elección del tamaño de batch adecuado es un aspecto importante en el entrenamiento de las redes generativas. Un tamaño de lote demasiado pequeño puede generar resultados inestables o sobreajustar el modelo, mientras que un tamaño de lote demasiado grande puede consumir más recursos computacionales y ralentizar el proceso de entrenamiento.

Para lograr una estimación correcta de este valor, se llevaron a cabo una serie de experimentos para determinar el tamaño de lote (batch size) óptimo que proporcionaría resultados más realistas en la generación de muestras utilizando la arquitectura GAN. Durante estos experimentos, se entrenó la GAN con diferentes tamaños de lote durante 200 épocas.

Tras analizar los resultados de esta experimentación, se concluyó que las muestras generadas tenían una mayor calidad y realismo cuando se utilizaba un tamaño de lote de 64 tal y como se ve en la Figura 5.4.

A medida que se aumenta el número de épocas, la GAN tiene la oportunidad de explorar y ajustar aún más la distribución de las imágenes de entrenamiento por lo que tras realizar las pruebas iniciales y obtener resultados prometedores con un tamaño de lote de 64, se decidió proceder a un entrenamiento más exhaustivo de la GAN. En este entrenamiento prolongado, se mantuvo el tamaño de lote en 64 y se aumentó el número de épocas a 1000.



Figura 5.4: Imágenes generadas para el tamaño de Batch estimado

Al prolongar el entrenamiento, se busca que la GAN continúe aprendiendo y refinando su capacidad para generar imágenes de mayor calidad y realismo. Sin embargo, tal y como se puede apreciar en la Figura 5.6, a partir de las 500 épocas (32000 iteraciones) las mejoras cualitativas que se consiguen son mínimas (Figura 5.5). De todas formas, el aumento en el número de épocas logra capturar más detalles y sutilezas en las imágenes generadas, así como lograr una mayor estabilidad y coherencia en la calidad de las muestras generadas.

Con la obtención de imágenes de calidad, que a simple vista pueden ser asociadas con marcas existentes, fue inmediato realizar un análisis de las predicciones obtenidas al pasarlas por la red de clasificación. Obteniendo de esa forma la marca a la que podrían pertenecer las imágenes generadas y el porcentaje de confianza con el que el clasificador los puede relacionar con dicha marca.

Como se puede observar en la Figura 5.8 los porcentajes de confianza en las predicciones realizadas son bajos. Esto se debe a que las imágenes generadas utilizan una distribución de probabilidad aprendida a partir de todos los datos de entrenamiento, lo que significa que los vehículos generados no presentan características distintivas de marcas individuales, sino un conjunto de características que al no ser elementos distintivos que diferencien realmente el diseño de otras marcas, ofrecen una aproximación cualitativa clave para sentar la creación de un nuevo lenguaje de diseño en estos elementos. Además, al aplicar LIME sobre las instancias generadas (Figura 5.7), nos dimos cuenta de que el clasificador identifica componentes tanto positivos como negativos para varias marcas, lo que confirma la baja probabilidad en las predicciones realizadas por el clasificador.

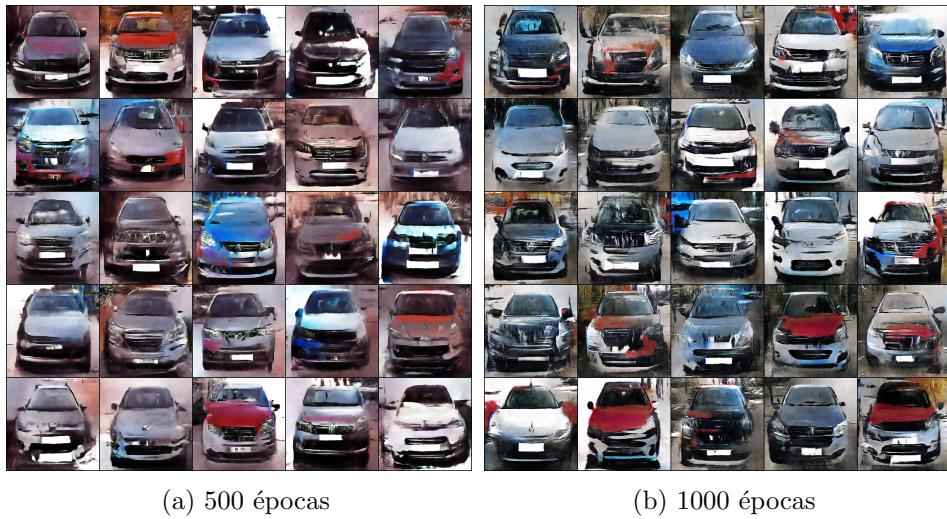


Figura 5.5: Imágenes generadas por la GAN para el conjunto de datos completo

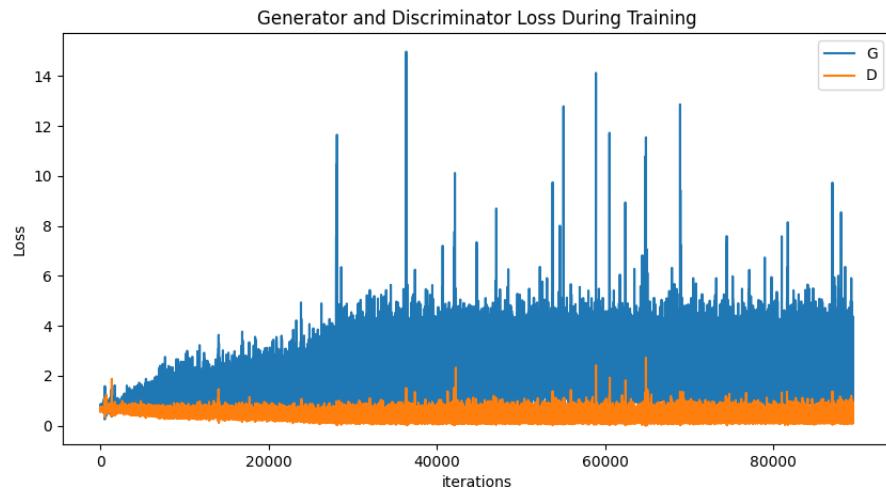


Figura 5.6: Gráfica que muestra la función de pérdida de generador y discriminador a lo largo del entrenamiento de la GAN

5.1. GANs para la generación automática de imágenes de vehículos

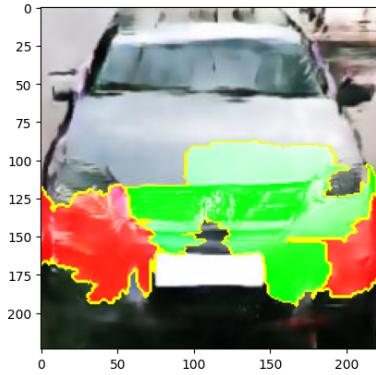


Figura 5.7: Análisis de imágenes generadas por la GAN con técnicas de explicabilidad. En la imagen podemos ver el resultado de la aplicación de LIME sobre una instancia generada donde vemos la presencia de diversas clases en el diseño del vehículo

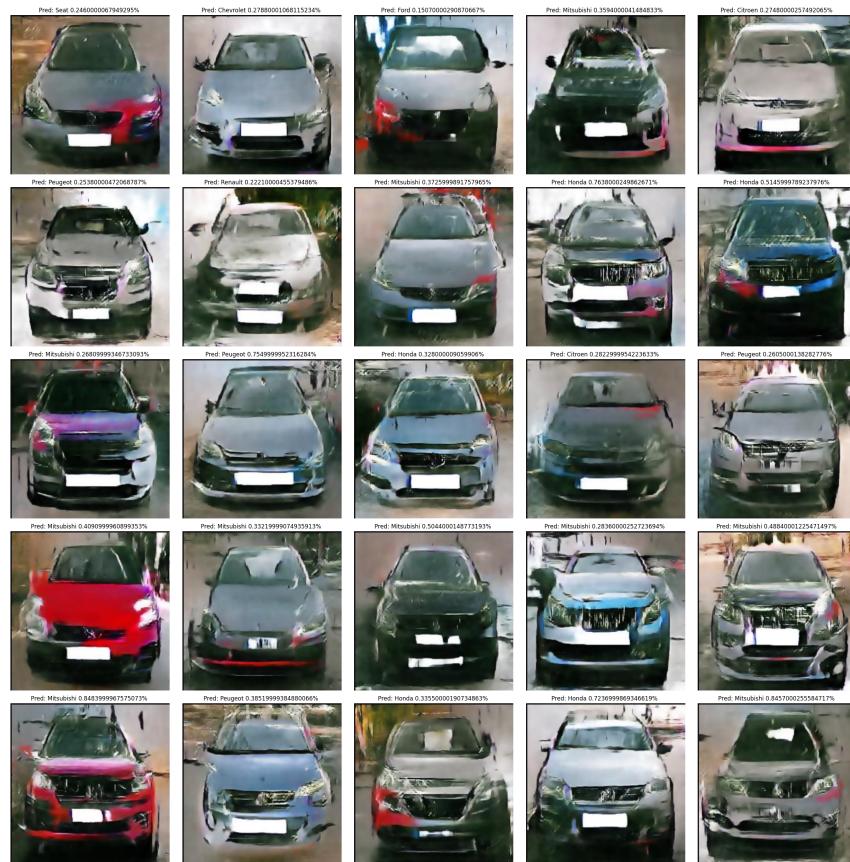


Figura 5.8: En la imagen podemos ver las imágenes generadas junto con su etiqueta inferida por el clasificador y la confianza aportada para dicha predicción.

Al realizar el entrenamiento con una base de datos que presenta una gran variedad tanto intra como interclase, las imágenes generadas siguen una distribución que combina características de todas las instancias de forma que puedan ser clasificadas como vehículos por el discriminador, pero sin mostrar características distintivas de marcas específicas.

Además, se observa que la mayoría de las imágenes generadas son clasificadas como pertenecientes a clases como Peugeot, Citroën o Honda, a pesar de tener menos instancias de su clase en la base de datos. Esto se debe a que estas marcas tienen un diseño más genérico en comparación con otras marcas como Audi, Mercedes o BMW, que tienen características más distintivas.

Los resultados obtenidos nos llevaron a intentar afinar aún más las imágenes generadas, buscando lograr resultados que, aunque novedosos, mantuvieran las características distintivas de las marcas. Para lograr esto, se planteó reducir la dimensionalidad de la base de datos utilizada en el entrenamiento. Específicamente, se llevó a cabo un entrenamiento con solo dos clases: BMW y Mercedes.

Como era de esperar, el entrenamiento de la red requirió un número mucho mayor de épocas para generar resultados con el mismo nivel de detalle y calidad que cuando se utilizó el conjunto de datos completo. A pesar de esto, los resultados de este experimento mostrados en la Figura 5.9 fueron satisfactorios, ya que se obtuvieron imágenes que se ajustaban mejor al estilo de BMW y Mercedes.



Imágenes generadas por la GAN para el conjunto reducido de imágenes BMW y Mercedes



Comparativa de uno de los modelos generados con un modelo con características similares

Figura 5.9: GAN para el conjunto reducido de imágenes de Bmw y Mercedes. A la luz de los resultados, podemos ver como no todas las imágenes son satisfactorias pero encontramos algunos resultados interesantes.

5.2. Transferencia de estilos con Stylegan

Otro enfoque que vamos a considerar para la generación automática de diseños de vehículos es la transferencia de estilos, que proporciona un mayor control sobre las imágenes generadas al combinar explícitamente características de estilo de diferentes marcas o modelos. En este enfoque, se entrena un modelo de manera que aprenda automáticamente a separar los diferentes estilos presentes en las imágenes, los cuales definen la apariencia y estética de las mismas. A continuación se pueden combinar estas características de estilo con el contenido extraído de la imagen original para generar imágenes que sean una fusión de ambos aspectos.

Los diferentes estilos a considerar hacen referencia a la profundidad de las características de estilo que se transfieren a las imágenes. A continuación se definen los diferentes niveles de estilos que podrían distinguirse en las imágenes de vehículos.

- Coarse styles (detalles 'gruesos'): Forma del frontal, paragolpes y parabrisas, posición del vehículo, espejos.
- Middle styles (detalles intermedios): Parrilla, faros y forma del capó.
- Fine styles (detalles finos): Colores, iluminación

Si bien esta arquitectura nos brinda un gran potencial para generar imágenes realistas, su alto costo computacional y temporal (para el hardware disponible, un entrenamiento adecuado nos podría llevar más de un par de meses) nos impide presentar los resultados finales de la red. Por lo tanto, hemos optado por una aproximación en la que se aplica una 'fuerza' alta a los estilos, lo que nos permite obtener imágenes con mayores variaciones, pero con el inconveniente de que algunas pueden quedar distorsionadas o rotas, como se puede apreciar en la Figura 5.10.



Figura 5.10: Imágenes erróneas generadas por Stylegan

Algunas de las imágenes generadas por esta arquitectura son extremadamente realistas, hasta el punto en que un humano no podría distinguirlas de diseños reales presentes en el mercado. Estos resultados son muy prometedores para el campo del diseño automotriz, como se explorará en el siguiente capítulo. Tal y como podemos ver en la Figura 5.11 las combinaciones de características pertenecientes a distintos modelos y marcas nos proporciona una gran variedad de diseños que, sin romper completamente con la estética de las marcas, aporta un toque novedoso a los vehículos pudiendo comprobar como quedarían los modelos con el uso de nuevos colores, formas etc



Figura 5.11: Evolución de diseño en Stylegan con distintas características

Capítulo 6

Casos prácticos

En este capítulo se abordarán las diversas aplicaciones de los modelos previamente expuestos para facilitar las tareas de diseño y análisis en el ámbito automotriz.

En primer lugar, se abordará la aplicación de explainers para la detección y análisis de las características que definen el lenguaje de diseño de una marca o modelo en particular. Además se explorará su aplicación para la detección de posibles casos de plagio entre marcas con el objetivo de preservar los derechos de autoría.

Posteriormente, se examinarán los diferentes usos de las redes generativas antagónicas (GAN) en el diseño automático de nuevos modelos de vehículos, proporcionando a los diseñadores prototipos que pueden servir como inspiración. En esta Sección se distinguirá entre el uso de las GAN para la generación automática de bocetos de diseño de vehículos y la aplicación de la transferencia de estilos para obtener diseños realistas para nuevos modelos surgidos de la combinación de características.

6.1. Aplicación práctica de explainers para análisis del lenguaje de diseño de las marcas

En esta primera Sección, se realizará un análisis de las posibles aplicaciones de la explicabilidad de modelos de clasificación en tareas de análisis del lenguaje de diseño de vehículos.

En primer lugar, se presentará un enfoque orientado al análisis de características de vehículos, mediante la extracción de características distintivas de la línea de diseño propia de una marca, así como las características de interés de un modelo concreto que goce de gran aceptación en el mercado.

En la segunda Sección, se abordará la tarea de detección de plagio entre marcas, utilizando SHAP para detectar elementos característicos de una marca que estén presentes en otras.

6.1.1. Detección de características de interés

En este apartado procederemos a representar un ejemplo de caso de uso real de la explicabilidad de modelos de cara al estudio y análisis de las características que definen la línea de diseño de una marca concreta en nuestro caso este análisis será realizado para Audi por su diseño distintivo y un número mayor de instancias para dicha clase. Dicho estudio puede ser realizado también para detectar patrones distintivos en modelos que han suscitado especial interés en los compradores haciendo uso de los clasificadores entrenados para especificidad MM.

Partimos del modelo DenseNet121, que fue entrenado en la Sección 4.3.3 para especificidad de marca, y le aplicamos la técnica de explicabilidad SHAP para generar una explicación sobre el modelo completo. Posteriormente se le pasa una batería de imágenes de la marca Audi obteniendo de una forma gráfica y clara cuales son las regiones que hacen que el clasificador diferencie que un vehículo es de esta marca y no de cualquier otra.

Esta información resulta realmente útil para un diseñador de la marca, ya que conservar la identidad distintiva de la misma en los nuevos modelos que se lanzarán al mercado garantiza una diferenciación respecto a los demás competidores y asegura mantener el estilo que atraiga a su público objetivo.

En la Figura 6.1 se muestra la distribución de los píxeles junto con sus valores de Shapley. Aquellos píxeles con valores de Shapley altos representan características que contribuyen significativamente a la predicción de que la imagen pertenezca a la clase Audi.

Específicamente, podemos observar que los píxeles con una alta contribución para Audi se concentran en la región de la parrilla, lo que nos lleva a concluir que la forma de esta es la característica más distintiva e intrínseca del diseño de los vehículos de dicha marca. De este modo, en el desarrollo de futuros modelos, se podrá optar por una parrilla que mantenga un alto grado de similitud con la forma de los modelos anteriores, garantizando así que la estética de dicho vehículo siga la línea de diseño de Audi .

Es inevitable preguntarnos ¿Es posible potenciar las regiones de interés de una marca en los modelos generativos?. Sería posible si definimos la matriz de ruido que se le pasa al generador y dejando sin ruido la región que queremos conservar. De esta forma, en el caso de Audi deberíamos dejar la zona de la parrilla sin ruido y así podríamos generar imágenes sintéticas en las que se conserve esa zona sin alterar.

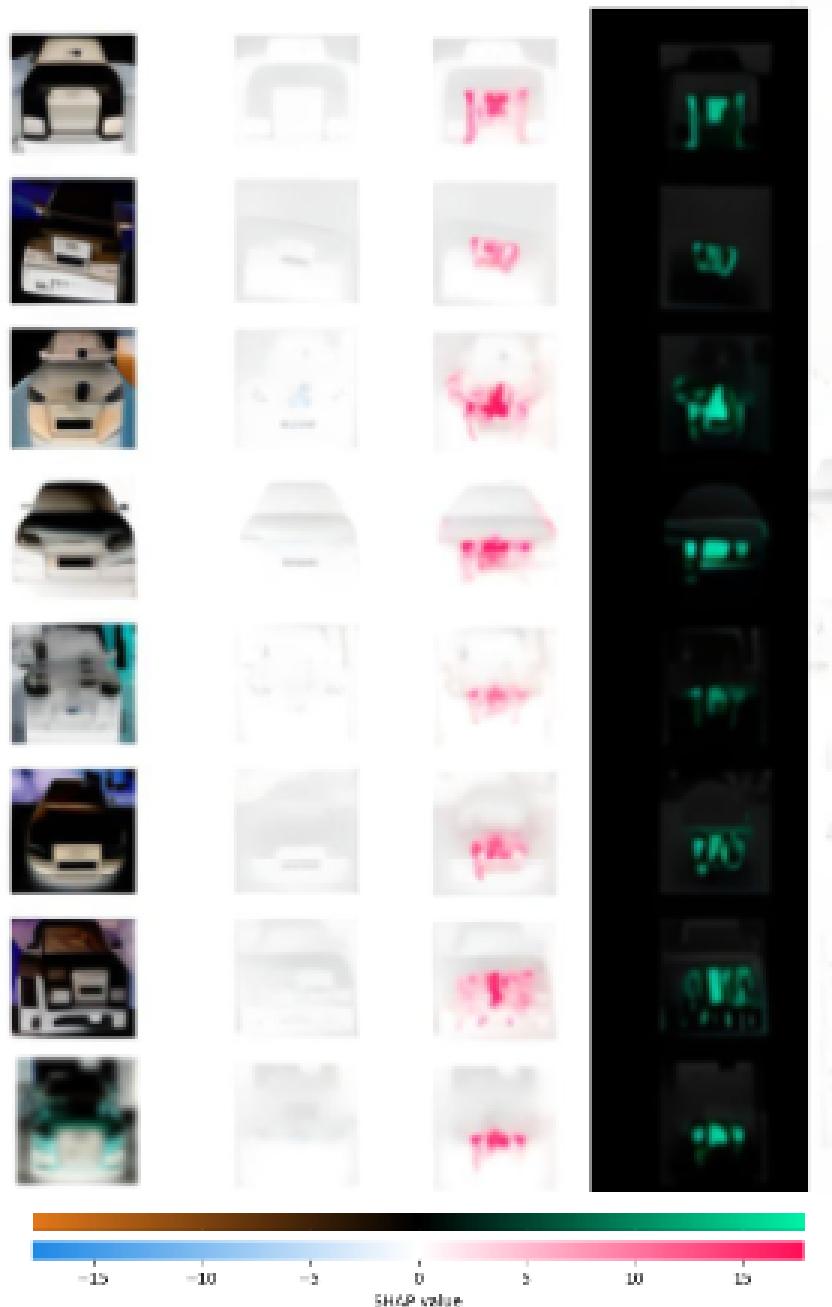


Figura 6.1: En la imagen se muestra el análisis de características realizado sobre la marca Audi mediante la aplicación de SHAP. En la primera columna, encontramos las instancias originales, mientras que en las columnas 3 y 4 se muestra la distribución de los valores de Shapley. La columna 4 corresponde a una inversión de los colores de la columna 3 para permitir una visualización más clara de la distribución obtenida.

6.1.2. Detección de plagio entre marcas

El otro gran interés de las marcas para mantener su sello distintivo y diferenciarse de las demás, es evitar que otras marcas plagien las características distintivas de sus modelos. Estas características son atractivas para los compradores, por lo que otras marcas podrían crear modelos similares a un costo inferior suponiendo una disminución en las ventas de la marca original.

Para evitar esto se proponen 2 técnicas. La primera primero consiste en realizar un análisis de las distribuciones de valores de shapley obtenidos de aplicar el explainer de SHAP al modelo de clasificación. Este análisis puede llevarse a cabo desde 2 puntos de vista distintos. Un primer enfoque que consiste en pasar la marca original al explicador, lo que nos proporcionará, para un subconjunto de imágenes de esa marca, las regiones o características que indican que la instancia pertenece a la marca y las que indican que pertenece a otra. De esta manera, obtenemos las características que podrían haber sido plagiadas y que deben ser analizadas con mayor detalle. Por otro lado, también se puede utilizar el enfoque opuesto. Al pasar una marca sospechosa al explicador, podemos identificar qué características de marcas externas tienen una gran similitud con nuestra marca y han generado incertidumbre en el clasificador.

Continuando con el ejemplo de Audi, supongamos ahora que queremos detectar posibles plagios presentes en un conjunto de 8 modelos de Audi con el resto de marcas. Para ello aplicamos SHAP explainer sobre el modelo de clasificación con especificidad de marca obteniendo los resultados mostrados en la Figura 6.2a al mostrar dicho explainer sobre el conjunto de imágenes de vehículos Audi. Podemos observar algunas similitudes leves con marcas como Citroen o Dacia, sin embargo, la más preocupante es la similitud con Hyundai (marcada en azul), donde para todos los modelos evaluados se comparten una gran cantidad de detalles. Ante este resultado, procedemos a investigar más a fondo y obtener las distribuciones de los valores de Shapley para las 2 clases con mayor probabilidad para todas las instancias tanto para el subconjunto de imágenes de Audi como para uno de Hyundai. Estas contribuciones se muestran en la Figura 6.2b, y nos indican que los modelos de Audi tienden a tener a Hyundai como la segunda probabilidad más alta y que las regiones que generan incertidumbre se encuentran principalmente en la zona del embellecedor de la parrilla y el logotipo. Esto, junto con el hecho de que para el análisis de Hyundai solo encontramos 1 instancia en la que hay alguna duda con Audi, nos lleva a pensar que efectivamente Audi estaría copiando ciertos detalles de estilo de Hyundai.

Sin embargo, debemos tener en cuenta otros factores, como el porcentaje de probabilidad que el clasificador asigna a cada clase para las instancias estudiadas. A la luz de esas probabilidades, que se muestran en el Cuadro 6.1, podemos concluir que no existe una base cualitativa firme para indicar que

Instancia	1	2	3	4	5	6	7
% Audi	0.9985	0.9986	0.977	0.9955	0.9996	0.9990	0.9992
% Hyundai	1.91×10^{-4}	3.12×10^{-4}	1.88×10^{-3}	2.77×10^{-3}	6.35×10^{-5}	1.57×10^{-4}	2.82×10^{-4}

Cuadro 6.1: Probabilidades Audi Vs Hyundai

Marca	Similitud de otras marcas con Audi			Similitud de Audi con otras marcas		
	Nº instancias	Probabilidad media	Probabilidad máxima	Nº instancias	Probabilidad media	Probabilidad Máxima
Chevrolet	1	0.0183	0.0123	0	-	-
Hyundai	0	-	-	1	0.032	0.032
Otras	0	-	-	0	-	-

Cuadro 6.2: En este cuadro se muestran las marcas ajenas junto con el número de instancias que han sido incorrectamente asignadas a Audi, así como la probabilidad estimada de pertenecer a nuestra marca calculada, como el promedio de los porcentajes que ha obtenido nuestra marca para las instancias de la clase ajena, y el mayor valor de esta. Además se muestra también el caso contrario.

hay tal plagio de características por parte de Audi, ya que las probabilidades de que las instancias pertenezcan a Hyundai, inferidas por el clasificador, son inferiores al 1 %.

La segunda técnica consiste en el uso del script *PlagiarismDetector.py* que se ha implementado (consultar Código fuente en la carpeta '*Auxiliary scripts*' del repositorio del proyecto ¹) para la detección de clases sospechosas de plagio. El script mencionado calcula el número de instancias que tienen más de un porcentaje x de probabilidad para la predicción hecha por el clasificador de pertenecer a nuestra marca, pero en realidad son de otra. Además, se identificarán las instancias de nuestra clase que han superado ese umbral de probabilidad de pertenecer a otras marcas. Esto nos proporciona información sobre la similitud de estas marcas con la nuestra y nos ayuda a identificar posibles casos de plagio o confusión.

En el Cuadro 6.2 podemos observar los resultados obtenidos de aplicar dicho script a Audi, que es la marca que estamos investigando, para un 1 % de similitud. En dicha tabla solamente se muestran aquellas marcas para las que se detecta ese porcentaje de similitud a pesar de que el script produce como salida para todas las clases. De esta manera, obtendremos un listado con las diferentes marcas y su grado de similitud con la nuestra. Para el caso de Audi encontramos que no tiene más de un 2 % de similitud con ninguna marca con lo que se puede confirmar la conclusión obtenida para el enfoque anterior.

¹<https://github.com/anariasnav/Car-design-analisis/tree/efeddef12fda221b1d43554a39fb4343fdee7e24/Auxiliary%20scripts>

6.1. Aplicación práctica de explainers para análisis del lenguaje de diseño de las marcas

74

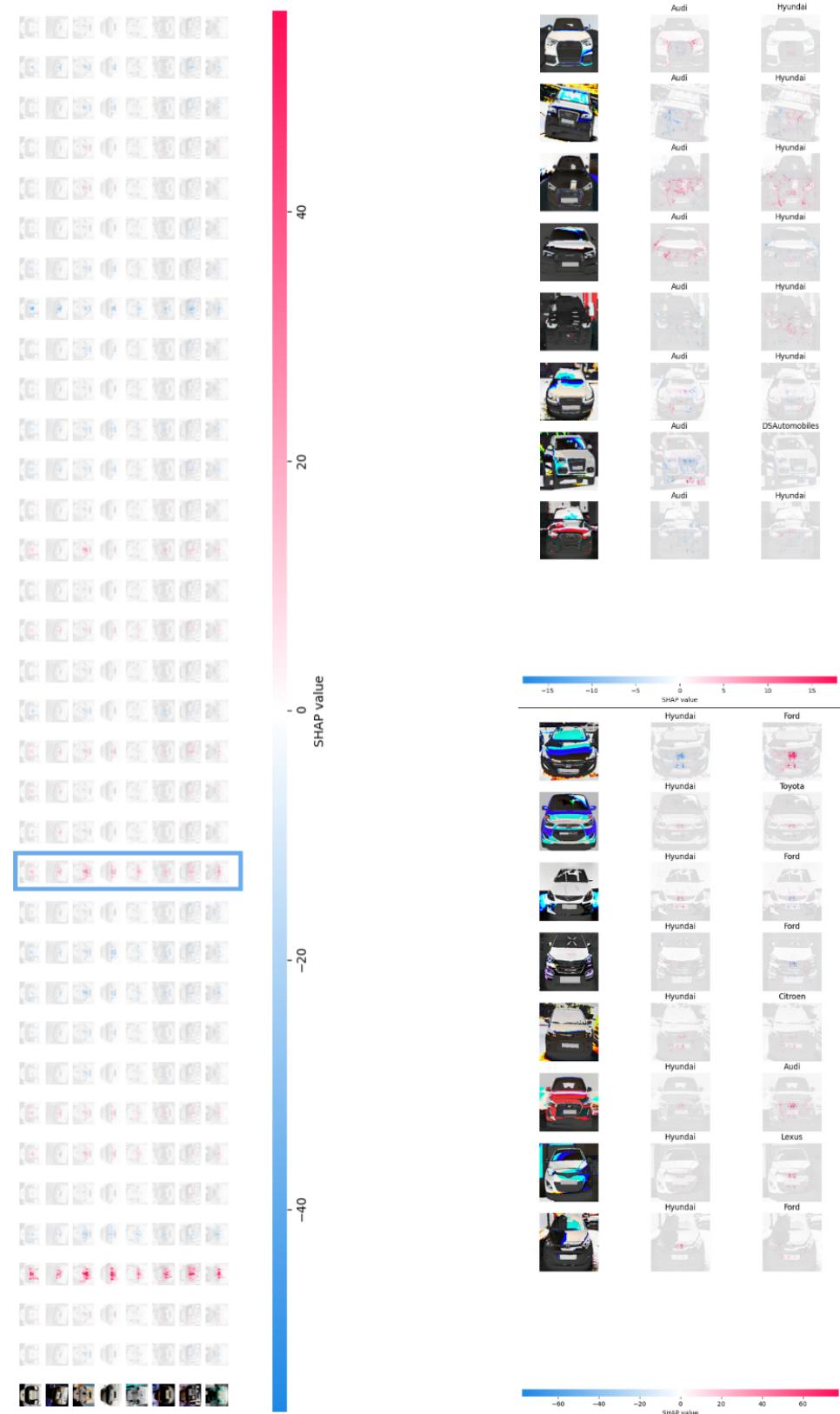


Figura 6.2: Análisis de plagio para Audi

6.2. Uso de redes de generación automática de imágenes para la asistencia al diseño automotriz

Hasta ahora se han presentado técnicas auxiliares que facilitan y contribuyen al diseño de vehículos mediante el análisis de las características de los vehículos de una marca. Estas aplicaciones son de gran interés para las marcas por la información que pueden extraer y que resulta crucial para mejorar y estabilizar la línea de diseño de la marca, así como para una gestión comercial efectiva.

Sin embargo, hasta el momento, las estrategias propuestas no tienen un impacto directo en el diseño de nuevos modelos. En esta Sección, se abordará la aplicación de las arquitecturas de generación automática de imágenes presentadas en el capítulo 5 a las tareas de diseño de modelos para una marca específica. Para ello, se distinguirá entre la generación de modelos diseños basados en la distribución de los datos de entrenamiento utilizando las GAN entrenadas y el uso de transferencia de estilos para combinar características.

6.2.1. Aplicación de arquitecturas GAN al diseño de vehículos

En los resultados experimentales del anterior capítulo pudimos comprobar como al reducir el conjunto de datos, si bien es cierto que se requiere un número mayor de épocas para generar resultados de calidad, las imágenes se ajustaban mucho mejor a las clases conservadas y se obtenían diseños mucho más cercanos a las características de la línea de diseño.

En este punto resaltaremos los 2 posibles usos de esta arquitectura de cara a la generación automática de bocetos que pudieran servir como inspiración a los diseñadores.

En el caso de marcas de nueva creación o sin un lenguaje de diseño distintivo consolidado en el tiempo, podría ser útil el uso del GAN entrenado con toda la base de datos. Así se lograrían obtener diseños que, al no seguir unas características únicas de marca, pueden romper con lo establecido hasta ahora y servir de base para el establecimiento de una línea de diseño propia.

Por otra parte, una GAN entrenada con conjuntos de datos reducidos sería más apropiada para el diseño de modelos que prosigan con una línea de diseño anterior. Estos modelos añadirán detalles nuevos o pequeñas modificaciones pero conservando las características distintivas de la marca. Es una aproximación mucho más conservadora pero aún así eficaz a la hora de generar modelos novedosos tal y como podemos observar en la Figura 6.3.



Figura 6.3: Modelos generados para BMW y Mercedes

En ambos casos, las imágenes generadas representan una serie de propuestas que pueden ser utilizadas como fuente de inspiración para el diseñador, lo que resulta en una reducción significativa tanto en tiempo como en costos. Estas imágenes generadas proporcionan ideas para el proceso de diseño, permitiendo explorar diferentes estilos y enfoques sin tener que invertir recursos adicionales en la creación de prototipos físicos o en iteraciones costosas. Por lo tanto, representan una herramienta valiosa que puede agilizar y optimizar el proceso creativo y el desarrollo de nuevos modelos.

6.2.2. Obtención de diseños por transferencia de estilos

En esta Sección, exploraremos las distintas utilidades que nos proporciona la transferencia de estilos mostrando ejemplos de los resultados obtenidos por Stylegan para cada una de ellas.

En el Anexo B se pueden ver varios ejemplos de las imágenes generadas por la arquitectura StyleGAN para diferentes combinaciones de estilo, para distintas marcas. De la misma manera en que una marca real consideraría adoptar alguno de los modelos generados como propio, se ejemplificará para un diseño de interés concreto el proceso de elección y análisis. Se realizará un breve análisis de plagio pasando el diseño por el clasificador para determinar el porcentaje de similitud con otras marcas. También se utilizarán los explainers para evaluar la intensidad de las características imitadas. De esta manera, podremos concluir sobre la viabilidad de un modelo interesante.

Por otra parte, una marca podría no solo adoptar modelos completos sino incluir colores, secciones etc Para ello se podría usar el conjunto de imágenes generado, independientemente de la instancia que se este modificando, y conservar los colores o secciones que se generan como pasos intermedios entre imágenes.



Figura 6.4: Diseños generados con interés en el color

Color

Tal y como expusimos anteriormente, dentro de la transferencia de estilos podemos diferenciar 3 niveles en función de las características que se combinan. En este primer apartado procedemos a analizar como la combinación a nivel de '*coarse styles*' puede ayudar en las tareas de diseño.

El color es uno de los aspectos visuales más destacados de un vehículo y puede ser determinante para captar la atención de los clientes potenciales. La elección de un color novedoso, atractivo y llamativo puede hacer que un vehículo destaque entre la multitud y genere interés en los compradores. Por ello la elección de los colores en los que estará disponible un modelo juega un papel crucial tanto en el proceso de diseño como en la venta del mismo. No solo afecta la atracción visual y la expresión de estilo, sino que también puede influir en la percepción de la marca y tener consideraciones prácticas y funcionales.

Por ello, la arquitectura Stylegan supone una mejora con respecto al diseño tradicional ya que nos da la posibilidad de ver una paleta de colores amplia y novedosa sobre vehículos reales permitiéndonos hacernos una idea del resultado final sin necesidad de múltiples prototipos. En la Figura 6.4 se han recogido un par de ejemplos de colores interesantes que en la actualidad no se encuentran en el mercado y que podrían suscitar el interés de los fabricantes de vehículos.



Figura 6.5: Imágenes distorsionadas

Imágenes 'Rotas'

Tal y como indicamos anteriormente, se ha optado por una alta combinación de estilos que proporciona imágenes con mayor variabilidad, pero con la consecuencia de que algunas imágenes resultan distorsionadas o 'rotas'.

Puede parecer obvio descartar estas imágenes, ya que no son realistas por lo que se reduce considerablemente su utilidad. Si bien esto es cierto, las imágenes que no están demasiado corruptas pueden aportar ideas igualmente válidas para inspirar al diseñador, por lo que se conservan y deben tenerse en consideración. En la Figura 6.5 se pueden ver algunas imágenes generadas por la arquitectura que, a pesar de ser imprecisas y tener cierto grado de distorsión, contienen regiones interesantes que podrían incluirse en modelos futuros.

Diseños completos

Cuando se usan las imágenes generadas por Stylegan para obtener diseños completos, debemos distinguir que aunque las imágenes están formadas por combinación de características a partir de una imagen de origen perteneciente a nuestra marca no tienen porque conservar patrones característicos de la misma, pudiendo generar diseños totalmente rompedores y sin relación aparente con ninguna marca en concreto. En la Figura 6.6 se muestran algunos de los modelos realistas generados por Stylegan.

Los modelos generados por esta arquitectura son, en ocasiones, hiperrealistas, como se ha podido comprobar en múltiples ejemplos. Esto podría llevar a considerar la posibilidad de utilizar directamente las imágenes generadas como diseños finales. Sin embargo, debido a la naturaleza inherente de la transferencia de estilos, antes de incorporar uno de los modelos generados por Stylegan a la línea de producción, es necesario llevar a cabo una serie de comprobaciones previas.



Figura 6.6: Diseños realistas generados por Stylegan

Mercedes	Mini	Opel	Tesla	Jaguar
0.9996	7.94×10^{-5}	3.84×10^{-5}	3.19×10^{-5}	2.79×10^{-5}

Cuadro 6.3: Probabilidad de un diseño de interés para diversas marcas

Estas comprobaciones implican realizar un análisis de plagio sobre el modelo. Dado que las imágenes generadas son una combinación de estilos tomados de diversas imágenes, no se puede asumir, incluso partiendo de una imagen de un modelo propio de la marca, que no existan similitudes con otros modelos existentes que podrían constituir un delito de plagio.

Consideremos que en esta ocasión representamos a Mercedes y nos disponemos a diseñar y lanzar al mercado un nuevo modelo. Generamos con Stylegan un conjunto de imágenes de las cuales estimamos que las que más se adaptan a nuestra línea de diseño son las que encontramos en la Figura 5.11. El primer paso será escoger, entre los posibles diseños acordes a nuestro estilo, aquel con una estimación de acogida en el mercado mejor. Tras esto podemos proceder a realizar un análisis para dicho diseño mediante su clasificación con las redes entrenadas y la evaluación de sus características.

Tal y como podemos apreciar en el Cuadro 6.3, las predicciones realizadas por el clasificador son bastante buenas para nuestra marca y sin una presencia notable de ninguna otra marca. Por tanto, podemos concluir que el modelo de Mercedes que ha suscitado nuestro interés no solo sigue la línea de diseño propia sino que el porcentaje de similitud con otras marcas es muy reducido.

Al analizar los resultados de explicabilidad obtenidos sobre la imagen del modelo de interés (Figura 6.7), podemos observar que no presenta similitudes preocupantes con otras marcas. Mantiene su diseño característico en la forma de la parrilla y el capó, pero ha sufrido modificaciones principalmente en los faros y paragolpes.

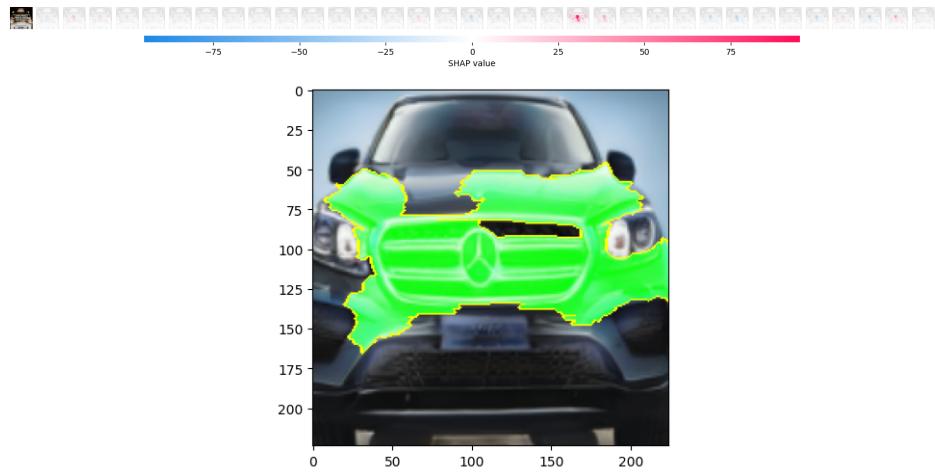


Figura 6.7: Análisis del modelo Mercedes de interés usando explainers

La distribución de probabilidades inferida por el clasificador no siempre será tan clara como en este caso, ya que puede haber ocasiones en las que se encuentren modelos con más del 40 % de probabilidad para otra clase. En estos casos, es fundamental utilizar explainers que nos ayuden a comprender las características que nos llevan a considerar una instancia como ajena. De esta forma se logrará alcanzar una verdadera comprensión de si las similitudes detectadas son características cruciales del vehículo o pequeños detalles que no provocan un verdadero problema de plagio.

Capítulo 7

Conclusiones y trabajos Futuros

El objetivo principal de este trabajo era en primera instancia la asistencia al diseño automotriz haciendo uso de algoritmos de IA. Para concluir este proyecto en este capítulo serán expuestas las conclusiones obtenidas durante el desarrollo del mismo además de las posibles mejoras futuras.

7.1. Conclusiones

En el Capítulo 6 han sido expuestas posibles aplicaciones de los modelos generativos y técnicas de explicabilidad como asistencia al diseño, satisfaciendo los objetivos 2 y 3 y logrando resultados muy prometedores para el campo del diseño automotriz. Sin embargo, al responder a la pregunta del objetivo 4: '¿Es posible que la IA sustituya, con su nivel actual, a los diseñadores de vehículos?', la respuesta es clara: todavía estamos lejos de alcanzar una independencia total en la generación automática de diseños.

La detección de imágenes corruptas, la identificación de modelos demasiado similares a los existentes, así como el refinamiento y diseño de modelos más realistas, siguen siendo tareas que requieren de la intervención humana. En este enfoque, este proyecto busca proporcionar un conjunto de herramientas auxiliares que reduzcan tanto el tiempo como los costos económicos asociados con el diseño de nuevos modelos de vehículos.

La combinación de tecnología de vanguardia con la experiencia y la visión creativa de los diseñadores tiene el potencial de impulsar la evolución y la excelencia en el diseño de vehículos, creando así una experiencia única para los usuarios y fortaleciendo la competitividad de las marcas en el mercado global.

7.2. Trabajo futuro

Si bien es cierto que los resultados del presente trabajo son muy satisfactorios, queda un largo camino por explorar que nos puede llevar a alcanzar un nivel de calidad y precisión superior.

A continuación se explican algunas de las propuestas de mejora que se proponen en la línea de investigación del proyecto.

- **Mayor calidad en el conjunto de datos a utilizar:** Aumentar el número de instancias y clases podría conseguir una identificación y análisis de las marcas más preciso. Además la inclusión de imágenes tomadas desde diferentes puntos de vista y perspectivas aportaría diversidad en las imágenes de entrenamiento, permitiendo un análisis completo del vehículo y no solo del frontal.
- **Segmentación de imágenes:** La inclusión de técnicas de segmentación de imágenes sobre los conjuntos de entrenamiento y las instancias a evaluar nos permitiría que la red de clasificación aprenda características únicamente del vehículo. Tal y como hemos comprobado durante el desarrollo de este proyecto, las características en las que se fija el clasificador no son exclusivamente del vehículo, debido a convoluciones, pooling etc, por ello segmentando el vehículo se obtendrá información independiente del contexto de cada coche aspirando así a un mejor desempeño de los modelos entrenados.
- **Profundizar en Stylegan:** Tal y como se ha comprobado, Stylegan nos proporciona modelos realistas con una gran variabilidad e interés para las marcas. Se propone en primer lugar el análisis de resultados para un entrenamiento completo de la arquitectura. Además podría ser interesante probar con otros conjuntos de datos tales como:
 - Reducción en el número de marcas para producir resultados más específicos. Por ejemplo para 2 marcas o modelos concretos.
 - Combinar el dataset de entrenamiento original con las imágenes generadas por la GAN, permitiendo combinar características de modelos reales con los bocetos obtenidos. De esta forma se podrán obtener modelos con mayor realismo pero que incluyan características novedosas que no formen parte de vehículos existentes.
- **Gan OVA:** Puede ser un enfoque interesante refinar las imágenes generadas por la GAN haciendo uso de la arquitectura de generación condicional de imágenes [28] en la que se sustituye el discriminador habitual de verdadero/falso por un clasificador One-Vs-All(GAN-OVA) con el que se puede distinguir cada dato de entrada a su etiqueta.

- **Diffusion models [3]:** Los modelos de difusión son otra de las técnicas que está suponiendo el estado del arte en técnicas de generación automática de imágenes. Si bien en este proyecto no han sido evaluadas, por su necesidad de una componente textual y su relativamente baja innovación en las imágenes generadas, puede ser fructífero realizar cierta experimentación con estos modelos que frente a las GAN producen resultados más realistas y comprobar si los resultados obtenidos innovan lo suficiente como para aplicarlos a tareas de diseño.

Bibliografía

- [1] Antonia Creswell et al. “Generative adversarial networks: An overview”. En: *IEEE signal processing magazine* 35.1 (2018), págs. 53-65.
- [2] Arun Das y Paul Rad. *Opportunities and Challenges in Explainable Artificial Intelligence (XAI): A Survey*. 2020. arXiv: 2006.11371 [cs.CV].
- [3] Prafulla Dhariwal y Alex Nichol. *Diffusion Models Beat GANs on Image Synthesis*. 2021. arXiv: 2105.05233 [cs.LG].
- [4] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].
- [5] Roberto Gozalo-Brizuela y Eduardo C. Garrido-Merchan. *ChatGPT is not all you need. A State of the Art Review of large Generative AI models*. 2023. arXiv: 2301.04655 [cs.LG].
- [6] Margherita Grandini, Enrico Bagli y Giorgio Visani. “Metrics for multi-class classification: an overview”. En: *arXiv preprint arXiv:2008.05756* (2020).
- [7] Jiuxiang Gu et al. “Recent advances in convolutional neural networks”. En: *Pattern recognition* 77 (2018), págs. 354-377.
- [8] David Gunning y David Aha. “DARPA’s explainable artificial intelligence (XAI) program”. En: *AI magazine* 40.2 (2019), págs. 44-58.
- [9] Kaiming He et al. *Deep residual learning for image recognition*. 2016.
- [10] Gao Huang et al. “Convolutional networks with dense connectivity”. En: *IEEE transactions on pattern analysis and machine intelligence* 44.12 (2019), págs. 8704-8716.
- [11] Gao Huang et al. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV].
- [12] Tero Karras, Samuli Laine y Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: 1812.04948 [cs.NE].
- [13] Tero Karras et al. *Alias-Free Generative Adversarial Networks*. 2021. arXiv: 2106.12423 [cs.CV].

- [14] Diederik P Kingma y Jimmy Ba. “Adam: A method for stochastic optimization”. En: *arXiv preprint arXiv:1412.6980* (2014).
- [15] Yann LeCun, Yoshua Bengio y Geoffrey Hinton. “Deep learning”. En: *nature* 521.7553 (2015), págs. 436-444.
- [16] Scott M Lundberg y Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. En: *Advances in Neural Information Processing Systems 30*. Ed. por I. Guyon et al. Curran Associates, Inc., 2017, págs. 4765-4774. URL: <http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf>.
- [17] Tianyu Pang et al. “Rethinking softmax cross-entropy loss for adversarial robustness”. En: *arXiv preprint arXiv:1905.10626* (2019).
- [18] Rafael Prieto Meléndez et al. “EL MODELO NEURONAL DE McCULLOCH Y PITTS Interpretación Comparativa del Modelo”. En: oct. de 2000.
- [19] Alec Radford, Luke Metz y Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].
- [20] Marco Tulio Ribeiro, Sameer Singh y Carlos Guestrin. “”Why should i trust you?”. Explaining the predictions of any classifier”. En: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, págs. 1135-1144.
- [21] Herbert Robbins y Sutton Monro. “A stochastic approximation method”. En: *The annals of mathematical statistics* (1951), págs. 400-407.
- [22] D. E. Rumelhart, G. E. Hinton y R. J. Williams. “Learning representations by back-propagating errors”. En: *Nature* 323.6088 (1986), págs. 533-536. DOI: 10.1038/323533a0.
- [23] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.
- [24] Avanti Shrikumar, Peyton Greenside y Anshul Kundaje. *Learning Important Features Through Propagating Activation Differences*. 2019. arXiv: 1704.02685 [cs.CV].
- [25] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.
- [26] W.S.McCulloch y W.Pitts. “A logical calculus of the ideas immanent in neurons activity”. En: 1943.
- [27] Eyal Winter. “The shapley value”. En: *Handbook of game theory with economic applications* 3 (2002), págs. 2025-2054.
- [28] Xiangrui Xu, Yaqin Li y Cao Yuan. *Conditional Image Generation with One-Vs-All Classifier*. 2020. arXiv: 2009.08688 [cs.CV].

- [29] Zhi-Hua Zhou. *Machine learning*. Springer Nature, 2021.

Apéndice A

Manual de usuario

En las siguientes secciones de este anexo, se desarrollará un pequeño manual de uso de los distintos scripts auxiliares implementados durante el desarrollo de este proyecto para manejar los resultados obtenidos de los modelos entrenados. Se explicará su función principal, funcionamiento y parámetros.

Los códigos fuente se pueden encontrar en el repositorio GitHub del proyecto a través de <https://github.com/anariasnav/Car-design-analisis>.

A.1. modCarpetas.py - reestructuración de la base de datos.

La base de datos en su estado original se encuentra contenida en una carpeta llamada especificidad MMA que contiene 2 subconjuntos: train y test. Sin embargo, de cara al entrenamiento con la base de datos y el uso de una estrategia de validación cruzada, hace necesaria una reestructuración de la base de datos en distintas especificidades donde cada una tenga 3 subconjuntos de datos: train, validation y test.

modCarpetas.py es un script que nos permite realizar dicha reestructuración leyendo toda la estructura de directorios y subdirectorios que 'cuelgan' de la ruta original para especificidad MMA y agrupa las clases que contengan el mismo modelo o marca. De igual forma se separa el conjunto de test en test y validación.

La estimación de la agrupación se realiza observando el nombre de la carpeta que define una clase y copiando todas las instancias de esa carpeta a otra carpeta con el nombre igual que la actual pero eliminando el año o el año y el modelo.

92 A.2. Denoising.py - reducción de ruido en imágenes generadas

Por su parte la división en train, test y val se realiza copiando la primera instancia de cada clase de test en validación, de tal forma que tendremos un número de instancias en validación igual al número de subclases de año que contenga la nueva clase.

A.2. Denoising.py - reducción de ruido en imágenes generadas

Es un script para la reducción de ruido en imágenes generadas por la GAN a través de un filtro bilateral.

Este script toma como entrada 2 parámetros:

- –inDir: Es la ruta de acceso al directorio que contiene las imágenes a limpiar por el script.
- –out: Directorio donde se almacenan las imágenes sin ruido tras haber pasado por el filtro bilateral. Su formato ha de ser ./nombre/ La ausencia del / provocará que se copien las imágenes en el directorio donde se ejecuta el script.

La implementación del código es bastante intuitiva. Se cargan los parámetros de entrada al script haciendo uso de `argparse` y se procede a crear la carpeta de salida en caso de no existir. Posteriormente se recorre el sistema de archivos desde la ruta indicada leyendo cada uno de los archivos de imagen a limpiar. Se aplica `bilateralFilter` de OpenCV¹ sobre cada imagen y posteriormente se almacenan en el directorio de salida.

A.3. Split.py - división de rejillas generadas a imágenes individuales

Ante la necesidad de separar las rejillas de conjuntos de imágenes generadas por GAN y styleGAN para poder hacer uso de las imágenes individuales que se han generado, se implementó este script que toma como parámetros la ruta al directorio que contiene las imágenes a separar (`-inDir`) y la carpeta donde habrán de almacenarse las imágenes individuales (`out`).

La implementación consiste en tras haber obtenido los parámetros y creado el directorio de salida en caso de no existir, se leen iterativamente las imágenes originales obteniendo sus dimensiones. Se calculan las dimensiones

¹cv2.bilateralFilter - https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga9d7064d478c95d60003cf839430737ed

que habrán de tener las imágenes originales para dividir la cuadrícula en n filas y m columnas y posteriormente se almacenan cada una de las imágenes individuales obtenidas de realizar `PIL.Image.crop` con las coordenadas de la subimagen como parámetros.

A.4. PlagiarismDetector.py - detección de similitudes entre marcas

Es un script para la obtención de información de similitud entre múltiples marcas haciendo uso del clasificador de marca entrenado.

Toma como parámetros la marca base sobre la que se comprobarán las similitudes (`-Marca`) por defecto es Audi y el porcentaje mínimo de probabilidad que el clasificador ha de darle a una marca para considerarlo una similitud (`-Prob`) por defecto es 0.01 (un 1%).

Inicialmente se establecen una serie de variables que son de interés para todo el programa como son el tamaño de imagen, tamaño de *batch* y directorio del que se leen las imágenes de test.

Tras esto se carga el conjunto de test, haciendo uso de *Dataloader* junto con las transformaciones básicas de redimensionado, normalización y transformación a tensor, y el modelo haciendo uso de `torch.load` y el archivo que almacena los pesos entrenados (`DenseNetM.pt`). Con esto se puede proceder a recorrer el conjunto de test calculando para cada instancia las 5 clases con mayor probabilidad y los valores de dicha probabilidad. Se comprueba si la instancia cumple:

- Es de la clase a analizar pero ha sido etiquetado como otra para una probabilidad mayor a la indicada
- Es de una clase ajena pero ha sido etiquetada como propia de la clase estudiada con una probabilidad mayor.

En cualquiera de estos 2 casos se incrementará el contador de instancias con probabilidad de plagio y se almacenará dicha probabilidad. Finalmente se calcula la probabilidad media y máxima para cada clase y se muestran los resultados por pantalla. Dichos resultados contiene información sobre: el número de instancias de cada clase que han sido etiquetadas como pertenecientes a la marca investigada, su probabilidad media y máxima de sufrir dicha clasificación, instancias que siendo de la clase base han sido etiquetadas como ajenas y su probabilidad media y máxima.

Apéndice B

Resultados obtenidos de los modelos generativos

B.1. Resultados GAN

Figura B.1: Imágenes generadas por la GAN para el conjunto completo de datos de entrenamiento



Figura B.2: Imágenes generadas por la GAN para el conjunto reducido de imágenes de Bmw

B.2. Resultados StyleGAN



Figura B.3: Posibles diseños generados a partir de una misma imagen



Figura B.4: Diseños con color de interés

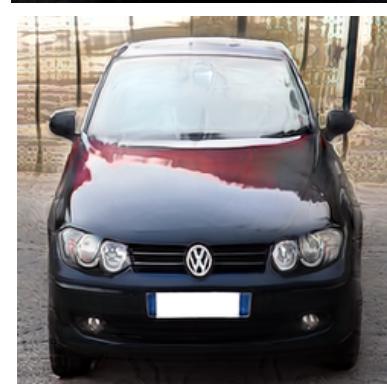
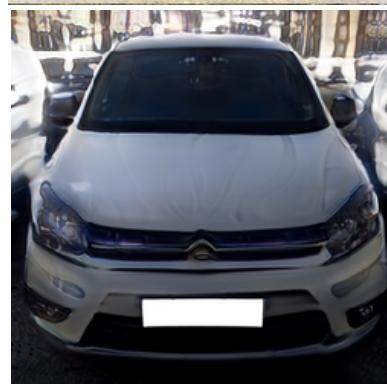




Figura B.6: Modelos de interés

