



UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

INTELIGENCIA ARTIFICIAL EXPLICABLE (XAI) EN APRENDIZAJE AUTOMÁTICO

Alumno: Juan Francisco Molina Martos

Tutor: Cristóbal José Carmona del Jesús
Cotutora: María José del Jesús Díaz
Dpto: Departamento de Informática

Septiembre, 2021



Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Cristóbal José Carmona del Jesús, tutor, y María José del Jesús Díaz, cotutora del Trabajo Fin de Grado titulado: Inteligencia artificial explicable (XAI) en aprendizaje automático, que presenta Juan Francisco Molina Martos, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2021

El alumno:

Los tutores:

CARMONA
DEL JESUS
CRISTOBAL
JOSE -
75102603J

Firmado
digitalmente por
CARMONA DEL
JESUS CRISTOBAL
JOSE - 75102603J
Fecha: 2021.09.10
17:29:45 +02'00'

DEL JESUS
DIAZ MARIA
JOSE -
26485651R

Firmado
digitalmente por
DEL JESUS DIAZ
MARIA JOSE -
26485651R
Fecha: 2021.09.10
18:48:48 +02'00'

Juan Francisco Molina Martos

Cristóbal José Carmona del Jesús y María

José del Jesús Díaz

Índice

1. INTRODUCCIÓN	10
2. ESTADO DEL ARTE	18
2.1. Deep Learning.....	18
2.1.1. Modelo de la red neuronal.....	19
2.1.2. Tipo de la red neuronal	20
2.1.3. Arquitectura de la red neuronal	23
2.1.3.1. Redes convolucionales.....	23
2.1.3.2. Autocodificadores	25
2.1.3.3. Máquinas de Boltzmann restringidas	26
2.1.3.4. Redes LSTM.....	27
2.1.3.5. Perceptrón multicapa.....	28
2.1.3.6. Redes Generativas Antagónicas.....	29
2.1.4. Validación de los modelos de Deep Learning.....	29
2.1.5. Comparación de redes neuronales y sus características.....	32
2.2. XAI	33
2.2.1. Vocabulario	33
2.2.2. Objetivos de XAI	34
2.2.3. Niveles de transparencia en modelos de Aprendizaje Automático.....	37
2.2.4. Métodos a posteriori para obtención de modelos explicables.....	38
2.2.5. Modelos de Aprendizaje Automático transparentes.....	40
2.2.6. Modelos propensos a utilizar métodos a posteriori	42
2.2.6.1. Métodos a posteriori no dependientes del modelo	42
2.2.6.2. Métodos a posteriori dependientes del modelo	43
2.3. Keras	46
2.3.1. Descripción de Keras	47
2.3.2. Modelos de Aprendizaje Profundo en Keras	49
2.3.3. XAI en Keras.....	49
3. SISTEMA	52
3.1. Estructura del proyecto	53
3.2. Modelos de Aprendizaje Profundo.....	55
3.2.1. Autoencoder.....	55
3.2.1.1. Esquema general	56
3.2.1.2. Parámetros configurables.....	56
3.2.1.3. Implementación de la estructura del modelo.....	57
3.2.2. CNN	57

3.2.2.1.	Esquema general	58
3.2.2.2.	Parámetros configurables	58
3.2.2.3.	Implementación de la estructura del modelo	59
3.2.3.	GAN	59
3.2.3.1.	Esquema general	60
3.2.3.2.	Parámetros configurables	61
3.2.3.3.	Implementación de la estructura del modelo	61
3.2.4.	LSTM	62
3.2.4.1.	Esquema general	62
3.2.4.2.	Parámetros configurables	62
3.2.4.3.	Implementación de la estructura del modelo	63
3.2.5.	MLP	63
3.2.5.1.	Esquema general	64
3.2.5.2.	Parámetros configurables	64
3.2.5.3.	Implementación de la estructura del modelo	64
3.2.6.	RNN	64
3.2.6.1.	Esquema general	65
3.2.6.2.	Parámetros configurables	65
3.2.6.3.	Implementación de la estructura del modelo	66
3.3.	Modelos de XAI	66
3.3.1.	Autoencoder y GAN	66
3.3.2.	CNN y MLP	66
3.3.2.1.	SHAP	67
3.3.2.2.	LIME	68
3.3.3.	LSTM Y RNN	68
3.3.3.1.	EEUUTwitter	69
3.3.3.2.	IMDB	70
4.	EXPERIMENTACIÓN	71
4.1.	Datos	72
4.2.	Resultados	73
4.2.1.	Autoencoder	73
4.2.1.1.	Elementos relevantes en la salida por consola	73
4.2.1.2.	Ejecución sin XAI	74
4.2.2.	CNN	74
4.2.2.1.	Elementos relevantes en la salida por consola	74
4.2.2.2.	Ejecución sin XAI	74

4.2.2.3.	Ejecución con SHAP	75
4.2.2.4.	Ejecución con LIME	76
4.2.3.	GAN	76
4.2.3.1.	Elementos relevantes en la salida por consola	77
4.2.3.2.	Ejecución sin XAI	77
4.2.4.	LSTM	77
4.2.4.1.	Elementos relevantes en la salida por consola	78
4.2.4.2.	Ejecución sin XAI	78
4.2.4.3.	Ejecución con XAI	79
4.2.4.3.1.	IMDB	80
4.2.4.3.2.	EEUUTwitter	81
4.2.5.	MLP	82
4.2.5.1.	Elementos relevantes en la salida por consola	82
4.2.5.2.	Ejecución sin XAI	82
4.2.5.3.	Ejecución con SHAP	83
4.2.5.4.	Ejecución con LIME	84
4.2.6.	RNN	84
4.2.6.1.	Elementos relevantes en la salida por consola	84
4.2.6.2.	Ejecución sin XAI	85
4.2.6.3.	Ejecución con XAI	86
4.2.6.3.1.	IMDB	87
4.2.6.3.2.	EEUUTwitter	88
5.	ANÁLISIS DE RESULTADOS	89
5.1.	Clasificación de imágenes	89
5.2.	Clasificación de textos	90
6.	CONCLUSIONES	93
	ANEXO I: GUÍA DE INSTALACIÓN	95
	Python	95
	PyCharm	97
	CUDA	98
	Keras	102
	Librerías de XAI	102
	SHAP	102
	Lime	102
	ANEXO II: CÓDIGO, EJECUCIÓN Y SALIDA DE LOS MODELOS DE APRENDIZAJE AUTOMÁTICO Y MÉTODOS DE XAI	103

Autoencoder	103
Código.....	103
Salida por consola.....	106
Imágenes obtenidas	120
CNN.....	120
Código.....	120
Salida por consola.....	127
Imágenes obtenidas	133
GAN.....	134
Código.....	134
Salida por consola.....	140
Imágenes obtenidas	144
LSTM.....	144
Código.....	144
Salida por consola.....	151
Imágenes obtenidas	157
MLP	160
Código.....	160
Salida por consola.....	166
Imágenes obtenidas	172
RNN.....	173
Código.....	173
Salida por consola.....	180
Imágenes obtenidas	187
Bibliografía	189

Índice de ilustraciones

Ilustración 1. Aumento de los datos anuales.	10
Ilustración 2. Fases de la preparación de los datos.	11
Ilustración 3. Fases de la reducción de la dimensionalidad de los datos.	12
Ilustración 4. Tipos de técnicas de minería web.	13
Ilustración 5. Publicaciones de IA en revistas.	15
Ilustración 6. Inversión privada en IA.	15
Ilustración 7. Aplicaciones del Aprendizaje Profundo.	16
Ilustración 8. Esquema de una red neuronal recurrente.	21
Ilustración 9. Esquema de una red neuronal de base radial.	21
Ilustración 10. Esquema de una red de Kohonen.	22
Ilustración 11. Ejemplo de red convolucional.	24
Ilustración 12. Procedimiento de un autocodificador.	25
Ilustración 13. Estructura de una máquina de Boltzmann restringida.	26
Ilustración 14. Estructura de una red LSTM.	28
Ilustración 15. Esquema de un perceptrón multicapa.	28
Ilustración 16. Esquema de una red generativa antagónica (GAN).	29
Ilustración 17. Posibles niveles de transparencia en un modelo transparente.	38
Ilustración 18. Métodos a posteriori para obtención de modelos explicables.	40
Ilustración 19. Diferentes métodos explicativos a posteriori en Redes Neuronales Convolucionales.	45
Ilustración 20. Impacto de Keras en múltiples ámbitos en comparación con PyTorch.	48
Ilustración 21. Muestra de la predicción chow con Grad-CAM.	50
Ilustración 22. Muestra de la predicción gato egipcio con Grad-CAM.	50
Ilustración 23. Método de XAI en AI Platform.	51
Ilustración 24. Ilustración del proyecto.	54
Ilustración 25. Esquema general de Autoencoder.	56
Ilustración 26. Esquema general de CNN.	58
Ilustración 27. Esquema general de GAN.	60
Ilustración 28. Esquema general de LSTM.	62
Ilustración 29. Esquema general de MLP.	64
Ilustración 30. Esquema general de RNN.	65
Ilustración 31. Proceso de KDD.	71
Ilustración 32. Ejecución de Autoencoder con MNIST sin XAI.	74
Ilustración 33. Ejecución de Autoencoder con Fashion MNIST sin XAI.	74
Ilustración 34. Ejecución de CNN con MNIST sin XAI.	75
Ilustración 35. Ejecución de CNN con Fashion MNIST sin XAI.	75
Ilustración 36. Ejecución de CNN con MNIST usando SHAP.	75
Ilustración 37. Ejecución de CNN con Fashion MNIST usando SHAP.	76
Ilustración 38. Ejecución de CNN con MNIST usando LIME.	76
Ilustración 39. Ejecución de CNN con Fashion MNIST usando LIME.	76
Ilustración 40. Ejecución de GAN con MNIST sin XAI.	77
Ilustración 41. Ejecución de GAN con Fashion MNIST sin XAI.	77
Ilustración 42. Ejecución de LSTM con IMDB aplicando SHAP y LIME.	80
Ilustración 43. Ejecución de LSTM con EEUU Twitter aplicando SHAP y LIME.	81
Ilustración 44. Ejecución de MLP con MNIST sin XAI.	82
Ilustración 45. Ejecución de MLP con Fashion MNIST sin XAI.	83

Ilustración 46. Ejecución de MLP con MNIST usando SHAP.....	83
Ilustración 47. Ejecución de MLP con Fashion MNIST usando SHAP.	83
Ilustración 48. Ejecución de MLP con MNIST usando LIME.	84
Ilustración 49. Ejecución de MLP con Fashion MNIST usando LIME.....	84
Ilustración 50. Ejecución de RNN con IMDB aplicando SHAP y LIME.	87
Ilustración 51. Ejecución de RNN con EEUUTwitter aplicando SHAP y LIME.	88
Ilustración 52. 1º análisis de SHAP en imágenes.	89
Ilustración 53. 1º análisis de LIME en imágenes.....	89
Ilustración 54. 2º análisis de SHAP en imágenes.	90
Ilustración 55. 2º análisis de LIME en imágenes.....	90
Ilustración 56. 1º análisis de SHAP y LIME en textos.	91
Ilustración 57. 2º análisis de SHAP y LIME en textos.	92
Ilustración 58. 1º paso para instalación de Python.....	95
Ilustración 59. 2º paso para instalación de Python.....	96
Ilustración 60. 3º paso para instalación de Python.....	96
Ilustración 61. Último paso para instalación de Python.....	97
Ilustración 62. 1º paso para instalación de PyCharm.....	97
Ilustración 63. 2º paso para instalación de PyCharm.....	98
Ilustración 64. 1º paso para instalación de CUDA.	99
Ilustración 65. 2º paso para la instalación de CUDA.....	99
Ilustración 66. 3º paso para la instalación de CUDA.....	100
Ilustración 67. 4º paso para la instalación de CUDA.....	100
Ilustración 68. 5º paso para la instalación de CUDA.....	101

Índice de tablas

Tabla 1. Comparación de redes neuronales y sus características.	33
Tabla 2. Impacto de los objetivos de XAI.....	37
Tabla 3. Descripción de modelos transparentes dependiendo de su grado de transparencia.	41
Tabla 4. Comparación entre SHAP y LIME.....	93

1. INTRODUCCIÓN

En el contexto tecnológico actual, la generación de una inmensa cantidad de datos por parte de una amplia variedad de dispositivos (internet de las cosas, teléfonos móviles...) es una realidad latente. El empleo de este flujo masivo de datos es tratado en el campo de la ciencia de datos, campo interdisciplinar (cuyas disciplinas principales son estadística e informática) que extrae conocimiento de datos en diversas condiciones (estructurados o no estructurados, con un bajo o alto ruido en la información que contienen...) para ser utilizado en un amplio rango de aplicaciones (finanzas, medicina, ciudades inteligentes...) (Dhar, 2013). La tendencia al aumento del uso de datos se refleja en esta ilustración, en la que se realiza una estimación del aumento de los datos utilizados mundialmente hasta 2025:



Ilustración 1. Aumento de los datos anuales.

Adaptado de Figura 1, Reinsel, D., Gantz, J., & Rydning, J. (2018). The Digitization of the World From Edge to Core.

La Ciencia de datos, aunque presente con mucha fuerza en la actualidad, comenzó a gestarse mucho antes, acuñándose el término en la década de los 70 (Naur, 1974). Sin embargo, la irrupción del *big data*, que se puede definir como el conjunto de datos cuyo tamaño supera considerablemente la capacidad de captura, almacenado, gestión y análisis del software convencional de bases de datos (J. García et al., 2018), ha dotado a la ciencia de datos de una popularidad destacable en los últimos años.

La primera característica a destacar de la ciencia de datos, que en diversas ocasiones no se le da la importancia necesaria, es el preprocesamiento de los datos.

Este preprocesamiento de los datos permite emplear correctamente y obtener el mejor rendimiento de las diversas técnicas de ciencia de datos que se mostrarán a continuación, por lo que su relevancia es palpable dentro del campo.

A grandes rasgos, existen dos pasos en el preprocesamiento de los datos, que son la preparación de los datos y la reducción de dimensionalidad de los datos. Para sintetizar las diferentes fases de ambas etapas, primeramente, se mostrarán las fases de la preparación de los datos y, posteriormente, se mostrarán las fases de la reducción de dimensionalidad de los datos (S. García et al., 2015):

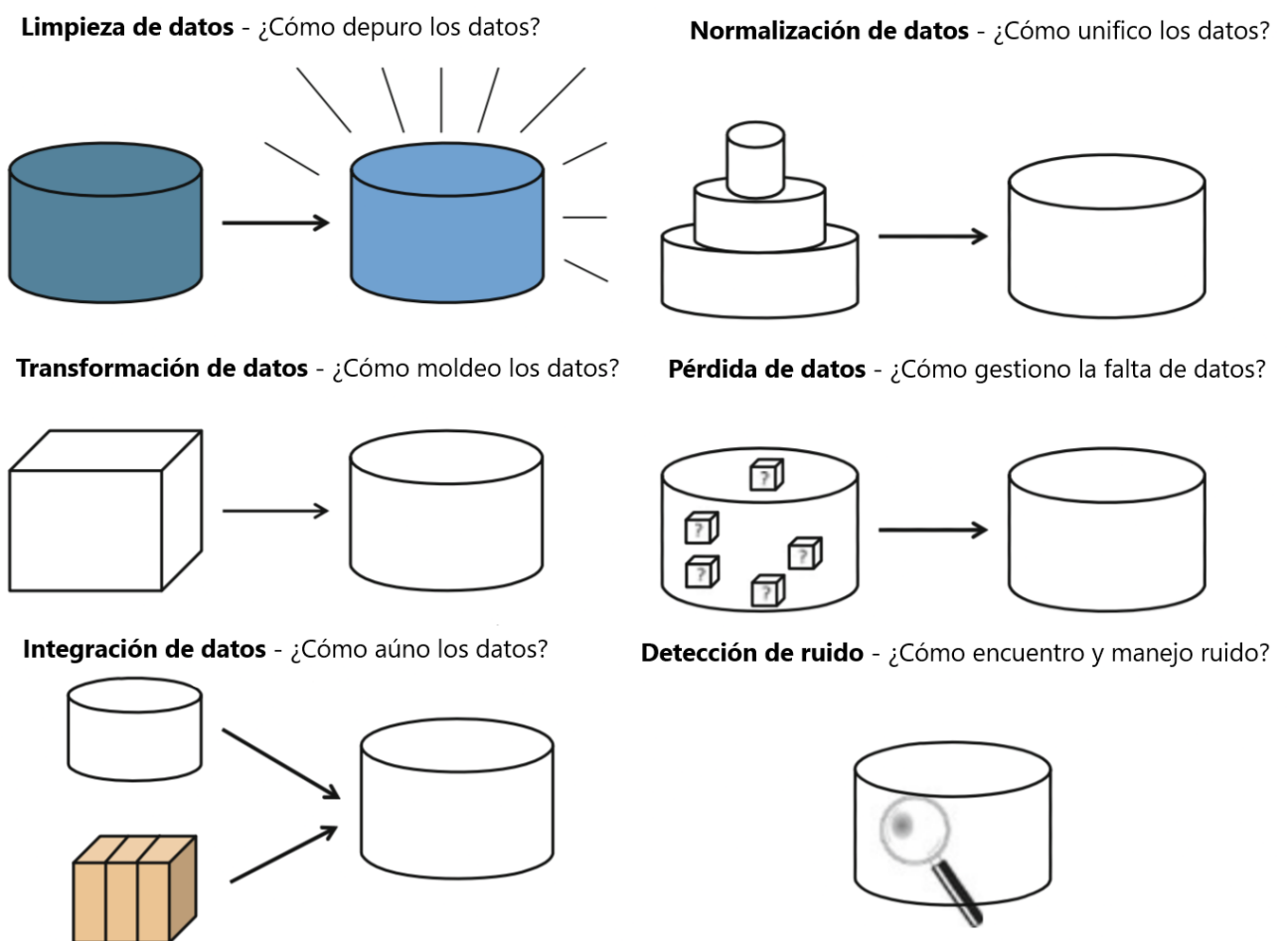


Ilustración 2. Fases de la preparación de los datos.

Adaptado de Figura 1.3, García, S., Luengo, J., & Herrera, F. (2015). Data Preprocessing in Data Mining. En Intelligent Systems Reference Library (Vol. 72). Springer International Publishing.

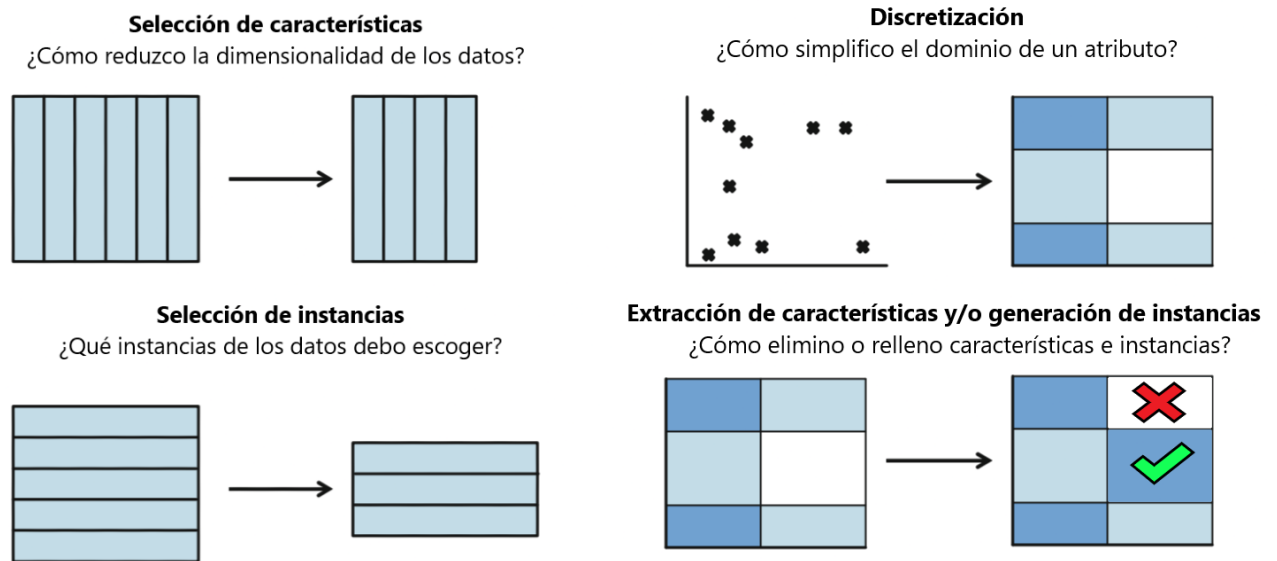


Ilustración 3. Fases de la reducción de la dimensionalidad de los datos.

Adaptado de Figura 1.4, García, S., Luengo, J., & Herrera, F. (2015). Data Preprocessing in Data Mining. En Intelligent Systems Reference Library (Vol. 72). Springer International Publishing.

Una vez el preprocesamiento ha sido abordado, cabe destacar que existen dos tipos de aprendizaje en ciencia de datos (Pérez, 2021):

- Aprendizaje supervisado: Cada dato contiene el valor de la clase a la que corresponde.
- Aprendizaje no supervisado: Cada dato no contiene el valor de la clase a la que corresponde.

Es necesario indicar que, dentro del campo de la ciencia de datos, existe un aspecto importante que debe ser tratado, conocido como minería de datos. Esta minería de datos puede definirse como el conjunto de técnicas y herramientas aplicadas al proceso trivial de extraer y presentar el conocimiento implícito, previamente desconocido, potencialmente útil y humanamente comprensible, a partir de grandes conjuntos de datos, con el objeto de predecir de forma autorizada tendencias y comportamientos y/o descubrir de forma automatizada modelos previamente desconocidos (Frawley et al., 1992). Esta minería de datos está formada por un amplio compendio de técnicas, que quedan sintetizadas en la siguiente imagen:

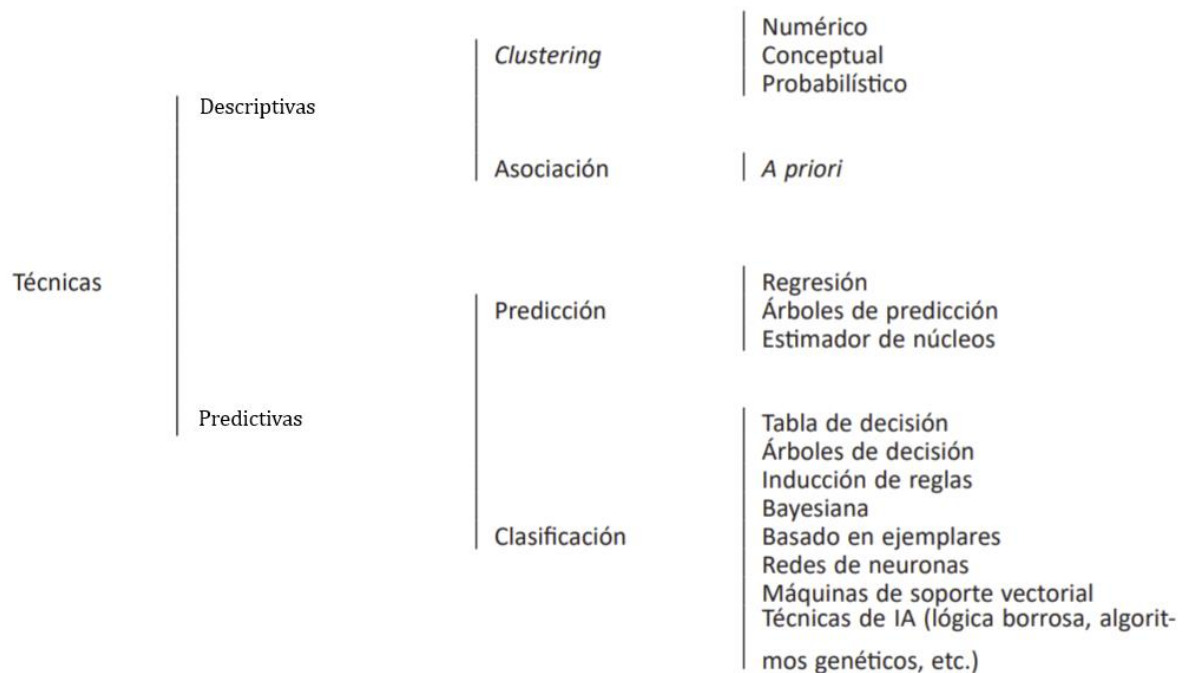


Ilustración 4. Tipos de técnicas de minería web.

Adaptado de Fig 6.1, García, J., Molina, J., Berlanga, A., Bustamante, Á., Padilla, W., & Patricio, M. (2018). Ciencia de datos, Técnicas analíticas y aprendizaje estadístico. www.alfaomega.com.co.

De cara a profundizar en este tipo de técnicas, cabe incidir en la clasificación entre técnicas descriptivas y técnicas predictivas (J. García et al., 2018):

- **Técnicas descriptivas:** El objetivo de estas técnicas es obtener un modelo que permita hallar patrones o resumir los datos, es decir, no pretender predecir nuevos datos a partir de la información recabada. En este tipo de técnicas, se suele utilizar principalmente aprendizaje no supervisado.
- **Técnicas predictivas:** El objetivo de estas técnicas es obtener un modelo que permita aproximar posibles valores a partir de los datos de los que ya se dispone. En este tipo de técnicas, se suele utilizar principalmente aprendizaje supervisado.

En las técnicas descriptivas, se encuentran principalmente 2 posibilidades:

- Clustering: El proceso de clustering puede ser visto como la identificación de clases similares de datos. Mediante el clustering, se puede identificar regiones en el espacio de datos con las mismas características, encontrando patrones, correlaciones... En ocasiones, este tipo de técnica es utilizada para realizar un preprocesamiento del conjunto de datos y obtener un subconjunto de datos que luego será utilizado en una técnica predictiva de clasificación (Rahman, 2018).
- Asociación: Esta técnica busca encontrar reglas y conexiones en grandes conjuntos de datos. Normalmente, se encuentran una gran cantidad de asociaciones, aunque la mayoría carecen de valor, por lo que se deben seleccionar aquellas que aporten información de calidad (Rahman, 2018).

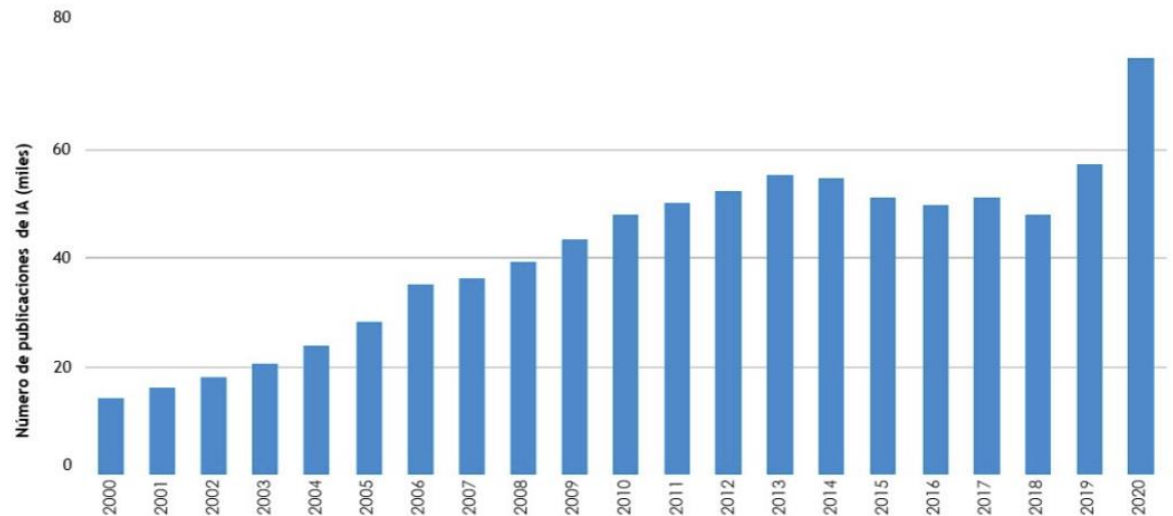
En el caso de las técnicas predictivas, se encuentran principalmente 2 posibilidades:

- Predicción: Esta técnica intenta determinar los valores de una o varias variables a partir de un conjunto de datos. Problemas más sencillos pueden abordarse mediante regresión lineal, mientras que problemas más complejos necesitarán árboles de predicción o estimador de núcleos (J. García et al., 2018).
- Clasificación: Esta técnica es la más popular del conjunto de técnicas de minería de datos. Consiste en definir una serie de clases, de forma que el modelo creado a partir del conjunto de datos de entrada indicará a que clase pertenece un nuevo dato. Una de las vertientes más destacadas es el Aprendizaje Profundo o *Deep Learning*.

El *Deep Learning* (Dechter, 1986), o Aprendizaje Profundo, ha sufrido un crecimiento extraordinario en estos últimos años. Esta tendencia queda reflejada en las siguientes imágenes, que sugieren un aumento considerable en áreas tan importantes como investigación o inversión económica en el campo de la Inteligencia Artificial, en el cual el Aprendizaje Profundo desempeña un gran papel:

NÚMERO DE PUBLICACIONES DE IA EN REVISTAS, 2000-20

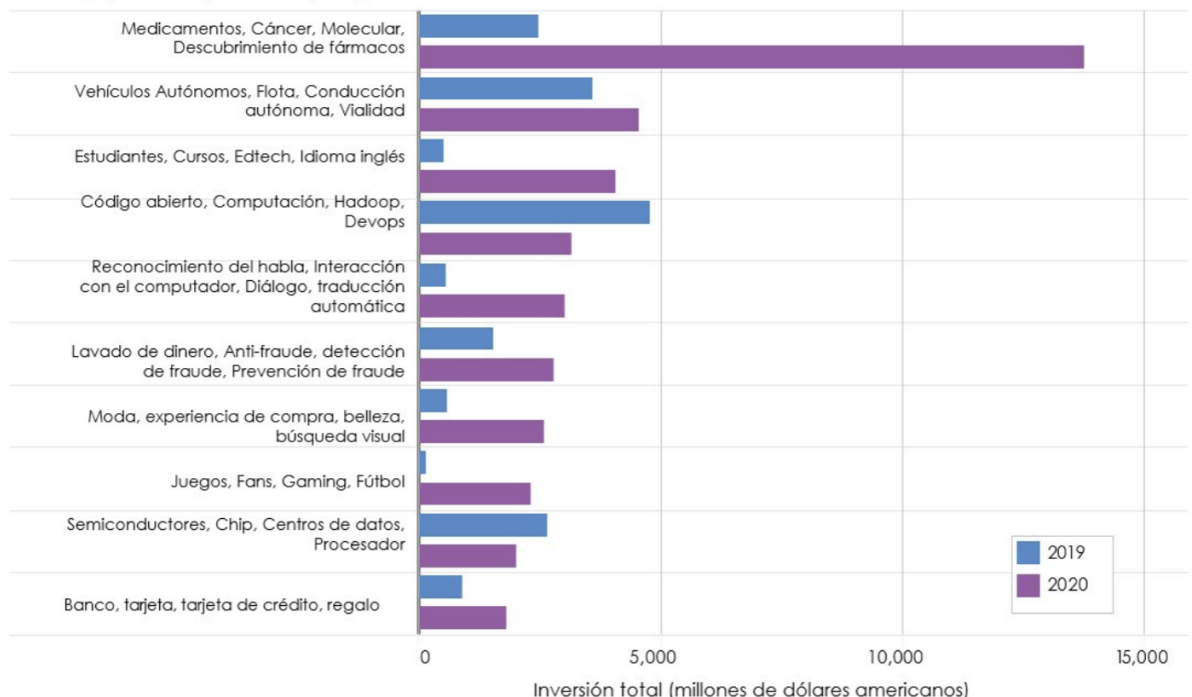
Fuente: Microsoft Academic Graph, 2020 | Gráfica: Informe AI Index 2021

**Ilustración 5. Publicaciones de IA en revistas.**

Adaptado de Figura 1.1.1a, Daniel Zhang, Saurabh Mishra, Erik Brynjolfsson, John Etchemendy, Deep Ganguli, Barbara Grosz, Terah Lyons, James Manyika, Juan Carlos Niebles, Michael Sellitto, Yoav Shoham, Jack Clark, and Raymond Perrault, "The AI Index 2021 Annual Report," Junta Directiva del AI Index, Human-Centered AI Institute, Universidad de Stanford, Stanford, California, Marzo 2021. Traducción al castellano por el Instituto de Inteligencia Artificial.

INVERSIÓN PRIVADA GLOBAL en IA por ÁREA PRIORITARIA, 2019 vs 2020

Fuente: CapIQ, Crunchbase y NetBase Quid, 2020 | Gráfica: Informe AI Index 2021

**Ilustración 6. Inversión privada en IA.**

Adaptado de Figura 3.2.6, Daniel Zhang, Saurabh Mishra, Erik Brynjolfsson, John Etchemendy, Deep Ganguli, Barbara Grosz, Terah Lyons, James Manyika, Juan Carlos Niebles, Michael Sellitto, Yoav Shoham, Jack Clark,

and Raymond Perrault, "The AI Index 2021 Annual Report," Junta Directiva del AI Index, Human-Centered AI Institute, Universidad de Stanford, Stanford, California, Marzo 2021. Traducción al castellano por el Instituto de Inteligencia Artificial.

El sector del *Deep Learning* juega un papel importante en la sociedad actual, debido a que es utilizado en diversas áreas de gran impacto en la población, como medicina o reconocimiento de voz. A continuación, se muestra una ilustración que presenta una parte de la pluralidad temática que aborda el Aprendizaje Profundo:

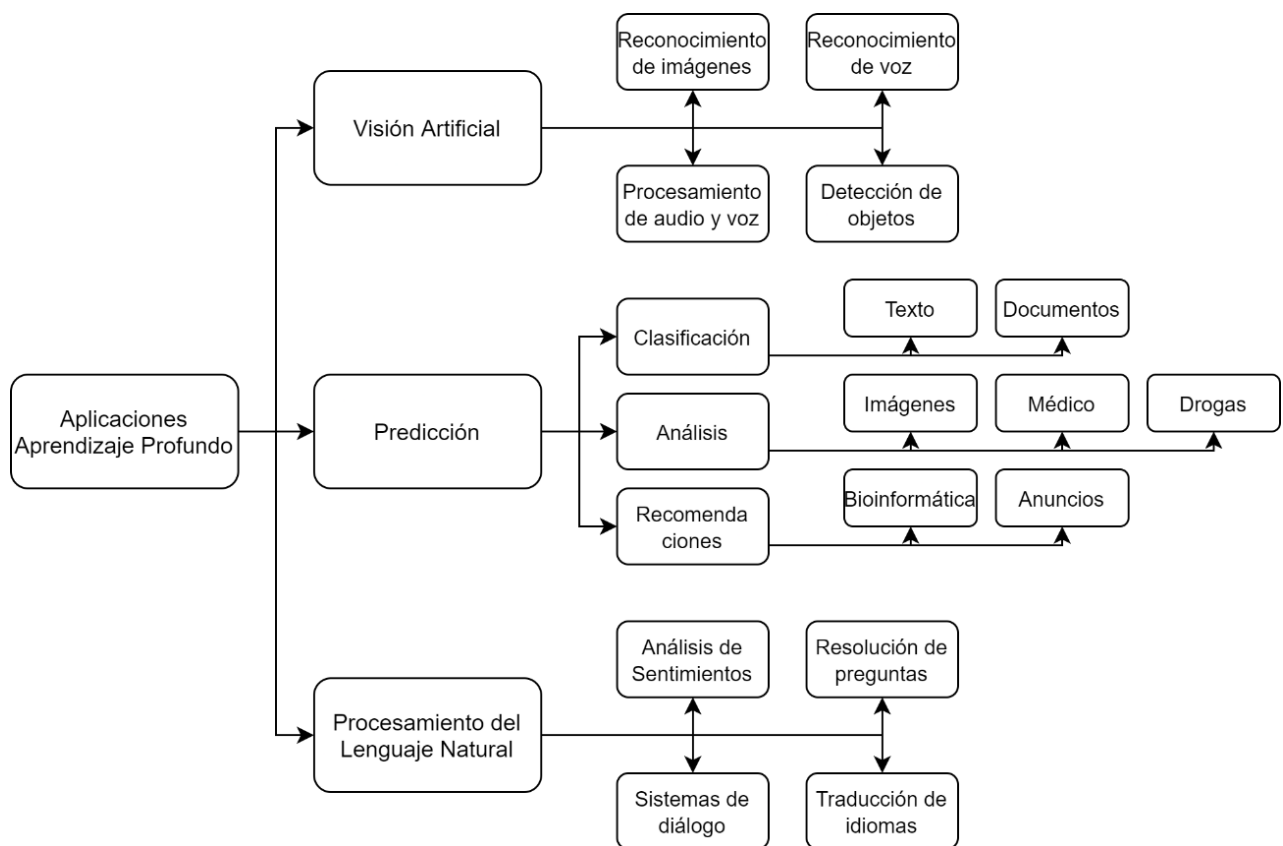


Ilustración 7. Aplicaciones del Aprendizaje Profundo.

No obstante, no todo es luz en el campo de *Deep Learning*. Uno de los factores que atenúa las bondades del Aprendizaje Profundo es su naturaleza opaca. Los modelos de *Deep Learning* actuales, como las redes neuronales profundas, son complejos modelos de caja negra, debido a los cientos de capas y millones de parámetros que los conforman, por lo que no se puede comprender el procedimiento exacto por lo que el sistema ha obtenido un resultado u otro. Esta falta de transparencia, unida a la desconfianza innata del ser humano en aquello que no se puede interpretar o controlar, dificulta la entrada de estos sistemas en campos

donde se necesite respaldar el resultado obtenido, como medicina, finanzas o vehículos autónomos (Barredo Arrieta et al., 2020).

XAI (Explainable Artificial Intelligence) (Gunning et al., 2019) se postula para lidiar con la situación expuesta previamente, proponiendo crear una serie de técnicas que:

- Permitan producir modelos más explicables o técnicas que permitan a modelos no explicables serlo, manteniendo en ambos casos un buen rendimiento.
- Permita a los humanos entender, gestionar y confiar en los últimos modelos de Inteligencia Artificial.

1.1. Objetivos

Este Trabajo de Fin de Grado tiene los siguientes objetivos:

- Estudiar el área de XAI en la actualidad, a partir de la revisión de la bibliografía.
- Obtener métodos que permitan hacer más explicables los modelos de *Deep Learning*.
- Aplicación de los métodos obtenidos para analizar los resultados.
- Extracción de conclusiones, describiendo las ventajas y desventajas de los métodos y mostrando las líneas de trabajo abiertas.

2. ESTADO DEL ARTE

2.1. Deep Learning

Una de las arquitecturas más frecuentes en el Aprendizaje Profundo son las redes neuronales. Estas redes neuronales, inspiradas en el funcionamiento del cerebro humano, consisten en el aglutinamiento y conexión de un número determinado de capas formadas por una unidad mínima denominada neurona (aquellas redes con un número alto de capas son conocidas como redes neuronales profundas). Las conexiones se realizan entre neuronas de las diferentes capas que conforman la red neuronal, y el conocimiento de la red neuronal reside en estas. (Shrestha & Mahmood, 2019)

El conocimiento de una red neuronal es producto de un proceso de entrenamiento, existiendo, a grandes rasgos, tres posibilidades (Shrestha & Mahmood, 2019):

- Entrenamiento supervisado: Los datos pertenecientes al conjunto de datos contienen una etiqueta, que indican el tipo de dato dentro de este conjunto (en el caso de un conjunto de imágenes de números manuscritos, esta etiqueta indicaría el número que representa la imagen). El principal algoritmo (junto con sus variantes para amoldarse a diferentes tipos de redes neuronales) es el conocido como retropropagación (*backpropagation*). La retropropagación busca minimizar una función de coste (función que indica el desempeño de la red neuronal mediante la etiqueta de los datos) para mejorar el rendimiento utilizando el concepto de descenso del gradiente para esta labor. La disminución del valor de la función de coste se logra a partir de la modificación de los parámetros (pesos y sesgos) de las conexiones neuronales a partir de los estímulos proporcionados por los datos de entrenamiento, moldeando en mayor o menor medida estos parámetros en función del resultado deseado. Esta retropropagación comienza en las conexiones entre la última y penúltima capa de neuronas, modificando los parámetros de las conexiones para variar el valor de activación de las neuronas que posibilitan obtener una salida correcta

para el dato en concreto que se está tratando, repitiéndose este proceso para todas las conexiones de las capas que conforman la red neuronal.

- No supervisado: En este tipo de entrenamiento los datos del conjunto de datos no disponen de una etiqueta. Estas redes neuronales son entrenadas previamente con modelos generativos (que se definirán posteriormente) como las máquinas de Boltzmann restringidas (RBM, que se definirán en un apartado siguiente) y, posteriormente, pueden ser mejoradas mediante algoritmos supervisados.
- Semi-supervisado: En este tipo de entrenamiento, se produce una complementación del entrenamiento no supervisado (con datos no etiquetados) con entrenamiento supervisado (con datos etiquetados).

Estas redes neuronales ofrecen una alta polivalencia gracias a los distintos tipos existentes, lo que permite seleccionar un tipo de red neuronal con una mayor eficacia para el problema concreto a resolver. En consecuencia, se indagará en múltiples clasificaciones de aspectos referentes a las redes neuronales, lo que permitirá un mayor aprendizaje sobre estas.

2.1.1. Modelo de la red neuronal

Atendiendo al modelo de la red neuronal, esta se puede clasificar en (Shrestha & Mahmood, 2019):

- Discriminativos: El recorrido de los datos del conjunto de datos comienza en la capa de entrada y termina en la capa de salida. Son utilizados en aprendizaje supervisado para problemas de clasificación o regresión.
- Generativos: El recorrido de los datos del conjunto de datos comienza en la capa de salida y termina en la capa de entrada. Son utilizados en el entrenamiento previo de algoritmos no supervisados y en problemas probabilísticos.

De forma general, cuando hay datos etiquetados disponibles, se utiliza el modelo discriminativo, debido a que proporcionan un entrenamiento efectivo,

mientras que, en el caso de no disponer de datos etiquetados, se utiliza el modelo generativo.

2.1.2. Tipo de la red neuronal

Una red neuronal puede clasificarse en uno de los siguientes tipos (Shrestha & Mahmood, 2019):

- Red neuronal prealimentada (*feedforward neural network*): La información fluye en una dirección desde la capa de entrada hasta la capa neuronal de salida a través de las capas intermedias ocultas (si las hay), por lo que no forman círculos ni bucles. Este tipo de redes son usadas en una gran cantidad de aplicaciones, como reconocimiento de imágenes o reconstrucción de modelos 3D.
- Red neuronal recurrente (*recurrent neural network*): En este tipo de redes neuronales, se forman ciclos entre las capas, por lo que la salida de una capa se convierte en la entrada de la siguiente capa. Como normalmente solo hay una capa en este tipo de redes neuronales, la salida de esta capa se convierte en una entrada para ella misma, formando un bucle, lo que permite que estados previos influyan en la salida final. La topología de este tipo de redes permite obtener una secuencia de valores a partir de una secuencia de datos entrantes, característica útil para aplicaciones que procesan una secuencia de datos temporal como reconocimiento de voz o clasificación imagen a imagen de un vídeo. Para ilustrar este tipo de redes, se mostrará una imagen demostrativa:

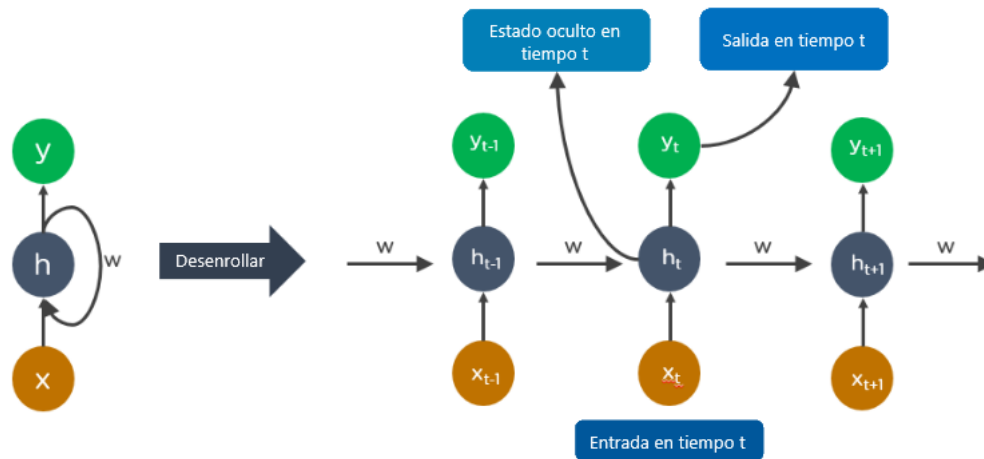


Ilustración 8. Esquema de una red neuronal recurrente.

Adaptado de Simplilearn. (2021). Top 10 Deep Learning Frameworks You Should Know in 2021. simplilearn.com. https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm?source=sl_frs_nav_playlist_video_clicked.

- Red neuronal de base radial (*radial basis function neural network*): Las redes neuronales de base radial están formadas por capas de entrada, de salida e intermedias ocultas. Estas capas ocultas incluyen una función de base radial y cada neurona representa un centroide de un clúster. La red aprende a designar los valores de entrada a un centroide y la capa de salida combina las salidas de las funciones de base radial y los pesos de las conexiones neuronales para la realización de inferencia o clasificación. Este tipo de redes son usadas diversas tareas como clasificación, aproximación de funciones o problemas de predicción de series temporales. A continuación, se mostrará un esquema sobre este tipo de redes:

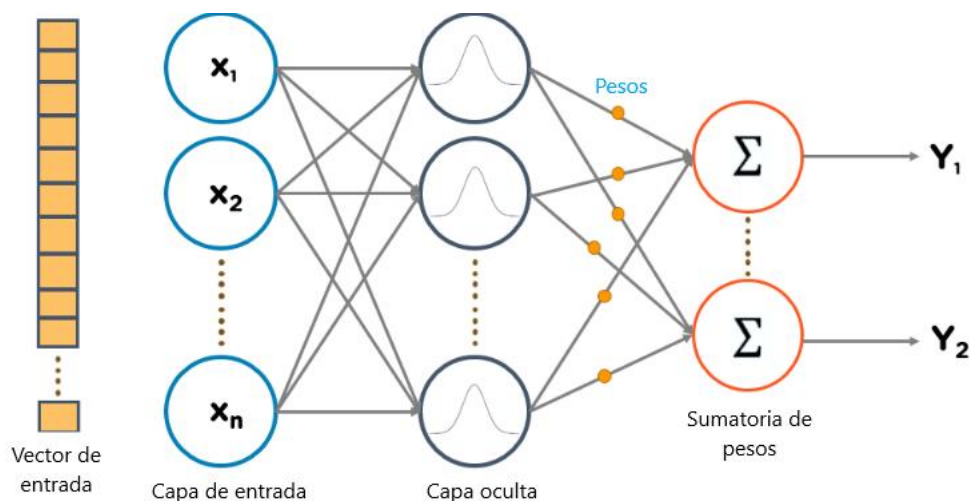


Ilustración 9. Esquema de una red neuronal de base radial.

Adaptado de Simplilearn. (2021). Top 10 Deep Learning Frameworks You Should Know in 2021. simplilearn.com. https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm?source=sl_frs_nav_playlist_video_clicked.

- Redes de Kohonen (*Kohonen self-organizing neural network*): Este tipo de redes neuronales consisten en dos capas, una de entrada con forma bidimensional y otra de salida, totalmente conectadas entre sí, indicando los pesos la posición de la neurona en la capa de salida. El funcionamiento de la red neuronal consiste en que, ante datos de entrada con características comunes, se activan neuronas situadas en zonas próximas de la capa de salida, y tanto estas neuronas que se activan como las neuronas vecinas reciben una actualización en sus pesos para reflejar esta correspondencia. Este tipo de redes neuronales tienen la estructura reflejada en la siguiente figura:

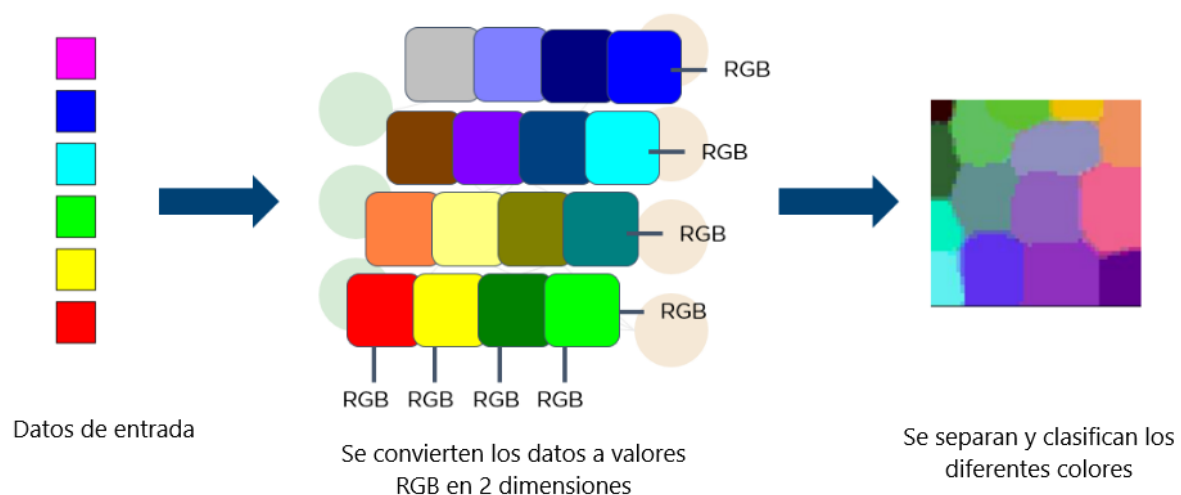


Ilustración 10. Esquema de una red de Kohonen.

Adaptado de Simplilearn. (2021). Top 10 Deep Learning Frameworks You Should Know in 2021. simplilearn.com. https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm?source=sl_frs_nav_playlist_video_clicked.

- Redes neuronales modulares (*modular neural network*): Las redes neuronales modulares se conforman de una serie de redes neuronales independientes de menor tamaño, cuyos valores de salida son posteriormente combinados para obtener la salida de la red neuronal modular.

2.1.3. Arquitectura de la red neuronal

Numerosas arquitecturas han sido desarrolladas a lo largo de los años para resolver problemas de distintos campos o casos de uso (como la utilización de redes convolucionales para visión artificial y reconocimiento de imágenes o redes neuronales recurrentes para problemas de series temporales), lo que posibilita realizar una elección de una arquitectura que se amolde a las necesidades del problema a resolver. Sin embargo, en problemas más generales en los cuales múltiples arquitecturas pueden obtener un buen rendimiento, como un problema de clasificación, no existe una arquitectura que despunte por encima del resto. Las arquitecturas más comunes en redes neuronales son las redes convolucionales (CNN), autocodificadores, máquinas de Boltzmann restringidas (RBM) y redes LSTM (Shrestha & Mahmood, 2019).

2.1.3.1. Redes convolucionales

Las redes convolucionales (LeCun et al., 1990) se basan en la corteza visual humana, siendo la red neuronal predilecta en áreas como Visión Artificial y reconocimiento de vídeo, utilizándose además en otras áreas como Procesamiento del Lenguaje Natural o descubrimiento de drogas.

Las redes convolucionales reciben datos N-dimensionales de entrada (N suele ser 2 o 3) y están formadas por una serie de capas, que pueden ser de convolución, de muestreo (*subsampling*), completamente conectadas y de normalización. Cada tipo de capa cumple una funcionalidad dentro de la red neuronal convolucional:

- Capa de convolución: Este tipo de capa está formada por neuronas asociadas a una posición de la imagen y conectadas con las neuronas que representan las posiciones vecinas. Estas neuronas se comportan como máscaras de convolución (su utilización obtiene un nuevo valor aplicando una operación predefinida a los valores vecinos) cuyos coeficientes son los pesos de las conexiones con las neuronas que representan las posiciones vecinas. La utilización de estas capas permite realizar la extracción de características de los datos de entrada.

- Capa de muestreo (*subsampling*): Este tipo de capa permite reducir el tamaño de los datos de entrada (por lo que reduce también el tamaño de la red), introduciéndose normalmente entre dos capas de convolución. Realizan operaciones como el máximo o la media sobre un área, reduciendo esta área al valor obtenido por la operación.
- Capa completamente conectada: Este tipo de capa, que se suele encontrar al final de la red neuronal, es la que se encarga de procesar la información extraída por las capas anteriores y mostrar la salida deseada (en un problema de clasificación, indicaría a que clase pertenece el dato de entrada).
- Capa de normalización: Este tipo de capa permite normalizar los datos de la red neuronal.

A continuación, se muestra una ilustración de una red neuronal convolucional utilizada en el reconocimiento de caracteres.

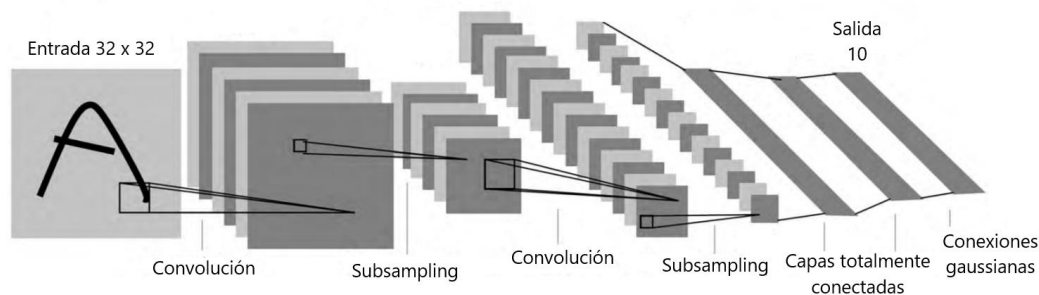


Ilustración 11. Ejemplo de red convolucional.

Adaptado de Fig 4, Shrestha, A., & Mahmood, A. (2019). Review of deep learning algorithms and architectures. *IEEE Access*, 7, 53040-53065. <https://doi.org/10.1109/ACCESS.2019.2912200>.

En esta imagen se puede apreciar una red convolucional formada por dos capas de convolución y muestreo, junto a dos capas completamente conectadas y una capa con conexiones gaussianas. Esta estructura permite que, al indicarse una imagen 32x32, se obtenga una predicción de que tipo de carácter es el que se ha introducido.

2.1.3.2. Autocodificadores

Los autocodificadores o *autoencoders* (Galinari et al., 1987) son un tipo de red neuronal que utilizan algoritmos no supervisados para representar en la salida de la red neuronal aquel dato con el que se ha alimentado la red neuronal en la entrada (Shrestha & Mahmood, 2019). Para la realización de esta tarea, utiliza un codificador (*encoder*) que comprime la entrada y obtiene características de esta, características que son utilizadas en otro fragmento posterior de la red neuronal denominado decodificador (*decoder*).

Debido a la capacidad de extracción de características de los *autoencoders*, estos son utilizados frecuentemente en el entrenamiento previo de redes neuronales cuando no se disponen de muchos datos etiquetados o para crear datos de salida similares a los de entrada.

A modo de ilustración, se mostrará un esquema que representa un *autoencoder*:

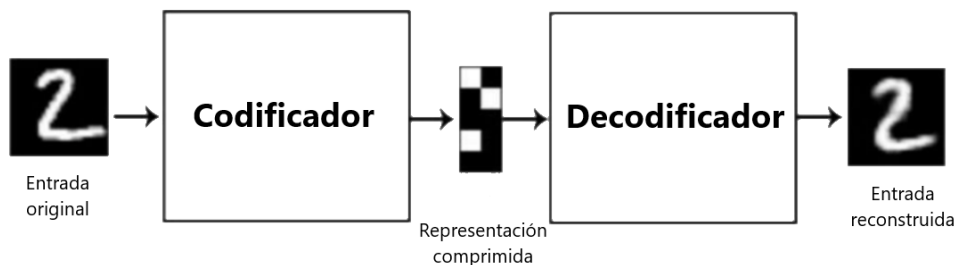


Ilustración 12. Procedimiento de un autocodificador.

Adaptado de Building Autoencoders in Keras. (s. f.). Recuperado 7 de septiembre de 2021, de <https://blog.keras.io/building-autoencoders-in-keras.html>.

En este esquema se puede apreciar como el autoencoder, mediante la entrada y con el uso del codificador, se obtiene una representación comprimida. Esta representación comprimida es utilizada posteriormente por el decodificador, obteniendo un resultado parecido al original aunque haya sufrido un proceso de reconstrucción a partir de una cantidad de información notablemente inferior.

2.1.3.3. Máquinas de Boltzmann restringidas

Las máquinas de Boltzmann restringidas (Hinton et al., 1983) son un tipo de red neuronal en el que se utilizan algoritmos no supervisados para crear modelos generativos no lineales a partir de datos sin etiquetar. La estructura de este tipo de redes neuronales está constituida por dos capas, una denominada capa visible y la otra denominada capa oculta, estando cada neurona de la capa visible conectada con todas las neuronas de la capa oculta y no habiendo conexiones entre neuronas de la misma capa. La estructura de este tipo de redes neuronales se puede apreciar en la siguiente ilustración:

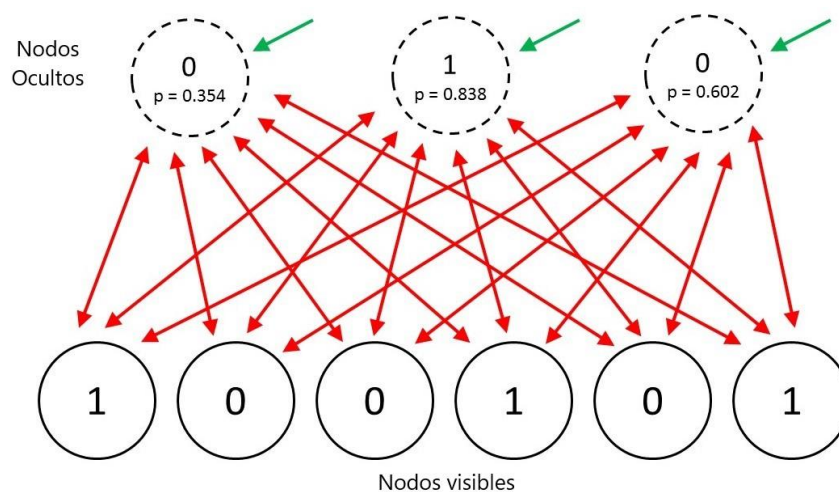


Ilustración 13. Estructura de una máquina de Boltzmann restringida.

Adaptado de Restricted Boltzmann Machine Tutorial | Deep Learning Concepts | Edureka. (s. f.). Recuperado 7 de septiembre de 2021, de <https://www.edureka.co/blog/restricted-boltzmann-machine-tutorial/>.

Este tipo de redes neuronales son entrenadas para que aprendan la distribución de probabilidad de los datos de entrada y, una vez han sido entrenadas (incluso con la utilización de algoritmos supervisados para optimizar la red una vez realizado el proceso del algoritmo no supervisado), son aplicadas principalmente en una de las dos siguientes situaciones:

- Se utiliza la representación obtenida en la capa oculta como preprocesamiento de los datos de entrada para otra red.
- Se utilizan los parámetros obtenidos en la máquina de Boltzmann restringidas para inicializar una red neuronal prealimentada (*feedforward neural network*).

2.1.3.4. Redes LSTM

Las redes neuronales LSTM (Hochreiter & Schmidhuber, 1997), o Long Short-Term Memory, son un tipo de redes neuronales recurrentes por lo que, a diferencia de las arquitecturas previamente mencionadas, pueden almacenar información de estados previos y ser usadas en problemas en los que se requiera la utilización de estos estados previos. Las redes neuronales LSTM superan una limitación de las redes neuronales recurrentes, el desvanecimiento de gradientes (concepto que será tratado en un apartado posterior), gracias a las puertas (*gates*) de la estructura que conforma estas redes, conocida como celdas de memoria (*memory cell*) o simplemente celdas (*cell*). Esta *memory cell* permite diferenciar las redes LSTM con respecto a otras redes recurrentes, ya que las redes LSTM permiten almacenar información de estados a largo plazo, mientras que otras redes recurrentes se centran más en información de estados a corto plazo. Una *memory cell* contiene *gates* de tres tipos diferentes:

- Puerta de entrada (*input gate*): Controla que información puede o no entrar en la *memory cell*.
- Puerta de olvido (*forget gate*): Gestiona que información permanece o no en la *memory cell*.
- Puerta de salida (*output gate*): Selecciona la información utilizada para calcular la salida de la *memory cell*.

Cabe incidir en que existe una variación de menor tamaño de LSTM, debido a la eliminación de las puertas de salida, conocida como GRU que permite obtener incluso un mejor resultado que LSTM en algunos conjuntos de datos sencillos. Para finalizar el apartado, se mostrará un esquema de una red LSTM:

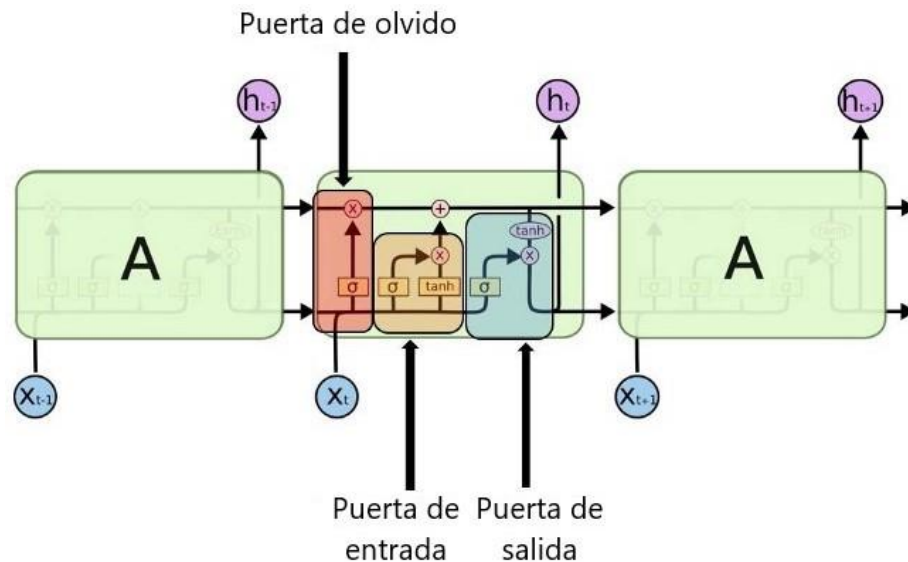


Ilustración 14. Estructura de una red LSTM.

Adaptado de *What is LSTM , peephole LSTM and GRU?* | by Jaimin Mungalpara | Nerd For Tech | Medium. (s. f.). Recuperado 7 de septiembre de 2021, de <https://medium.com/nerd-for-tech/what-is-lstm-peephole-lstm-and-gru-77470d84954b>.

2.1.3.5. Perceptrón multicapa

Los perceptrones multicapa, mejor conocidos como MLP (*Multilayer Perceptron*) son un tipo de red neuronal prealimentada con numerosas capas, una de salida, una de entrada y múltiples capas ocultas. Este tipo de redes neuronales utilizan aprendizaje supervisado y un enfoque discriminativo, siendo su principal mecanismo de aprendizaje la retropropagación basada en descenso del gradiente. Este tipo de redes neuronales son utilizadas en múltiples aplicaciones como clasificación y reconocimiento de voz o imagen.

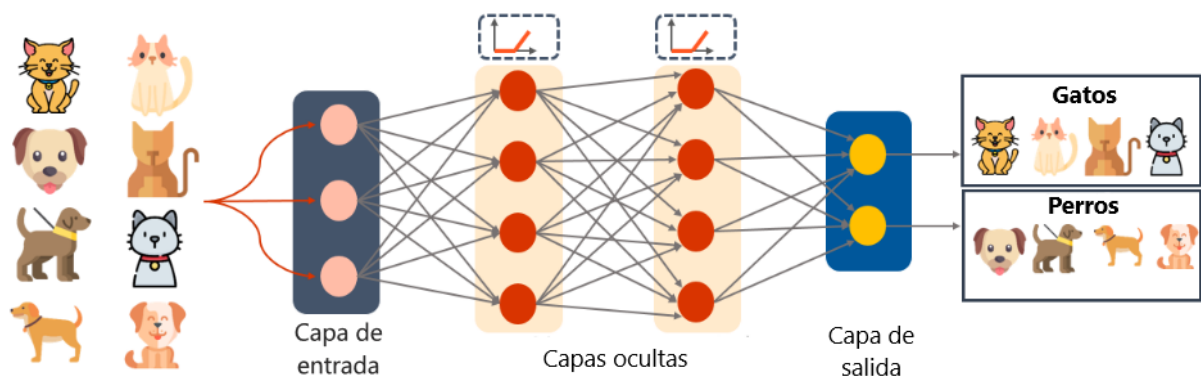


Ilustración 15. Esquema de un perceptrón multicapa.

Adaptado de Simplilearn. (2021). *Top 10 Deep Learning Frameworks You Should Know in 2021*. simplilearn.com. https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm?source=sl_frs_nav_playlist_video_clicked.

2.1.3.6. Redes Generativas Antagónicas

Las Redes Generativas Antagónicas, mejor conocidas como GAN (Goodfellow et al., 2014), o *Generative Adversarial Networks*, consiste en una mezcla de redes neuronales generativas y discriminativas. La red neuronal generativa obtiene nuevos datos falsos a partir de la información de entrada, utilizando entrenamiento no supervisado, y la red neuronal discriminativa intentar discernir si los datos son reales (del conjunto de datos de entrenamiento) o si esos datos son generados por la red neuronal generativa. Posteriormente, con la predicción obtenida, se entrena tanto la red neuronal generativa para engañar a la red neuronal discriminativa y a la red neuronal discriminativa para distinguir los datos reales de los falsos.

Las aplicaciones de este tipo de redes son múltiples, como el escalado de imágenes de baja resolución a alta resolución, crear imágenes realistas a partir de texto, renderizar objetos 3D, crear fotografías de caras humanas, entre otros.

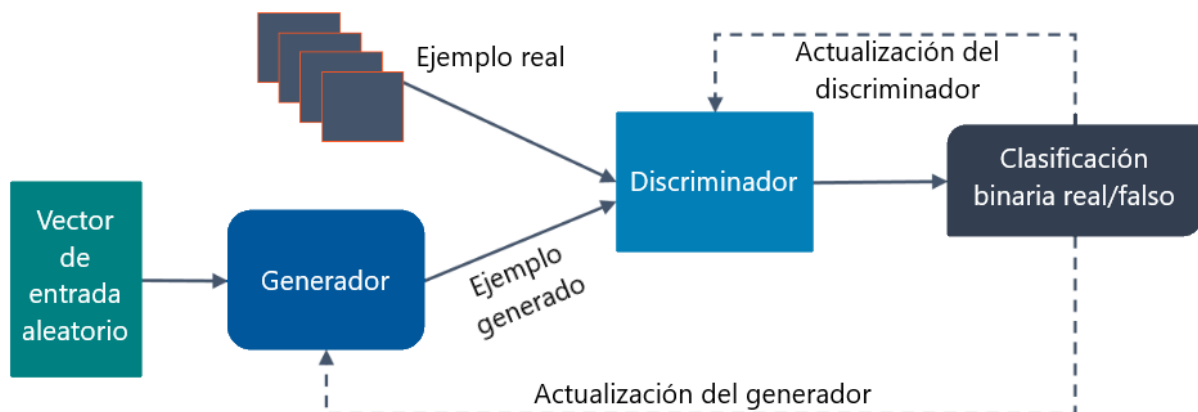


Ilustración 16. Esquema de una red generativa antagónica (GAN).

Adaptado de Simplilearn. (2021). Top 10 Deep Learning Frameworks You Should Know in 2021. [simplilearn.com. https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm?source=sl_frs_nav_playlist_video_clicked](https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm?source=sl_frs_nav_playlist_video_clicked).

2.1.4. Validación de los modelos de Deep Learning

La validación de los modelos de *Deep Learning* es una parte vital de las redes neuronales, debido a que, conforme las redes neuronales eran cada vez más profundas, el entrenamiento de forma manual cada vez era más difícil de realizar. Por consiguiente, debido a las múltiples desventajas del entrenamiento manual, como su complejidad, poca adaptabilidad o peor rendimiento respecto a

entrenamiento automático, esta validación es realizada actualmente mediante algoritmos automáticos de entrenamiento, cuyo objetivo es encontrar los valores óptimos de los parámetros de las conexiones neuronales para resolver el problema planteado. Cabe señalar los siguientes algoritmos de entrenamientos más comunes (Shrestha & Mahmood, 2019):

- **Descenso del gradiente (*Gradient descent*):** El descenso del gradiente es la idea detrás de una gran cantidad de redes neuronales, basándose una gran parte de algoritmos de retropropagación (*backpropagation*) en este concepto. El descenso del gradiente es un algoritmo de optimización enfocado a la minimización de la función objetivo de la red neuronal. El algoritmo de *retropropagación* utiliza la regla de la cadena y derivadas parciales para determinar el error del cambio del valor en cada uno de los pesos y, posteriormente, estos pesos son ajustados para minimizar la función objetivo en cada iteración del conjunto de datos de entrenamiento, finalizando el algoritmo cuando no se reduzca más la función objetivo. Cabe señalar que, debido a la iteración completa del conjunto de datos, este concepto del descenso de gradiente puro es ineficiente, por lo que surge la variante descenso estocástico del gradiente.
- **Descenso estocástico del gradiente (*Stochastic gradient descent*):** Esta variante del descenso del gradiente divide el conjunto de entrenamiento en pequeños subconjuntos y actualiza los pesos al finalizar el procesamiento de cada uno de estos. Esta variante del descenso del gradiente es la más comúnmente implementada debido a su rápida convergencia en un mínimo y su eficiencia con respecto a la variante pura.
- ***Momentum*:** En el descenso estocástico del gradiente se utiliza un multiplicador fijo del gradiente para calcular la actualización de los pesos, lo que puede provocar problemas como aumentar la probabilidad de una convergencia prematura a un mínimo local o retrasar la convergencia si el gradiente es irregular. Debido a ello, este algoritmo

introduce una variable velocidad que es la media del gradiente con un descenso exponencial con respecto a este, suavizando los problemas indicados anteriormente.

- **Algoritmo Levenberg-Marquardt:** Este algoritmo está enfocado a resolver problemas de mínimos cuadrados no lineales como el ajuste de curvas. Para realizar esta labor, el algoritmo Levenberg-Marquardt utiliza una combinación del concepto de descenso del gradiente (empleado para reducir la suma de los errores cuadrados actualizando los parámetros de la función en dirección al descenso más elevado) y el método Gauss-Newton (minimiza el error asumiendo que la función es localmente cuadrática y encuentra el mínimo).
- **Retropropagación en el tiempo (*backpropagation through time*):** Este algoritmo es el estándar para las redes neuronales recurrentes. Una red neuronal recurrente al “desenrollarse” parece similar a una red neuronal prealimentada, pero realmente no es así, ya que la red neuronal recurrente tiene los mismos pesos para cada capa y representan el proceso de entrenamiento durante el tiempo. El proceso de retropropagación en el tiempo calcula los gradientes respecto a unos pesos específicos en cada capa de la red recurrente “desenrollada” y, posteriormente, hace la media del mismo peso para las diferentes actualizaciones durante el tiempo y las modifica para asegurar la uniformidad de los valores de los pesos de cada capa.
- ***Contrastive divergence:*** Este algoritmo es utilizado en modelos probabilísticos como la máquina de Boltzmann restringida. Este algoritmo nació a partir de la lentitud de convergencia del método de Monte Carlo basado en cadenas de Marko, demostrando que, si únicamente se realizan unos cuantos pasos en la cadena de Markov, el aprendizaje todavía podía funcionar bien, acercándose al rendimiento de la solución con el cálculo de la cadena completa.

- *Competitive learning*: Este algoritmo busca la competición entre las diferentes neuronas que conforman la red neuronal, iniciando la mayoría de parámetros de estas neuronas con los mismos valores excepto algún parámetro con valores ligeramente diferentes que le permita respuestas ligeramente variables. Una vez iniciadas, se tiene que fomentar la competición entre las neuronas para que responda una neurona u otra a los subconjuntos del conjunto de datos de entrada, teniendo en cuenta que se ha de añadir un límite máximo de la capacidad que puede tener una neurona para responder, es decir, se debe limitar su “fuerza”.

2.1.5. Comparación de redes neuronales y sus características

Tipo	Arquitectura	Modelo	Tipo de entrenamiento	Algoritmo de entrenamiento	Aplicaciones comunes
RNP	Convolutacional	Discriminativo	Supervisado	Retropropagación basada en descenso del gradiente	Clasificación y reconocimiento de imágenes
RNP	Autocodificadores	Generativo	No supervisado	Retropropagación	Reducción de dimensionalidad y codificación
RNP	Boltzmann	Generativo con optimización discriminativa	No supervisado	<i>Contrastive divergence</i>	Reducción de dimensionalidad, aprendizaje de características y modelado de temas
RNP	GAN	Generativo y discriminativo	No supervisado	Retropropagación	Mejora de imágenes, creación de datos falsos realistas, reconstrucción de modelos 3D
RNP	Perceptrón multicapa	Discriminativo	Supervisado	Retropropagación basada en	Clasificación y reconocimiento

				descenso del gradiente	de voz e imagen.
RNR	LSTM	Discriminativo	Supervisado	Descenso del gradiente y Retropropagación en el tiempo	Procesamiento del Lenguaje Natural y traducción de idiomas
RNBR	Red de función de base radial	Discriminativo	Supervisado y no supervisado	Agrupamiento mediante K-means y función de mínimos cuadrados	Aproximación de funciones y predicción de series temporales
Kohonen	Nodos dispuestos en una red hexagonal o rectangular	Generativo	No supervisado	<i>Competitive learning</i>	Reducción de dimensionalidad, problemas de optimización y análisis de agrupamiento

Tabla 1. Comparación de redes neuronales y sus características.

Adaptado de Tabla 2, Shrestha, A., & Mahmood, A. (2019). Review of deep learning algorithms and architectures. IEEE Access, 7, 53040-53065. <https://doi.org/10.1109/ACCESS.2019.2912200>.

2.2. XAI

Para habilitar la utilización de los modelos de *Deep Learning* en ciertos sectores con una alta necesidad de comprender el modelo, tal y como se indica en la introducción, se ahondará en diferentes apartados correspondientes al campo de *XAI*.

2.2.1. Vocabulario

Antes de profundizar en el campo de *XAI*, cabe recalcar diferentes palabras que serán utilizadas a lo largo de los siguientes apartados con el significado indicado y que, aunque similares, contienen diversos matices que las diferencia:

- *Understandability* (entendible): Característica de un modelo que permite a un humano entender su funcionamiento sin explicar sus algoritmos o estructura interna.
- *Comprehensibility* (comprensible): Capacidad de un modelo de representar el conocimiento obtenido de forma que un humano pueda entenderlo.
- *Interpretability* (Interpretable): Capacidad de explicar un significado de forma entendible para un humano.
- *Explainability* (Explicable): Asociado al término de explicación, es decir, interfaz entre humanos y un responsable de la toma de decisiones, siendo tanto una representación precisa del responsable de la toma de decisiones como comprensible para los humanos.
- *Transparency* (Transparente): Un modelo es transparente si es entendible por sí mismo. Posteriormente, se desglosará los diferentes grados de transparencia que puede tener un modelo.

2.2.2. Objetivos de XAI

Existen diferentes objetivos abordados en numerosos artículos académicos por los cuales se busca obtener un modelo explicable, aunque prácticamente ninguno se pone de acuerdo en que objetivos un modelo explicable debería cumplir. Por lo tanto, se mostrará una clasificación de aquellos objetivos más frecuentes para dilucidar el porqué de la búsqueda de un modelo explicable de Aprendizaje Automático:

- *Trustworthiness* (confiable): Diversos autores indican que ser confiable es el objetivo principal de un modelo explicable de Inteligencia Artificial. Sin embargo, declarar un modelo como explicable solo por su capacidad de ser confiable puede no ser una definición completa, aunque si es necesaria esta cualidad en un modelo explicable.

- *Causality* (causalidad): La causalidad entre variables de datos es otro objetivo buscado en XAI, aunque un modelo de Aprendizaje Automático solo descubre correlaciones, por lo que puede no ser suficiente para encontrar una relación causa-efecto. Sin embargo, causalidad implica correlación, por lo que estas correlaciones pueden validar resultados de técnicas de descubrimiento de causalidades, o aportar un primer punto de vista en las posibles causalidades de los datos.
- *Transferability* (transferibilidad): En muchas ocasiones, el objetivo de obtener un modelo explicable es entender mejor los conceptos obtenidos por este modelo, para reutilizar la información obtenida en otros ámbitos. Este concepto de transferibilidad es necesario para un modelo explicable, pero no cada modelo transferible es un modelo explicable.
- *Informativeness* (informativo): Los modelos de Aprendizaje Automático son usados para ayudar una toma de decisiones, siendo diferente el problema afrontado por estos modelos respecto al problema afrontado por los usuarios del modelo. En consecuencia, se necesita una gran cantidad de información del modelo para poder extrapolar la solución dada por el modelo a la solución que el usuario del modelo debe tomar, evitando equivocaciones.
- *Confidence* (confianza): Entendiendo confianza como generalización de la robustez y estabilidad de un sistema, este elemento siempre debe encontrarse en un sistema donde se espera fiabilidad. Esta estabilidad es un elemento fundamental si se quieren obtener interpretaciones de un modelo, debido a que un modelo no estable no producirá interpretaciones confiables, por lo que un modelo explicable debe tener información sobre la confianza de su proceso y estructura.
- *Fairness* (justo): La obtención de un modelo explicable permite gestionar la justicia o el análisis ético de los resultados a mano, por lo que la

obtención de esta característica en el modelo actúa como un puente para poder obtener justicia en los resultados arrojados.

- **Accessibility** (accesibilidad): La posibilidad de obtener un modelo explicable permite a aquellos usuarios no familiarizados con técnicas de Aprendizaje Automático a comprender mejor el modelo, lo que permite que estos usuarios sean más propensos a involucrarse en el proceso de desarrollo y mejora de este, además de facilitar la utilización del modelo por parte de estos usuarios no avanzados.
- **Interactivity** (Interactividad): La posibilidad de interactuar y modificar el modelo es mucho más amplia cuando se dispone de un modelo explicable. Por lo tanto, para aquellos modelos en los que el usuario final es de gran importancia, como en el caso del apartado anterior, esta interactividad es lo que asegura el éxito del modelo.
- **Privacy awareness** (conciencia de privacidad): No entender la información que es capturada y almacenada en el modelo puede abrir una brecha de privacidad, y, si se obtiene información sobre las relaciones del modelo por usuarios externos, se puede obtener información de los datos utilizados por los creadores del modelo. Por consiguiente, obtener modelos explicables permite gestionar con mayor facilidad la privacidad del modelo.

Cabe destacar que, para conocer el impacto de los diversos objetivos en la literatura actualmente, se incorporará una tabla (cuyos datos fueron rescatados en 2020) que contiene el número de artículos académicos que tratan cada objetivo en un conjunto amplio de artículos relacionados con XAI:

Objetivo	Nº Referencias
Trustworthiness	9
Causality	6
Transferability	51
Informativeness	112
Confidence	15
Fairness	14
Accessibility	31
Interactivity	8

Privacy awareness 1

Tabla 2. Impacto de los objetivos de XAI.

Adaptado de Tabla 1, Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information Fusion, 58, 82-115. <https://doi.org/10.1016/j.inffus.2019.12.012>.

Como se puede apreciar a partir de los datos en la tabla, los objetivos más buscados actualmente son *informativeness* (informativo) y *transferability* (transferabilidad), mientras que *privacy awareness* (conciencia de privacidad) es el objetivo más olvidado en la literatura.

2.2.3. Niveles de transparencia en modelos de Aprendizaje Automático

Existen diversos modelos de Aprendizaje Automático que son explicables por sí mismos. Sin embargo, el nivel de transparencia del modelo puede diferir, distinguiéndose tres niveles de transparencia (Barredo Arrieta et al., 2020):

- **Simulable:** Habilidad del modelo de ser simulado o pensado por un humano, por lo que tanto la complejidad como la extensión del modelo (como un modelo basado en reglas con demasiadas reglas) juegan un papel importante en este aspecto. Por lo tanto, los modelos más escuetos serán más interpretables que los modelos más densos. Este nivel de transparencia es el más alto de los tres e incorpora el resto de niveles de transparencia.
- **Descomponible:** Habilidad de poder explicar cada una de las partes del modelo (datos de entrada, parámetros y cálculo) y de que estas partes sean entendibles por un humano sin la ayuda de herramientas adicionales. Este nivel de transparencia incorpora el último nivel de transparencia.
- **Transparente algorítmicamente:** Habilidad del usuario de entender el proceso seguido por el modelo para obtener una salida a partir de los datos de entrada. En consecuencia, este modelo debe ser completamente explorable mediante métodos y análisis matemáticos.

A modo de ilustración, se mostrará una imagen que ejemplifica los diversos niveles de transparencia que puede tener un modelo:

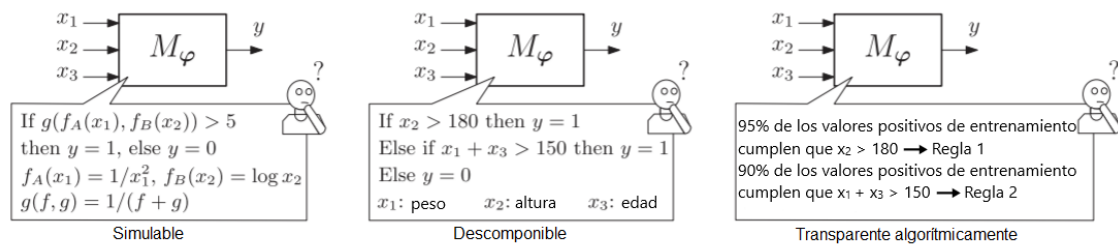


Ilustración 17. Posibles niveles de transparencia en un modelo transparente.

Adaptado de Figura 3, Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Benetot, A., Tabik, S., Barbado, A., García, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). *Explainable Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI*. *Information Fusion*, 58, 82-115. <https://doi.org/10.1016/j.inffus.2019.12.012>.

2.2.4. Métodos a posteriori para obtención de modelos explicables

Existen una gran cantidad de modelos de Aprendizaje Automático que no son interpretables por sí mismos. Debido a ello, existen diversos métodos utilizados para obtener modelos explicables a partir de modelos que no son explicables (Barredo Arrieta et al., 2020):

- **Explicación mediante texto:** Se generan explicaciones textuales que ayudan a explicar los resultados del modelo, incluyendo también cualquier método que genere símbolos que representen el funcionamiento del modelo.
- **Explicación visual:** El objetivo de este tipo de métodos es visualizar el comportamiento del modelo. La mayoría de estos métodos de visualización se utilizan junto a técnicas de reducción de dimensionalidad para poder obtener una visualización simple que pueda ser interpretada por un humano. Junto a estas visualizaciones, se suelen utilizar otras técnicas para mejorar su entendimiento, siendo este tipo de métodos los más indicados para que los usuarios no familiarizados con modelos de Aprendizaje Automático comprendan los entresijos que los conforman.

- Explicación local: Este tipo de métodos segmenta el espacio de soluciones y proporciona explicaciones a subespacios de soluciones menos complejos que son relevantes para todo el modelo. Estas explicaciones pueden estar formadas con otras técnicas, pero estas técnicas solo abordarán una parte del funcionamiento del sistema.
- Explicación mediante ejemplos: Se generan ejemplos del resultado generado por un modelo en concreto, permitiendo mejorar el entendimiento de este al poder observar las relaciones del modelo de una forma más clara.
- Explicación por simplificación: Se crea un nuevo sistema más simple partiendo del modelo que se quiere explicar. Este nuevo sistema busca ser muy parecido al sistema original, disminuyendo la complejidad y obtenido un rendimiento similar, lo que puede aportar un beneficio colateral al poder implementar esta variante simplificada del modelo, reduciendo el tiempo de desarrollo necesario.
- Explicación de la relevancia de características: Este tipo de métodos intentan clarificar el funcionamiento del modelo obteniendo un valor que indica la importancia de las variables del modelo sobre el resultado obtenido. Por consiguiente, la comparación de estos valores permitirá al usuario conocer el impacto de las diferentes variables del modelo al producir el resultado.

De cara a completar la explicación de los métodos a posteriori, se muestra una imagen que ilustra los diferentes tipos de métodos para la obtención de modelos explicables:

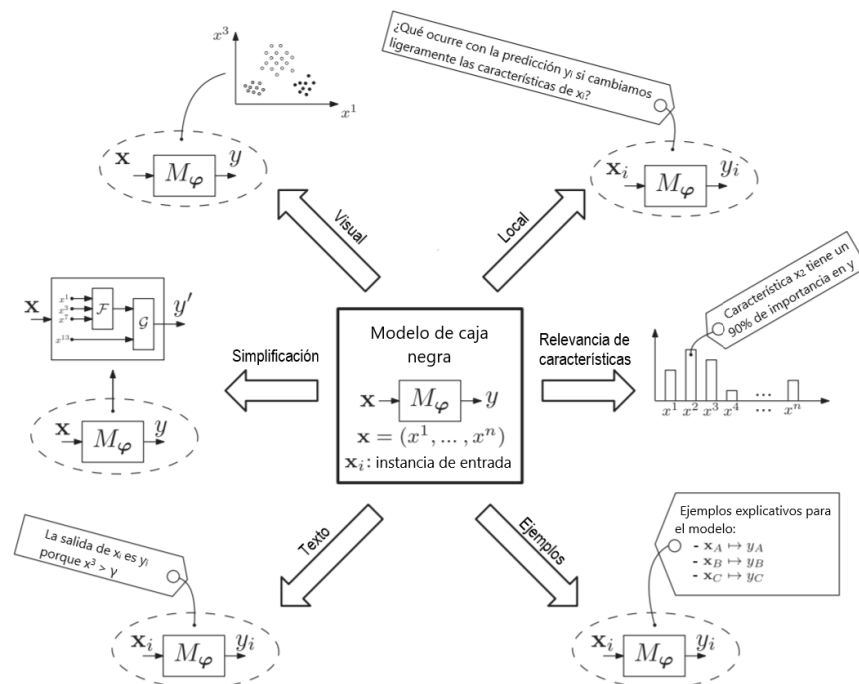


Ilustración 18. Métodos a posteriori para obtención de modelos explicables.

Adaptado de Figura 4, Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information Fusion, 58, 82-115. <https://doi.org/10.1016/j.inffus.2019.12.012>.

2.2.5. Modelos de Aprendizaje Automático transparentes

Existen numerosos modelos de Aprendizaje Automático que son transparentes, y por lo tanto son explicables sin necesidad de métodos a posteriori. A continuación, se expondrán los principales modelos de Aprendizaje Automático que serán descritos dependiendo del nivel de transparencia que posean:

Modelo	Simulable	Descomponible	Transparente algorítmicamente
Regresión lineal/logística	Los sistemas de predicción son legibles por una persona, y las interacciones entre estos sistemas se mantienen al mínimo.	Las variables son todavía legibles, pero el número de interacciones y sistemas de predicción han crecido para forzar la descomposición.	Las variables y las interacciones son muy complejas, por lo que tienen que ser analizadas con herramientas matemáticas.
Árboles de decisión	Un humano puede simular y obtener la	El modelo contiene reglas que no alteran	Las reglas son legibles por el ser humano y explican el

	predicción sin conocimiento matemático.	los datos y preservan su legibilidad.	conocimiento obtenido de los datos, permitiendo entender el proceso de predicción.
K vecinos más cercanos (K-NN)	La complejidad del modelo (número de variables, mecanismo de similaridad utilizado...) entra dentro de las capacidades humanas de simulación.	La complejidad del modelo es demasiado alta para ser simulado completamente, pero puede ser descompuesto y analizado por separado.	La complejidad del modelo es tan alta que el usuario debe utilizar métodos matemáticos y estadísticos para analizar el modelo.
Sistemas de aprendizaje basado en reglas	Las variables son legibles y el número de reglas es gestionable por un humano sin ayuda externa.	El número de reglas se vuelve muy grande, por lo que debe descomponerse en subconjuntos para poder analizarse.	Las reglas han aumentado tanto su complejidad y el número de reglas es tan grande que se necesitan herramientas matemáticas para inspeccionar el comportamiento del modelo.
Modelos generalizados aditivos (GAMs)	Variables, interacciones y funciones de suavizado deben estar limitadas a la capacidad de entendimiento del ser humano.	Las interacciones se vuelven muy complejas para ser emuladas, por lo que se necesitan técnicas de descomposición para analizar el modelo.	Debido a la complejidad, las interacciones y las variables no pueden ser analizadas sin utilizar herramientas estadísticas y matemáticas.
Modelos Bayesianos	Relaciones estadísticas entre variables y las variables en sí mismas deben ser entendibles por la audiencia a la que el sistema esté dirigido.	Las relaciones estadísticas involucran tantas variables que deben descomponerse para facilitar su análisis.	Aunque descompuestas, las relaciones estadísticas y los sistemas de predicción son demasiado complejos, por lo que solo podrán ser analizados con herramientas matemáticas.

Tabla 3. Descripción de modelos transparentes dependiendo de su grado de transparencia.

Adaptado de Tabla 2, Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable Explainable

2.2.6. Modelos propensos a utilizar métodos a posteriori

Cuando los modelos de Aprendizaje Automático no cumplen las características para ser transparentes, se han de utilizar métodos a posteriori como los abordados en apartados anteriores. Sin embargo, dependiendo del tipo de modelo de Aprendizaje Automático, se podrán aplicar métodos específicos para el tipo de modelo, además de la posibilidad de utilizar una serie de métodos a priori que no dependen del modelo en el que se utilicen.

2.2.6.1. Métodos a posteriori no dependientes del modelo

Los métodos a posteriori que pueden utilizarse en cualquier modelo que no es transparente son los siguientes (Barredo Arrieta et al., 2020):

- Explicación por simplificación: Este tipo de método es el más utilizado dentro del grupo de métodos no dependientes del modelo, estando también presente dentro de este la explicación local, debido a que las representaciones simplificadas en ocasiones solo corresponden a una parte del modelo. Algunas de las técnicas de explicación por simplificación más utilizadas son LIME y G-REX.
- Explicación de la relevancia de características: En los últimos años se han escrito una cantidad de artículos académicos similar a los métodos de explicación por simplificación no dependientes del modelo, lo que significa que es un campo activo en XAI. Uno de los algoritmos más utilizados en este ámbito es SHAP.
- Explicación visual: Este tipo de métodos son menos comunes debido a que, al buscar obtener una representación visual de cualquier tipo de modelo de Aprendizaje Automático, crear visualizaciones solo a partir de los valores de entrada y salida es algo muy complejo. Por lo tanto, la mayoría de métodos de explicación visual trabaja en conjunto con métodos de explicación de relevancia de características, aportando la

información que el usuario final obtendrá (este trabajo conjunto puede apreciarse en la ilustración previa).

2.2.6.2. Métodos a posteriori dependientes del modelo

Existen una gran cantidad de métodos que están adheridos a un modelo en concreto. Entre los modelos con una mayor relevancia en cuanto a métodos a posteriori específicos cabe señalar los siguientes (Barredo Arrieta et al., 2020):

- **Ensamblador de árboles:** Estos modelos normalmente utilizan métodos de explicación por simplificación del modelo o métodos de explicación de la relevancia de las características. En el caso de la simplificación, varios artículos académicos tratan la idea de obtener un ensamblador de árboles más simple que facilite la interpretación, utilizando el complejo para el trabajo de clasificación. Por parte de los métodos de explicación de relevancia de las características, un artículo académico trata sobre la medición del descenso de la precisión y el aumento del error de los árboles a partir del cambio aleatorio de una variable.
- **Máquinas de vectores de soporte:** Estos modelos normalmente utilizan métodos de explicación por simplificación del modelo o métodos de explicación local. En el caso de la simplificación, existen diversos artículos científicos que varían en la utilización o no de diferentes elementos del modelo, incorporando únicamente los vectores de soporte, en otros se incorporan tanto los vectores de soporte como el conjunto de datos de entrada... Por parte de los métodos de explicación local, aumentó el número de artículos académicos enfocados en ellos, aunque los métodos de explicación por simplificación tienen más recorrido en la literatura. Sin embargo, todos los artículos académicos tratados por las referencias son anteriores a 2018, lo que puede indicar que estos modelos ya se entienden, o que la proliferación de *Deep Learning* ha mermado el interés en métodos a posteriori en las máquinas de vectores de soporte.

- **Red Neuronal Multicapa:** Estos modelos normalmente utilizan métodos de explicación por simplificación del modelo, métodos de explicación de la relevancia de las características o métodos de explicación visual. Por parte de los métodos de explicación por simplificación, se han propuesto muchos métodos para redes neuronales con una única capa oculta, pero pocos con muchas capas ocultas (como DeepRED, que descompone la red a nivel de neurona para obtener árboles y reglas de decisión). Otro enfoque seguido por los métodos de explicación por simplificación del modelo es obtener información de la red neuronal, como sus gradientes, para generar modelos más simples. Debido a que los métodos de explicación por simplificación son más complejos conforme avanza el número de capas, los métodos de explicación de la relevancia de las características se han convertido paulatinamente más populares. Dentro de estos trabajos cabe destacar el algoritmo DeepTaylor que realiza una descomposición de Taylor profunda para obtener la relevancia de los valores de entrada, además de otro denominado DeepLIFT que compara los valores de activación de las neuronas con un valor de referencia y obtiene información dependiendo de la diferencia. Otra vertiente de este tipo de métodos se basa en comprobar teóricamente los métodos disponibles (como DeConvNet o LRP), descubrir fallos en la mayoría de estos, y postular métodos nuevos que no tienen los fallos de los métodos estudiados (como PatternNet, PatternAttribution y gradientes integrados). Para finalizar, cabe señalar que las técnicas de visualización suelen utilizarse junto a otro tipo de métodos.
- **Red Neuronal Convolutiva:** Estos modelos normalmente utilizan métodos de explicación de la relevancia de las características o métodos de explicación visual. Dentro de los métodos de explicación de la relevancia existen principalmente dos vertientes, una dirigida a entender el proceso de decisión a partir de modificar la imagen de entrada, y otra dirigida a entender la representación en las capas intermedias del mundo exterior de forma general, no teniendo que estar asociado a valores de entrada concretos. En la primera vertiente, se utilizan algoritmos ya citados anteriormente como LRP o Deconvnet, además de

otros diferentes como Grad-CAM. En el caso de la segunda vertiente, existen artículos que obtienen la imagen más representativa para una neurona de salida concreta, o artículos que buscan reconstruir la imagen a partir de las representaciones internas de las diversas capas de la red neuronal convolucional. Finalmente, cabe destacar que las técnicas de visualización suelen utilizarse junto a otro tipo de métodos, como puede apreciarse en la siguiente ilustración:

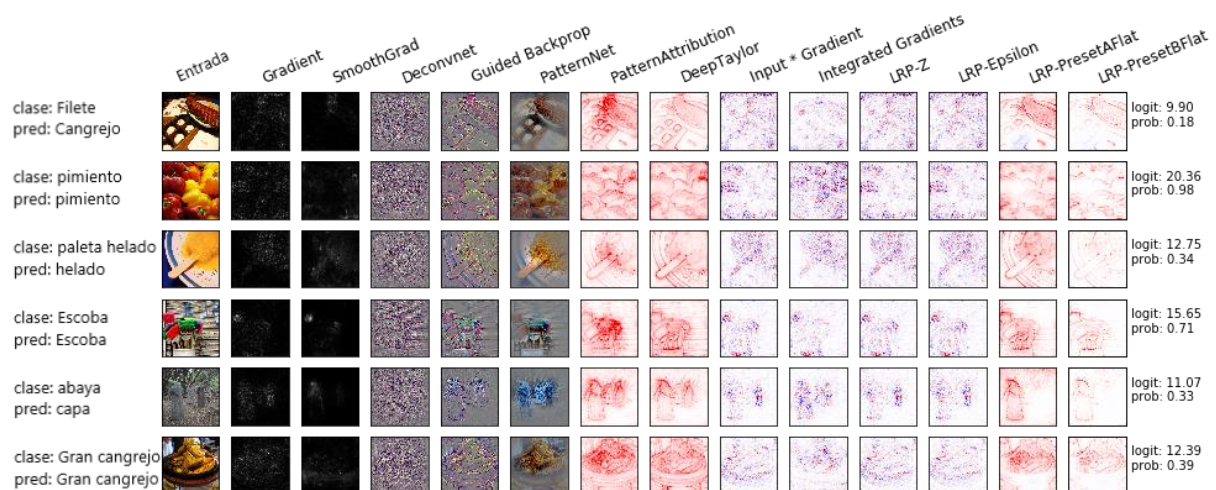


Ilustración 19. Diferentes métodos explicativos a posteriori en Redes Neuronales Convolucionales.

Adaptado de GitHub - albermax/innvestigate: A toolbox to iNNvestigate neural networks' predictions! (s. f.). Recuperado 9 de junio de 2021, de <https://github.com/albermax/innvestigate>.

- **Red Neuronal Recurrente:** Estos modelos normalmente utilizan métodos de explicación de la relevancia de las características y métodos de explicación local. Pocos aportes se han realizado en el caso de las redes neuronales recurrentes, dividiéndose estos aportes en explicar que es lo que ha aprendido el modelo (mediante métodos de explicación de la relevancia de las características) y en modificar la arquitectura del modelo para obtener información sobre las decisiones que realizan (mediante métodos de explicación local). En la primera vertiente, un ejemplo es la extensión de LRP a este tipo de redes neuronales, que utiliza las conexiones multiplicativas presentes en redes LSTM y GRU. En la segunda vertiente, un artículo académico combina un modelo oculto de Markov (*HMM*) con una red neuronal recurrente, para obtener la interpretabilidad de un modelo oculto de Markov y la precisión de una red neuronal recurrente.

2.2.6.3. Métodos a posteriori con presencia en el trabajo

En los siguientes apartados, se utilizarán diversos métodos de XAI para obtener una mayor comprensión sobre los modelos de *Deep Learning*. Debido a ello, es necesario abordar el fundamento teórico de aquellos métodos que serán empleados posteriormente en el trabajo:

- SHAP: El método exacto utilizado basado en los valores de Shapley se denomina DeepSHAP. Este método obtiene un rendimiento computacional superior con respecto a otras implementaciones calculando los valores de Shapley mediante el uso del método DeepLIFT (que permite descomponer la salida de la predicción del modelo a partir de un valor de entrada realizando la retropropagación de la contribución de todas las neuronas de la red neuronal para cada característica del valor de entrada). Al utilizar DeepLIFT para obtener un tiempo de ejecución menor, este método solo está disponible para modelos de Aprendizaje Profundo. Lo que aporta la explicación para el modelo son los valores de Shapley, de forma que cada valor indica la contribución media de una característica en concreto para todas las posibles combinaciones.
- LIME: LIME obtiene una explicación de los modelos de Aprendizaje Profundo a partir de comprender que sucede con las predicciones de los modelos cuando se le aportan datos con variaciones. Para ello, LIME genera un nuevo conjunto de datos a partir de los datos con variaciones y las predicciones del modelo de Aprendizaje Profundo, entrenando un modelo de caja blanca con este conjunto de datos. Este modelo de caja blanca es el utilizado para obtener información sobre el modelo de Aprendizaje Profundo al que se le ha utilizado el método LIME.

2.3. Keras

Keras es una interfaz de programación de aplicaciones (API) de *Deep Learning* escrita en Python, ejecutándose sobre la plataforma de *Machine Learning* TensorFlow.

2.3.1. Descripción de Keras

A continuación, cabe destacar diversas características importantes de Keras:

- **Facilidad de uso:** En Keras prima la facilidad de uso, minimizando las acciones necesarias por el usuario y teniendo una curva de aprendizaje poco pronunciada. Sin embargo, aunque sencillo de utilizar, permite realizar una gran cantidad de ajustes debido al uso de TensorFlow.
- **Gran impacto en la comunidad:** La facilidad de uso y polivalencia que caracteriza a Keras le ha llevado a ser la solución de *Aprendizaje Automático* más utilizada en la actualidad, encontrándose en empresas como Netflix o Uber y en organizaciones científicas importantes como el CERN (Organización Europea para la Investigación Nuclear) o la NASA (Administración Nacional de Aeronáutica y el Espacio). El impacto de Keras en la comunidad del Aprendizaje Automático con respecto a su mayor competidor, PyTorch, puede verse reflejada en la siguiente imagen:

Métricas de frameworks de Deep Learning en 2020

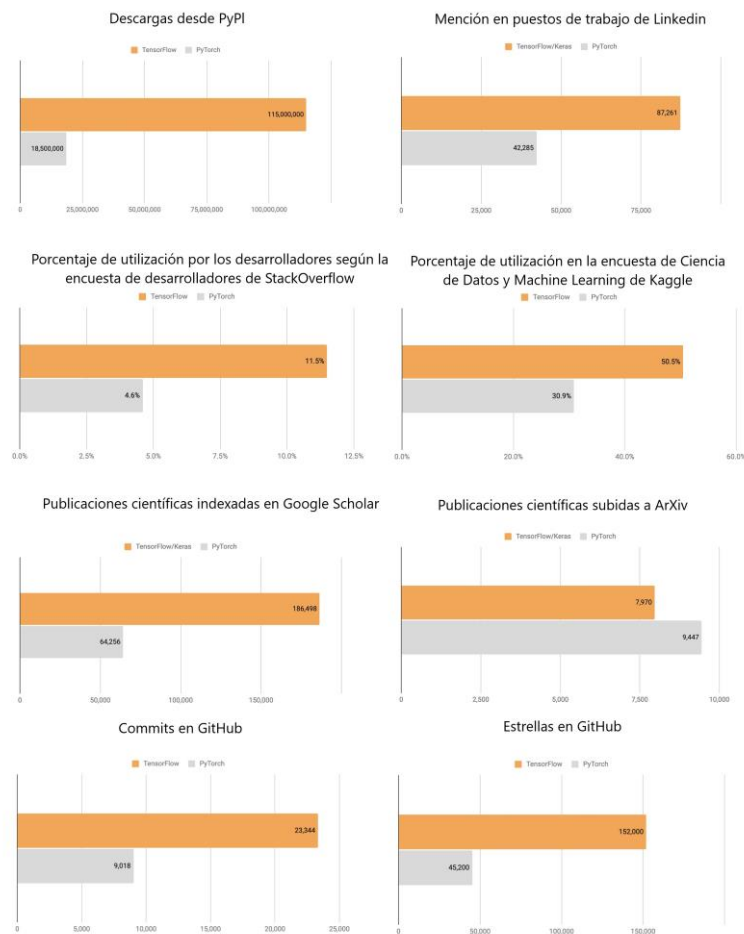


Ilustración 20. Impacto de Keras en múltiples ámbitos en comparación con PyTorch.

Adaptado de *Why choose Keras?* (s. f.). Recuperado 9 de junio de 2021, de https://keras.io/why_keras/.

- Paso a producción sencillo: Los modelos generados con Keras pueden ser desplegados en múltiples plataformas:
 - En un servidor mediante Python o Node.js.
 - En un servidor TFX / TS Serving.
 - En un navegador mediante TF.js.
 - En Android y iOS mediante TF Lite o Apple's CoreML.
 - En Raspberry Pi, Edge TPU y sistemas embebidos.
- Escalabilidad y soporte: Keras es escalable, permitiendo ejecutar modelos en grandes clústeres de tarjetas gráficas y conjuntos de TPUs. Cabe señalar que en Keras existe soporte para entrenamiento con

precisión mixta en TPUs y las últimas gráficas de NVIDIA, lo que permite duplicar la velocidad de entrenamiento e inferencia.

- Keras es el centro de un ecosistema mayor: Keras forma parte de un ecosistema variado, que comprende desde tecnologías de despliegue mencionadas previamente como técnicas de prototipado, entrenamiento escalable de modelos en la nube, extensiones de TensorFlow...

2.3.2. Modelos de Aprendizaje Profundo en Keras

En Keras, encontramos una gran variedad de modelos de Aprendizaje Profundo, tales como redes neuronales convolucionales, redes neuronales recurrentes (como LSTM), autocodificadores... Cabe recalcar que existen una gran cantidad de ejemplos codificados en la página oficial, lo que aporta aún más facilidad a la hora de desarrollar modelos de Aprendizaje Profundo.

2.3.3. XAI en Keras

En los ejemplos oficiales, prácticamente no encontramos ejemplos sobre implementaciones de diversos métodos de XAI para los diversos modelos de caja negra implementados en Keras. Cabe incidir en uno de los pocos ejemplos oficiales encontrados, que utiliza Grad-CAM, algoritmo mencionado previamente en el apartado de métodos a posteriori específicos de redes neuronales convolucionales. El resultado de este algoritmo se puede apreciar en las siguientes ilustraciones:



Ilustración 21. Muestra de la predicción chow con Grad-CAM.

*Adaptado de Chollet, F. (2021). Grad-CAM class activation visualization. En Keras.
https://keras.io/examples/vision/grad_cam/.*



Ilustración 22. Muestra de la predicción gato egipcio con Grad-CAM.

*Adaptado de Chollet, F. (2021). Grad-CAM class activation visualization. En Keras.
https://keras.io/examples/vision/grad_cam/.*

En el caso de recurrir a librerías externas para la obtención de métodos de XAI, existen diversas alternativas que pueden ser utilizadas con los modelos de Keras. Cabe incidir en shap, librería que incorpora el algoritmo SHAP y otros ajustes para entender múltiples tipos de modelos de forma más sencilla y en la librería innvestigate, que incorpora múltiples métodos de XAI como PatternNet, Deconvnet o

versiones de LRP (la ilustración 21 muestra diversos métodos implementados en esta librería).

Por último, cabe destacar una beta en la plataforma AI Platform de Google, denominada Explainable AI, que utiliza diversos métodos de explicación de relevancia de características, como gradientes integrados o SHAP, pudiendo utilizarse estos métodos con modelos de Keras. A continuación, se muestra una ilustración de XRAI, uno de los métodos de explicación de la relevancia de las características disponibles en esta plataforma:

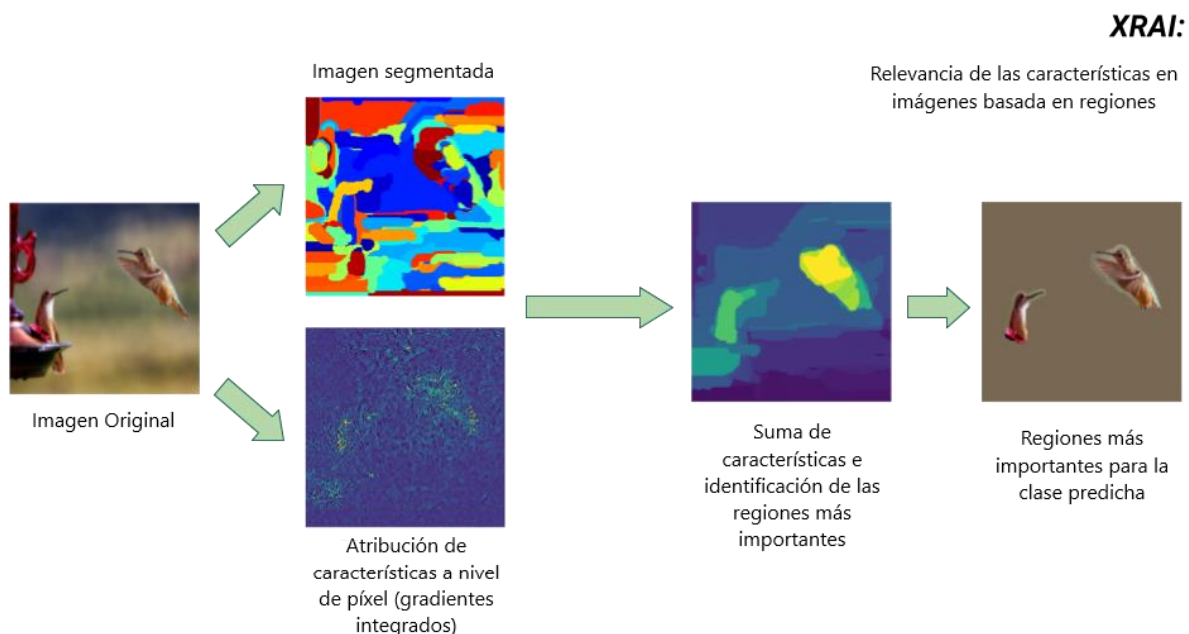


Ilustración 23. Método de XAI en AI Platform.

Adaptado de Chollet, F. (2021). *Grad-CAM class activation visualization*. En Keras.
https://keras.io/examples/vision/grad_cam/.

3. SISTEMA

La creación de los diversos sistemas, que permitirán mostrar el estado actual de *XAI*, conllevó la toma de una serie de decisiones.

La primera decisión consistió en el lenguaje de programación a utilizar. Esta selección del lenguaje está altamente condicionada por la calidad y facilidad de uso tanto de las herramientas a emplear como del propio lenguaje, debido a que, al ser un desarrollador novicio en este tipo de sistemas, la curva de aprendizaje debía ser lo más liviana posible. En consonancia, gracias a la experiencia amable de Python para desarrolladores que se inician en este, junto a la existencia de utilidades a alto nivel altamente respaldadas, el lenguaje de programación seleccionado es Python.

La segunda decisión consistió en que herramienta utilizar para obtener modelos de Aprendizaje Profundo. Esta selección de la herramienta fue condicionada al ser mi primera toma de contacto con el mundo del Aprendizaje Profundo, por lo que Keras, *API* de Tensorflow a alto nivel que se centra en la facilidad de uso, se convirtió en el instrumento que permitió adentrarme en la creación de los sistemas.

La tercera decisión estuvo dirigida a la elección del entorno de desarrollo. Esta decisión fue eminentemente de carácter personal, al no influir ninguna condición del proyecto en la elección, por lo que se seleccionó PyCharm, un entorno de desarrollo polivalente en el que la universidad presta una licencia gratuita. Es necesario indicar que poseo una tarjeta gráfica con cierta potencia, lo que acelera el tiempo de entrenamiento de los modelos en comparación al uso del procesador del equipo. Sin embargo, en el caso de que no hubiese dispuesto de esa posibilidad, existe un entorno de desarrollo alternativo que habría seleccionado, Google Colab. Google Colab es un entorno de desarrollo en la nube creado por Google, en el que se puede utilizar una gráfica para el procesamiento de los scripts sin ningún coste, lo que posibilita un desarrollo ágil en aquellos equipos que no dispongan de los recursos necesarios para un entrenamiento rápido de los modelos de Aprendizaje Profundo.

Una vez estas decisiones fueron tomadas, se creó el proyecto que albergaría los diferentes modelos de Aprendizaje Profundo. El procedimiento de creación del proyecto en PyCharm no contuvo ningún paso con dificultad, por lo que, una vez creado, se crearon los ficheros Python que albergarían los diferentes modelos de Aprendizaje Profundo.

A lo largo de esta sección, se tratarán dos apartados de los sistemas que han sido creados. En primer lugar, se mostrará la estructura del proyecto, indicando tanto los diferentes ficheros que lo conforman como que contiene cada uno de estos ficheros. Más adelante, se abordará diversos aspectos de los sistemas, como sus parámetros o el código que crea los modelos de Aprendizaje Profundo, además de un esquema que recoge la información expuesta para cada tipo de modelo.

3.1. Estructura del proyecto

La estructura del proyecto consiste en doce ficheros Python, conteniendo cada fichero un modelo de Aprendizaje Profundo y un conjunto de datos en concreto para este modelo. Estos doce ficheros contienen uno de los siguientes 6 modelos de Aprendizaje Profundo:

- Autoencoder, cuya función es eliminar el ruido de las imágenes.
- Red convolucional, cuya función es realizar una predicción de las imágenes.
- GAN, cuya función es generar imágenes parecidas a las imágenes del conjunto de datos.
- LSTM, cuya función es realizar una clasificación de la polaridad de los textos.
- MLP, cuya función es realizar una predicción de las imágenes.
- RNN, cuya función es realizar una clasificación de la polaridad de los textos.

Cabe recalcar que, junto a estos ficheros Python, se guarda el fichero correspondiente a un conjunto de datos y, en dos tipos de modelo, ficheros html que nos permitirán mostrar las técnicas de XAI. La siguiente ilustración muestra la estructura del proyecto:

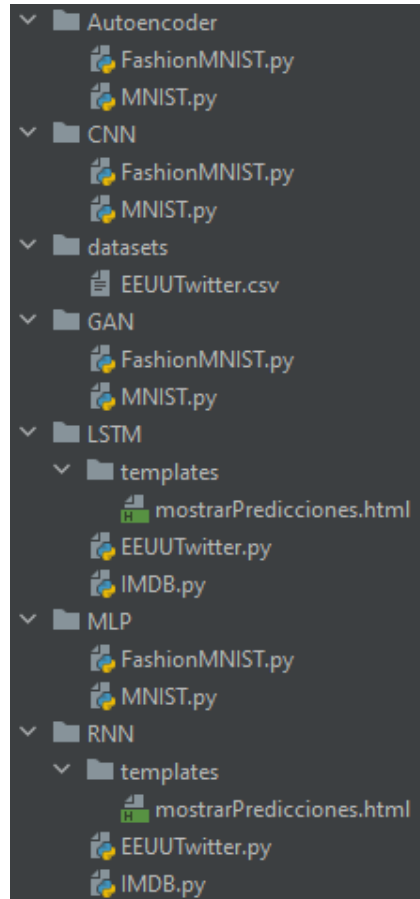


Ilustración 24. Ilustración del proyecto

Aunque se podría haber gestionado los diferentes conjuntos de datos dentro de un solo fichero Python, esta división en 2 ficheros permite separar claramente las diferencias en los scripts ante el uso de diferentes conjuntos de datos. Una porción de código que muestra estas diferencias es el siguiente:

```
Conjunto de datos IMDB
# Se cargan los datos del conjunto de datos. X contiene la información
textual e Y contiene las clases.
(X_entrenamiento, Y_entrenamiento), (X_test, Y_test) =
imdb.load_data(num_words=maxTamVocabulario)

Conjunto de datos EEUUTwitter
# Se pasa los textos a secuencias.
tokenizer = Tokenizer(num_words=maxTamVocabulario, split=' ')
tokenizer.fit_on_texts(conjuntoDatos['text'].values)
X = tokenizer.texts_to_sequences(conjuntoDatos['text'].values)
# Se rellenan los textos.
X = pad_sequences(X)
# Se pasan los valores de polaridad a un array simple, para poder
utilizar un modelo con una función objetivo binaria.
valoresNoTratados = pd.get_dummies(conjuntoDatos['sentiment']).values
Y = []
for elemento in valoresNoTratados:
    if elemento[0] == 1:
        Y.append(0)
    else:
        Y.append(1)
Y = np.array(Y)
# Se cargan los datos del conjunto de datos. X contiene la información
textual e Y contiene las clases.
X_entrenamiento, X_test, Y_entrenamiento, Y_test = train_test_split(X,
Y, test_size = 0.2)
```

En esta parte del código se puede apreciar que, mientras en el caso de un conjunto de datos IMDB el preprocesamiento ya ha sido realizado, por lo que no se tiene que implementar, en el caso del conjunto de datos EEUUTwitter ha de realizarse este preprocesamiento.

3.2. Modelos de Aprendizaje Profundo

A continuación, se indicarán diversos aspectos importantes de la implementación de los diversos modelos de Aprendizaje Profundo.

3.2.1. Autoencoder

Dentro de la implementación de este modelo, cabe mostrar los siguientes aspectos:

3.2.1.1. Esquema general

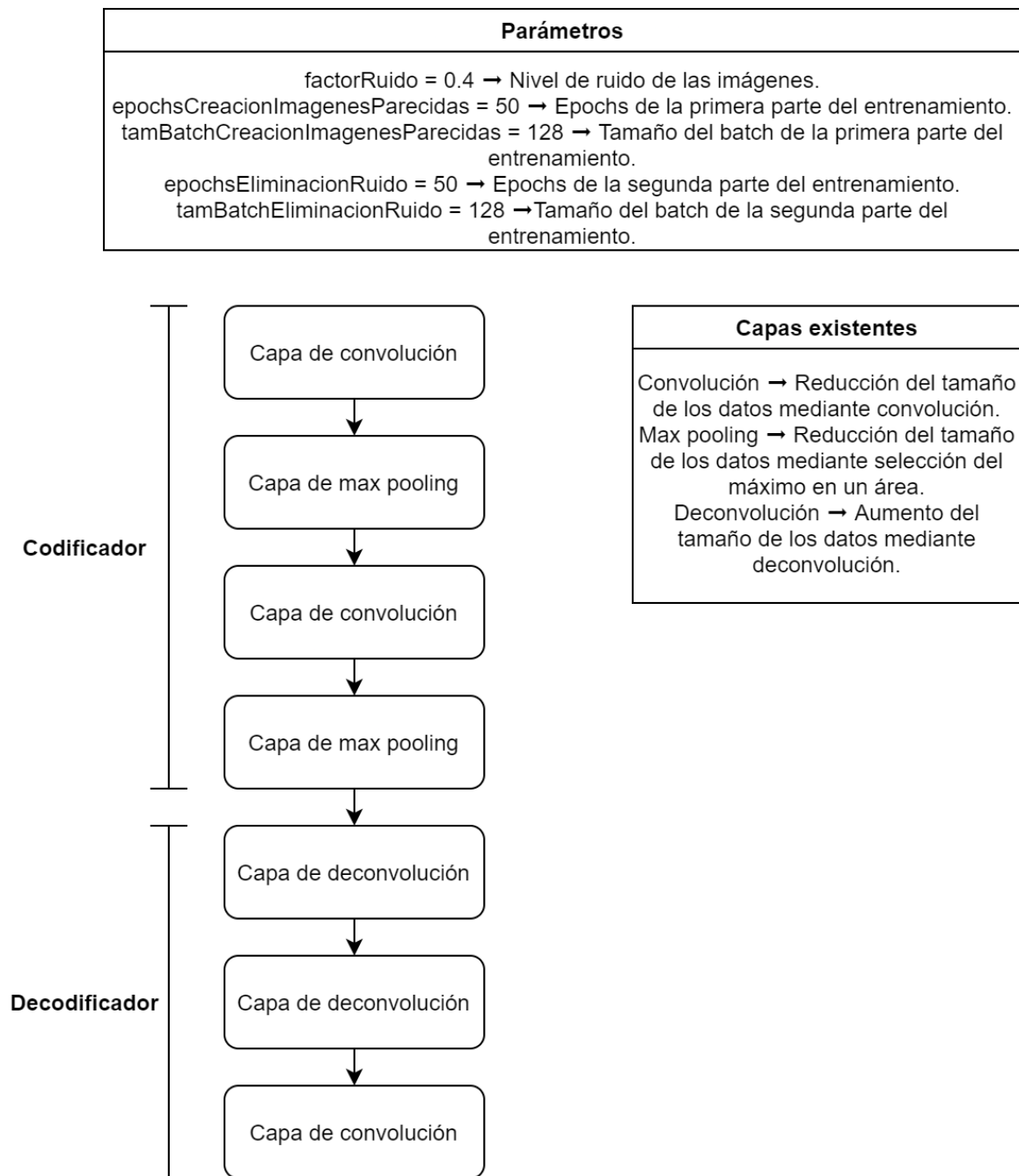


Ilustración 25. Esquema general de Autoencoder.

3.2.1.2. Parámetros configurables

Los parámetros configurables disponibles en este modelo son los siguientes:

```

factorRuido = 0.4
epochsCreacionImagenesParecidas = 50
tamBatchCreacionImagenesParecidas = 128
epochsEliminacionRuido = 50
tamBatchEliminacionRuido = 128
  
```

Estos parámetros nos permiten modificar el nivel del ruido que tendrán las imágenes a las que se les eliminará el ruido, y tanto los *epochs* como el tamaño del *batch* en las dos partes de entrenamiento del modelo (creación de imágenes parecidas y eliminación del ruido de las imágenes).

3.2.1.3. Implementación de la estructura del modelo

El código de la implementación de la estructura del modelo es realizado acorde a la ilustración:

```
# Codificador.
x = layers.Conv2D(32, (3, 3), activation="relu",
padding="same")(formaDatosEntrada)
x = layers.MaxPooling2D((2, 2), padding="same")(x)
x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(x)
x = layers.MaxPooling2D((2, 2), padding="same")(x)

# Decodificador.
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu",
padding="same")(x)
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu",
padding="same")(x)
x = layers.Conv2D(1, (3, 3), activation="sigmoid", padding="same")(x)
```

La parte del codificador está formada por dos capas de convolución y dos capas de *max pooling* (que permite reducir el tamaño de los datos seleccionando el valor máximo de un área), mientras que la parte del decodificador está formada por dos capas de deconvolución y una última capa de convolución.

3.2.2. CNN

Dentro de la implementación de este modelo, cabe destacar los siguientes aspectos:

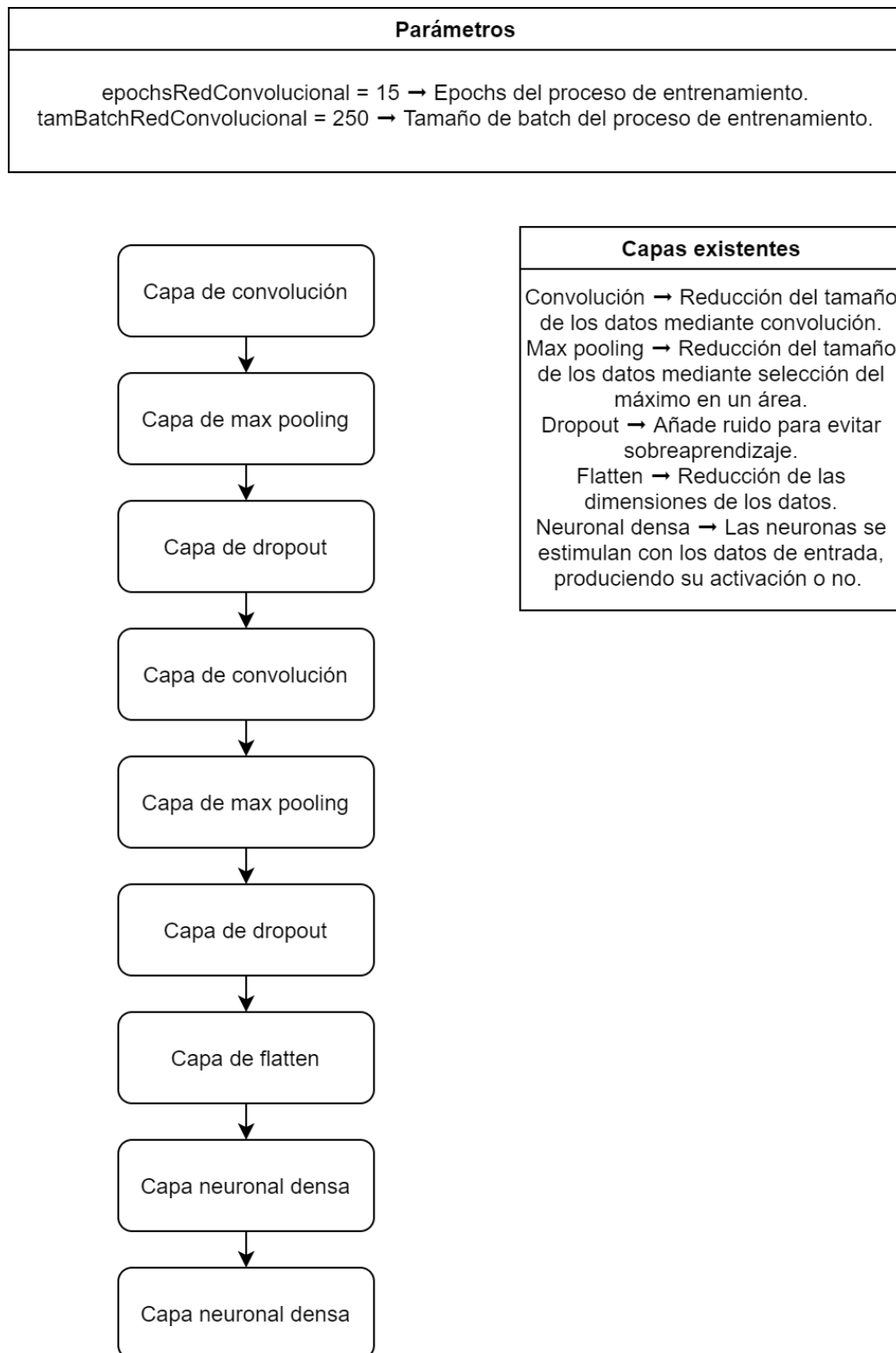
3.2.2.1. Esquema general

Ilustración 26. Esquema general de CNN.

3.2.2.2. Parámetros configurables

Los parámetros configurables disponibles en este modelo son los siguientes:

```
epochsRedConvolucional = 15  
tamBatchRedConvolucional = 250
```

Estos parámetros nos permiten modificar tanto el número de *epochs* como el tamaño del *batch* para el entrenamiento de la red convolucional.

3.2.2.3. Implementación de la estructura del modelo

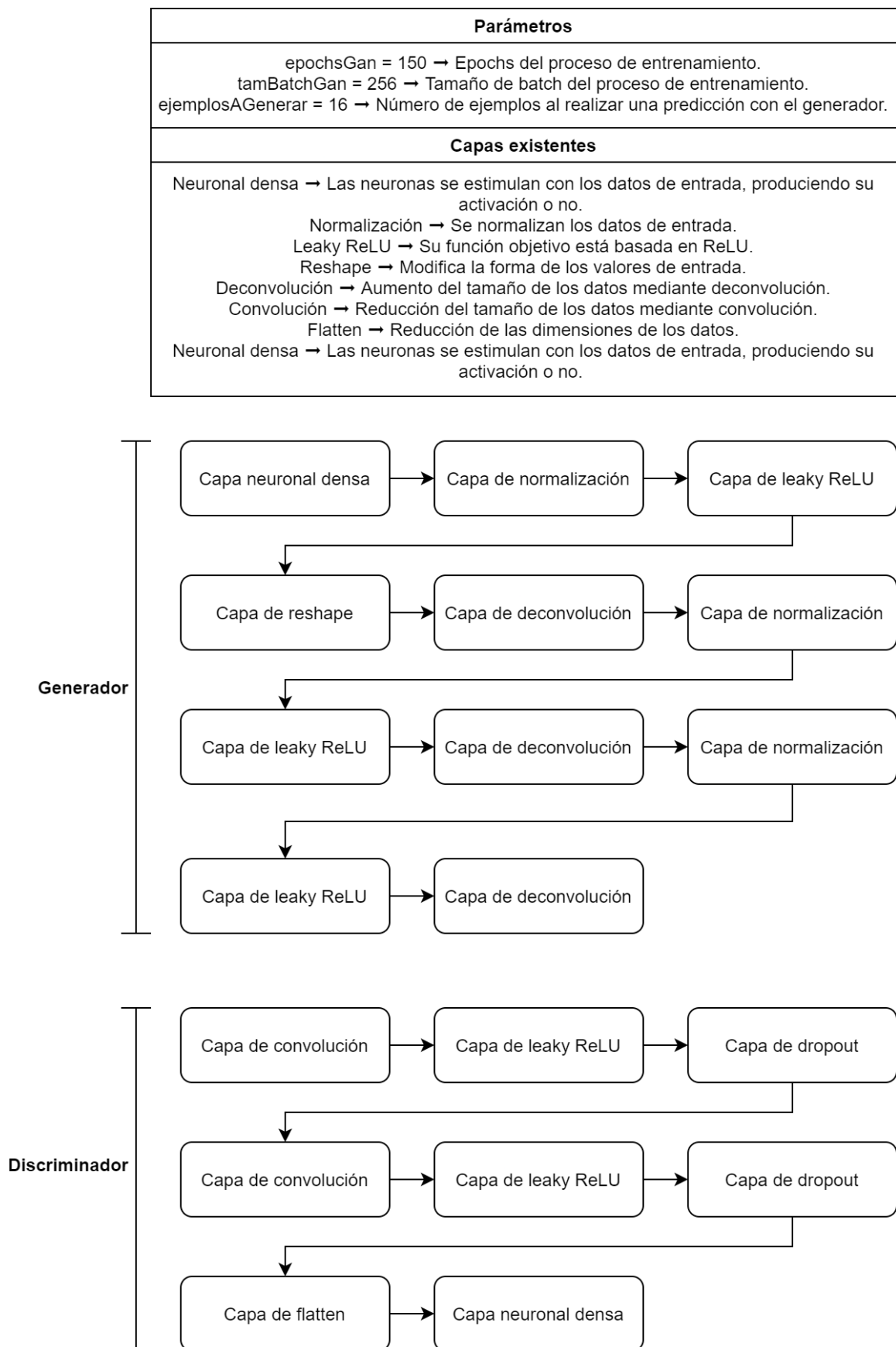
El código de la implementación de la estructura del modelo es realizado acorde a la ilustración:

```
modelo.add(Conv2D(32, kernel_size=(3, 3), activation='relu',  
input_shape=formaDatosEntrada))  
modelo.add(MaxPooling2D(pool_size=(2, 2)))  
modelo.add(Dropout(0.25))  
modelo.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))  
modelo.add(MaxPooling2D(pool_size=(2, 2)))  
modelo.add(Dropout(0.25))  
modelo.add(Flatten())  
modelo.add(Dense(256, activation='relu'))  
modelo.add(Dense(numClases, activation='softmax'))
```

La red convolucional está formada por una serie de capas de convolución, de *max pooling* (que permite reducir el tamaño de los datos seleccionando el valor máximo de un área), de *dropout* (que permite añadir ruido, lo que reduce la posibilidad de sobreaprendizaje), neuronales densas y de *flatten* (que permiten convertir los datos multidimensionales en datos unidimensionales).

3.2.3. GAN

Dentro de la implementación de este modelo, cabe mostrar los siguientes aspectos:

3.2.3.1. Esquema general**Ilustración 27. Esquema general de GAN**

3.2.3.2. Parámetros configurables

Los parámetros configurables disponibles en este modelo son los siguientes:

```
epochsGan = 150  
tamBatchGan = 256  
ejemplosAGenerar = 16
```

Estos parámetros nos permiten modificar tanto el número de *epochs* como el tamaño del *batch* para el entrenamiento del GAN, además del número de ejemplos generados en la muestra del funcionamiento del modelo.

3.2.3.3. Implementación de la estructura del modelo

El código de la implementación de la estructura del modelo es realizado acorde a la ilustración:

```
# Generador  
modelo.add(layers.Dense(7 * 7 * 256, use_bias=False,  
input_shape=(100,)))  
modelo.add(layers.BatchNormalization())  
modelo.add(layers.LeakyReLU())  
modelo.add(layers.Reshape((7, 7, 256)))  
modelo.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1),  
padding='same', use_bias=False))  
modelo.add(layers.BatchNormalization())  
modelo.add(layers.LeakyReLU())  
modelo.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2),  
padding='same', use_bias=False))  
modelo.add(layers.BatchNormalization())  
modelo.add(layers.LeakyReLU())  
modelo.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2),  
padding='same', use_bias=False, activation='tanh'))  
# Discriminador  
modelo.add(layers.Conv2D(64, (5, 5), strides=(2, 2),  
padding='same', input_shape=[28, 28, 1]))  
modelo.add(layers.LeakyReLU())  
modelo.add(layers.Dropout(0.3))  
modelo.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))  
modelo.add(layers.LeakyReLU())  
modelo.add(layers.Dropout(0.3))  
modelo.add(layers.Flatten())  
modelo.add(layers.Dense(1))
```

El GAN está formado por un generador de imágenes y un discriminador, que clasifica las imágenes como reales o generadas. Tanto el generador como el discriminador comparten capas de *leaky ReLU* (capa con una función objetivo basada en ReLU) y neuronales densas. Cabe señalar que, además de estas capas, el generador contiene capas de normalización, de deconvolución y de *reshape* (que permiten la modificación de la forma de los valores de entrada), mientras que el

discriminador contiene capas de convolución, de *dropout* (que permite añadir ruido, lo que reduce la posibilidad de sobreaprendizaje) y de *flatten* (que permiten convertir los datos multidimensionales en datos unidimensionales).

3.2.4. LSTM

Dentro de la implementación de este modelo, cabe destacar los siguientes aspectos:

3.2.4.1. Esquema general

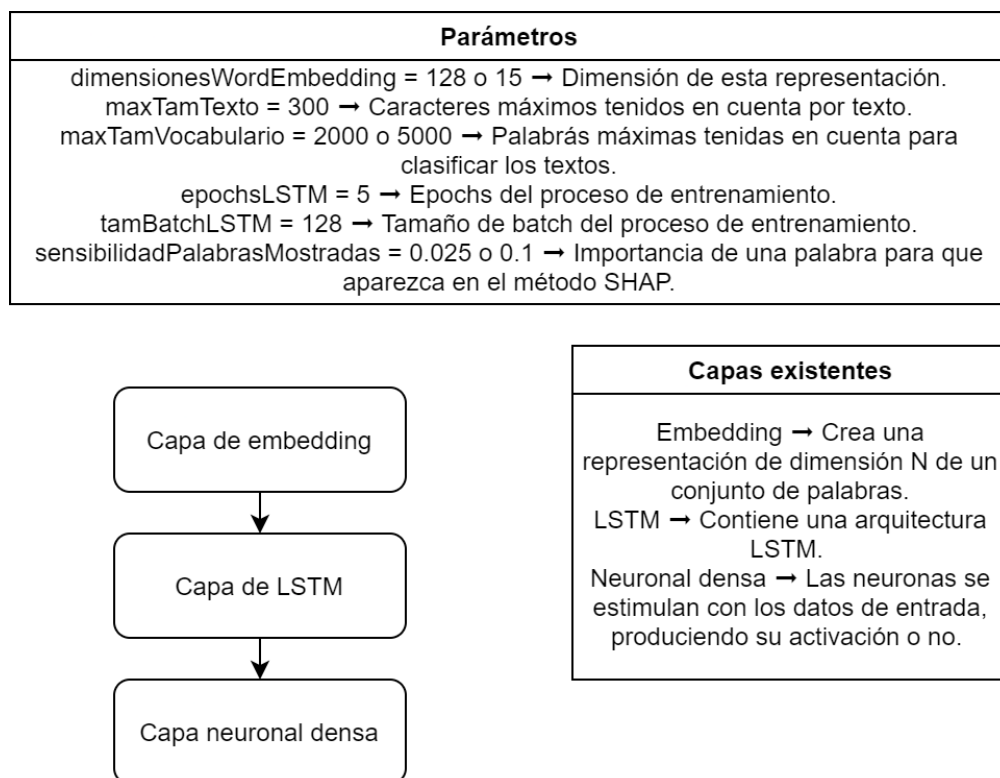


Ilustración 28. Esquema general de LSTM.

3.2.4.2. Parámetros configurables

Los parámetros configurables disponibles en este modelo son los siguientes:

```
dimensionesWordEmbedding = 128 o 15
maxTamTexto = 300
maxTamVocabulario = 2000 o 5000
epochsLSTM = 5
tamBatchLSTM = 128
sensibilidadPalabrasMostradas = 0.025 o 0.1
```

Estos parámetros nos permiten modificar la dimensión de la capa de *word embedding* (que crea una representación de un conjunto de palabras), la cantidad

máxima de caracteres que será tomada en cuenta en cada texto, el número máximo de palabras que se tendrán en cuenta para clasificar los textos, el número de *epochs* y el tamaño del *batch* para el entrenamiento del LSTM. Los valores en las dimensiones del *word embedding* y en el tamaño máximo del vocabulario difieren debido a que están optimizados para los dos conjuntos de datos que se emplean en este modelo.

Por último, cabe destacar el parámetro de sensibilidad, que indica como de importantes han de ser las palabras en el análisis de polaridad para que aparezcan representadas en el método de SHAP.

3.2.4.3. Implementación de la estructura del modelo

El código de la implementación de la estructura del modelo es realizado acorde a la ilustración:

```
modelo.add(Embedding(maxTamVocabulario, dimensionesWordEmbedding,  
input_length=maxTamTexto))  
modelo.add(LSTM(10))  
modelo.add(Dense(1, activation='sigmoid'))
```

El LSTM está formado por una capa de *word embedding* (que crea una representación de un conjunto de palabras), de LSTM y neuronal densa.

3.2.5. MLP

Dentro de la implementación de este modelo, cabe mostrar los siguientes aspectos:

3.2.5.1. Esquema general

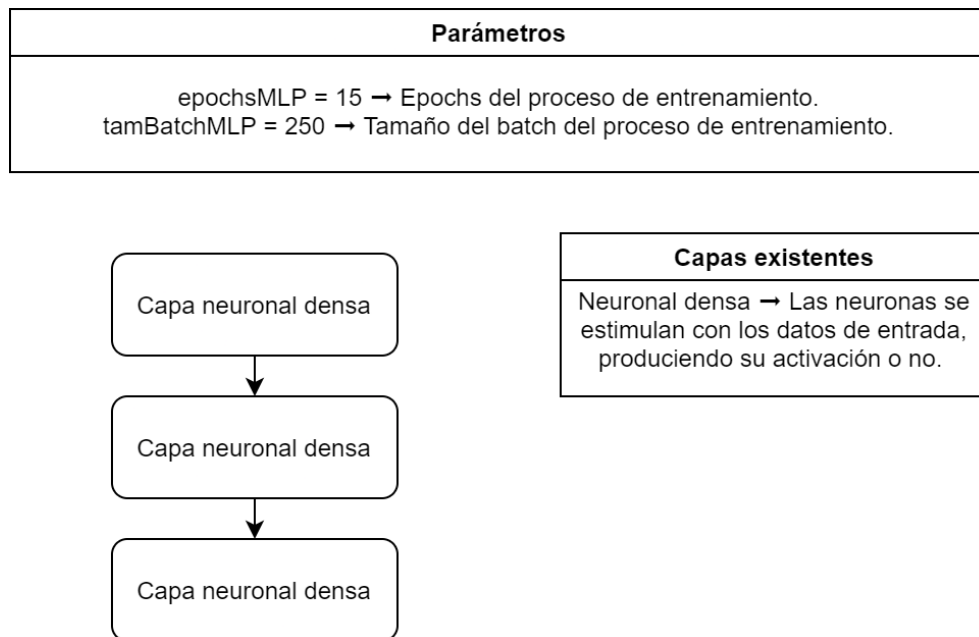


Ilustración 29. Esquema general de MLP.

3.2.5.2. Parámetros configurables

Los parámetros configurables disponibles en este modelo son los siguientes:

```
epochsMLP = 15
tamBatchMLP = 250
```

Estos parámetros nos permiten modificar tanto el número de *epochs* como el tamaño del *batch* para el entrenamiento del perceptrón multicapa.

3.2.5.3. Implementación de la estructura del modelo

El código de la implementación de la estructura del modelo es realizado acorde a la ilustración:

```

modelo.add(Dense(350, input_shape=formaDatosEntrada,
activation='relu'))
modelo.add(Dense(50, activation='relu'))
modelo.add(Dense(numClases, activation='softmax'))
          
```

El perceptrón multicapa está formado únicamente por capas neuronales densas.

3.2.6. RNN

Dentro de la implementación de este modelo, cabe destacar los siguientes aspectos:

3.2.6.1. Esquema general

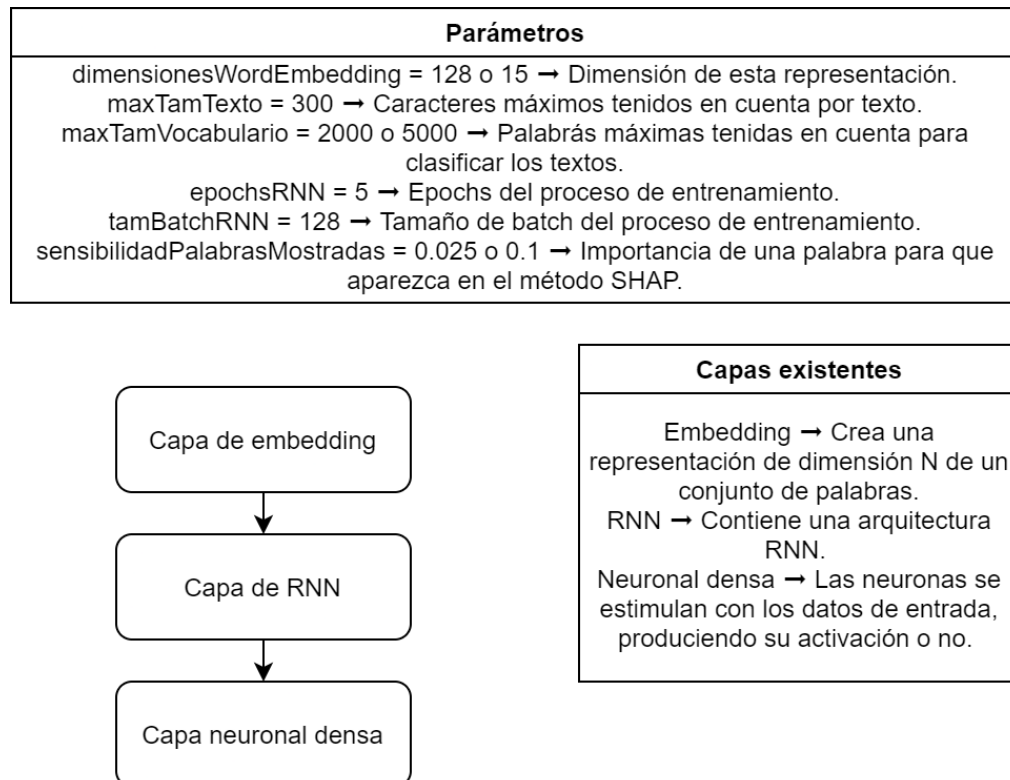


Ilustración 30. Esquema general de RNN.

3.2.6.2. Parámetros configurables

Los parámetros configurables disponibles en este modelo son los siguientes:

```

dimensionesWordEmbedding = 128 o 15
maxTamTexto = 300
maxTamVocabulario = 2000 o 5000
epochsRNN = 5
tamBatchRNN = 128
sensibilidadPalabrasMostradas = 0.025 o 0.1
  
```

Estos parámetros nos permiten modificar la dimensión de la capa de *word embedding* (que crea una representación de un conjunto de palabras), la cantidad máxima de caracteres que será tomada en cuenta en cada texto, el número máximo de palabras que se tendrán en cuenta para clasificar los textos, el número de *epochs* y el tamaño del *batch* para el entrenamiento del RNN. Los valores en las dimensiones del *word embedding* y en el tamaño máximo del vocabulario difieren

debido a que están optimizados para los dos conjuntos de datos que se emplean en este modelo.

Por último, cabe destacar el parámetro de sensibilidad, que indica como de importantes han de ser las palabras en el análisis de polaridad para que aparezcan representadas en el método de SHAP.

3.2.6.3. Implementación de la estructura del modelo

El código de la implementación de la estructura del modelo es realizado acorde a la ilustración:

```
modelo.add(Embedding(maxTamVocabulario, dimensionesWordEmbedding,
input_length=maxTamTexto))
modelo.add(SimpleRNN(16, return_sequences=False, activation="relu"))
modelo.add(Dense(1, activation='sigmoid'))
```

El RNN está formado por una capa de *word embedding* (que crea una representación de un conjunto de palabras), de RNN y neuronal densa.

3.3. Modelos de XAI

3.3.1. Autoencoder y GAN

Cabe indicar que, en el caso de estos modelos de Aprendizaje Profundo, no se ha podido encontrar una librería que aporte mecanismos de XAI capaces de amoldarse a los problemas que resuelven. Esto es debido a que problemas más comunes, como clasificación de imágenes o clasificación de textos, son los que monopolizan la atención de los desarrolladores, por lo que los problemas de eliminación de ruido en imágenes (Autoencoder) y generación de nuevas imágenes a partir de imágenes previas (GAN) no son abarcados de forma solvente en librerías.

3.3.2. CNN y MLP

El problema que resuelven ambos modelos de Aprendizaje Profundo es el de clasificación de imágenes. A continuación, se muestra las partes del sistema que gestionan los métodos de XAI:

3.3.2.1. SHAP

```
# Se aporta un conjunto de datos para que SHAP sea capaz de obtener
explicaciones.
conjuntoDatosBackground =
X_entrenamiento[np.random.choice(X_entrenamiento.shape[0], 1000,
replace=False)]
# Se le pasan los datos de background a SHAP.
e = shap.DeepExplainer(modelo, conjuntoDatosBackground)
# Se pasan las imágenes predecidas anteriormente con la forma correcta
para obtener sus valores de SHAP.
valoresShap = e.shap_values(imagenesSeleccionadas.reshape(3,28,28,1) o
.reshape(3,784))
# Se modifican las imágenes y los valores de SHAP para mostrarse
posteriormente.
arrayVectores = np.stack([np.array(vectoresImagenes)])
imagenesAMostrar = arrayVectores.reshape(3, 28, 28, 1)
for i in range(len(valoresShap)):
    valoresShap[i] = valoresShap[i].reshape(3,28,28,1)
# Se añade el tipo de valor de cada columna para mejorar la
visualización.
nuevoGrupoClases = []
for i in range(len(labelsSeleccionadas[0])):
    nuevoGrupoClases.append(valoresClases[i] + "\n" +
str(labelsSeleccionadas[0][i]))
# Se modifica el primer conjunto de valores para reflejar las clases.
labelsSeleccionadas[0] = nuevoGrupoClases
""" Se muestra las imágenes con sus valores de SHAP superpuestos, que
representan aquellos puntos de importancia en la
predicción, además de un 0 o 1 que indica el valor real del elemento.
"""
shap.image_plot(valoresShap, -imagenesAMostrar, labelsSeleccionadas)
```

3.3.2.2. LIME

```
# Se crea una función que gestiona las imágenes para poder utilizar
correctamente LIME.
def funcionPrediccion(images):
    arrayImg = []
    for img in images:
        img = img.reshape(28,28,3)
        img = img[0:29,0:29,0:1]
        arrayImg.append(img)
    imagenesAPredecir = np.array(arrayImg)
    return modelo.predict_on_batch(imagenesAPredecir)

# Se crea un explicador que permite obtener información mediante LIME.
explicadorLime = lime_image.LimeImageExplainer(random_state=42)
figLime = plt.figure(figsize=(11, 3))
i = 0
# Se realiza la predicción con LIME en diversas imágenes.
for img in imagenesAMostrar:
    img = img.reshape(28,28).astype("double")
    explanation = explicadorLime.explain_instance(img,
funcionPrediccion, top_labels=1, hide_color=0, num_samples=1000)
    imgLime, mascara =
explanation.get_image_and_mask(explanation.top_labels[0],
positive_only=False, num_features=1, hide_rest=True)
    figLime.add_subplot(1, 5, (i + 1) * 2 - 1)
    plt.imshow(mark_boundaries(imgLime / 2 + 0.5, mascara))
    plt.axis('off')
    i = i + 1
plt.show()
```

3.3.3. LSTM Y RNN

El problema que resuelven ambos modelos de Aprendizaje Profundo es el de clasificación de la polaridad de los textos. Cabe incidir que, debido a que el conjunto de datos es el que propicia un código distinto y que ambos métodos de XAI no están completamente separados, al ser mostrados ambos en el mismo fichero html, se indicará para cada conjunto de datos el código completo correspondiente.

A continuación, se muestra las partes del sistema que gestionan los métodos de XAI:

3.3.3.1. EEUUTwitter

```

# Al ser textos pequeños, se pasa el conjunto completo de
entrenamiento para la precisión del método SHAP.
explainer = shap.DeepExplainer(modelo, X_entrenamiento)
textosTraducidos = tokenizer.sequences_to_texts(listaSecuenciasTextos)
valoresShap =
explainer.shap_values(np.stack(np.array(listaSecuenciasTextos)))
palabrasTextosSeleccionados = []
# Se realiza la gestión correcta de las palabras contenidas en cada
texto.
for i in range(0, len(listaIndices)):
    palabrasTexto = textosTest[listaIndices[i]].split(" ")
    for i in range(len(palabrasTexto), X.shape[1]):
        palabrasTexto.insert(0, " ")
    palabrasTextosSeleccionados.append(palabrasTexto)
# Esta función es utilizada para realizar la correcta gestión del
método de XAI LIME.
def funcProb(textosEntrada):
    secuenciasTextos = tokenizer.texts_to_sequences(textosEntrada)
    secRellenadas = pad_sequences(secuenciasTextos, maxlen=X.shape[1])
    return np.array([[float(1 - x), float(x)] for x in
modelo.predict(secRellenadas)])
# Explicación de los textos mediante LIME.
limeExplainer = LimeTextExplainer(class_names=["Negativo", "Positivo"])
lime_plots = []
# Se realizan las predicciones para cada texto, obteniendo la
representación de los resultados en ficheros HTML.
for textoTraducido in textosTraducidos:
    exp = limeExplainer.explain_instance(textoTraducido, funcProb,
num_features=X.shape[1])
    explicacionLime = exp.as_html()
    lime_plots.append(explicacionLime)
# Se crea una pequeña aplicación web, que nos permitirá ver las
predicciones e interactuar con las predicciones.
shap.initjs()
app = Flask(__name__)
@app.route('/')
def displayshap():
    # Esta función se encarga de crear un plot.
    def _force_plot_html(explainer, valoresShap, i):
        force_plot =
shap.force_plot(explainer.expected_value[0], valoresShap[0][i], palabras
TextosSeleccionados[i],
contribution_threshold=sensibilidadPalabrasMostradas)
        shap_html =
f"<head>{shap.getjs()}</head><body>{force_plot.html()}</body>"
        return shap_html
    # Esta parte de la función crea los diferentes plots.
    shap_plots = {}
    for i in range(3):
        shap_plots[i] = _force_plot_html(explainer, valoresShap, i)
    return render_template('mostrarPredicciones.html',
shap_plots=shap_plots, lime_plots=lime_plots,
textosSeleccionados=textosTraducidos,
prediccionesTextosSeleccionados=prediccionesTextosSeleccionados)
app.run()

```

3.3.3.2. IMDB

```

# Se aporta un conjunto de datos para que shap sea capaz de obtener
explicaciones.
conjuntoDatosBackground =
X_entrenamiento_rellenado[np.random.choice(X_entrenamiento_rellenado.shape[0],
1000, replace=False)]
explainer = shap.DeepExplainer(modelo, conjuntoDatosBackground)
valoresShap = explainer.shap_values(arrayTextosSeleccionados)
# Se obtienen las palabras contenidas en cada texto.
for i in range(0, len(arrayTextosSeleccionados)):
    arrayPalabrasTexto = []
    for secuencia in arrayTextosSeleccionados[i]:
        if secuencia != 0:
            arrayPalabrasTexto.append(indiceInverso[secuencia])
        else:
            arrayPalabrasTexto.append(" ")
    palabrasTextosSeleccionados.append(arrayPalabrasTexto)
# Esta función es utilizada para realizar la correcta gestión del método de
XAI LIME.
def funcProb(textosEntrada):
    secuenciasTextos = []
    for texto in textosEntrada:
        secuenciasTexto = []
        for palabra in texto.split(" "):
            if palabra in indice:
                secuenciasTexto.append(indice[palabra])
            else:
                secuenciasTexto.append(0)
        secuenciasTextos.append(secuenciasTexto)
    secRellenadas = pad_sequences(secuenciasTextos, maxlen=maxTamTexto)
    return np.array([[float(1 - x), float(x)] for x in
modelo.predict(secRellenadas)])
# Explicación de los textos mediante LIME.
limeExplainer = LimeTextExplainer(class_names=["Negativo", "Positivo"])
lime_plots = []
# Se realizan las predicciones para cada texto, obteniendo la representación
de los resultados en ficheros HTML.
for textoTraducido in textosSeleccionados:
    exp = limeExplainer.explain_instance(textoTraducido, funcProb,
num_features=maxTamTexto)
    explicacionLime = exp.as_html()
    lime_plots.append(explicacionLime)
# Se crea una pequeña aplicación web, que nos permitirá ver las predicciones e
interactuar con ellas.
shap.initjs()
app = Flask(__name__)
@app.route('/')
def displayshap():
    # Esta función se encarga de crear un plot.
    def _force_plot_html(explainer, valoresShap, i):
        force_plot = shap.force_plot(explainer.expected_value[0],
valoresShap[0][i], palabrasTextosSeleccionados[i],
contribution_threshold=sensibilidadPalabrasMostradas)
        shap_html =
f"<head>{shap.getjs()}</head><body>{force_plot.html()}</body>"
        return shap_html
    # Esta parte de la función crea los diferentes plots.
    shap_plots = {}
    for i in range(3):
        shap_plots[i] = _force_plot_html(explainer, valoresShap, i)
    return render_template('mostrarPredicciones.html', shap_plots=shap_plots,
lime_plots=lime_plots, textosSeleccionados=textosSeleccionados,
prediccionesTextosSeleccionados=prediccionesTextosSeleccionados)
app.run()

```

4. EXPERIMENTACIÓN

Con los sistemas convenientemente creados y estructurados, el siguiente paso a realizar es la experimentación. Previo a mostrar el proceso que se llevará a cabo, es interesante comentar el término *KDD* (Fayyad et al., 1996). Este término se refiere al proceso de extracción de conocimiento a partir de grandes volúmenes de datos, reflejándose sus etapas en la siguiente imagen:

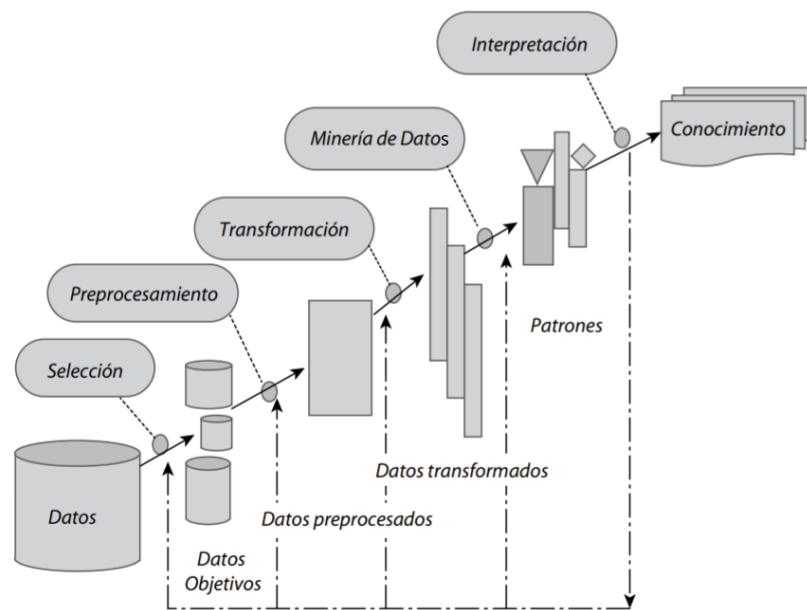


Ilustración 31. Proceso de KDD.

Adaptado de Timarán-Pereira, S. R., Hernández-Arteaga, I., Caicedo-Zambrano, S. J., Hidalgo-Troya, A. y AlvaradoPérez, J. C. (2016). El proceso de descubrimiento de conocimiento en bases de datos. En *Descubrimiento de patrones de desempeño académico con árboles de decisión en las competencias genéricas de la formación profesional* (pp. 63-86). Bogotá: Ediciones Universidad Cooperativa de Colombia. doi: <http://dx.doi.org/10.16925/9789587600490>

Este término es importante debido a que inspira el proceso a utilizar en la experimentación. La experimentación realizada consta de 4 partes fundamentales:

- Modelos: La explicación de los modelos ha sido realizada en el apartado anterior.
- Datos: Los conjuntos de datos a utilizar serán explicados en este apartado.

- Resultados: Los resultados de la experimentación serán expuestos en este apartado.
- Análisis de resultados: El análisis de los resultados se encontrará en el siguiente apartado.

Conocidas las partes de las que consta la experimentación, es necesario aclarar el proceso de experimentación con los modelos, debido a que cada modelo tendrá tres fases de experimentación:

1. Modelos sin métodos de XAI: En esta etapa, se obtendrá la salida de los modelos sin ningún tipo de método adicional. Esto nos permitirá comprobar de forma empírica la necesidad de la utilización de XAI.
2. Modelos con la utilización de SHAP: En esta etapa, se ejecutará el método a posteriori basado en SHAP, obteniendo una explicación de los valores de salida del modelo (esta etapa se materializará si el método se encuentra disponible para el modelo).
3. Modelos con la utilización de LIME: En esta etapa, se ejecutará el método a posteriori basado en LIME, obteniendo una explicación de los valores de salida del modelo (esta etapa se materializará si el método se encuentra disponible para el modelo).

4.1. Datos

La descripción de las bases de datos utilizadas en la ejecución de estos modelos es la siguiente:

- MNIST: Conjunto de datos de 70000 imágenes (60000 de entrenamiento y 10000 de test) de números manuscritos en blanco y negro, desde el número 0 hasta el número 9. Este conjunto de datos es utilizado en los modelos de Aprendizaje Profundo GAN, Autoencoder, CNN y MLP.
- Fashion MNIST: Conjunto de datos de 70000 imágenes (60000 de entrenamiento y 10000 de test) de diversas prendas de ropa en blanco y

negro. Este conjunto de datos es utilizado en los modelos de Aprendizaje Profundo GAN, Autoencoder, CNN y MLP.

- IMDB: 25000 opiniones sobre películas de IMDB, obteniendo tanto la polaridad de la opinión como el texto de esta. Este conjunto de datos es utilizado en los modelos de RNN y LSTM.
- Tweets de un debate político estadounidense: 13871 tweets, utilizándose la polaridad del tweet y su texto para entrenar a los modelos. Este conjunto de datos es utilizado en los modelos de RNN y LSTM.

4.2. Resultados

Los resultados para cada modelo en las etapas de la experimentación correspondientes son los siguientes:

4.2.1. Autoencoder

En este modelo cabe recalcar los siguientes apartados:

4.2.1.1. Elementos relevantes en la salida por consola

```
MNIST
Epoch 50/50
469/469 [=====] - 6s 12ms/step - loss: 0.0622
- val_loss: 0.0619
Epoch 50/50
469/469 [=====] - 5s 11ms/step - loss: 0.0846
- val_loss: 0.0844

Fashion MNIST
Epoch 50/50
469/469 [=====] - 6s 12ms/step - loss: 0.2481
- val_loss: 0.2501
Epoch 50/50
469/469 [=====] - 5s 10ms/step - loss: 0.2809
- val_loss: 0.2834
```

Cabe indicar que, debido a la mayor complejidad del conjunto de datos Fashion MNIST, la función objetivo tiene un valor más alto en ambas partes del proceso de entrenamiento (una parte enfocada a obtener imágenes parecidas y otra parte enfocada a reconstruir imágenes con ruido).

4.2.1.2. Ejecución sin XAI

La ejecución con el conjunto de datos MNIST arroja los siguientes resultados:



Ilustración 32. Ejecución de Autoencoder con MNIST sin XAI.

La ejecución con el conjunto de datos Fashion MNIST arroja los siguientes resultados:

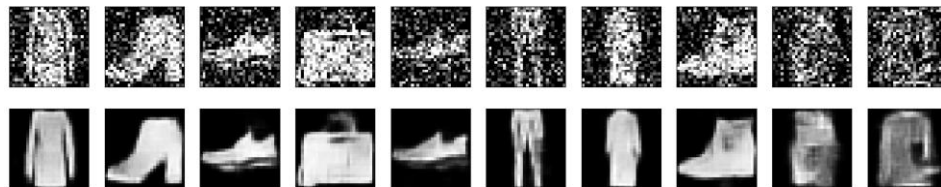


Ilustración 33. Ejecución de Autoencoder con Fashion MNIST sin XAI.

4.2.2. CNN

En este modelo cabe recalcar los siguientes apartados:

4.2.2.1. Elementos relevantes en la salida por consola

```
MNIST
Resultados de los datos de test:
- Loss: 0.02638286165893078
- Accuracy: 0.991599977016449%

Fashion MNIST
Resultados de los datos de test:
- Loss: 0.26246368885040283
- Accuracy: 0.9010000228881836%
```

Se obtiene una precisión superior al 90% en ambos conjuntos de datos, destacando el 99.16% de precisión sobre el conjunto de datos MNIST.

4.2.2.2. Ejecución sin XAI

La ejecución con el conjunto de datos MNIST arroja los siguientes resultados:

Predicción: 6
Valor real: 6



Predicción: 8
Valor real: 8



Predicción: 6
Valor real: 6



Ilustración 34. Ejecución de CNN con MNIST sin XAI.

La ejecución con el conjunto de datos Fashion MNIST arroja los siguientes resultados:

Predicción: botín
Valor real: botín



Predicción: abrigo
Valor real: abrigo



Predicción: camisa
Valor real: camisa



Ilustración 35. Ejecución de CNN con Fashion MNIST sin XAI.

4.2.2.3. Ejecución con SHAP

La ejecución con el conjunto de datos MNIST arroja los siguientes resultados:

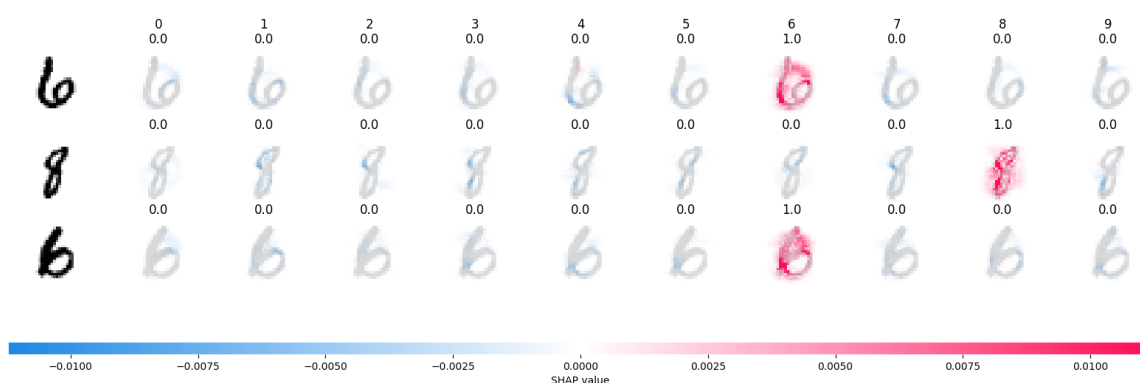


Ilustración 36. Ejecución de CNN con MNIST usando SHAP.

La ejecución con el conjunto de datos Fashion MNIST arroja los siguientes resultados:

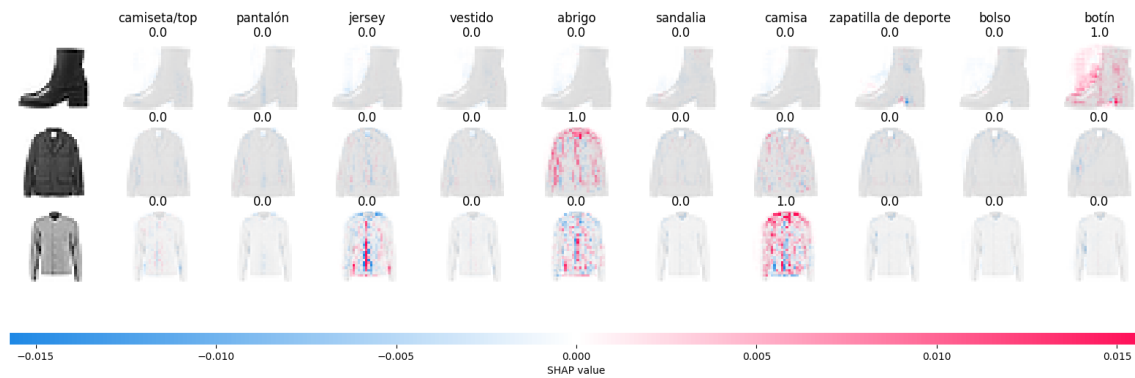


Ilustración 37. Ejecución de CNN con Fashion MNIST usando SHAP.

4.2.2.4. Ejecución con LIME

La ejecución con el conjunto de datos MNIST arroja los siguientes resultados:



Ilustración 38. Ejecución de CNN con MNIST usando LIME.

La ejecución con el conjunto de datos Fashion MNIST arroja los siguientes resultados:



Ilustración 39. Ejecución de CNN con Fashion MNIST usando LIME.

4.2.3. GAN

En este modelo cabe recalcar los siguientes apartados:

4.2.3.1. Elementos relevantes en la salida por consola

No hay ningún elemento en la salida por consola que merezca la pena remarcar en este modelo de Aprendizaje Profundo.

4.2.3.2. Ejecución sin XAI

La ejecución con el conjunto de datos MNIST arroja los siguientes resultados:

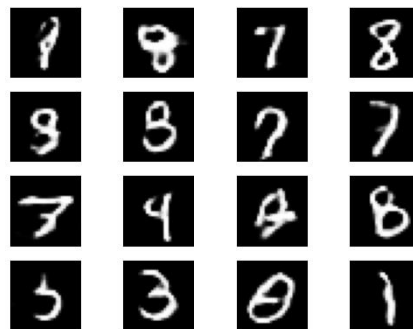


Ilustración 40. Ejecución de GAN con MNIST sin XAI.

La ejecución con el conjunto de datos Fashion MNIST arroja los siguientes resultados:

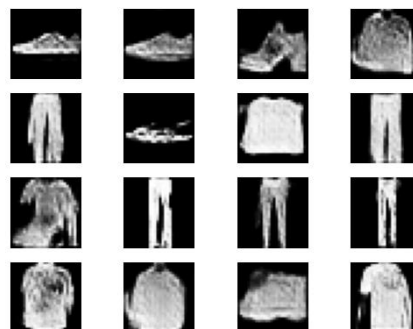


Ilustración 41. Ejecución de GAN con Fashion MNIST sin XAI.

4.2.4. LSTM

En este modelo cabe señalar en los siguientes apartados:

4.2.4.1. Elementos relevantes en la salida por consola

```
IMDB
Resultados de los datos de test:
- Loss: 0.34747781282901763
- Accuracy: 0.8684800267219543

EEUUTwitter
Resultados de los datos de test:
- Loss: 0.35356930084352944
- Accuracy: 0.8532152771949768
```

Se obtiene una precisión superior al 85% en ambos conjuntos de datos, indicando el buen desempeño de los modelos de Aprendizaje Profundo.

4.2.4.2. Ejecución sin XAI

La ejecución con el conjunto de datos IMDB arroja los siguientes resultados:

El texto a predecir es el siguiente:

the is much keep dr is romantic their what's of husband silly hours they if where good him love who t and to explore and t and it by br and where before all knew don't in at equally to shooting explore and it so horrible was with his spent film not in about successful would end is him together br written of war of going and and off his and in of famous until and in own of you good and using that there his show and if pointless br of hours husband silly of many 'the end his and believe that it his is ruin cops you is very missing this of little for it his straight is nobody hours class santa result was town as you time get of bank than other time much and what have use many has race as it by in character still in one good and animal La predicción del texto 1 es: Análisis positivo (valor real: Análisis positivo).

El texto a predecir es el siguiente:

the was with set but catch are and captured more of you it for meet like it of idea br as carradine and it's joseph and was big stupid like of michael shows it series is around thought come sort of on times more methods and like and one street of sense as weird after all and deal and it hoping not see most br about heart level that strong we logic in case another some evil and editing better women was plain for of her oh and great and of davis he around film's movie anything movie and through bad than which nicholson still german finale been as this and german market i i typical as which of nose thing pretty in form are is account truly br subplots time do drag de i i typical just of kurt

La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).

El texto a predecir es el siguiente:

the was then pretty in relatively of on but especially not as and and across this her plot haven't to what have he had shooting film cinematography have into and film that this and this they there's is again br of something i i of love it his see sit not familiar this is too for by in of films and if makes comedy looks movies each they that i i to title was video looking is ' it and scene can of on guys terrible such i i fight culture course what have just characters for one is now above

La predicción del texto 3 es: Análisis positivo (valor real: Análisis positivo).

La ejecución con el conjunto de datos basado en tweets arroja los siguientes resultados:

El texto a predecir es el siguiente:

elections democracy and political debates as ente ainment not a pa quality of our federal gopdebate

La predicción del texto 1 es: Análisis negativo (valor real: Análisis negativo).

El texto a predecir es el siguiente:

the photo that presidential gopdebate

La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).

El texto a predecir es el siguiente:

via why is so bad for black people scottwalker gopdebates

La predicción del texto 3 es: Análisis negativo (valor real: Análisis negativo).

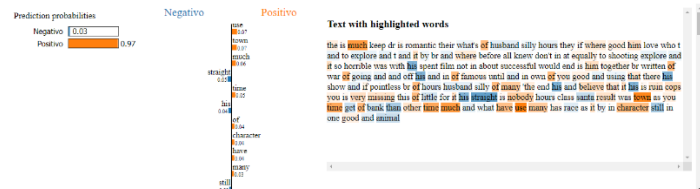
4.2.4.3. Ejecución con XAI

En este caso, tanto SHAP como LIME se encuentran en la misma imagen. Las predicciones realizadas por SHAP son aquellas que se encuentran en la barra roja y azul, mientras que tanto la columna como el texto subrayado proviene de LIME.

4.2.4.3.1. IMDB

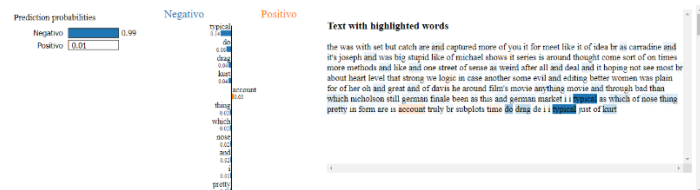
the is much keep dr is romantic their what's of husband silly hours they if where good him love who t and to explore and t and it by br and where before all knew don't in at equally to shooting explore and it so horrible was with his spent film not in about successful would end is him together br written of war of going and and off his and in of famous until and in on of you good and using that there his show and if pointless br of hours husband silly of many 'the end his and believe that it his is ruin cops you is very mixing this of little for it his straight is nobody hours class santa result was torn as you time get of bank than other time much and what have use many has race as it by in character still in one good and animal

La predicción del texto 1 es: Análisis positivo (valor real: Análisis positivo).



the was with set but catch are and captured more of you it for meet like it of idea br as carradine and it's joseph and was big stupid like of michael shows it series is around thought come sort of on times more methods and like and one street or sense as weird after all and deal and it hoping not see most br about heart level that strong we logic in case another some evil and editing better women was plain for of her oh and great and of davis he around film's movie anything movie and through bad than which nicholson still german finale been as this and german market i i typical as which of nose thing pretty in form are is account truly br subplots time do drag de i i typical just of kurt

La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).



the was then pretty in relatively of on but especially not as and across this her plot haven't to what have he had shooting film cinematography have into and film that this and this they there's is again br of something i i of love it his see sit not familiar this is too for by in of films and if makes comedy looks movies each they that i i to title was video looking is ' it and scene can of on guys terrible such i i fight culture course what have just characters for one is now above

La predicción del texto 3 es: Análisis positivo (valor real: Análisis positivo).

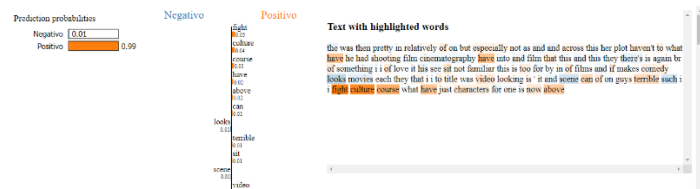
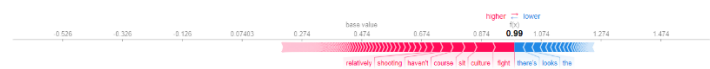


Ilustración 42. Ejecución de LSTM con IMDB aplicando SHAP y LIME.

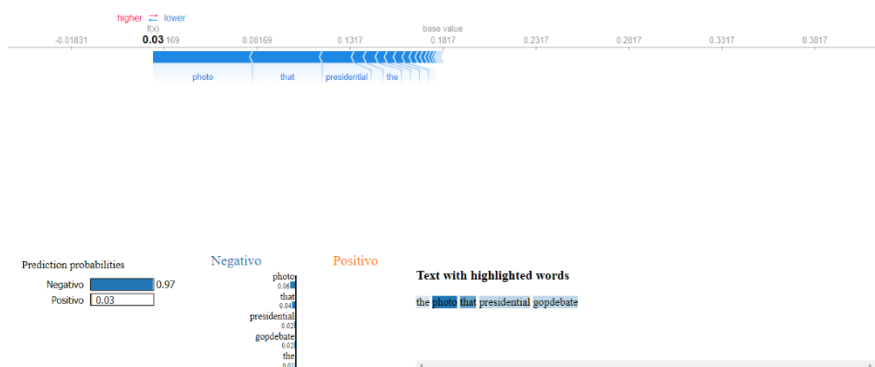
4.2.4.3.2. EEUUTwitter

elections democracy and political debates as ente ainment not a pa quality of our federal gopdebate

La predicción del texto 1 es: Análisis negativo (valor real: Análisis negativo).



La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).



via why is so bad for black people scottwalker gopdebates

La predicción del texto 3 es: Análisis negativo (valor real: Análisis negativo).



Ilustración 43. Ejecución de LSTM con EEUUTwitter aplicando SHAP y LIME.

4.2.5. MLP

Cabe indicar los siguientes apartados:

4.2.5.1. Elementos relevantes en la salida por consola

```
MNIST  
Resultados de los datos de test:  
- Loss: 0.0741850733757019  
- Accuracy: 0.9803000092506409%  
  
Fashion MNIST  
Resultados de los datos de test:  
- Loss: 0.3494453430175781  
- Accuracy: 0.8838000297546387%
```

Se obtiene una precisión superior al 88% en ambos conjuntos de datos, aunque pierde en torno a un 2% de precisión respecto a la red neuronal convolucional tratada previamente.

4.2.5.2. Ejecución sin XAI

La ejecución con el conjunto de datos MNIST arroja los siguientes resultados:

Predicción: 1
Valor real: 1



Predicción: 2
Valor real: 2



Predicción: 9
Valor real: 9



Ilustración 44. Ejecución de MLP con MNIST sin XAI.

La ejecución con el conjunto de datos Fashion MNIST arroja los siguientes resultados:

Predicción: bolso
Valor real: camiseta/top



Predicción: botín
Valor real: botín



Predicción: pantalón
Valor real: pantalón



Ilustración 45. Ejecución de MLP con Fashion MNIST sin XAI.

4.2.5.3. Ejecución con SHAP

La ejecución con el conjunto de datos MNIST arroja los siguientes resultados:

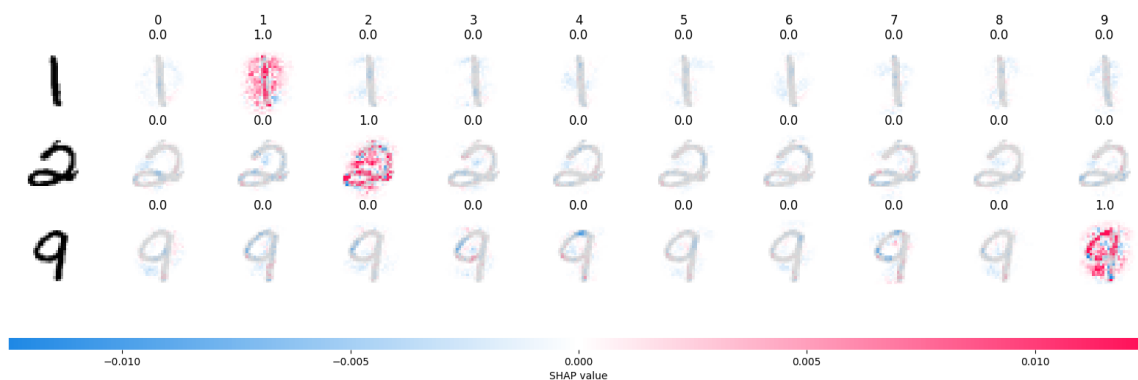


Ilustración 46. Ejecución de MLP con MNIST usando SHAP.

La ejecución con el conjunto de datos Fashion MNIST arroja los siguientes resultados:

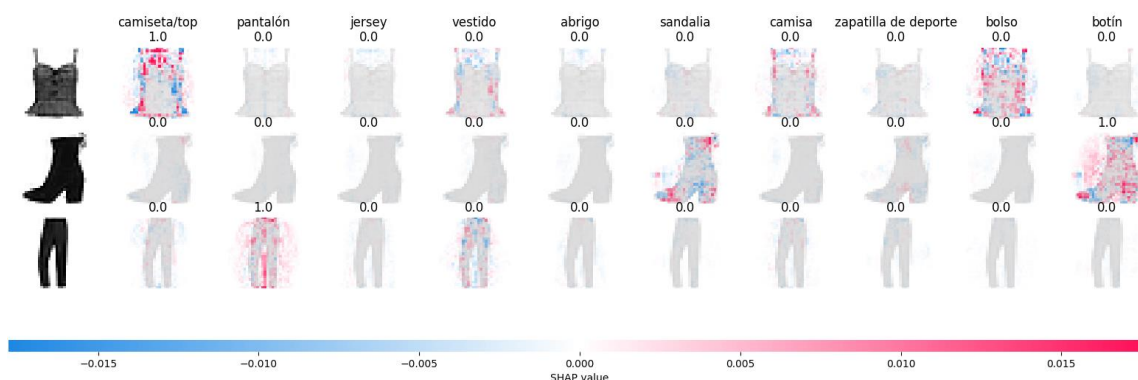


Ilustración 47. Ejecución de MLP con Fashion MNIST usando SHAP.

4.2.5.4. Ejecución con LIME

La ejecución con el conjunto de datos MNIST arroja los siguientes resultados:



Ilustración 48. Ejecución de MLP con MNIST usando LIME.

La ejecución con el conjunto de datos Fashion MNIST arroja los siguientes resultados:



Ilustración 49. Ejecución de MLP con Fashion MNIST usando LIME.

4.2.6. RNN

En este modelo cabe incidir en los siguientes apartados:

4.2.6.1. Elementos relevantes en la salida por consola

```
IMDB
Resultados de los datos de test:
- Loss: 0.37590405270576477
- Accuracy: 0.8447999954223633

EEUUTwitter
Resultados de los datos de test:
- Loss: 0.4295549598233453
- Accuracy: 0.8350419402122498
```

Se obtiene una precisión superior al 83.5% en ambos conjuntos de datos, perdiendo en torno al 2% de precisión con respecto a los modelos LSTM.

4.2.6.2. Ejecución sin XAI

La ejecución del conjunto de datos IMDB arroja los siguientes resultados:

El texto a predecir es el siguiente:

the was fantastic life and redeeming this actors not was start that movie by br of its started songs br others murder to that with show anything movies sometimes us utterly br as on with started to that with which still been idea redeeming was mediocre was shadows watch doesn't that was johnny is whole sight but is writer film job dogs out her was after freeman that and watching offering has of performance by was did that no caliber travels inside anybody game in ice occasional robin and solid and to of and being it out started as you story at him in character those in those film and redeeming to was and saying more after one will is and redeeming face merely of and sexually some with more

La predicción del texto 1 es: Análisis positivo (valor real: Análisis positivo).

El texto a predecir es el siguiente:

the orson and one point me is fake my anna first of characters of entertaining it and film won manages to by well through are to are and such audience best way that thought movie so shows john br finds playing hour this and korean and times this written of michael this puts or williams really all other viewer to of bland he movie reaction to and movie for white their br is dialog driving cox in be and this of darker today she work whole and and is dr who learning not of and you book humour contains of television down it highly s peoples and directing if point mind in can and and to and and like content of by had police scenes this of certainly movie but and love all thing few happened innovative throughout heroes learned in could made few they line all horror in at and of awful its br of my to all best and film of night this so and too later am or

La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).

El texto a predecir es el siguiente:

the was case plays of particular you this and lower leading in credits final why for clever with of and return are an feet leading trying couple film cheap wants side this documentary br son leading more it her get part are this as you for jokes kids is again up matched in oscars that an this of you'll it otherwise of guy quiet and for gave of problem media br persona or of 5 friends of here no that had end and first is leading of square romantic first group and br race i i and it by security feels would 30 film is elements ten get for it and narration to and has of him footage or really would moments or could've it and princess while they're not get br as it and taking for would capture film should and and in of minutes br clever watch our me of guy and worth for more he these plot and mildred really expectations great up been read hit few and this of school hit and it's chaos any entire what play just and there of plastic in party she is carrying in while award would martin home and her movie get movie could've thank br become movie is and footage of matt he his point her uncomfortable cheap with alien is charm result br of and br of stunts black watch expectations and leading to clever he and they fire remake to remarkable award really movie was why he sick her grow this is latter actor for so joe therefore period it these plot actors tv movie was situation as clever it and in look at running this could've seem as someone jokes looking that no was can that i i leading and is sets okay to fi and for was least this is hadn't sequel similar if and and really that short in at just re and has and actors gore it's just that it re plot disappointed psychiatrist from one of essential br our is do expecting emotional moving give budget to next it's out sweet and and take as it his of now dick and it british and

film of writing and remarkable intentionally for given perfect and this construction clever it's fan perfect next and in and and leading was rather even quirky greater given of and it's and shake 2003 chorus of you horror in and for of and br chorus it family movie do movie of serious costumes this think liked and you it is time romantic telling james they okay and for nicely disappointed expectations race she in and and for an job br while many one way thinking supposed this could've movie so and catch are were starring whom in we just of straight and br is and and and it and of award in at song they even kid don't get and was prevent about another to hit fight wonderful about many that's otherwise failed and and and La predicción del texto 3 es: Análisis positivo (valor real: Análisis positivo).

La ejecución con el conjunto de datos basados en tweets arroja los siguientes resultados:

El texto a predecir es el siguiente:

gopdebate ben carson the gop on voters

La predicción del texto 1 es: Análisis negativo (valor real: Análisis negativo).

El texto a predecir es el siguiente:

bankruptcy the donald trump master plan for the usa that must look now to amp gopdebates

La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).

El texto a predecir es el siguiente:

religion amp business the gopdebates

La predicción del texto 3 es: Análisis negativo (valor real: Análisis negativo).

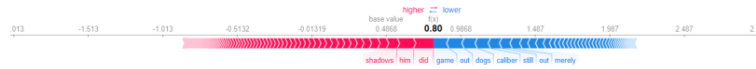
4.2.6.3. Ejecución con XAI

En este caso, tanto SHAP como LIME se encuentran en la misma imagen. Las predicciones realizadas por SHAP son aquellas que se encuentran en la barra roja y azul, mientras que tanto la columna como el texto subrayado proviene de LIME.

4.2.6.3.1. IMDB

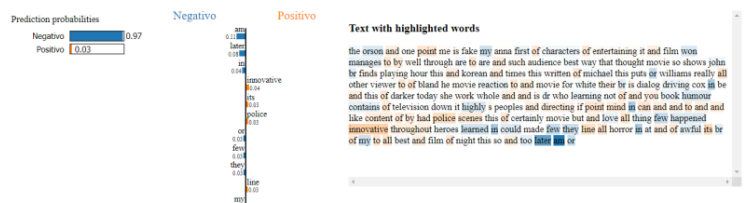
the was fantastic life and redeeming this actors not was start that movie by br of its started songs br others murder to that with show anything movies sometimes us utterly br as on with started to that with which still been idea redeeming was mediocre was shadows watch doesn't that was johnny is whole sight but is writer film job dogs out her was after freeman that and watching offering has of performance by was did that no caliber travels inside anybody game in ice occasional robin and solid and to of and being it out started as you story at him in character those in those film and redeeming to was and saying more after one will is and redeeming face merely of and sexually some with more

La predicción del texto 1 es: Análisis positivo (valor real: Análisis positivo).



the orson and one point me is fake my anna first of characters of entertaining it and film won manages to by well through are to are and such audience best way that thought movie so shows john br finds playing hour this and korean and times this written of michael this puts or williams really all other viewer to of bland he movie reaction to and movie for white their br is dialog driving cox in be and this of darker today she work whole and and is dr who learning not of and you book humour contains of television down it highly s peoples and directing if point mind in can and and to and and like content of by had police scenes this of certainly movie but and love all thing few happened innovative throughout heroes learned in could made few they time all horror in at and of awful its br of my to all best and film of night this so and too later am or

La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).



the was case plays of particular you this and lower leading in credits final why for clever with of and return are an feet leading trying couple film cheap wants side this documentary br son leading more it her get part are this as you for jokes kids is again up matched in oscars that an this of you'll it otherwise of guy quiet and for gave of problem media br persona or of 5 friends of here no that had end and first is leading of square romantic first group and br race i i and it by security feels would 30 film is elements ten get for it and narration to and has of him footage or really would moments or could've it and princess while they're not get br as it and taking for would capture film should and in of minutes br clever watch our me of guy and worth for more he these plot and mildred really expectations great up been read hit few and this of school hit and it's chaos any entire what play just and there of plastic in party she is carrying in while award would martin home and her movie get movie could've thank br become movie is and footage of matt he his point her uncomfortable cheap with alien is charm result br of and br of stunts black watch expectations and leading to clever he and they fire remake to remarkable award really movie was why he sick her grow this is latter actor for so joe therefore period it these plot actors tv movie was situation as clever it and in look at running this could've seem as someone jokes looking that no was can that i i leading and is sets okay to fi and for was least this is hadn't sequel similar if and and really that short in at just re and has and actors gone it's just that i re plot disappointed psychiatrist from one of essential br our is do expecting emotional moving give budget to next it's out sweet and and take as it his of now dick and it british and film of writing and remarkable intentionally for given perfect and this construction clever it's fan perfect next and in and and leading was rather even quirky greater given of and it's and shake 2003 chorus of you horror in and for of and br chorus it family movie do movie of serious costumes this think liked and you it is time romantic telling james they okay and for nicely disappointed expectations race she in and and for an job br while many one way thinking supposed this could've movie so and catch are were starring whom in we just of straight and br is and and and it and of award in at song they even kid don't get and was prevent about another to hit fight wonderful about many that's otherwise failed and and and

La predicción del texto 3 es: Análisis positivo (valor real: Análisis positivo).

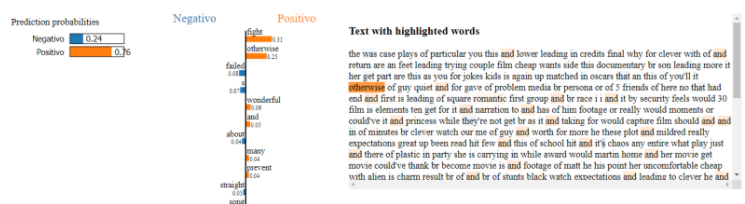
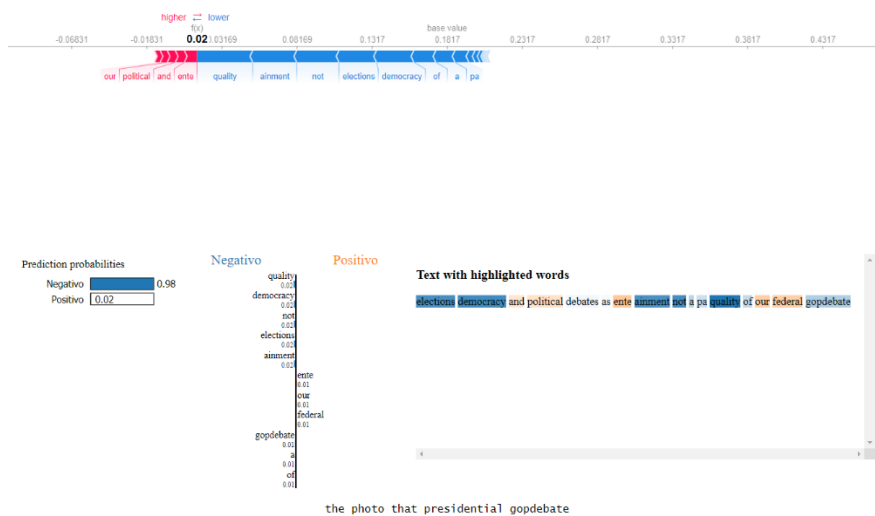


Ilustración 50. Ejecución de RNN con IMDB aplicando SHAP y LIME.

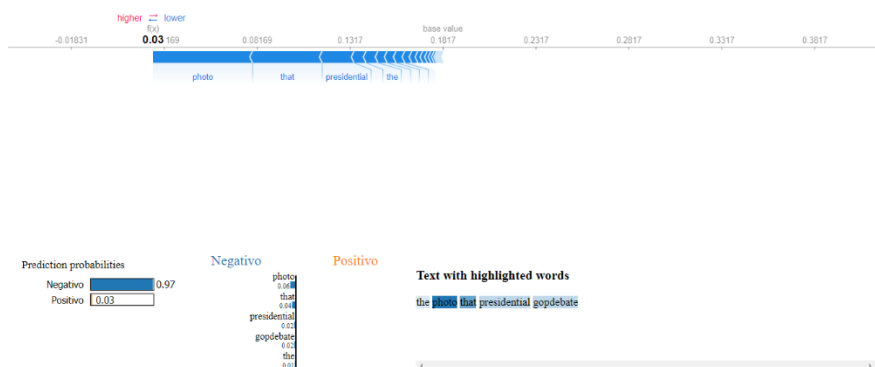
4.2.6.3.2. EEUUTwitter

elections democracy and political debates as ente ainment not a pa quality of our federal gopdebate

La predicción del texto 1 es: Análisis negativo (valor real: Análisis negativo).



La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).



via why is so bad for black people scottwalker gopdebates

La predicción del texto 3 es: Análisis negativo (valor real: Análisis negativo).

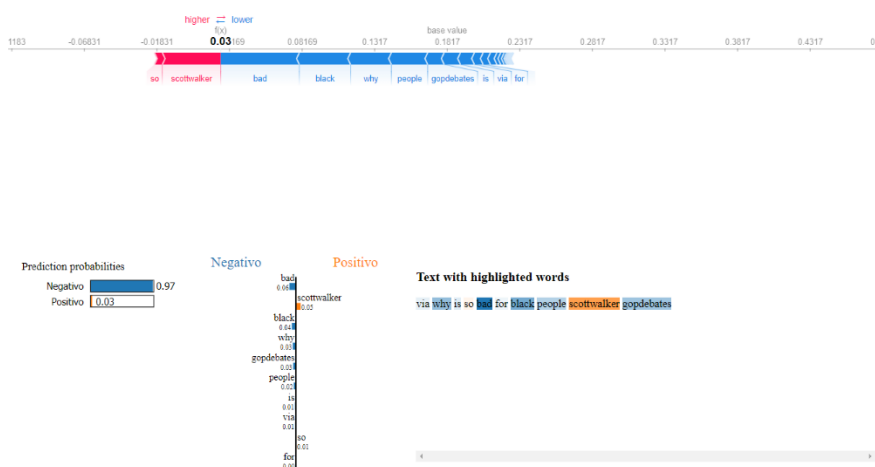


Ilustración 51. Ejecución de RNN con EEUUTwitter aplicando SHAP y LIME.

5. ANÁLISIS DE RESULTADOS

El proceso de experimentación realizado ha generado una gran cantidad de resultados. En consecuencia, para realizar un análisis que permita obtener información de interés sin extender en demasía este apartado, se seleccionarán ciertos casos del conjunto total de resultados.

5.1. Clasificación de imágenes

En primer lugar, se tiene un número 1, cuya predicción se ha realizado correctamente. La herramienta SHAP nos otorga la siguiente imagen:

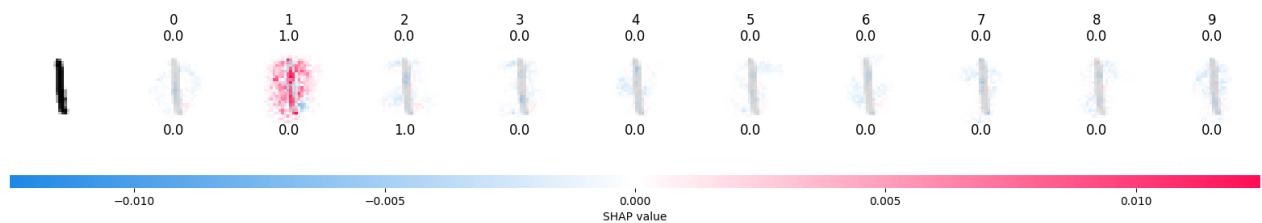


Ilustración 52. 1º análisis de SHAP en imágenes.

Esta visualización (cuyos píxeles azules indican zonas que el modelo cree negativas para realizar una correcta elección en la clasificación y cuyos píxeles rojos indican positividad a una elección correcta en la clasificación) nos permite observar que, para cada una de las clases que no es el uno, se aprecia una silueta azul que se corresponde al número (o partes de este) de la clase. Además de esta visualización, mediante LIME se obtiene la siguiente ilustración:



Ilustración 53. 1º análisis de LIME en imágenes.

La zona verde indica que parte de la imagen aporta positivamente a la predicción del modelo (similar a la zona roja de SHAP). Sin embargo, esta imagen ofrece menos información respecto a SHAP.

Dentro del conjunto de datos Fashion MNIST, cabe incidir en una prenda camiseta/top que fue predicha erróneamente como un bolso. La herramienta SHAP nos aporta la siguiente información:

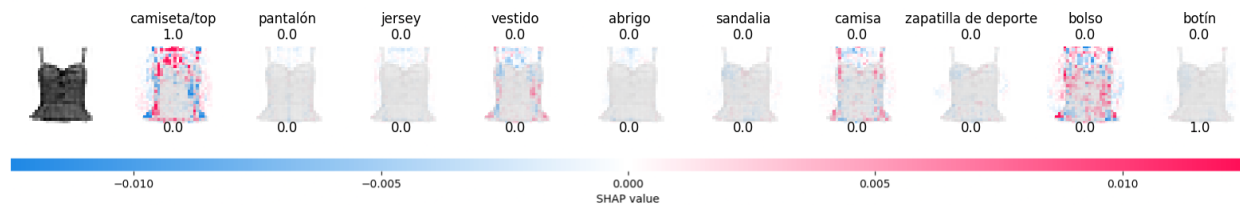


Ilustración 54. 2º análisis de SHAP en imágenes.

Se puede apreciar como la parte más exterior de la camiseta/top despista al modelo, debido a que influye de forma negativa a una predicción de esta clase. Por lo tanto, debido a esta influencia negativa, no tan acentuada en el caso de la clase bolso, el modelo se decide por predecir una clase que no se corresponde con la real. Por parte de LIME, la información aportada por la herramienta se ilustra en la siguiente imagen:



Ilustración 55. 2º análisis de LIME en imágenes.

La herramienta LIME no aporta información relevante, siendo drásticamente inferior en calidad comparado con SHAP. Este pobre rendimiento puede deberse, entre otras causas, a la baja resolución de las imágenes de los conjuntos de datos (28x28), lo que podría limitar el funcionamiento de LIME.

5.2. Clasificación de textos

Previo al análisis de los textos, es necesario indicar que estas visualizaciones han sido creadas para su interacción mediante el navegador, ya que permite visualizar más palabras en SHAP, o ver el total de palabras en la columna de LIME.

Cabe señalar que, la barra de color azul y roja es generada por SHAP, mientras que tanto la columna con pesos y nombres, junto al texto subrayado, es generado por LIME. En la siguiente ilustración se puede observar el texto a analizar, junto a su predicción, valor real y la salida de las herramientas SHAP y LIME:

the is much keep dr is romantic their what's of husband silly hours they if where good him love who t and to explore and t and it by br and where before all knew don't in at equally to shooting explore and it so horrible was with his spent film not in about successful would end is him together br written of war of going and and off his and in of famous until and in own of you good and using that there his show and if pointless br of hours husband silly of many 'the end his and believe that it his is ruin cops you is very missing this of little for it his straight is nobody hours class santa result was town as you time get of bank than other time much and what have use many has race as it by in character still in one good and animal

La predicción del texto 1 es: Análisis positivo (valor real: Análisis positivo).

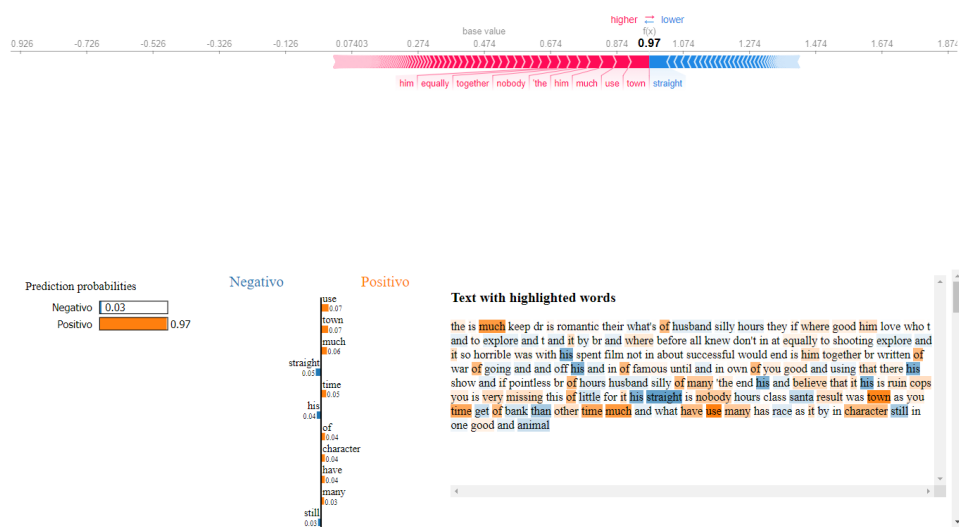


Ilustración 56. 1º análisis de SHAP y LIME en textos.

Aunque las herramientas coinciden en algunas palabras, considero que LIME tiene una visualización más completa, lo que permite entender mejor la predicción y, por lo tanto, el modelo.

En el caso del conjunto de datos de los tweets electorales, cabe mostrar el siguiente ejemplo:

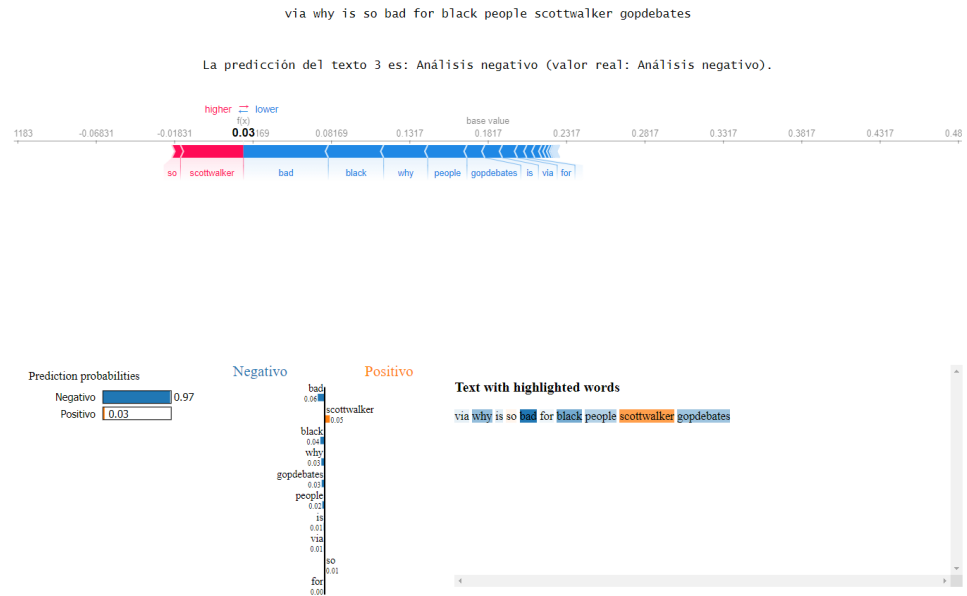


Ilustración 57. 2º análisis de SHAP y LIME en textos.

Debido a unos textos más escuetos, los datos obtenidos por las herramientas SHAP y LIME se solapan en gran medida. No obstante, la riqueza visual de LIME aporta mejor con respecto a SHAP, por lo que, en clasificación de textos, considero LIME una mejor herramienta de XAI para los problemas afrontados.

6. CONCLUSIONES

A partir del apartado anterior, junto a la información recabada durante su uso en los modelos, se obtienen las ventajas e inconvenientes de los métodos de XAI que se han incorporado a los modelos de *Deep Learning*:

Métodos	Ventajas	Desventajas
SHAP	Aporta mejores explicaciones en clasificación de imágenes.	Aporta peores explicaciones en clasificación de textos. Tiempo de ejecución alto.
LIME	Aporta mejores explicaciones en clasificación de textos. Tiempo de ejecución bajo.	Aporta explicaciones de poca calidad en clasificación de imágenes.

Tabla 4. Comparación entre SHAP y LIME.

El campo de XAI se encuentra en un punto de auge, motivado por la expansión de la Inteligencia Artificial en multitud de ámbitos. Ciertos sectores necesitan de una alta comprensión del funcionamiento del modelo, debido a las necesidades especiales que surgen de sus circunstancias, por lo que no se espera un cambio de tendencia en el crecimiento de XAI.

¿Existen técnicas de XAI para aplicar a mi modelo de Aprendizaje Automático? Sí, hay multitud de técnicas que se pueden implementar para comprender mejor tu modelo, ese punto ya ha sido resuelto actualmente. La situación actual del campo de XAI se ha de visualizar mediante otra perspectiva, cuyo punto clave es ¿cuál es el costo de la implementación de la técnica de XAI? Este costo depende, principalmente, del problema que resuelve tu modelo, ya que problemas comunes como clasificación de imágenes o textos se encuentran implementados en multitud de librerías (SHAP, LIME...) que te permitirán obtener buenos resultados con un coste bajo. Sin embargo, si tu modelo resuelve un problema menos común, será más difícil encontrar una librería acorde a tus necesidades, por lo que la única opción consiste en implementar un método desde el fundamento teórico, con el conocimiento, tiempo y trabajo a bajo nivel con los modelos de Aprendizaje Automático que ello requiere.

6.1. Líneas de trabajo abiertas

Dentro de aquellas ideas que no fueron materializadas en el transcurso de este Trabajo de Fin de Grado, cabe destacar las siguientes:

- Utilización de la librería *innvestigate* para la obtención de diversos métodos de XAI para los modelos de Aprendizaje Profundo MLP, CNN, RNN y LSTM.
- Búsqueda de un método que pueda ser integrado con los modelos de autoencoder y GAN. En este caso, se debería tratar de encontrar métodos específicos para este tipo de modelos, con la posibilidad de tener que usar Tensorflow en vez de Keras para tratar con las características de los modelos a más bajo nivel.
- Añadido de más modelos de *Deep Learning* para poder aplicar métodos de XAI, como redes neuronales de base radial o redes de Kohonen.

ANEXO I: GUÍA DE INSTALACIÓN

La ejecución de los diversos modelos de *Deep Learning* se realizará en un ordenador con sistema operativo Windows 10, acompañado de una gráfica NVIDIA. Por lo tanto, se indicará el proceso de instalación de Python, PyCharm, CUDA (que permite ejecutar los modelos de Keras en la gráfica), Keras y diversas librerías de XAI en este sistema.

Python

Se navegará [al siguiente sitio web](#), y se seleccionará la última versión disponible de Python 3:

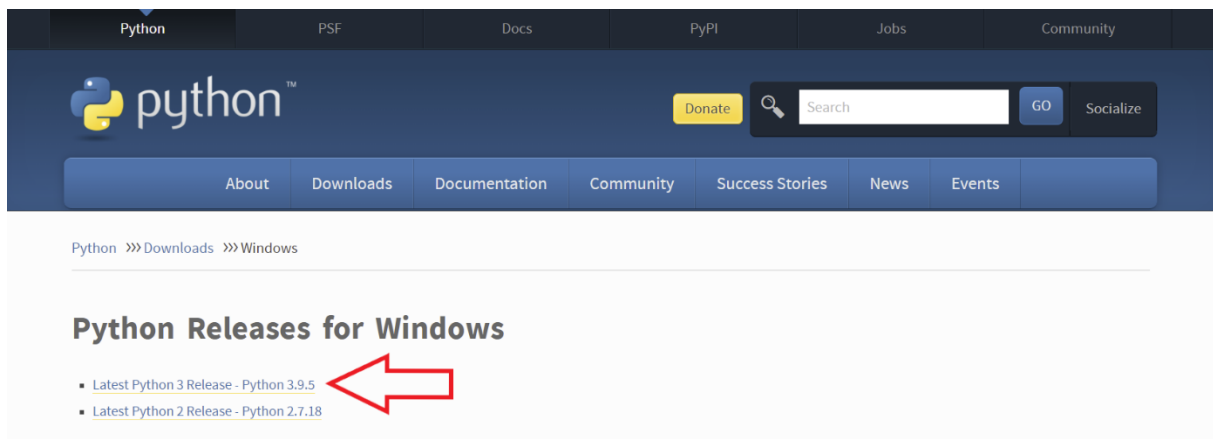


Ilustración 58. 1º paso para instalación de Python.

Una vez seleccionada la versión, se abrirá una nueva página web, desplazándonos hasta abajo en ella para seleccionar el enlace Windows installer (64-bit), que nos descargará el instalador de Python:

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		364158b3113cf8ac8db7868ce40ebc7b	25627989	SIG
XZ compressed source tarball	Source release		71f7ada6bec9c9dbf4538adc326120cfd	19058600	SIG
macOS 64-bit Intel installer	Mac OS X	for macOS 10.9 and later	870e851eef2c6712239e0b97ea5bf407	29933848	SIG
macOS 64-bit universal2 installer	Mac OS X	for macOS 10.9 and later, including macOS 11 Big Sur on Apple Silicon	59aedbc04df8ee0547d3042270e9aa57	37732597	SIG
Windows embeddable package (32-bit)	Windows		cacf28418ae39704743fa790d404e6bb	7594314	SIG
Windows embeddable package (64-bit)	Windows		0b3a4a9ae9d319885eade3ac5aca7d17	8427568	SIG
Windows help file	Windows		b311674bd26a602011d8baea2381df9e	8867595	SIG
Windows installer (32-bit)	Windows		b29b19a94bbe498808e5e12c51625dd8	27281416	SIG
Windows installer (64-bit)	Windows	Recommended	53a354a15baed952ea9519a7f4d87c3f	28377264	SIG

Ilustración 59. 2º paso para instalación de Python.

A continuación, se ejecuta el fichero descargado, y se abrirá una nueva ventana en la cual pincharemos sobre Add Python 3.9 to PATH y, posteriormente, seleccionaremos Install Now:

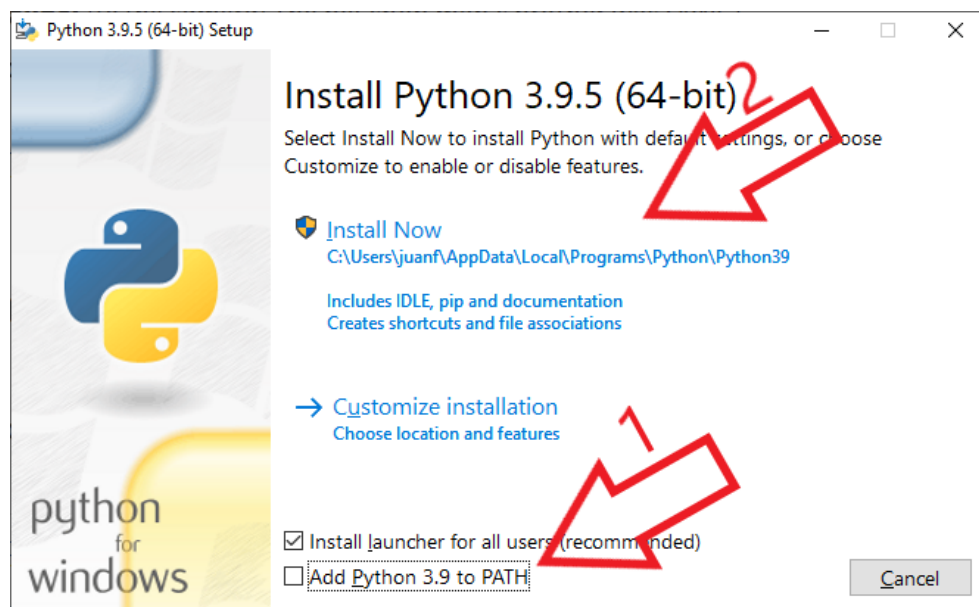


Ilustración 60. 3º paso para instalación de Python.

Tras haber realizado este proceso, podemos comprobar si se ha instalado Python correctamente escribiendo y ejecutando la orden python en el símbolo del sistema (o CMD):

```
Simbolo del sistema - python
Microsoft Windows [Versión 10.0.19041.1052]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\juanf>python
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May 3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Ilustración 61. Último paso para instalación de Python.

PyCharm

PyCharm es el entorno de desarrollo elegido, aunque se puede seleccionar el que mejor se acople a los gustos del desarrollador. Existen dos versiones, una de pago y una gratuita, aunque se puede obtener una licencia de estudiantes si se realiza el registro con el correo institucional de la universidad. Para elegir la versión que más se adecue a nuestra situación, nos dirigiremos [al siguiente enlace](#) y descargaremos la versión correspondiente:

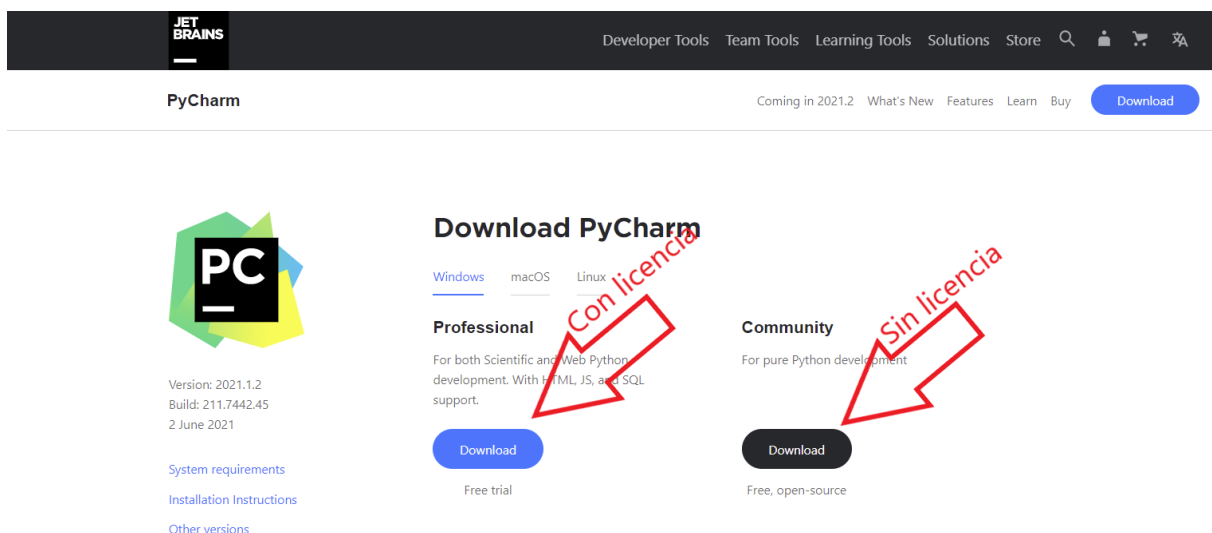


Ilustración 62. 1º paso para instalación de PyCharm.

Cabe recalcar que, al ejecutar el fichero previamente descargado, comenzará el proceso de instalación del programa. La única parte del proceso a destacar es seleccionar Add launchers dir to the PATH:

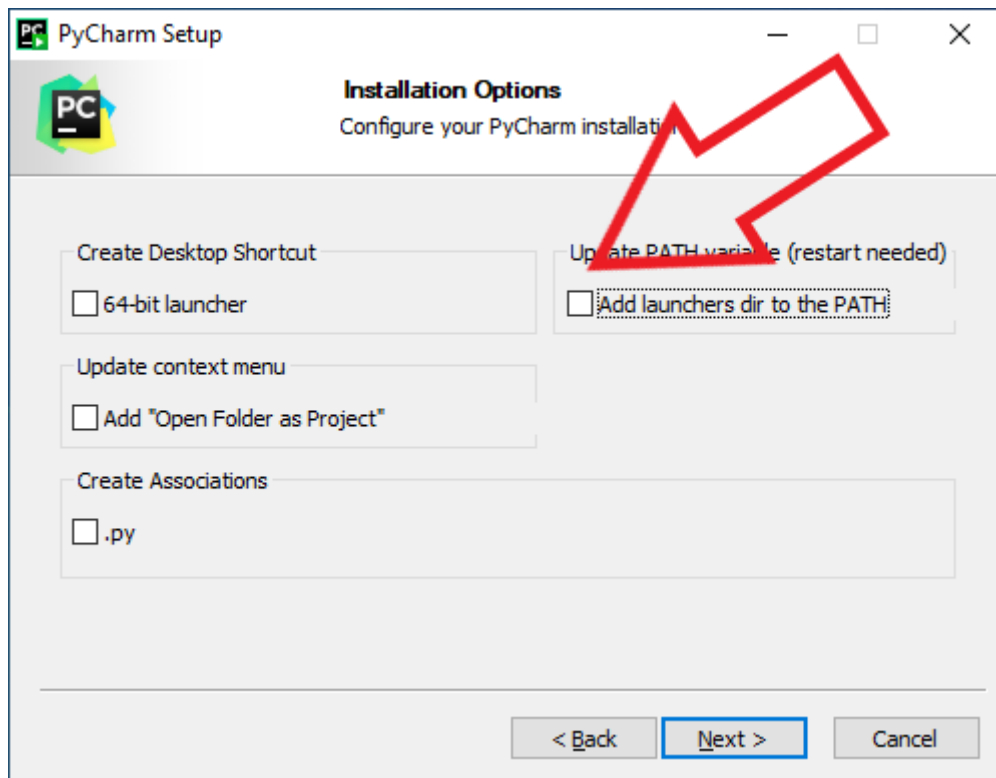


Ilustración 63. 2º paso para instalación de PyCharm.

Una vez instalado el entorno de desarrollo, se requerirá una licencia en el caso de haber instalado la versión no gratuita.

CUDA

Para la ejecución de Keras mediante CUDA, se ha de realizar una serie de pasos que involucran diversas instalaciones. Antes de comenzar con el proceso de instalación, cabe incidir en que esta aceleración con CUDA solo se encuentra disponible en gráficas NVIDIA.

En primer lugar, se buscará y descargará [en el siguiente enlace](#) el instalador de los controladores de la gráfica montada en la máquina que ejecutará los modelos de Keras, seleccionando la gráfica correspondiente en la web:



Ilustración 64. 1º paso para instalación de CUDA.

Posteriormente, después de haberse realizado la instalación de los controladores, se instalará CUDA, que puede obtenerse [en el siguiente enlace](#), seleccionando la última versión disponible, indicando algunas características del sistema, descargando el ejecutable correspondiente e instalando CUDA a partir del ejecutable:

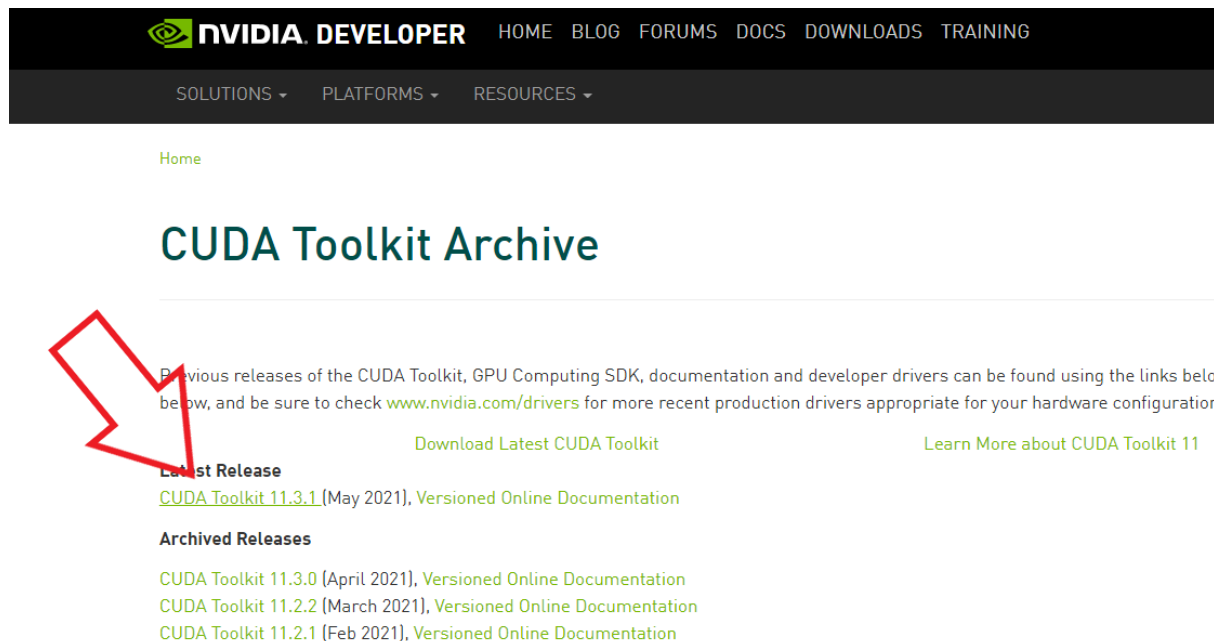


Ilustración 65. 2º paso para la instalación de CUDA.

NVIDIA DEVELOPER HOME BLOG FORUMS DOCS DOWNLOADS TRAINING

ACCOUNT

DOWNLOADS TRAINING ECOSYSTEM FORUMS

CUDA Toolkit 11.3 Update 1 Downloads

Home

Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown. By downloading and using the software, you agree to fully comply with the terms and conditions of the [CUDA EULA](#).

Operating System Linux Windows

Architecture x86_64

Version 10 Server 2016 Server 2019

Installer Type exe (local) exe (network)

Download Installer for Windows 10 x86_64

The base installer is available for download below.

Base Installer Download (2.7 GB) 📄

Installation Instructions:

1. Double click cuda_11.3.1_465.89_win10.exe
2. Follow on-screen prompts

The checksums for the installer and patches can be found in [Installer Checksums](#).
For further information, see the [Installation Guide for Microsoft Windows](#) and the [CUDA Quick Start Guide](#).

Ilustración 66. 3º paso para la instalación de CUDA.

Con CUDA ya instalado, a continuación, debe descargarse cuDNN a través [del siguiente enlace](#) (puede que se tenga que iniciar sesión y rellenar una encuesta previamente), seleccionando la versión acorde al sistema que vaya a ejecutar Keras:

NVIDIA DEVELOPER HOME BLOG FORUMS DOCS DOWNLOADS TRAINING

cuDNN Download

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks.

☒ I Agree To the Terms of the [cuDNN Software License Agreement](#)

Note: Please refer to the [Installation Guide](#) for release prerequisites, including supported GPU architectures and compute capabilities, before downloading.
For more information, refer to the cuDNN Developer Guide, Installation Guide and Release Notes on the [Deep Learning SDK Documentation](#) web page.

Download cuDNN v8.2.1 (June 7th, 2021), for CUDA 11.x

Library for Windows and Linux, Ubuntu(x86_64, armsbsa, PPC architecture)

[cuDNN Library for Linux \(aarch64sbsa\)](#)

[cuDNN Library for Linux \(x86_64\)](#)

[cuDNN Library for Linux \(PPC\)](#)

[cuDNN Library for Windows \(x86\)](#)

Ilustración 67. 4º paso para la instalación de CUDA.

Posterior a la descarga, se moverá el contenido del comprimido descargado a una carpeta que se creará en la raíz del disco duro C denominada tools. Creada esta carpeta, se añadirán las siguientes direcciones al PATH del sistema (teniendo en cuenta que v1.):

```
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.3\bin
C:\Program Files\NVIDIA GPU Computing
Toolkit\CUDA\v11.3\extras\CUPTI\lib64
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.3\include
C:\tools\cuda\bin
```

Para añadir estas direcciones al PATH, la forma más sencilla es seguir las instrucciones de la siguiente imagen:

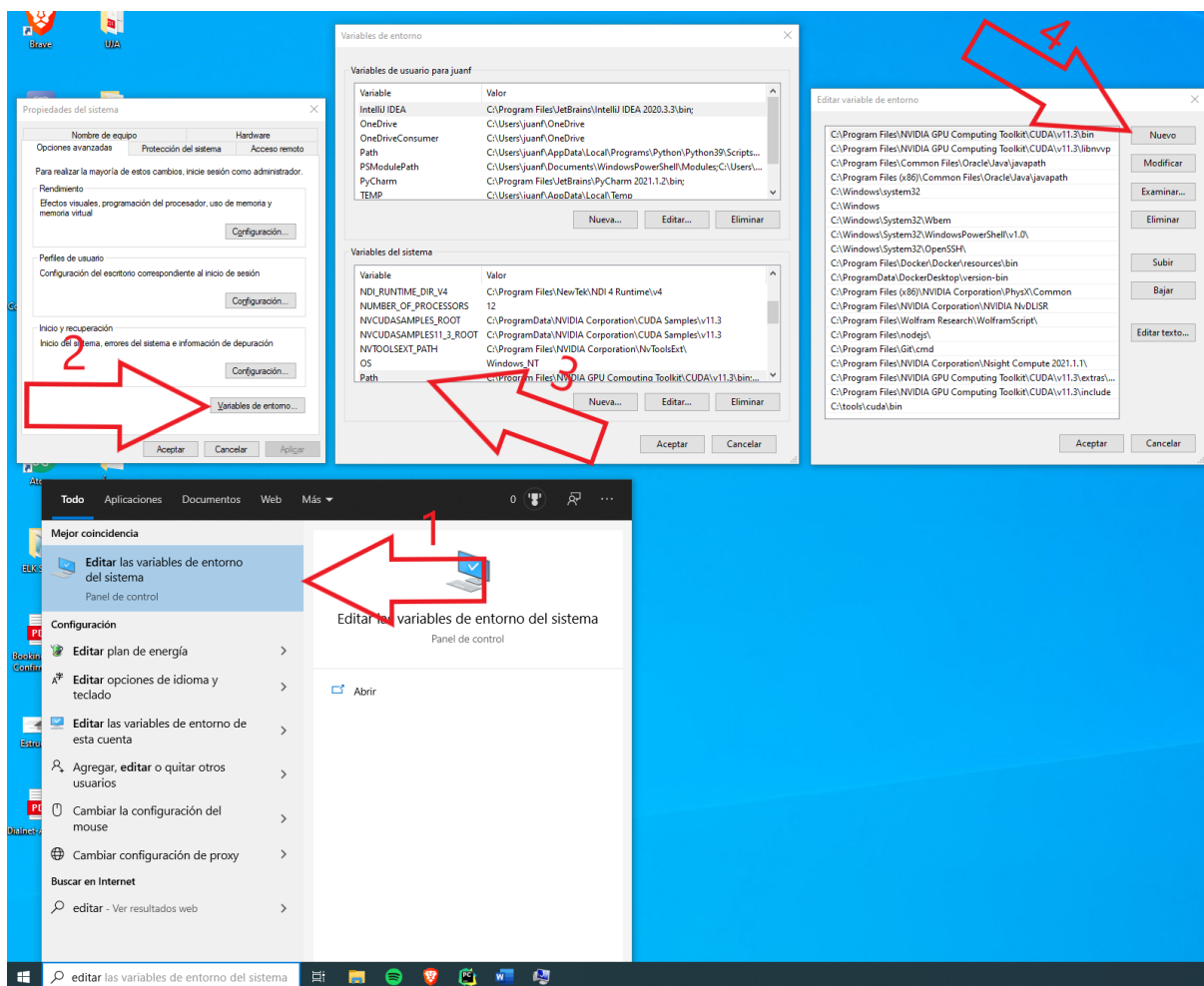


Ilustración 68. 5º paso para la instalación de CUDA.

Posteriormente, una vez instalado Keras, comprobaremos que la gráfica está detectada correctamente, por lo que se comprobará que este proceso se ha realizado correctamente.

Keras

Para instalar Keras, simplemente se debe introducir en el símbolo del sistema la orden `pip install tensorflow-gpu`. Una vez realizado, se ejecutará el siguiente código para comprobar que todo está correcto:

```
import sys
import tensorflow.keras
import tensorflow as tf
print(f"Versión de Keras: {tensorflow.keras.__version__}")
print(f"Versión de TensorFlow: {tf.__version__}")
print(f"Versión de Python {sys.version}")
gpu = len(tf.config.list_physical_devices('GPU'))>0
print("La tarjeta gráfica está", "disponible." if gpu else "no disponible.")
```

Una salida que indicaría que todo el proceso se ha realizado correctamente, finalizando el proceso de instalación de todos los apartados abordados:

```
Versión de Keras: 2.5.0
Versión de TensorFlow: 2.5.0
Versión de Python 3.9.5 (tags/v3.9.5:0a7dcdb, May 3 2021, 17:27:52)
[MSC v.1928 64 bit (AMD64)]
La tarjeta gráfica está disponible.
```

Librerías de XAI

SHAP

Para la instalación de SHAP, se ha de obtener la librería shap, en el caso de utilizar pip sería:

```
pip install shap
```

Lime

Para la instalación de lime, se ha de obtener la librería lime, en el caso de utilizar pip sería:

```
pip install lime
```

ANEXO II: CÓDIGO, EJECUCIÓN Y SALIDA DE LOS MODELOS DE APRENDIZAJE AUTOMÁTICO Y MÉTODOS DE XAI

Autoencoder

Código

Fashion MNIST

```
# Este import es utilizado para realizar diversas operaciones.
import numpy as np
# Este import nos permite mostrar las imágenes a predecir.
import matplotlib.pyplot as plt
# Este import nos permite obtener las diversas capas de las que dispone
keras.
from tensorflow.keras import layers
# Este import nos permite obtener el dataset de números manuscritos del
MNIST.
from tensorflow.keras.datasets import fashion_mnist
# Este import nos permite crear un modelo.
from tensorflow.keras.models import Model

# Parámetros de ejecución.
factorRuido = 0.4
epochsCreacionImagenesParecidas = 50
tamBatchCreacionImagenesParecidas = 128
epochsEliminacionRuido = 50
tamBatchEliminacionRuido = 128

# Esta función permite normalizar y modificar la forma del array indicado.
def preprocesamientoImagenes(array):
    array = array.astype("float32") / 255.0
    array = np.reshape(array, (len(array), 28, 28, 1))
    return array

# Esta función permite añadir ruido al array indicado.
def ponerRuido(array):
    datosConRuido = array + factorRuido * np.random.normal(
        loc=0.0, scale=1.0, size=array.shape
    )
    return np.clip(datosConRuido, 0.0, 1.0)

# Tomamos las imágenes del MNIST.
(X_entrenamiento, _), (X_test, _) = fashion_mnist.load_data()

# Se normaliza y modifica la forma del conjunto de datos.
X_entrenamiento = preprocesamientoImagenes(X_entrenamiento)
X_test = preprocesamientoImagenes(X_test)

# Se crea una copia de los datos con ruido.
X_entrenamiento_ruido = ponerRuido(X_entrenamiento)
X_test_ruido = ponerRuido(X_test)

# Esta función permite mostrar 10 imágenes de cada array indicado.
def mostrarImagenes(array1, array2):
    n = 10

    indices = np.random.randint(len(array1), size=n)
    imagenes1 = array1[indices, :]
    imagenes2 = array2[indices, :]
```

```

plt.figure(figsize=(20, 4))
for i, (image1, image2) in enumerate(zip(imagenes1, imagenes2)):
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(image1.reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(image2.reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()

# Se indica la forma de entrada
formaDatosEntrada = layers.Input(shape=(28, 28, 1))

# Codificador.
x = layers.Conv2D(32, (3, 3), activation="relu",
padding="same")(formaDatosEntrada)
x = layers.MaxPooling2D((2, 2), padding="same")(x)
x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(x)
x = layers.MaxPooling2D((2, 2), padding="same")(x)

# Decodificador.
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu",
padding="same")(x)
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu",
padding="same")(x)
x = layers.Conv2D(1, (3, 3), activation="sigmoid", padding="same")(x)

# Autoencoder.
autoencoder = Model(formaDatosEntrada, x)
autoencoder.compile(optimizer="adam", loss="binary_crossentropy")

# Se entrena el autoencoder para obtener imágenes parecidas a las
originales.
autoencoder.fit(
    x=X_entrenamiento,
    y=X_entrenamiento,
    epochs=epochsCreacionImagenesParecidas,
    batch_size=tamBatchCreacionImagenesParecidas,
    shuffle=True,
    validation_data=(X_test, X_test),
)

# Se entrena el autoencoder para reconstruir las imágenes con ruido.
autoencoder.fit(
    x=X_entrenamiento_ruido,
    y=X_entrenamiento,
    epochs=epochsCreacionImagenesParecidas,
    batch_size=tamBatchCreacionImagenesParecidas,
    shuffle=True,
    validation_data=(X_test_ruido, X_test),
)

prediccion = autoencoder.predict(X_test_ruido)
mostrarImagenes(X_test_ruido, prediccion)

```

MNIST

```

# Este import es utilizado para realizar diversas operaciones.
import numpy as np
# Este import nos permite mostrar las imágenes a predecir.
import matplotlib.pyplot as plt
# Este import nos permite obtener las diversas capas de las que dispone
keras.
from tensorflow.keras import layers
# Este import nos permite obtener el dataset de números manuscritos del
MNIST.
from tensorflow.keras.datasets import mnist
# Este import nos permite crear un modelo.
from tensorflow.keras.models import Model

# Parámetros de ejecución.
factorRuido = 0.4
epochsCreacionImagenesParecidas = 50
tamBatchCreacionImagenesParecidas = 128
epochsEliminacionRuido = 50
tamBatchEliminacionRuido = 128

# Esta función permite normalizar y modificar la forma del array indicado.
def preprocesamientoImagenes(array):
    array = array.astype("float32") / 255.0
    array = np.reshape(array, (len(array), 28, 28, 1))
    return array

# Esta función permite añadir ruido al array indicado.
def ponerRuido(array):
    datosConRuido = array + factorRuido * np.random.normal(
        loc=0.0, scale=1.0, size=array.shape
    )
    return np.clip(datosConRuido, 0.0, 1.0)

# Tomamos las imágenes del MNIST.
(X_entrenamiento, _), (X_test, _) = mnist.load_data()

# Se normaliza y modifica la forma del conjunto de datos.
X_entrenamiento = preprocesamientoImagenes(X_entrenamiento)
X_test = preprocesamientoImagenes(X_test)

# Se crea una copia de los datos con ruido.
X_entrenamiento_ruido = ponerRuido(X_entrenamiento)
X_test_ruido = ponerRuido(X_test)

# Esta función permite mostrar 10 imágenes de cada array indicado.
def mostrarImagenes(array1, array2):
    n = 10

    indices = np.random.randint(len(array1), size=n)
    imagenes1 = array1[indices, :]
    imagenes2 = array2[indices, :]

    plt.figure(figsize=(20, 4))
    for i, (image1, image2) in enumerate(zip(imagenes1, imagenes2)):
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(image1.reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

```

```

ax = plt.subplot(2, n, i + 1 + n)
plt.imshow(image2.reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)

plt.show()

# Se indica la forma de entrada
formaDatosEntrada = layers.Input(shape=(28, 28, 1))

# Codificador.
x = layers.Conv2D(32, (3, 3), activation="relu",
padding="same")(formaDatosEntrada)
x = layers.MaxPooling2D((2, 2), padding="same")(x)
x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(x)
x = layers.MaxPooling2D((2, 2), padding="same")(x)

# Decodificador.
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu",
padding="same")(x)
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu",
padding="same")(x)
x = layers.Conv2D(1, (3, 3), activation="sigmoid", padding="same")(x)

# Autoencoder.
autoencoder = Model(formaDatosEntrada, x)
autoencoder.compile(optimizer="adam", loss="binary_crossentropy")

# Se entrena el autoencoder para obtener imágenes parecidas a las
originales.
autoencoder.fit(
    x=X_entrenamiento,
    y=X_entrenamiento,
    epochs=epochsCreacionImagenesParecidas,
    batch_size=tamBatchCreacionImagenesParecidas,
    shuffle=True,
    validation_data=(X_test, X_test),
)

# Se entrena el autoencoder para reconstruir las imágenes con ruido.
autoencoder.fit(
    x=X_entrenamiento_ruido,
    y=X_entrenamiento,
    epochs=epochsCreacionImagenesParecidas,
    batch_size=tamBatchCreacionImagenesParecidas,
    shuffle=True,
    validation_data=(X_test_ruido, X_test),
)

prediccion = autoencoder.predict(X_test_ruido)
mostrarImagenes(X_test_ruido, prediccion)

```

Salida por consola

Fashion MNIST

```

C:\Users\juanf\PycharmProjects\pythonProject\venv\Scripts\python.exe
C:/Users/juanf/Desktop/TFG/Ficheros/Keras/pythonProject/Autoencoder/Fashion
MNIST.py

```

```
2021-09-03 01:52:54.562479: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 01:53:04.405423: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library nvcuda.dll
2021-09-03 01:53:04.478349: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 01:53:04.478723: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 01:53:04.524162: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 01:53:04.524363: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
2021-09-03 01:53:04.562599: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cufft64_10.dll
2021-09-03 01:53:04.567451: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library curand64_10.dll
2021-09-03 01:53:04.574374: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusolver64_11.dll
2021-09-03 01:53:04.584454: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusparse64_11.dll
2021-09-03 01:53:04.587039: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 01:53:04.587903: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 01:53:04.589142: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-03 01:53:04.590861: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 01:53:04.591255: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 01:53:06.246321: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 01:53:06.246529: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264] 0
```

```
2021-09-03 01:53:06.246656: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0:  N
2021-09-03 01:53:06.248571: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
2021-09-03 01:53:07.430045: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the
MLIR Optimization Passes are enabled (registered 2)
Epoch 1/50
2021-09-03 01:53:08.174839: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 01:53:09.031904: I
tensorflow/stream_executor/cuda/cuda_dnn.cc:359] Loaded cuDNN version 8100
2021-09-03 01:53:10.070305: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 01:53:11.279440: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
469/469 [=====] - 10s 12ms/step - loss: 0.3018 -
val_loss: 0.2715
Epoch 2/50
469/469 [=====] - 5s 11ms/step - loss: 0.2655 -
val_loss: 0.2637
Epoch 3/50
469/469 [=====] - 5s 11ms/step - loss: 0.2592 -
val_loss: 0.2596
Epoch 4/50
469/469 [=====] - 5s 11ms/step - loss: 0.2566 -
val_loss: 0.2578
Epoch 5/50
469/469 [=====] - 6s 12ms/step - loss: 0.2551 -
val_loss: 0.2566
Epoch 6/50
469/469 [=====] - 6s 13ms/step - loss: 0.2541 -
val_loss: 0.2556
Epoch 7/50
469/469 [=====] - 5s 11ms/step - loss: 0.2532 -
val_loss: 0.2551
Epoch 8/50
469/469 [=====] - 5s 11ms/step - loss: 0.2526 -
val_loss: 0.2544
Epoch 9/50
469/469 [=====] - 6s 12ms/step - loss: 0.2521 -
val_loss: 0.2539
Epoch 10/50
469/469 [=====] - 5s 11ms/step - loss: 0.2517 -
val_loss: 0.2535
Epoch 11/50
469/469 [=====] - 6s 12ms/step - loss: 0.2513 -
val_loss: 0.2532
Epoch 12/50
469/469 [=====] - 5s 11ms/step - loss: 0.2510 -
val_loss: 0.2530
Epoch 13/50
469/469 [=====] - 5s 12ms/step - loss: 0.2508 -
val_loss: 0.2527
Epoch 14/50
```

```
469/469 [=====] - 6s 12ms/step - loss: 0.2505 -  
val_loss: 0.2525  
Epoch 15/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2503 -  
val_loss: 0.2523  
Epoch 16/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2502 -  
val_loss: 0.2521  
Epoch 17/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2500 -  
val_loss: 0.2519  
Epoch 18/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2499 -  
val_loss: 0.2518  
Epoch 19/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2497 -  
val_loss: 0.2517  
Epoch 20/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2496 -  
val_loss: 0.2516  
Epoch 21/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2495 -  
val_loss: 0.2515  
Epoch 22/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2494 -  
val_loss: 0.2516  
Epoch 23/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2493 -  
val_loss: 0.2513  
Epoch 24/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2492 -  
val_loss: 0.2513  
Epoch 25/50  
469/469 [=====] - 6s 13ms/step - loss: 0.2492 -  
val_loss: 0.2512  
Epoch 26/50  
469/469 [=====] - 6s 13ms/step - loss: 0.2491 -  
val_loss: 0.2511  
Epoch 27/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2490 -  
val_loss: 0.2511  
Epoch 28/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2490 -  
val_loss: 0.2509  
Epoch 29/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2489 -  
val_loss: 0.2510  
Epoch 30/50  
469/469 [=====] - 5s 12ms/step - loss: 0.2489 -  
val_loss: 0.2509  
Epoch 31/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2488 -  
val_loss: 0.2509  
Epoch 32/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2488 -  
val_loss: 0.2508  
Epoch 33/50  
469/469 [=====] - 5s 10ms/step - loss: 0.2487 -  
val_loss: 0.2508  
Epoch 34/50
```

```
469/469 [=====] - 5s 11ms/step - loss: 0.2487 -  
val_loss: 0.2507  
Epoch 35/50  
469/469 [=====] - 5s 10ms/step - loss: 0.2486 -  
val_loss: 0.2506  
Epoch 36/50  
469/469 [=====] - 5s 10ms/step - loss: 0.2486 -  
val_loss: 0.2507  
Epoch 37/50  
469/469 [=====] - 5s 10ms/step - loss: 0.2485 -  
val_loss: 0.2505  
Epoch 38/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2485 -  
val_loss: 0.2506  
Epoch 39/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2485 -  
val_loss: 0.2505  
Epoch 40/50  
469/469 [=====] - 5s 12ms/step - loss: 0.2484 -  
val_loss: 0.2506  
Epoch 41/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2484 -  
val_loss: 0.2504  
Epoch 42/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2484 -  
val_loss: 0.2506  
Epoch 43/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2483 -  
val_loss: 0.2505  
Epoch 44/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2483 -  
val_loss: 0.2503  
Epoch 45/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2483 -  
val_loss: 0.2504  
Epoch 46/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2482 -  
val_loss: 0.2502  
Epoch 47/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2482 -  
val_loss: 0.2503  
Epoch 48/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2482 -  
val_loss: 0.2502  
Epoch 49/50  
469/469 [=====] - 6s 13ms/step - loss: 0.2482 -  
val_loss: 0.2502  
Epoch 50/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2481 -  
val_loss: 0.2501  
Epoch 1/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2981 -  
val_loss: 0.2935  
Epoch 2/50  
469/469 [=====] - 5s 12ms/step - loss: 0.2898 -  
val_loss: 0.2906  
Epoch 3/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2879 -  
val_loss: 0.2893  
Epoch 4/50
```

```
469/469 [=====] - 5s 11ms/step - loss: 0.2867 -  
val_loss: 0.2883  
Epoch 5/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2859 -  
val_loss: 0.2878  
Epoch 6/50  
469/469 [=====] - 5s 10ms/step - loss: 0.2854 -  
val_loss: 0.2872  
Epoch 7/50  
469/469 [=====] - 6s 13ms/step - loss: 0.2849 -  
val_loss: 0.2868  
Epoch 8/50  
469/469 [=====] - 6s 13ms/step - loss: 0.2845 -  
val_loss: 0.2868  
Epoch 9/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2842 -  
val_loss: 0.2862  
Epoch 10/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2839 -  
val_loss: 0.2865  
Epoch 11/50  
469/469 [=====] - 5s 12ms/step - loss: 0.2836 -  
val_loss: 0.2857  
Epoch 12/50  
469/469 [=====] - 5s 10ms/step - loss: 0.2834 -  
val_loss: 0.2855  
Epoch 13/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2832 -  
val_loss: 0.2853  
Epoch 14/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2831 -  
val_loss: 0.2853  
Epoch 15/50  
469/469 [=====] - 5s 10ms/step - loss: 0.2829 -  
val_loss: 0.2850  
Epoch 16/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2828 -  
val_loss: 0.2849  
Epoch 17/50  
469/469 [=====] - 5s 10ms/step - loss: 0.2826 -  
val_loss: 0.2847  
Epoch 18/50  
469/469 [=====] - 5s 10ms/step - loss: 0.2824 -  
val_loss: 0.2849  
Epoch 19/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2824 -  
val_loss: 0.2844  
Epoch 20/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2823 -  
val_loss: 0.2844  
Epoch 21/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2822 -  
val_loss: 0.2849  
Epoch 22/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2821 -  
val_loss: 0.2849  
Epoch 23/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2820 -  
val_loss: 0.2841  
Epoch 24/50
```

```
469/469 [=====] - 5s 11ms/step - loss: 0.2819 -  
val_loss: 0.2841  
Epoch 25/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2819 -  
val_loss: 0.2841  
Epoch 26/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2818 -  
val_loss: 0.2840  
Epoch 27/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2817 -  
val_loss: 0.2840  
Epoch 28/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2817 -  
val_loss: 0.2839  
Epoch 29/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2816 -  
val_loss: 0.2841  
Epoch 30/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2816 -  
val_loss: 0.2842  
Epoch 31/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2815 -  
val_loss: 0.2838  
Epoch 32/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2815 -  
val_loss: 0.2839  
Epoch 33/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2815 -  
val_loss: 0.2838  
Epoch 34/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2814 -  
val_loss: 0.2837  
Epoch 35/50  
469/469 [=====] - 5s 12ms/step - loss: 0.2813 -  
val_loss: 0.2838  
Epoch 36/50  
469/469 [=====] - 6s 13ms/step - loss: 0.2813 -  
val_loss: 0.2836  
Epoch 37/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2813 -  
val_loss: 0.2835  
Epoch 38/50  
469/469 [=====] - 6s 12ms/step - loss: 0.2812 -  
val_loss: 0.2836  
Epoch 39/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2812 -  
val_loss: 0.2835  
Epoch 40/50  
469/469 [=====] - 5s 12ms/step - loss: 0.2812 -  
val_loss: 0.2836  
Epoch 41/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2811 -  
val_loss: 0.2834  
Epoch 42/50  
469/469 [=====] - 5s 12ms/step - loss: 0.2811 -  
val_loss: 0.2834  
Epoch 43/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2811 -  
val_loss: 0.2833  
Epoch 44/50
```

```
469/469 [=====] - 6s 12ms/step - loss: 0.2810 -  
val_loss: 0.2833  
Epoch 45/50  
469/469 [=====] - 6s 13ms/step - loss: 0.2810 -  
val_loss: 0.2839  
Epoch 46/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2810 -  
val_loss: 0.2835  
Epoch 47/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2810 -  
val_loss: 0.2834  
Epoch 48/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2810 -  
val_loss: 0.2834  
Epoch 49/50  
469/469 [=====] - 5s 11ms/step - loss: 0.2810 -  
val_loss: 0.2833  
Epoch 50/50  
469/469 [=====] - 5s 10ms/step - loss: 0.2809 -  
val_loss: 0.2834
```

Process finished with exit code 0

MNIST

```
C:\Users\juanf\PycharmProjects\pythonProject\venv\Scripts\python.exe  
C:/Users/juanf/Desktop/TFG/Ficheros/Keras/pythonProject/Autoencoder/MNIST.py  
2021-09-03 01:31:18.182826: I  
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully  
opened dynamic library cudart64_110.dll  
2021-09-03 01:31:25.609027: I  
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully  
opened dynamic library nvcuda.dll  
2021-09-03 01:31:25.652763: I  
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with  
properties:  
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER  
computeCapability: 7.5  
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB  
deviceMemoryBandwidth: 417.29GiB/s  
2021-09-03 01:31:25.653136: I  
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully  
opened dynamic library cudart64_110.dll  
2021-09-03 01:31:25.672083: I  
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully  
opened dynamic library cublas_11.dll  
2021-09-03 01:31:25.672301: I  
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully  
opened dynamic library cublasLt64_11.dll  
2021-09-03 01:31:25.677729: I  
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully  
opened dynamic library cufft64_10.dll  
2021-09-03 01:31:25.680323: I  
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully  
opened dynamic library curand64_10.dll  
2021-09-03 01:31:25.685151: I  
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully  
opened dynamic library cusolver64_11.dll
```

```
2021-09-03 01:31:25.690641: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cuspars64_11.dll
2021-09-03 01:31:25.692590: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 01:31:25.692920: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 01:31:25.693439: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-03 01:31:25.694697: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 01:31:25.695265: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 01:31:26.477501: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 01:31:26.477724: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264] 0
2021-09-03 01:31:26.477849: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0: N
2021-09-03 01:31:26.478279: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
2021-09-03 01:31:27.284918: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the
MLIR Optimization Passes are enabled (registered 2)
Epoch 1/50
2021-09-03 01:31:28.095603: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 01:31:28.858990: I
tensorflow/stream_executor/cuda/cuda_dnn.cc:359] Loaded cuDNN version 8100
2021-09-03 01:31:29.787882: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 01:31:30.930460: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
469/469 [=====] - 10s 13ms/step - loss: 0.1293 -
val_loss: 0.0734
Epoch 2/50
469/469 [=====] - 6s 12ms/step - loss: 0.0719 -
val_loss: 0.0697
Epoch 3/50
469/469 [=====] - 5s 11ms/step - loss: 0.0695 -
val_loss: 0.0682
Epoch 4/50
```

```
469/469 [=====] - 5s 12ms/step - loss: 0.0683 -  
val_loss: 0.0673  
Epoch 5/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0674 -  
val_loss: 0.0666  
Epoch 6/50  
469/469 [=====] - 9s 20ms/step - loss: 0.0668 -  
val_loss: 0.0660  
Epoch 7/50  
469/469 [=====] - 81s 174ms/step - loss: 0.0662 -  
val_loss: 0.0655  
Epoch 8/50  
469/469 [=====] - 59s 125ms/step - loss: 0.0658 -  
val_loss: 0.0652  
Epoch 9/50  
469/469 [=====] - 39s 83ms/step - loss: 0.0655 -  
val_loss: 0.0650  
Epoch 10/50  
469/469 [=====] - 6s 14ms/step - loss: 0.0652 -  
val_loss: 0.0646  
Epoch 11/50  
469/469 [=====] - 7s 15ms/step - loss: 0.0649 -  
val_loss: 0.0644  
Epoch 12/50  
469/469 [=====] - 6s 13ms/step - loss: 0.0647 -  
val_loss: 0.0643  
Epoch 13/50  
469/469 [=====] - 6s 13ms/step - loss: 0.0645 -  
val_loss: 0.0639  
Epoch 14/50  
469/469 [=====] - 6s 13ms/step - loss: 0.0643 -  
val_loss: 0.0638  
Epoch 15/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0641 -  
val_loss: 0.0637  
Epoch 16/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0640 -  
val_loss: 0.0635  
Epoch 17/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0638 -  
val_loss: 0.0634  
Epoch 18/50  
469/469 [=====] - 5s 12ms/step - loss: 0.0637 -  
val_loss: 0.0633  
Epoch 19/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0636 -  
val_loss: 0.0631  
Epoch 20/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0635 -  
val_loss: 0.0630  
Epoch 21/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0634 -  
val_loss: 0.0629  
Epoch 22/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0633 -  
val_loss: 0.0630  
Epoch 23/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0632 -  
val_loss: 0.0629  
Epoch 24/50
```

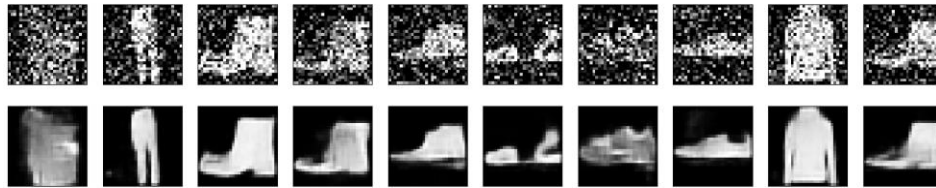
```
469/469 [=====] - 5s 11ms/step - loss: 0.0631 -  
val_loss: 0.0628  
Epoch 25/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0631 -  
val_loss: 0.0626  
Epoch 26/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0630 -  
val_loss: 0.0626  
Epoch 27/50  
469/469 [=====] - 5s 10ms/step - loss: 0.0629 -  
val_loss: 0.0626  
Epoch 28/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0629 -  
val_loss: 0.0625  
Epoch 29/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0628 -  
val_loss: 0.0625  
Epoch 30/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0628 -  
val_loss: 0.0624  
Epoch 31/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0627 -  
val_loss: 0.0624  
Epoch 32/50  
469/469 [=====] - 7s 15ms/step - loss: 0.0627 -  
val_loss: 0.0624  
Epoch 33/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0627 -  
val_loss: 0.0623  
Epoch 34/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0626 -  
val_loss: 0.0623  
Epoch 35/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0626 -  
val_loss: 0.0622  
Epoch 36/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0626 -  
val_loss: 0.0622  
Epoch 37/50  
469/469 [=====] - 6s 13ms/step - loss: 0.0625 -  
val_loss: 0.0622  
Epoch 38/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0625 -  
val_loss: 0.0621  
Epoch 39/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0625 -  
val_loss: 0.0621  
Epoch 40/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0624 -  
val_loss: 0.0621  
Epoch 41/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0624 -  
val_loss: 0.0620  
Epoch 42/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0624 -  
val_loss: 0.0621  
Epoch 43/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0624 -  
val_loss: 0.0621  
Epoch 44/50
```

```
469/469 [=====] - 5s 11ms/step - loss: 0.0623 -  
val_loss: 0.0620  
Epoch 45/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0623 -  
val_loss: 0.0620  
Epoch 46/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0623 -  
val_loss: 0.0620  
Epoch 47/50  
469/469 [=====] - 5s 12ms/step - loss: 0.0623 -  
val_loss: 0.0619  
Epoch 48/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0623 -  
val_loss: 0.0619  
Epoch 49/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0622 -  
val_loss: 0.0619  
Epoch 50/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0622 -  
val_loss: 0.0619  
Epoch 1/50  
469/469 [=====] - 6s 13ms/step - loss: 0.1002 -  
val_loss: 0.0931  
Epoch 2/50  
469/469 [=====] - 6s 13ms/step - loss: 0.0927 -  
val_loss: 0.0910  
Epoch 3/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0912 -  
val_loss: 0.0899  
Epoch 4/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0903 -  
val_loss: 0.0894  
Epoch 5/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0895 -  
val_loss: 0.0886  
Epoch 6/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0890 -  
val_loss: 0.0881  
Epoch 7/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0884 -  
val_loss: 0.0880  
Epoch 8/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0881 -  
val_loss: 0.0875  
Epoch 9/50  
469/469 [=====] - 6s 13ms/step - loss: 0.0877 -  
val_loss: 0.0870  
Epoch 10/50  
469/469 [=====] - 6s 13ms/step - loss: 0.0874 -  
val_loss: 0.0868  
Epoch 11/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0872 -  
val_loss: 0.0865  
Epoch 12/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0870 -  
val_loss: 0.0863  
Epoch 13/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0868 -  
val_loss: 0.0862  
Epoch 14/50
```

```
469/469 [=====] - 6s 13ms/step - loss: 0.0866 -  
val_loss: 0.0859  
Epoch 15/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0864 -  
val_loss: 0.0858  
Epoch 16/50  
469/469 [=====] - 6s 13ms/step - loss: 0.0863 -  
val_loss: 0.0857  
Epoch 17/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0862 -  
val_loss: 0.0859  
Epoch 18/50  
469/469 [=====] - 5s 12ms/step - loss: 0.0861 -  
val_loss: 0.0858  
Epoch 19/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0860 -  
val_loss: 0.0858  
Epoch 20/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0859 -  
val_loss: 0.0853  
Epoch 21/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0858 -  
val_loss: 0.0853  
Epoch 22/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0857 -  
val_loss: 0.0853  
Epoch 23/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0857 -  
val_loss: 0.0852  
Epoch 24/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0856 -  
val_loss: 0.0851  
Epoch 25/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0855 -  
val_loss: 0.0851  
Epoch 26/50  
469/469 [=====] - 5s 10ms/step - loss: 0.0855 -  
val_loss: 0.0850  
Epoch 27/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0854 -  
val_loss: 0.0850  
Epoch 28/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0853 -  
val_loss: 0.0851  
Epoch 29/50  
469/469 [=====] - 5s 12ms/step - loss: 0.0853 -  
val_loss: 0.0850  
Epoch 30/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0852 -  
val_loss: 0.0849  
Epoch 31/50  
469/469 [=====] - 5s 12ms/step - loss: 0.0852 -  
val_loss: 0.0850  
Epoch 32/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0851 -  
val_loss: 0.0848  
Epoch 33/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0851 -  
val_loss: 0.0849  
Epoch 34/50
```

```
469/469 [=====] - 5s 11ms/step - loss: 0.0851 -  
val_loss: 0.0848  
Epoch 35/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0850 -  
val_loss: 0.0847  
Epoch 36/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0850 -  
val_loss: 0.0847  
Epoch 37/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0850 -  
val_loss: 0.0847  
Epoch 38/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0849 -  
val_loss: 0.0846  
Epoch 39/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0849 -  
val_loss: 0.0845  
Epoch 40/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0848 -  
val_loss: 0.0846  
Epoch 41/50  
469/469 [=====] - 5s 10ms/step - loss: 0.0848 -  
val_loss: 0.0848  
Epoch 42/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0848 -  
val_loss: 0.0845  
Epoch 43/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0848 -  
val_loss: 0.0845  
Epoch 44/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0847 -  
val_loss: 0.0849  
Epoch 45/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0847 -  
val_loss: 0.0844  
Epoch 46/50  
469/469 [=====] - 5s 10ms/step - loss: 0.0847 -  
val_loss: 0.0844  
Epoch 47/50  
469/469 [=====] - 5s 10ms/step - loss: 0.0847 -  
val_loss: 0.0845  
Epoch 48/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0847 -  
val_loss: 0.0844  
Epoch 49/50  
469/469 [=====] - 6s 12ms/step - loss: 0.0847 -  
val_loss: 0.0844  
Epoch 50/50  
469/469 [=====] - 5s 11ms/step - loss: 0.0846 -  
val_loss: 0.0844
```

Process finished with exit code 0

Imágenes obtenidasFashion MNISTMNIST**CNN****Código**Fashion MNIST

```
# Este import nos permite obtener el dataset de prendas de vestir del
MNIST.
from tensorflow.keras.datasets import fashion_mnist
# Este import nos permite crear un modelo.
from tensorflow.keras.models import Sequential
""" Este import nos permite añadir capas neuronales densas, capas de
dropout, capas que conectan las capas
convolucionales con las capas neuronales densas, capas de convolución y
capas de max pooling."""
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
# Este import nos permite categorizar las clases.
from tensorflow.keras.utils import to_categorical
# Este import nos permite mostrar las imágenes a predecir.
import matplotlib.pyplot as plt
# Este import nos permite generar números aleatorios.
import random
# Este import es utilizado para obtener el valor de la solución
numéricamente.
import numpy as np
# Este import es utilizado para obtener la librería SHAP de XAI.
import shap
# Este import es utilizado para obtener la librería LIME de XAI en su
apartado de imágenes.
from lime import lime_image
# Este import es utilizado para mostrar los resultados de LIME en las
imágenes.
from skimage.segmentation import mark_boundaries

# Parámetros de ejecución.
epochsRedConvolucional = 15
tamBatchRedConvolucional = 250

# Valores de las clases del conjunto de datos.
```

```
valoresClases =
["camiseta/top","pantalón","jersey","vestido","abrigo","sandalia","camisa",
"zapatilla de deporte",
    "bolso","botín"]

# Esta función permite obtener la descripción de la clase que se ha
predecido.
def obtenerClase(clasePredecida):
    return valoresClases[clasePredecida]

# Esta variable indica el número de clases posibles en el conjunto de datos
de MNIST (de 0 a 9).
numClases = 10

# Estas variables indican las dimensiones de las imágenes de entrada.
alturaImagen = 28
anchuraImagen = 28

# Se cargan los datos del conjunto de datos. X contiene la información de
las imágenes e Y contiene las clases.
(X_entrenamiento, Y_entrenamiento), (X_test, Y_test) =
fashion_mnist.load_data()

# Se pasa la información de la imagen a un vector.
X_entrenamiento = X_entrenamiento.reshape(X_entrenamiento.shape[0],
anchuraImagen, alturaImagen, 1)
X_test = X_test.reshape(X_test.shape[0], anchuraImagen, alturaImagen, 1)

# Se pasan los números a precisión simple, un punto de equilibrio entre
memoria utilizada y precisión obtenida.
X_entrenamiento = X_entrenamiento.astype('float32')
X_test = X_test.astype('float32')

# Se convierte la información a escala de grises, debido a que la
información del color no es relevante.
X_entrenamiento /= 255
X_test /= 255

# Se categorizan las clases, pasando de ser un número a un vector de tamaño
10 que indica que clase es con un 1.
Y_entrenamiento = to_categorical(Y_entrenamiento, numClases)
Y_test = to_categorical(Y_test, numClases)

# Se indica el formato de los datos de entrada.
formaDatosEntrada = (anchuraImagen, alturaImagen, 1)

# Creación del modelo
modelo = Sequential()
# Añadido de una capa de convolución, que aprende 32 filtros, con un kernel
de 3x3 y activación ReLU.
modelo.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=formaDatosEntrada))
# Añadido de una capa que realiza max pooling en un tamaño de 2x2.
modelo.add(MaxPooling2D(pool_size=(2, 2)))
# Añadido de una capa de dropout, que añade algo de ruido, reduciendo la
posibilidad de sobreaprendizaje.
modelo.add(Dropout(0.25))
# Añadido de una capa de convolución, que aprende 64 filtros, con un kernel
de 3x3 y activación ReLU.
modelo.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
# Añadido de una capa que realiza max pooling en un tamaño de 2x2.
```

```

modelo.add(MaxPooling2D(pool_size=(2, 2)))
# Añadido de una capa de dropout, que añade algo de ruido, reduciendo la
# posibilidad de sobreaprendizaje.
modelo.add(Dropout(0.25))
# Añadido de una capa de flatten, que permite transformar los datos a un
# vector.
modelo.add(Flatten())
# Añadido de una capa neuronal con 256 neuronas de salida y activación
# ReLU.
modelo.add(Dense(256, activation='relu'))
# Añadido de una capa neuronal 10 neuronas de salida y activación softmax.
modelo.add(Dense(numClases, activation='softmax'))

# Se indica la función objetivo, el optimizador y la métrica a mostrar.
modelo.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Se entrena el modelo, con 10 epochs, el tamaño de los subconjuntos y un
# 20% de los datos usados para optimización.
modelo.fit(X_entrenamiento, Y_entrenamiento, epochs=epochsRedConvolucional,
batch_size=tamBatchRedConvolucional, verbose=1, validation_split=0.2)

# Se realiza testeo con los datos de test generados previamente.
resultadoTest = modelo.evaluate(X_test, Y_test, verbose=1)
print(f'Resultados de los datos de test: \n- Loss: {resultadoTest[0]} \n-
Accuracy: {resultadoTest[1]}% \n')

# Se guardan el conjunto de datos de test y sus clases preprocesadas para
# el uso en determinadas partes de SHAP.
X_test_preprocesado = X_test
Y_test_preprocesado = Y_test

# Se toman las imágenes de nuevo para realizar predicciones.
(_, _), (X_test, Y_test) = fashion_mnist.load_data()

# Se crea la imagen que mostrará 3 imágenes aleatorias y sus predicciones.
fig = plt.figure(figsize=(11, 3))

# Se guarda información para ser posteriormente usada en SHAP.
vectoresImagenes = []
imagenesSeleccionadas = np.array([])
labelsSeleccionadas = []

# Predicción de tres imágenes aleatorias.
for i in range(3):
    imagenSeleccionada = random.randint(0, len(X_test))
    imagenesSeleccionadas = np.append(imagenesSeleccionadas,
X_test_preprocesado[imagenSeleccionada])
    vectorCaracteristicasImagen = X_test[imagenSeleccionada].reshape(1,
anchuraImagen, alturaImagen, 1)
    caracteristicasPreprocesadasImagen =
X_test_preprocesado[imagenSeleccionada].reshape(1, anchuraImagen,
alturaImagen, 1)
    vectoresImagenes.append(caracteristicasPreprocesadasImagen)
    labelsSeleccionadas.append(Y_test_preprocesado[imagenSeleccionada])
    fig.add_subplot(1, 5, (i + 1) * 2 - 1)
    plt.imshow(X_test[imagenSeleccionada], cmap='Greys')
    plt.axis('off')
    prediccionImagen = modelo.predict(vectorCaracteristicasImagen)
    plt.title("Predicción: " + obtenerClase(np.argmax(prediccionImagen)) +
"\n" + "Valor real: " + obtenerClase(
Y_test[imagenSeleccionada]))

```

```

plt.show()

# Se aporta un conjunto de datos para que SHAP sea capaz de obtener
explicaciones.
conjuntoDatosBackground =
X_entrenamiento[np.random.choice(X_entrenamiento.shape[0], 1000,
replace=False)]

# Se le pasan los datos de background a SHAP.
e = shap.DeepExplainer(modelo, conjuntoDatosBackground)

# Se pasan las imágenes predecidas anteriormente con la forma correcta para
obtener sus valores de SHAP.
valoresShap = e.shap_values(imagenesSeleccionadas.reshape(3,28,28,1))

# Se modifican las imágenes y los valores de SHAP para mostrarse
posteriormente.
arrayVectores = np.stack([np.array(vectoresImagenes)])
imagenesAMostrar = arrayVectores.reshape(3, 28, 28, 1)
for i in range(len(valoresShap)):
    valoresShap[i] = valoresShap[i].reshape(3,28,28,1)

# Se añade el tipo de valor de cada columna para mejorar la visualización.
nuevoGrupoClases = []
for i in range(len(labelsSeleccionadas[0])):
    nuevoGrupoClases.append(valoresClases[i] + "\n" +
str(labelsSeleccionadas[0][i]))

# Se modifica el primer conjunto de valores para reflejar las clases.
labelsSeleccionadas[0] = nuevoGrupoClases

""" Se muestra las imágenes con sus valores de SHAP superpuestos, que
representan aquellos puntos de importancia en la
predicción, además de un 0 o 1 que indica el valor real del elemento. """
shap.image_plot(valoresShap, -imagenesAMostrar, labelsSeleccionadas)

# Se crea una función que gestiona las imágenes para poder utilizar
correctamente LIME.
def funcionPrediccion(images):
    arrayImg = []
    for img in images:
        img = img.reshape(28,28,3)
        img = img[0:29,0:29,0:1]
        arrayImg.append(img)
    imagenesAPredecir = np.array(arrayImg)
    return modelo.predict_on_batch(imagenesAPredecir)

# Se crea un explicador que permite obtener información mediante LIME.
explicadorLime = lime_image.LimeImageExplainer(random_state=42)
figLime = plt.figure(figsize=(11, 3))
i = 0
# Se realiza la predicción con LIME en diversas imágenes.
for img in imagenesAMostrar:
    img = img.reshape(28,28).astype("double")
    explanation = explicadorLime.explain_instance(img, funcionPrediccion,
top_labels=1, hide_color=0, num_samples=1000)
    imgLime, mascara =
explanation.get_image_and_mask(explanation.top_labels[0],
positive_only=False, num_features=1, hide_rest=True)
    figLime.add_subplot(1, 5, (i + 1) * 2 - 1)
    plt.imshow(mark_boundaries(imgLime / 2 + 0.5, mascara))

```

```
plt.axis('off')
i = i + 1
plt.show()
```

MNIST

```
# Este import nos permite obtener el dataset de números manuscritos del
MNIST.
from tensorflow.keras.datasets import mnist
# Este import nos permite crear un modelo.
from tensorflow.keras.models import Sequential
""" Este import nos permite añadir capas neuronales densas, capas de
dropout, capas que conectan las capas
convolucionales con las capas neuronales densas, capas de convolución y
capas de max pooling."""
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
# Este import nos permite categorizar las clases.
from tensorflow.keras.utils import to_categorical
# Este import nos permite mostrar las imágenes a predecir.
import matplotlib.pyplot as plt
# Este import nos permite generar números aleatorios.
import random
# Este import es utilizado para obtener el valor de la solución
numéricamente.
import numpy as np
# Este import es utilizado para obtener la librería SHAP de XAI.
import shap
# Este import es utilizado para obtener la librería LIME de XAI en su
apartado de imágenes.
from lime import lime_image
# Este import es utilizado para mostrar los resultados de LIME en las
imágenes.
from skimage.segmentation import mark_boundaries

# Parámetros de ejecución.
epochsRedConvolucional = 15
tamBatchRedConvolucional = 250

# Valores de las clases del conjunto de datos.
valoresClases = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

# Esta variable indica el número de clases posibles en el conjunto de datos
de MNIST (de 0 a 9).
numClases = 10

# Estas variables indican las dimensiones de las imágenes de entrada.
alturaImagen = 28
anchuraImagen = 28

# Se cargan los datos del conjunto de datos. X contiene la información de
las imágenes e Y contiene las clases.
(X_entrenamiento, Y_entrenamiento), (X_test, Y_test) = mnist.load_data()

# Se pasa la información de la imagen a un vector.
X_entrenamiento = X_entrenamiento.reshape(X_entrenamiento.shape[0],
anchuraImagen, alturaImagen, 1)
X_test = X_test.reshape(X_test.shape[0], anchuraImagen, alturaImagen, 1)

# Se pasan los números a precisión simple, un punto de equilibrio entre
memoria utilizada y precisión obtenida.
X_entrenamiento = X_entrenamiento.astype('float32')
```

```

X_test = X_test.astype('float32')

# Se convierte la información a escala de grises, debido a que la
información del color no es relevante.
X_entrenamiento /= 255
X_test /= 255

# Se categorizan las clases, pasando de ser un número a un vector de tamaño
10 que indica que clase es con un 1.
Y_entrenamiento = to_categorical(Y_entrenamiento, numClases)
Y_test = to_categorical(Y_test, numClases)

# Se indica el formato de los datos de entrada.
formaDatosEntrada = (anchuraImagen, alturaImagen, 1)

# Creación del modelo
modelo = Sequential()
# Añadido de una capa de convolución, que aprende 32 filtros, con un kernel
de 3x3 y activación ReLU.
modelo.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=formaDatosEntrada))
# Añadido de una capa que realiza max pooling en un tamaño de 2x2.
modelo.add(MaxPooling2D(pool_size=(2, 2)))
# Añadido de una capa de dropout, que añade algo de ruido, reduciendo la
posibilidad de sobreaprendizaje.
modelo.add(Dropout(0.25))
# Añadido de una capa de convolución, que aprende 64 filtros, con un kernel
de 3x3 y activación ReLU.
modelo.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
# Añadido de una capa que realiza max pooling en un tamaño de 2x2.
modelo.add(MaxPooling2D(pool_size=(2, 2)))
# Añadido de una capa de dropout, que añade algo de ruido, reduciendo la
posibilidad de sobreaprendizaje.
modelo.add(Dropout(0.25))
# Añadido de una capa de flatten, que permite transformar los datos a un
vector.
modelo.add(Flatten())
# Añadido de una capa neuronal con 256 neuronas de salida y activación
ReLU.
modelo.add(Dense(256, activation='relu'))
# Añadido de una capa neuronal 10 neuronas de salida y activación softmax.
modelo.add(Dense(numClases, activation='softmax'))

# Se indica la función objetivo, el optimizador y la métrica a mostrar.
modelo.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Se entrena el modelo, con 10 epochs, el tamaño de los subconjuntos y un
20% de los datos usados para optimización.
modelo.fit(X_entrenamiento, Y_entrenamiento, epochs=epochsRedConvolucional,
batch_size=tamBatchRedConvolucional, verbose=1, validation_split=0.2)

# Se realiza testeo con los datos de test generados previamente.
resultadoTest = modelo.evaluate(X_test, Y_test, verbose=1)
print(f'Resultados de los datos de test: \n- Loss: {resultadoTest[0]} \n-
Accuracy: {resultadoTest[1]}% \n')

# Se guardan el conjunto de datos de test y sus clases preprocesadas para
el uso en determinadas partes de SHAP.
X_test_preprocesado = X_test
Y_test_preprocesado = Y_test

```

```

# Se toman las imágenes de nuevo para realizar predicciones.
(_, _), (X_test, Y_test) = mnist.load_data()

# Se crea la imagen que mostrará 3 imágenes aleatorias y sus predicciones.
fig = plt.figure(figsize=(11, 3))

# Se guarda información para ser posteriormente usada en SHAP.
vectoresImagenes = []
imagenesSeleccionadas = np.array([])
labelsSeleccionadas = []

# Predicción de tres imágenes aleatorias.
for i in range(3):
    imagenSeleccionada = random.randint(0, len(X_test))
    imagenesSeleccionadas = np.append(imagenesSeleccionadas,
X_test_preprocesado[imagenSeleccionada])
    vectorCaracteristicasImagen = X_test[imagenSeleccionada].reshape(1,
anchuraImagen, alturaImagen, 1)
    caracteristicasPreprocesadasImagen =
X_test_preprocesado[imagenSeleccionada].reshape(1, anchuraImagen,
alturaImagen, 1)
    vectoresImagenes.append(caracteristicasPreprocesadasImagen)
    labelsSeleccionadas.append(Y_test_preprocesado[imagenSeleccionada])
    fig.add_subplot(1, 5, (i + 1) * 2 - 1)
    plt.imshow(X_test[imagenSeleccionada], cmap='Greys')
    plt.axis('off')
    prediccionImagen = modelo.predict(vectorCaracteristicasImagen)
    plt.title("Predicción: " + str(np.argmax(prediccionImagen)) + "\n" +
"Valor real: " + str(
        Y_test[imagenSeleccionada]))
plt.show()

# Se aporta un conjunto de datos para que SHAP sea capaz de obtener
explicaciones.
conjuntoDatosBackground =
X_entrenamiento[np.random.choice(X_entrenamiento.shape[0], 1000,
replace=False)]

# Se le pasan los datos de background a SHAP.
e = shap.DeepExplainer(modelo, conjuntoDatosBackground)

# Se pasan las imágenes predecidas anteriormente con la forma correcta para
obtener sus valores de SHAP.
valoresShap = e.shap_values(imagenesSeleccionadas.reshape(3,28,28,1))

# Se modifican las imágenes y los valores de SHAP para mostrarse
posteriormente.
arrayVectores = np.stack([np.array(vectoresImagenes)])
imagenesAMostrar = arrayVectores.reshape(3, 28, 28, 1)
for i in range(len(valoresShap)):
    valoresShap[i] = valoresShap[i].reshape(3,28,28,1)

# Se añade el tipo de valor de cada columna para mejorar la visualización.
nuevoGrupoClases = []
for i in range(len(labelsSeleccionadas[0])):
    nuevoGrupoClases.append(valoresClases[i] + "\n" +
str(labelsSeleccionadas[0][i]))

# Se modifica el primer conjunto de valores para reflejar las clases.
labelsSeleccionadas[0] = nuevoGrupoClases

```

```

""" Se muestra las imágenes con sus valores de SHAP superpuestos, que
representan aquellos puntos de importancia en la
predicción, además de un 0 o 1 que indica el valor real del elemento. """
shap.image_plot(valoresShap, -imagenesAMostrar, labelsSeleccionadas)

# Se crea una función que gestiona las imágenes para poder utilizar
correctamente LIME.
def funcionPrediccion(images):
    arrayImg = []
    for img in images:
        img = img.reshape(28,28,3)
        img = img[0:29,0:29,0:1]
        arrayImg.append(img)
    imagenesAPredecir = np.array(arrayImg)
    return modelo.predict_on_batch(imagenesAPredecir)

# Se crea un explicador que permite obtener información mediante LIME.
explicadorLime = lime_image.LimeImageExplainer(random_state=42)
figLime = plt.figure(figsize=(11, 3))
i = 0
# Se realiza la predicción con LIME en diversas imágenes.
for img in imagenesAMostrar:
    img = img.reshape(28,28).astype("double")
    explanation = explicadorLime.explain_instance(img, funcionPrediccion,
    top_labels=1, hide_color=0, num_samples=1000)
    imgLime, mascara =
    explanation.get_image_and_mask(explanation.top_labels[0],
    positive_only=False, num_features=1, hide_rest=True)
    figLime.add_subplot(1, 5, (i + 1) * 2 - 1)
    plt.imshow(mark_boundaries(imgLime / 2 + 0.5, mascara))
    plt.axis('off')
    i = i + 1
plt.show()

```

Salida por consola

Fashion MNIST

```

C:\Users\juanf\PycharmProjects\pythonProject\venv\Scripts\python.exe
C:/Users/juanf/Desktop/TFG/Ficheros/Keras/pythonProject/CNN/FashionMNIST.py
2021-09-03 03:04:06.400388: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 03:04:14.358811: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library nvcuda.dll
2021-09-03 03:04:14.399803: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 03:04:14.400285: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 03:04:14.416288: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 03:04:14.416498: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll

```

```
2021-09-03 03:04:14.422168: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cufft64_10.dll
2021-09-03 03:04:14.424643: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library curand64_10.dll
2021-09-03 03:04:14.429268: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusolver64_11.dll
2021-09-03 03:04:14.435103: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusparse64_11.dll
2021-09-03 03:04:14.436911: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 03:04:14.437370: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 03:04:14.438026: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-03 03:04:14.439637: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 03:04:14.440134: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 03:04:15.215255: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 03:04:15.215470: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264]          0
2021-09-03 03:04:15.215594: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0:      N
2021-09-03 03:04:15.216008: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
2021-09-03 03:04:15.559797: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the
MLIR Optimization Passes are enabled (registered 2)
Epoch 1/15
2021-09-03 03:04:16.197460: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 03:04:17.003067: I
tensorflow/stream_executor/cuda/cuda_dnn.cc:359] Loaded cuDNN version 8100
2021-09-03 03:04:17.924744: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 03:04:18.907475: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
```

```
192/192 [=====] - 6s 10ms/step - loss: 0.6946 -
accuracy: 0.7447 - val_loss: 0.4830 - val_accuracy: 0.8210
Epoch 2/15
192/192 [=====] - 2s 9ms/step - loss: 0.4481 -
accuracy: 0.8341 - val_loss: 0.3932 - val_accuracy: 0.8600
Epoch 3/15
192/192 [=====] - 2s 9ms/step - loss: 0.3956 -
accuracy: 0.8571 - val_loss: 0.3576 - val_accuracy: 0.8707
Epoch 4/15
192/192 [=====] - 2s 8ms/step - loss: 0.3565 -
accuracy: 0.8709 - val_loss: 0.3270 - val_accuracy: 0.8824
Epoch 5/15
192/192 [=====] - 1s 7ms/step - loss: 0.3300 -
accuracy: 0.8801 - val_loss: 0.3067 - val_accuracy: 0.8904
Epoch 6/15
192/192 [=====] - 2s 8ms/step - loss: 0.3094 -
accuracy: 0.8871 - val_loss: 0.2870 - val_accuracy: 0.8965
Epoch 7/15
192/192 [=====] - 1s 7ms/step - loss: 0.2934 -
accuracy: 0.8920 - val_loss: 0.2796 - val_accuracy: 0.8985
Epoch 8/15
192/192 [=====] - 1s 8ms/step - loss: 0.2778 -
accuracy: 0.8994 - val_loss: 0.2684 - val_accuracy: 0.9021
Epoch 9/15
192/192 [=====] - 2s 8ms/step - loss: 0.2676 -
accuracy: 0.9015 - val_loss: 0.2832 - val_accuracy: 0.8931
Epoch 10/15
192/192 [=====] - 2s 9ms/step - loss: 0.2568 -
accuracy: 0.9045 - val_loss: 0.2642 - val_accuracy: 0.9037
Epoch 11/15
192/192 [=====] - 2s 9ms/step - loss: 0.2482 -
accuracy: 0.9073 - val_loss: 0.2708 - val_accuracy: 0.8986
Epoch 12/15
192/192 [=====] - 2s 9ms/step - loss: 0.2374 -
accuracy: 0.9121 - val_loss: 0.2482 - val_accuracy: 0.9113
Epoch 13/15
192/192 [=====] - 2s 9ms/step - loss: 0.2308 -
accuracy: 0.9154 - val_loss: 0.2442 - val_accuracy: 0.9116
Epoch 14/15
192/192 [=====] - 2s 9ms/step - loss: 0.2224 -
accuracy: 0.9172 - val_loss: 0.2495 - val_accuracy: 0.9092
Epoch 15/15
192/192 [=====] - 2s 9ms/step - loss: 0.2159 -
accuracy: 0.9197 - val_loss: 0.2500 - val_accuracy: 0.9062
313/313 [=====] - 2s 5ms/step - loss: 0.2625 -
accuracy: 0.9010
Resultados de los datos de test:
- Loss: 0.26246368885040283
- Accuracy: 0.9010000228881836%
```

Using TensorFlow backend.

Your TensorFlow version is newer than 2.4.0 and so graph support has been removed in eager mode. See PR #1483 for discussion.

`tf.keras.backend.set\_learning\_phase` is deprecated and will be removed after 2020-10-11. To update it, simply pass a True/False value to the `training` argument of the `\_\_call\_\_` method of your layer or model.

WARNING:tensorflow:5 out of the last 13 calls to <function TFDeep.phi_symbolic.<locals>.grad_graph at 0x00000193EE2030D0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects

instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details. WARNING:tensorflow:5 out of the last 13 calls to <function TFDeep.phi_symbolic.<locals>.grad_graph at 0x00000193EE246A60> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

```
100%|██████████| 1000/1000 [00:01<00:00, 967.71it/s]
100%|██████████| 1000/1000 [00:00<00:00, 1156.44it/s]
100%|██████████| 1000/1000 [00:00<00:00, 1046.38it/s]
```

Process finished with exit code 0

MNIST

```
C:\Users\juanf\PycharmProjects\pythonProject\venv\Scripts\python.exe
C:/Users/juanf/Desktop/TFG/Ficheros/Keras/pythonProject/CNN/MNIST.py
2021-09-03 03:09:19.662773: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 03:09:27.633428: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library nvcuda.dll
2021-09-03 03:09:27.684419: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 03:09:27.684827: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 03:09:27.701692: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 03:09:27.701918: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
2021-09-03 03:09:27.707713: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cufft64_10.dll
2021-09-03 03:09:27.710293: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library curand64_10.dll
2021-09-03 03:09:27.715072: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusolver64_11.dll
2021-09-03 03:09:27.720927: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusparse64_11.dll
```

```
2021-09-03 03:09:27.722732: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 03:09:27.723050: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 03:09:27.723630: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-03 03:09:27.725032: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 03:09:27.725668: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 03:09:28.498232: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 03:09:28.498446: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264] 0
2021-09-03 03:09:28.498571: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0: N
2021-09-03 03:09:28.499051: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
2021-09-03 03:09:28.835773: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the
MLIR Optimization Passes are enabled (registered 2)
Epoch 1/15
2021-09-03 03:09:29.474927: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 03:09:30.256728: I
tensorflow/stream_executor/cuda/cuda_dnn.cc:359] Loaded cuDNN version 8100
2021-09-03 03:09:31.329943: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 03:09:32.360892: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
192/192 [=====] - 6s 10ms/step - loss: 0.3495 -
accuracy: 0.8971 - val_loss: 0.0912 - val_accuracy: 0.9738
Epoch 2/15
192/192 [=====] - 2s 8ms/step - loss: 0.0917 -
accuracy: 0.9711 - val_loss: 0.0634 - val_accuracy: 0.9816
Epoch 3/15
192/192 [=====] - 2s 8ms/step - loss: 0.0651 -
accuracy: 0.9806 - val_loss: 0.0488 - val_accuracy: 0.9869
Epoch 4/15
192/192 [=====] - 1s 8ms/step - loss: 0.0518 -
accuracy: 0.9841 - val_loss: 0.0436 - val_accuracy: 0.9869
Epoch 5/15
```

```

192/192 [=====] - 1s 8ms/step - loss: 0.0448 -
accuracy: 0.9860 - val_loss: 0.0414 - val_accuracy: 0.9878
Epoch 6/15
192/192 [=====] - 1s 8ms/step - loss: 0.0364 -
accuracy: 0.9889 - val_loss: 0.0347 - val_accuracy: 0.9903
Epoch 7/15
192/192 [=====] - 2s 8ms/step - loss: 0.0332 -
accuracy: 0.9898 - val_loss: 0.0324 - val_accuracy: 0.9906
Epoch 8/15
192/192 [=====] - 2s 8ms/step - loss: 0.0267 -
accuracy: 0.9913 - val_loss: 0.0318 - val_accuracy: 0.9918
Epoch 9/15
192/192 [=====] - 2s 9ms/step - loss: 0.0251 -
accuracy: 0.9921 - val_loss: 0.0330 - val_accuracy: 0.9911
Epoch 10/15
192/192 [=====] - 2s 9ms/step - loss: 0.0223 -
accuracy: 0.9930 - val_loss: 0.0313 - val_accuracy: 0.9910
Epoch 11/15
192/192 [=====] - 2s 9ms/step - loss: 0.0215 -
accuracy: 0.9931 - val_loss: 0.0301 - val_accuracy: 0.9918
Epoch 12/15
192/192 [=====] - 2s 8ms/step - loss: 0.0185 -
accuracy: 0.9938 - val_loss: 0.0412 - val_accuracy: 0.9889
Epoch 13/15
192/192 [=====] - 2s 8ms/step - loss: 0.0177 -
accuracy: 0.9939 - val_loss: 0.0330 - val_accuracy: 0.9920
Epoch 14/15
192/192 [=====] - 2s 8ms/step - loss: 0.0151 -
accuracy: 0.9947 - val_loss: 0.0301 - val_accuracy: 0.9919
Epoch 15/15
192/192 [=====] - 2s 8ms/step - loss: 0.0142 -
accuracy: 0.9952 - val_loss: 0.0328 - val_accuracy: 0.9914
313/313 [=====] - 2s 5ms/step - loss: 0.0264 -
accuracy: 0.9916
Resultados de los datos de test:
- Loss: 0.02638286165893078
- Accuracy: 0.991599977016449%

```

Using TensorFlow backend.

Your TensorFlow version is newer than 2.4.0 and so graph support has been removed in eager mode. See PR #1483 for discussion.

`tf.keras.backend.set\_learning\_phase` is deprecated and will be removed after 2020-10-11. To update it, simply pass a True/False value to the `training` argument of the `\_\_call\_\_` method of your layer or model.

WARNING:tensorflow:5 out of the last 13 calls to <function TFDeep.phi_symbolic.<locals>.grad_graph at 0x0000017CF6B550> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to

https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

WARNING:tensorflow:5 out of the last 13 calls to <function TFDeep.phi_symbolic.<locals>.grad_graph at 0x0000017D05562CA0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the

loop. For (2), @tf.function has `experimental_relax_shapes=True` option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to

https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

```
100%|██████████| 1000/1000 [00:00<00:00, 1038.70it/s]
100%|██████████| 1000/1000 [00:00<00:00, 1075.27it/s]
100%|██████████| 1000/1000 [00:00<00:00, 1041.03it/s]
```

Process finished with exit code 0

Imágenes obtenidas

Fashion MNIST

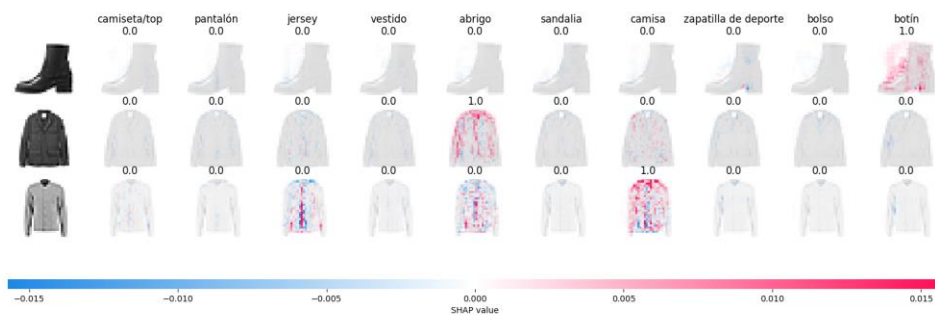
Predicción: botín
Valor real: botín



Predicción: abrigo
Valor real: abrigo



Predicción: camisa
Valor real: camisa



MNIST

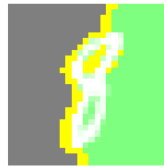
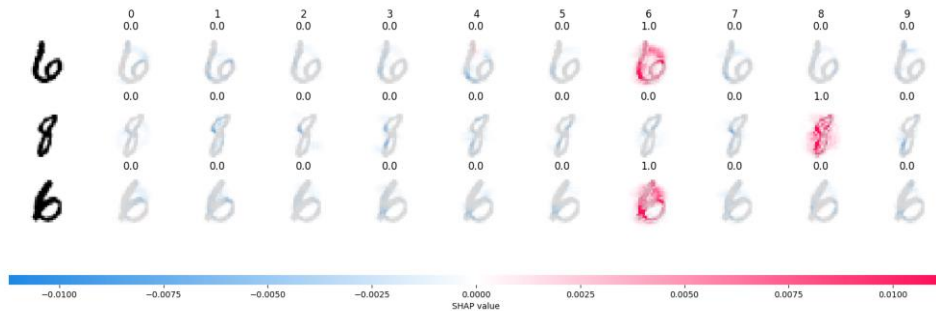
Predicción: 6
Valor real: 6



Predicción: 8
Valor real: 8



Predicción: 6
Valor real: 6

**GAN****Código**Fashion MNIST

Este import nos permite obtener las diferentes capas utilizadas para crear el modelo.

```
from tensorflow.keras import layers
```

Este import nos permite obtener TensorFlow.

```
import tensorflow as tf
```

Este import nos permite mostrar las imágenes generadas por el modelo.

```
import matplotlib.pyplot as plt
```

Este import nos permite ver el tiempo de cada epoch.

```
import time
```

Este import nos permite mostrar las imágenes generadas por el modelo.

```
from IPython import display
```

Parámetros de ejecución.

```
tamBatchGan = 256
```

```
epochsGan = 150
```

```
ejemplosAGenerar = 16
```

Variables utilizadas durante la ejecución del modelo.

```

tamBuffer = 60000
nivelRuido = 100

# Esta función se utiliza para crear el modelo del generador, parte del
GAN.
def crearGenerador():
    modelo = tf.keras.Sequential()
    modelo.add(layers.Dense(7 * 7 * 256, use_bias=False,
input_shape=(100,)))
    modelo.add(layers.BatchNormalization())
    modelo.add(layers.LeakyReLU())

    modelo.add(layers.Reshape((7, 7, 256)))
    assert modelo.output_shape == (None, 7, 7, 256)

    modelo.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1),
padding='same', use_bias=False))
    assert modelo.output_shape == (None, 7, 7, 128)
    modelo.add(layers.BatchNormalization())
    modelo.add(layers.LeakyReLU())

    modelo.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2),
padding='same', use_bias=False))
    assert modelo.output_shape == (None, 14, 14, 64)
    modelo.add(layers.BatchNormalization())
    modelo.add(layers.LeakyReLU())

    modelo.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2),
padding='same', use_bias=False, activation='tanh'))
    assert modelo.output_shape == (None, 28, 28, 1)

    return modelo

# Esta función se utiliza para crear el modelo del discriminador, parte del
GAN.
def crearDiscriminador():
    modelo = tf.keras.Sequential()
    modelo.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
input_shape=[28, 28, 1]))
    modelo.add(layers.LeakyReLU())
    modelo.add(layers.Dropout(0.3))

    modelo.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    modelo.add(layers.LeakyReLU())
    modelo.add(layers.Dropout(0.3))

    modelo.add(layers.Flatten())
    modelo.add(layers.Dense(1))

    return modelo

# Se crea el generador y el discriminador, además de sus optimizadores.
generador = crearGenerador()
discriminador = crearDiscriminador()

# Se definen 2 optimizadores, uno para el modelo discriminativo y otro para
el modelo generativo.
optimizadorGenerador = tf.keras.optimizers.Adam(1e-4)
optimizadorDiscriminador = tf.keras.optimizers.Adam(1e-4)

# Se obtiene un calculador del valor de la función objetivo.

```

```

calculadorEntropia = tf.keras.losses.BinaryCrossentropy(from_logits=True)

# Esta función nos permite calcular el valor de la función objetivo del
# generador.
def funcionObjetivoGenerador(datosGenerados):
    return calculadorEntropia(tf.ones_like(datosGenerados), datosGenerados)

# Esta función nos permite calcular el valor de la función objetivo del
# discriminador.
def funcionObjetivoDiscriminador(datosReales, datosGenerados):
    valorFuncionObjetivoReal =
    calculadorEntropia(tf.ones_like(datosReales), datosReales)
    valorFuncionObjetivoGenerado =
    calculadorEntropia(tf.zeros_like(datosGenerados), datosGenerados)
    valorFuncionObjetivoTotal = valorFuncionObjetivoReal +
    valorFuncionObjetivoGenerado
    return valorFuncionObjetivoTotal

# Permite generar y mostrar las imágenes del modelo, permitiendo apreciar
# el avance del modelo.
def mostrarImagen(modelo, valoresTestEntrada):
    # Training está a falso, indicando que se realizará un proceso de
    # inferencia.
    predicciones = modelo(valoresTestEntrada, training=False)

    for i in range(predicciones.shape[0]):
        plt.subplot(4, 4, i+1)
        plt.imshow(predicciones[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
        plt.axis('off')

    plt.show()

# Esta función permite realizar un paso en el entrenamiento.
@tf.function
def pasoEntrenamiento(imagenes):
    ruido = tf.random.normal([tamBatchGan, nivelRuido])

    with tf.GradientTape() as almacenGradientesGenerador, tf.GradientTape()
as almacenGradientesDiscriminador:
        imagenesGeneradas = generador(ruido, training=True)

        salidaValoresReales = discriminador(imagenes, training=True)
        salidaValoresGenerados = discriminador(imagenesGeneradas,
training=True)

        valorFuncionObjetivoGenerador =
funcionObjetivoGenerador(salidaValoresGenerados)
        valorFuncionObjetivoDiscriminador =
funcionObjetivoDiscriminador(salidaValoresReales, salidaValoresGenerados)

        gradientesGenerador =
almacenGradientesGenerador.gradient(valorFuncionObjetivoGenerador,
generador.trainable_variables)
        gradientesDiscriminador =
almacenGradientesDiscriminador.gradient(valorFuncionObjetivoDiscriminador,
discriminador.trainable_variables)

        optimizadorGenerador.apply_gradients(zip(gradientesGenerador,
generador.trainable_variables))
        optimizadorDiscriminador.apply_gradients(zip(gradientesDiscriminador,
discriminador.trainable_variables))

```

```

# Semilla para generar imágenes.
semilla = tf.random.normal([ejemplosAGenerar, nivelRuido])

# Esta función contiene el proceso de entrenamiento del modelo GAN (formado
por el modelo generativo y el discriminativo).
def entrenamiento(conjuntoDatos, epochs):
    for epoch in range(epochs):
        tiempoInicio = time.time()
        for batchImagen in conjuntoDatos:
            pasoEntrenamiento(batchImagen)
            # Muestra las imagen del epoch actual.
            display.clear_output(wait=True)
            mostrarImagen(generador, semilla)
            print ('El tiempo del epoch {} es {} segundos'.format(epoch + 1,
time.time()-tiempoInicio))
        # Genera la imagen del último epoch.
        display.clear_output(wait=True)
        mostrarImagen(generador, semilla)

# Se cargan los datos del conjunto de datos. X contiene la información de
las imágenes e Y contiene las clases.
(X_entrenamiento, Y_entrenamiento), (_, _) =
tf.keras.datasets.mnist.load_data()

# Se modifica la forma y se normaliza (de -1 a 1) el conjunto de datos de
entrenamiento.
X_entrenamiento = X_entrenamiento.reshape(X_entrenamiento.shape[0], 28, 28,
1).astype('float32')
X_entrenamiento = (X_entrenamiento - 127.5) / 127.5

# Se mezcla y se parten los datos en batches.
X_entrenamiento_modificado =
tf.data.Dataset.from_tensor_slices(X_entrenamiento).shuffle(tamBuffer).batch(tamBatchGan)

# Se llama el modelo para entrenarlo.
entrenamiento(X_entrenamiento_modificado, epochsGan)

# Debido a que el modelo tarda en entrenarse, se cargará el modelo ya
entrenado y se mostrará una generación de imágenes.
mostrarImagen(generador, semilla)

```

MNIST

```

# Este import nos permite obtener las diferentes capas utilizadas para
crear el modelo.
from tensorflow.keras import layers
# Este import nos permite obtener TensorFlow.
import tensorflow as tf
# Este import nos permite mostrar las imágenes generadas por el modelo.
import matplotlib.pyplot as plt
# Este import nos permite ver el tiempo de cada epoch.
import time
# Este import nos permite mostrar las imágenes generadas por el modelo.
from IPython import display

# Parámetros de ejecución.
tamBatchGan = 256
epochsGan = 150
ejemplosAGenerar = 16

```

```
# Variables utilizadas durante la ejecución del modelo.
tamBuffer = 60000
nivelRuido = 100

# Esta función se utiliza para crear el modelo del generador, parte del GAN.
def crearGenerador():
    modelo = tf.keras.Sequential()
    modelo.add(layers.Dense(7 * 7 * 256, use_bias=False,
input_shape=(100,)))
    modelo.add(layers.BatchNormalization())
    modelo.add(layers.LeakyReLU())

    modelo.add(layers.Reshape((7, 7, 256)))
    assert modelo.output_shape == (None, 7, 7, 256)

    modelo.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1),
padding='same', use_bias=False))
    assert modelo.output_shape == (None, 7, 7, 128)
    modelo.add(layers.BatchNormalization())
    modelo.add(layers.LeakyReLU())

    modelo.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2),
padding='same', use_bias=False))
    assert modelo.output_shape == (None, 14, 14, 64)
    modelo.add(layers.BatchNormalization())
    modelo.add(layers.LeakyReLU())

    modelo.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2),
padding='same', use_bias=False, activation='tanh'))
    assert modelo.output_shape == (None, 28, 28, 1)

    return modelo

# Esta función se utiliza para crear el modelo del discriminador, parte del GAN.
def crearDiscriminador():
    modelo = tf.keras.Sequential()
    modelo.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
input_shape=[28, 28, 1]))
    modelo.add(layers.LeakyReLU())
    modelo.add(layers.Dropout(0.3))

    modelo.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    modelo.add(layers.LeakyReLU())
    modelo.add(layers.Dropout(0.3))

    modelo.add(layers.Flatten())
    modelo.add(layers.Dense(1))

    return modelo

# Se crea el generador y el discriminador, además de sus optimizadores.
generador = crearGenerador()
discriminador = crearDiscriminador()

# Se definen 2 optimizadores, uno para el modelo discriminativo y otro para el modelo generativo.
optimizadorGenerador = tf.keras.optimizers.Adam(1e-4)
optimizadorDiscriminador = tf.keras.optimizers.Adam(1e-4)
```

```

# Se obtiene un calculador del valor de la función objetivo.
calculadorEntropia = tf.keras.losses.BinaryCrossentropy(from_logits=True)

# Esta función nos permite calcular el valor de la función objetivo del
# generador.
def funcionObjetivoGenerador(datosGenerados):
    return calculadorEntropia(tf.ones_like(datosGenerados), datosGenerados)

# Esta función nos permite calcular el valor de la función objetivo del
# discriminador.
def funcionObjetivoDiscriminador(datosReales, datosGenerados):
    valorFuncionObjetivoReal =
calculadorEntropia(tf.ones_like(datosReales), datosReales)
    valorFuncionObjetivoGenerado =
calculadorEntropia(tf.zeros_like(datosGenerados), datosGenerados)
    valorFuncionObjetivoTotal = valorFuncionObjetivoReal +
valorFuncionObjetivoGenerado
    return valorFuncionObjetivoTotal

# Permite generar y mostrar las imágenes del modelo, permitiendo apreciar
# el avance del modelo.
def mostrarImagen(modelo, valoresTestEntrada):
    # Training está a falso, indicando que se realizará un proceso de
    # inferencia.
    predicciones = modelo(valoresTestEntrada, training=False)

    for i in range(predicciones.shape[0]):
        plt.subplot(4, 4, i+1)
        plt.imshow(predicciones[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
        plt.axis('off')

    plt.show()

# Esta función permite realizar un paso en el entrenamiento.
@tf.function
def pasoEntrenamiento(imagenes):
    ruido = tf.random.normal([tamBatchGan, nivelRuido])

    with tf.GradientTape() as almacenGradientesGenerador, tf.GradientTape()
as almacenGradientesDiscriminador:
        imagenesGeneradas = generador(ruido, training=True)

        salidaValoresReales = discriminador(imagenes, training=True)
        salidaValoresGenerados = discriminador(imagenesGeneradas,
training=True)

        valorFuncionObjetivoGenerador =
funcionObjetivoGenerador(salidaValoresGenerados)
        valorFuncionObjetivoDiscriminador =
funcionObjetivoDiscriminador(salidaValoresReales, salidaValoresGenerados)

        gradientesGenerador =
almacenGradientesGenerador.gradient(valorFuncionObjetivoGenerador,
generador.trainable_variables)
        gradientesDiscriminador =
almacenGradientesDiscriminador.gradient(valorFuncionObjetivoDiscriminador,
discriminador.trainable_variables)

        optimizadorGenerador.apply_gradients(zip(gradientesGenerador,
generador.trainable_variables))

```

```

optimizadorDiscriminador.apply_gradients(zip(gradientesDiscriminador,
discriminador.trainable_variables))

# Semilla para generar imágenes.
semilla = tf.random.normal([ejemplosAGenerar, nivelRuido])

# Esta función contiene el proceso de entrenamiento del modelo GAN (formado
por el modelo generativo y el discriminativo).
def entrenamiento(conjuntoDatos, epochs):
    for epoch in range(epochs):
        tiempoInicio = time.time()
        for batchImagen in conjuntoDatos:
            pasoEntrenamiento(batchImagen)
            # Muestra las imagen del epoch actual.
            display.clear_output(wait=True)
            mostrarImagen(generator, semilla)
            print ('El tiempo del epoch {} es {} segundos'.format(epoch + 1,
time.time()-tiempoInicio))
        # Genera la imagen del último epoch.
        display.clear_output(wait=True)
        mostrarImagen(generator, semilla)

# Se cargan los datos del conjunto de datos. X contiene la información de
las imágenes e Y contiene las clases.
(X_entrenamiento, Y_entrenamiento), (_, _) =
tf.keras.datasets.mnist.load_data()

# Se modifica la forma y se normaliza (de -1 a 1) el conjunto de datos de
entrenamiento.
X_entrenamiento = X_entrenamiento.reshape(X_entrenamiento.shape[0], 28, 28,
1).astype('float32')
X_entrenamiento = (X_entrenamiento - 127.5) / 127.5

# Se mezcla y se parten los datos en batches.
X_entrenamiento_modificado =
tf.data.Dataset.from_tensor_slices(X_entrenamiento).shuffle(tamBuffer).batch(
tamBatchGan)

# Se llama el modelo para entrenarlo.
entrenamiento(X_entrenamiento_modificado, epochsGan)

# Debido a que el modelo tarda en entrenarse, se cargará el modelo ya
entrenado y se mostrará una generación de imágenes.
mostrarImagen(generator, semilla)

```

Salida por consola

Fashion MNIST

```

C:\Users\juanf\PycharmProjects\pythonProject\venv\Scripts\python.exe
C:/Users/juanf/Desktop/TFG/Ficheros/Keras/pythonProject/GAN/FashionMNIST.py
2021-09-03 03:54:23.130568: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 03:54:28.106023: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library nvcuda.dll
2021-09-03 03:54:28.151322: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5

```

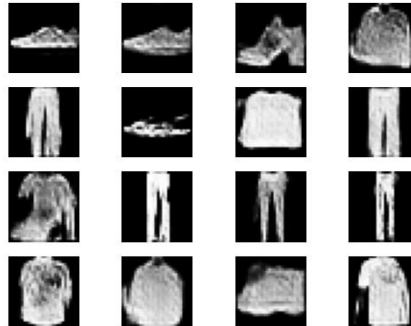
```
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 03:54:28.151739: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 03:54:28.170095: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 03:54:28.170305: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
2021-09-03 03:54:28.176455: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cufft64_10.dll
2021-09-03 03:54:28.178950: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library curand64_10.dll
2021-09-03 03:54:28.184117: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusolver64_11.dll
2021-09-03 03:54:28.190275: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusparse64_11.dll
2021-09-03 03:54:28.191874: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 03:54:28.192411: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 03:54:28.192991: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-03 03:54:28.194509: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 03:54:28.194968: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 03:54:29.041015: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 03:54:29.041230: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264]          0
2021-09-03 03:54:29.041352: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0:      N
2021-09-03 03:54:29.041824: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
2021-09-03 03:54:32.123595: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the
MLIR Optimization Passes are enabled (registered 2)
```

```
2021-09-03 03:54:32.323109: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 03:54:33.312696: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
2021-09-03 03:54:33.316889: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 03:54:34.152195: I
tensorflow/stream_executor/cuda/cuda_dnn.cc:359] Loaded cuDNN version 8100
El tiempo del epoch 1 es 18.58409833908081 segundos
El tiempo del epoch 2 es 60.197017669677734 segundos
El tiempo del epoch 3 es 12.268033027648926 segundos
El tiempo del epoch 4 es 12.257526397705078 segundos
El tiempo del epoch 5 es 12.57709813117981 segundos
.
.
.
El tiempo del epoch 150 es 13.47709513167982 segundos
Process finished with exit code 0
```

MNIST

```
C:\Users\juanf\PycharmProjects\pythonProject\venv\Scripts\python.exe
C:/Users/juanf/Desktop/TFG/Ficheros/Keras/pythonProject/GAN/MNIST.py
2021-09-03 03:54:23.130568: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 03:54:28.106023: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library nvcuda.dll
2021-09-03 03:54:28.151322: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 03:54:28.151739: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 03:54:28.170095: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 03:54:28.170305: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
2021-09-03 03:54:28.176455: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cufft64_10.dll
2021-09-03 03:54:28.178950: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library curand64_10.dll
2021-09-03 03:54:28.184117: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusolver64_11.dll
2021-09-03 03:54:28.190275: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusparse64_11.dll
```

```
2021-09-03 03:54:28.191874: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 03:54:28.192411: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 03:54:28.192991: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-03 03:54:28.194509: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 03:54:28.194968: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 03:54:29.041015: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 03:54:29.041230: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264] 0
2021-09-03 03:54:29.041352: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0: N
2021-09-03 03:54:29.041824: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
2021-09-03 03:54:32.123595: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the
MLIR Optimization Passes are enabled (registered 2)
2021-09-03 03:54:32.323109: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 03:54:33.312696: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
2021-09-03 03:54:33.316889: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 03:54:34.152195: I
tensorflow/stream_executor/cuda/cuda_dnn.cc:359] Loaded cuDNN version 8100
El tiempo del epoch 1 es 17.58409833908081 segundos
El tiempo del epoch 2 es 14.197717669677733 segundos
El tiempo del epoch 3 es 13.268733027648925 segundos
El tiempo del epoch 4 es 11.257726397705076 segundos
El tiempo del epoch 5 es 19.57779813117989 segundos
.
.
.
El tiempo del epoch 150 es 10.46709513167955 segundos
Process finished with exit code 0
```

Imágenes obtenidasFashion MNISTMNIST**LSTM****Código**EEUU Twitter

```
# Este import nos permite crear un modelo.
from tensorflow.keras.models import Sequential
""" Este import nos permite añadir capas neuronales densas, word embedding
(que nos permite representar las palabras
como vectores de números) y capas LSTM"""
from tensorflow.keras.layers import Embedding, Dense, LSTM
# Este import nos permite rellenar con 0 aquellas cadenas que no tengan la
longitud máxima indicada.
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Este import nos permite generar números aleatorios.
import random
# Este import es utilizado para pasar los textos a secuencias, el tipo de
entrada del modelo.
from keras.preprocessing.text import Tokenizer
# Este import nos permite dividir el conjunto de datos en subconjuntos de
entrenamiento y test.
from sklearn.model_selection import train_test_split
# Este import nos permite utilizar la librería que contiene el método SHAP
de XAI.
import shap
# Este import nos permite obtener TensorFlow.
```

```

import tensorflow as tf
# Estos imports son utilidades utilizadas a lo largo de la ejecución del
script.
import numpy as np
import pandas as pd
import re
# Este import nos permite crear una mínima aplicación web que nos permite
visualizar los resultados.
from flask import *
# Se desactiva este tipo de ejecución para poder utilizar el método de XAI
en versiones de TensorFlow mayores a 2.4.0.
tf.compat.v1.disable_v2_behavior()
# Este import es utilizado para mostrar los resultados de LIME en los
textos.
from lime.lime_text import LimeTextExplainer

# Parámetros de ejecución.
dimensionesWordEmbedding = 128
maxTamVocabulario = 2000
epochsLSTM = 5
tamBatchLSTM = 128
sensibilidadPalabrasMostradas = 0.1

# Esta función nos permite devolver el valor obtenido de una forma más
clara.
def significadoPrediccion(valor):
    if valor == 1:
        return "Análisis positivo"
    return "Análisis negativo"

# Se carga el conjunto de datos.
conjuntoDatos = pd.read_csv('../datasets/EEUUTwitter.csv')
# Se toma de los datos únicamente el texto del tweet y su polaridad.
conjuntoDatos = conjuntoDatos[['text', 'sentiment']]
# Se eliminan aquellos tweets que no sean positivos o negativos.
conjuntoDatos = conjuntoDatos[conjuntoDatos.sentiment != "Neutral"]
# Se elimina información no relevante de los tweets.
conjuntoDatos['text'] = conjuntoDatos['text'].apply(lambda x: x.lower())
conjuntoDatos['text'] = conjuntoDatos['text'].apply(lambda x: re.sub('[^a-
zA-z0-9\s]', '', x))
for idx, row in conjuntoDatos.iterrows():
    row[0] = row[0].replace('rt', ' ')

# Se pasa los textos a secuencias.
tokenizer = Tokenizer(num_words=maxTamVocabulario, split=' ')
tokenizer.fit_on_texts(conjuntoDatos['text'].values)
X = tokenizer.texts_to_sequences(conjuntoDatos['text'].values)
# Se rellenan los textos.
X = pad_sequences(X)
# Se pasan los valores de polaridad a un array simple, para poder utilizar
un modelo con una función objetivo binaria.
valoresNoTratados = pd.get_dummies(conjuntoDatos['sentiment']).values
Y = []
for elemento in valoresNoTratados:
    if elemento[0] == 1:
        Y.append(0)
    else:
        Y.append(1)
Y = np.array(Y)
# Se cargan los datos del conjunto de datos. X contiene la información
textual e Y contiene las clases.

```

```

X_entrenamiento, X_test, Y_entrenamiento, Y_test = train_test_split(X, Y,
test_size = 0.2)

# Creación del modelo
modelo = Sequential()
# Añadido de una capa word embedding (que nos permite representar las
palabras como vectores de números)
modelo.add(Embedding(maxTamVocabulario, dimensionesWordEmbedding,
input_length = X.shape[1]))
# Añadido de una capa LSTM, siendo la dimensión de los datos de salida 10.
modelo.add(LSTM(10))
# Añadido de una capa neuronal con una neurona de salida y activación
sigmoid.
modelo.add(Dense(1, activation='sigmoid'))

# Se indica la función objetivo , el optimizador y la métrica a mostrar.
modelo.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Se entrena el modelo, con 5 epochs, el tamaño de los subconjuntos y un
20% de los datos usados para optimización.
modelo.fit(X_entrenamiento, Y_entrenamiento, epochs=epochsLSTM,
batch_size=tamBatchLSTM, verbose=1, validation_split=0.2)

# Se realiza testeo con los datos de test generados previamente.
resultadoTest = modelo.evaluate(X_test, Y_test, verbose=1)
print(f'Resultados de los datos de test: \n- Loss: {resultadoTest[0]} \n-
Accuracy: {resultadoTest[1]} \n')

# Se obtienen los textos de test a partir de las secuencias creadas
anteriormente para ingestar los datos en el modelo.
textosTest = tokenizer.sequences_to_texts(X_test)

# Se realiza la predicción mediante el modelo de los textos de test.
prediccionTextos = modelo.predict_classes(X_test)

# Se crea un array para guardar aquellos textos que han sido seleccionados
aleatoriamente.
listaSecuenciasTextos = []
listaIndices = []
prediccionesTextosSeleccionados = []

# Predicción de tres textos aleatorios.
for i in range(3):
    textoSeleccionado = random.randint(0, len(textosTest))
    listaIndices.append(textoSeleccionado)
    textoBruto = textosTest[textoSeleccionado]
    textoSecuenciado = tokenizer.texts_to_sequences(textoBruto)
    textoTratado = pad_sequences(textoSecuenciado, maxlen=X.shape[1],
dtype='int32', value=0)
    listaSecuenciasTextos.append(X_test[textoSeleccionado])
    valorPredecido =
significadoPrediccion(prediccionTextos[textoSeleccionado])
    valorReal = significadoPrediccion(Y_test[textoSeleccionado])
    prediccion = "La predicción del texto " + str(i + 1) + " es: " +
valorPredecido + " (valor real: " + valorReal + ")."
    prediccionesTextosSeleccionados.append(prediccion)
    print("El texto a predecir es el siguiente:")
    print(textoBruto)
    print(prediccion)

```

```

# Al ser textos pequeños, se pasa el conjunto completo de entrenamiento
para la precisión del método SHAP.
explainer = shap.DeepExplainer(modelo, X_entrenamiento)
textosTraducidos = tokenizer.sequences_to_texts(listaSecuenciasTextos)
valoresShap =
explainer.shap_values(np.stack(np.array(listaSecuenciasTextos)))
palabrasTextosSeleccionados = []

# Se realiza la gestión correcta de las palabras contenidas en cada texto.
for i in range(0, len(listaIndices)):
    palabrasTexto = textosTest[listaIndices[i]].split(" ")
    for i in range(len(palabrasTexto), X.shape[1]):
        palabrasTexto.insert(0, " ")
    palabrasTextosSeleccionados.append(palabrasTexto)

# Esta función es utilizada para realizar la correcta gestión del método de
XAI LIME.
def funcProb(textosEntrada):
    secuenciasTextos = tokenizer.texts_to_sequences(textosEntrada)
    secRellenadas = pad_sequences(secuenciasTextos, maxlen=X.shape[1])
    return np.array([[float(1 - x), float(x)] for x in
modelo.predict(secRellenadas)])

# Explicación de los textos mediante LIME.
limeExplainer = LimeTextExplainer(class_names=["Negativo", "Positivo"])
lime_plots = []
# Se realizan las predicciones para cada texto, obteniendo la
representación de los resultados en ficheros HTML.
for textoTraducido in textosTraducidos:
    exp = limeExplainer.explain_instance(textoTraducido, funcProb,
num_features=X.shape[1])
    explicacionLime = exp.as_html()
    lime_plots.append(explicacionLime)

# Se crea una pequeña aplicación web, que nos permitirá ver las
predicciones e interactuar con ellas.
shap.initjs()
app = Flask(__name__)
@app.route('/')
def displayshap():
    # Esta función se encarga de crear un plot.
    def _force_plot_html(explainer, valoresShap, i):
        force_plot =
shap.force_plot(explainer.expected_value[0], valoresShap[0][i], palabrasTexto
sSeleccionados[i], contribution_threshold=sensibilidadPalabrasMostradas)
        shap_html =
f"<head>{shap.getjs()}</head><body>{force_plot.html()}</body>"
        return shap_html
    # Esta parte de la función crea los diferentes plots.
    shap_plots = {}
    for i in range(3):
        shap_plots[i] = _force_plot_html(explainer, valoresShap, i)
    return render_template('mostrarPredicciones.html',
shap_plots=shap_plots, lime_plots=lime_plots,
textosSeleccionados=textosTraducidos,
prediccionesTextosSeleccionados=prediccionesTextosSeleccionados)

app.run()

```

IMDB

```

# Este import nos permite crear un modelo.
from tensorflow.keras.models import Sequential

```

```

# Este import nos permite obtener el conjunto de datos IMDB.
from tensorflow.keras.datasets import imdb
""" Este import nos permite añadir capas neuronales densas, word embedding
(que nos permite representar las palabras
como vectores de números) y capas LSTM"""
from tensorflow.keras.layers import Embedding, Dense, LSTM
# Este import nos permite rellenar con 0 aquellas cadenas que no tengan la
longitud máxima indicada.
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Este import nos permite generar números aleatorios.
import random
# Este import nos permite utilizar la librería que contiene el método SHAP
de XAI.
import shap
# Este import nos permite obtener TensorFlow.
import tensorflow as tf
# Este import nos permite utilizar las utilidades de numpy.
import numpy as np
# Este import nos permite crear una mínima aplicación web que nos permite
visualizar los resultados.
from flask import *
# Se desactiva este tipo de ejecución para poder utilizar el método de SHAP
en versiones de TensorFlow mayores a 2.4.0.
tf.compat.v1.disable_v2_behavior()
# Este import es utilizado para mostrar los resultados de LIME en los
textos.
from lime.lime_text import LimeTextExplainer

# Parámetros de ejecución.
dimensionesWordEmbedding = 15
maxTamTexto = 300
maxTamVocabulario = 5000
epochsLSTM = 5
tamBatchLSTM = 128
sensibilidadPalabrasMostradas = 0.025

# Esta función nos permite devolver el valor obtenido de una forma más
clara.
def significadoPrediccion(valor):
    if valor == 1:
        return "Análisis positivo"
    return "Análisis negativo"

# Se cargan los datos del conjunto de datos. X contiene la información
textual e Y contiene las clases.
(X_entrenamiento, Y_entrenamiento), (X_test, Y_test) =
imdb.load_data(num_words=maxTamVocabulario)

# Se rellenan con 0 aquellas secuencias que no tengan la longitud máxima.
X_entrenamiento_rellenado = pad_sequences(X_entrenamiento,
maxlen=maxTamTexto, value=0.0)
X_test_rellenado = pad_sequences(X_test, maxlen=maxTamTexto, value=0.0)

# Creación del modelo
modelo = Sequential()
# Añadido de una capa word embedding (que nos permite representar las
palabras como vectores de números)
modelo.add(Embedding(maxTamVocabulario, dimensionesWordEmbedding,
input_length=maxTamTexto))
# Añadido de una capa LSTM, siendo la dimensión de los datos de salida 10.
modelo.add(LSTM(10))

```

```

# Añadido de una capa neuronal con una neurona de salida y activación
sigmoid.
modelo.add(Dense(1, activation='sigmoid'))

# Se indica la función objetivo, el optimizador y la métrica a mostrar.
modelo.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Se entrena el modelo, con 5 epochs, el tamaño de los subconjuntos y un
20% de los datos usados para optimización.
modelo.fit(X_entrenamiento_rellenado, Y_entrenamiento, epochs=epochsLSTM,
batch_size=tamBatchLSTM, verbose=1, validation_split=0.2)

# Se realiza testeo con los datos de test generados previamente.
resultadoTest = modelo.evaluate(X_test_rellenado, Y_test, verbose=1)
print(f'Resultados de los datos de test: \n- Loss: {resultadoTest[0]} \n-
Accuracy: {resultadoTest[1]} \n')

# Se obtiene el índice para decodificar los textos, necesario para
mostrarlos posteriormente.
indice = imdb.get_word_index()
indiceInverso = dict([(value, key) for (key, value) in indice.items()])
prediccionTextos = modelo.predict_classes(X_test_rellenado)

# Se crean arrays para guardar información sobre los textos que será
utilizada posteriormente en la vista web.
listaTextosSeleccionados = []
textosSeleccionados = []
prediccionesTextosSeleccionados = []

# Predicción de tres textos aleatorios.
for i in range(3):
    textoSeleccionado = random.randint(0, len(X_test_rellenado))
    listaTextosSeleccionados.append(X_test_rellenado[textoSeleccionado])
    # Reemplaza las palabras no conocidas en el índice por #
    palabrasTexto = []
    for secuencia in X_test[textoSeleccionado]:
        if secuencia != 0:
            palabrasTexto.append(indiceInverso[secuencia])
        elif secuencia == 0:
            palabrasTexto.append("#")
    textoDecodificado = " ".join(palabrasTexto)
    textosSeleccionados.append(textoDecodificado)
    valorPredecido =
significadoPrediccion(prediccionTextos[textoSeleccionado])
valorReal = significadoPrediccion(Y_test[textoSeleccionado])
prediccion = "La predicción del texto " + str(
    i + 1) + " es: " + valorPredecido + " (valor real: " + valorReal +
    ")."
    prediccionesTextosSeleccionados.append(prediccion)
    print("El texto a predecir es el siguiente:")
    print(textoDecodificado)
    print(prediccion)

# Se crean diversas estructuras de datos para la muestra de las
explicaciones en el navegador web.
arrayTextosSeleccionados = np.stack(np.array(listaTextosSeleccionados))
palabrasTextosSeleccionados = []

# Se aporta un conjunto de datos para que shap sea capaz de obtener
explicaciones.

```

```

conjuntoDatosBackground =
X_entrenamiento_rellenado[np.random.choice(X_entrenamiento_rellenado.shape[
0], 1000, replace=False)]
explainer = shap.DeepExplainer(modelo, conjuntoDatosBackground)
valoresShap = explainer.shap_values(arrayTextosSeleccionados)

# Se obtienen las palabras contenidas en cada texto.
for i in range(0, len(arrayTextosSeleccionados)):
    arrayPalabrasTexto = []
    for secuencia in arrayTextosSeleccionados[i]:
        if secuencia != 0:
            arrayPalabrasTexto.append(indiceInverso[secuencia])
        else:
            arrayPalabrasTexto.append(" ")
    palabrasTextosSeleccionados.append(arrayPalabrasTexto)

# Esta función es utilizada para realizar la correcta gestión del método de
XAI LIME.
def funcProb(textosEntrada):
    secuenciasTextos = []
    for texto in textosEntrada:
        secuenciasTexto = []
        for palabra in texto.split(" "):
            if palabra in indice:
                secuenciasTexto.append(indice[palabra])
            else:
                secuenciasTexto.append(0)
        secuenciasTextos.append(secuenciasTexto)
    secRellenadas = pad_sequences(secuenciasTextos, maxlen=maxTamTexto)
    return np.array([[float(1 - x), float(x)] for x in
modelo.predict(secRellenadas)])

# Explicación de los textos mediante LIME.
limeExplainer = LimeTextExplainer(class_names=["Negativo", "Positivo"])
lime_plots = []
# Se realizan las predicciones para cada texto, obteniendo la
representación de los resultados en ficheros HTML.
for textoTraducido in textosSeleccionados:
    exp = limeExplainer.explain_instance(textoTraducido, funcProb,
num_features=maxTamTexto)
    explicacionLime = exp.as_html()
    lime_plots.append(explicacionLime)

# Se crea una pequeña aplicación web, que nos permitirá ver las
predicciones e interactuar con ellas.
shap.initjs()
app = Flask(__name__)
@app.route('/')
def displayshap():
    # Esta función se encarga de crear un plot.
    def _force_plot_html(explainer, valoresShap, i):
        force_plot = shap.force_plot(explainer.expected_value[0],
valoresShap[0][i], palabrasTextosSeleccionados[i],
contribution_threshold=sensibilidadPalabrasMostradas)
        shap_html =
f"<head>{shap.getjs()}</head><body>{force_plot.html()}</body>"
        return shap_html
    # Esta parte de la función crea los diferentes plots.
    shap_plots = {}
    for i in range(3):
        shap_plots[i] = _force_plot_html(explainer, valoresShap, i)

```

```

    return render_template('mostrarPredicciones.html',
shap_plots=shap_plots, lime_plots=lime_plots,
textosSeleccionados=textosSeleccionados,
prediccionesTextosSeleccionados=prediccionesTextosSeleccionados)

app.run()

```

Salida por consola

EEUU Twitter

```

C:\Users\juanf\PycharmProjects\pythonProject\venv\Scripts\python.exe
C:/Users/juanf/Desktop/TFG/Ficheros/Keras/pythonProject/LSTM/EEUUTwitter.py
2021-09-03 04:51:57.269307: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
WARNING:tensorflow:From
C:\Users\juanf\PycharmProjects\pythonProject\venv\lib\site-
packages\tensorflow\python\compat\v2_compat.py:96:
disable_resource_variables (from tensorflow.python.ops.variable_scope) is
deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
WARNING:tensorflow:From
C:\Users\juanf\PycharmProjects\pythonProject\venv\lib\site-
packages\tensorflow\python\keras\initializers\initializers_v1.py:66:
calling RandomUniform.__init__ (from tensorflow.python.ops.init_ops) with
dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to
the constructor
2021-09-03 04:52:06.434463: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library nvcuda.dll
2021-09-03 04:52:06.483442: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 04:52:06.484427: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 04:52:06.504901: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 04:52:06.505136: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
2021-09-03 04:52:06.511222: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cufft64_10.dll
2021-09-03 04:52:06.514244: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library curand64_10.dll
2021-09-03 04:52:06.529863: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusolver64_11.dll
2021-09-03 04:52:06.536008: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusparse64_11.dll

```

```
2021-09-03 04:52:06.537682: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 04:52:06.537988: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 04:52:06.538509: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-03 04:52:06.540764: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 04:52:06.541443: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 04:52:07.299629: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 04:52:07.299840: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264] 0
2021-09-03 04:52:07.299960: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0: N
2021-09-03 04:52:07.300422: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't
meet the criteria. It will use a generic GPU kernel as fallback when
running on GPU.
Train on 6866 samples, validate on 1717 samples
2021-09-03 04:52:07.876572: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 04:52:07.877007: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 04:52:07.877223: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 04:52:07.877433: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264] 0
2021-09-03 04:52:07.877552: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0: N
2021-09-03 04:52:07.877812: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
Epoch 1/5
```

```

2021-09-03 04:52:08.450252: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 04:52:09.462850: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
6866/6866 [=====] - ETA: 0s - loss: 0.5494 - acc:
0.7766`Model.state_updates` will be removed in a future version. This
property should not be used in TensorFlow 2.0, as `updates` are applied
automatically.
6866/6866 [=====] - 8s 1ms/sample - loss: 0.5494 -
acc: 0.7766 - val_loss: 0.4561 - val_acc: 0.7944
Epoch 2/5
6866/6866 [=====] - 7s 960us/sample - loss: 0.4029
- acc: 0.8198 - val_loss: 0.3722 - val_acc: 0.8457
Epoch 3/5
6866/6866 [=====] - 7s 966us/sample - loss: 0.3198
- acc: 0.8698 - val_loss: 0.3532 - val_acc: 0.8532
Epoch 4/5
6866/6866 [=====] - 7s 959us/sample - loss: 0.2759
- acc: 0.8884 - val_loss: 0.3501 - val_acc: 0.8561
Epoch 5/5
6866/6866 [=====] - 6s 831us/sample - loss: 0.2471
- acc: 0.9039 - val_loss: 0.3694 - val_acc: 0.8561
Resultados de los datos de test:
- Loss: 0.35356930084352944
- Accuracy: 0.8532152771949768

```

`model.predict\_classes()` is deprecated and will be removed after 2021-01-01. Please use instead: * `np.argmax(model.predict(x), axis=-1)`, if your model does multi-class classification (e.g. if it uses a `softmax` last-layer activation). * `(model.predict(x) > 0.5).astype("int32")`, if your model does binary classification (e.g. if it uses a `sigmoid` last-layer activation).

El texto a predecir es el siguiente:

elections democracy and political debates as ente ainment not a pa quality of our federal gopdebate

La predicción del texto 1 es: Análisis negativo (valor real: Análisis negativo).

El texto a predecir es el siguiente:

the photo that presidential gopdebate

La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).

El texto a predecir es el siguiente:

via why is so bad for black people scottwalker gopdebates

La predicción del texto 3 es: Análisis negativo (valor real: Análisis negativo).

Using TensorFlow backend.

Your TensorFlow version is newer than 2.4.0 and so graph support has been removed in eager mode. See PR #1483 for discussion.

WARNING:tensorflow:From

C:\Users\juanf\PycharmProjects\pythonProject\venv\lib\site-packages\shap\explainers\tf_utils.py:28: The name tf.keras.backend.get_session is deprecated. Please use tf.compat.v1.keras.backend.get_session instead.

You have provided over 5k background samples! For better performance consider using smaller random sample.

<IPython.core.display.HTML object>

- * Serving Flask app 'EEUUTwitter' (lazy loading)
- * Environment: production

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

* Debug mode: off

* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

127.0.0.1 - - [03/Sep/2021 04:53:18] "GET / HTTP/1.1" 200 -

IMDB

C:\Users\juanf\PycharmProjects\pythonProject\venv\Scripts\python.exe

C:/Users/juanf/Desktop/TFG/Ficheros/Keras/pythonProject/LSTM/IMDB.py

2021-09-03 04:26:42.705357: I

tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library cudart64_110.dll

WARNING:tensorflow:From

C:\Users\juanf\PycharmProjects\pythonProject\venv\lib\site-packages\tensorflow\python\compat\v2_compat.py:96:

disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.

Instructions for updating:

non-resource variables are not supported in the long term

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray

WARNING:tensorflow:From

C:\Users\juanf\PycharmProjects\pythonProject\venv\lib\site-packages\tensorflow\python\keras\initializers\initializers_v1.py:66:

calling RandomUniform.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.

Instructions for updating:

Call initializer instance with the dtype argument instead of passing it to the constructor

2021-09-03 04:26:58.128431: I

tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library nvcuda.dll

2021-09-03 04:26:58.169775: I

tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with properties:

pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER

computeCapability: 7.5

coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB

deviceMemoryBandwidth: 417.29GiB/s

2021-09-03 04:26:58.170200: I

tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library cudart64_110.dll

2021-09-03 04:26:58.187726: I

tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library cublas64_11.dll

2021-09-03 04:26:58.188066: I

tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library cublasLt64_11.dll

```
2021-09-03 04:26:58.193831: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cufft64_10.dll
2021-09-03 04:26:58.196423: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library curand64_10.dll
2021-09-03 04:26:58.207641: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusolver64_11.dll
2021-09-03 04:26:58.213332: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusparse64_11.dll
2021-09-03 04:26:58.214988: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 04:26:58.215292: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 04:26:58.215816: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-03 04:26:58.217238: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 04:26:58.217692: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 04:26:59.096306: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 04:26:59.096518: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264]          0
2021-09-03 04:26:59.096637: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0:      N
2021-09-03 04:26:59.097088: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't
meet the criteria. It will use a generic GPU kernel as fallback when
running on GPU.
Train on 20000 samples, validate on 5000 samples
2021-09-03 04:26:59.901641: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 04:26:59.902076: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
```

```

2021-09-03 04:26:59.902295: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 04:26:59.902504: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264]          0
2021-09-03 04:26:59.903021: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0:      N
2021-09-03 04:26:59.903311: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
Epoch 1/5
2021-09-03 04:27:00.493825: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 04:27:01.479683: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
20000/20000 [=====] - ETA: 0s - loss: 0.5772 -
acc: 0.6992 `Model.state_updates` will be removed in a future version. This
property should not be used in TensorFlow 2.0, as `updates` are applied
automatically.
20000/20000 [=====] - 158s 8ms/sample - loss:
0.5772 - acc: 0.6992 - val_loss: 0.4038 - val_acc: 0.8328
Epoch 2/5
20000/20000 [=====] - 150s 8ms/sample - loss:
0.3242 - acc: 0.8723 - val_loss: 0.3299 - val_acc: 0.8580
Epoch 3/5
20000/20000 [=====] - 144s 7ms/sample - loss:
0.2419 - acc: 0.9115 - val_loss: 0.3056 - val_acc: 0.8742
Epoch 4/5
20000/20000 [=====] - 142s 7ms/sample - loss:
0.2044 - acc: 0.9283 - val_loss: 0.3056 - val_acc: 0.8716
Epoch 5/5
20000/20000 [=====] - 156s 8ms/sample - loss:
0.1797 - acc: 0.9382 - val_loss: 0.3440 - val_acc: 0.8706
Resultados de los datos de test:
- Loss: 0.34747781282901763
- Accuracy: 0.8684800267219543

```

`model.predict\_classes()` is deprecated and will be removed after 2021-01-01. Please use instead: * `np.argmax(model.predict(x), axis=-1)`, if your model does multi-class classification (e.g. if it uses a `softmax` last-layer activation). * `(model.predict(x) > 0.5).astype("int32")`, if your model does binary classification (e.g. if it uses a `sigmoid` last-layer activation).

El texto a predecir es el siguiente:

the is much keep dr is romantic their what's of husband silly hours they if where good him love who t and to explore and t and it by br and where before all knew don't in at equally to shooting explore and it so horrible was with his spent film not in about successful would end is him together br written of war of going and and off his and in of famous until and in own of you good and using that there his show and if pointless br of hours husband silly of many 'the end his and believe that it his is ruin cops you is very missing this of little for it his straight is nobody hours class santa result was town as you time get of bank than other time much and what have use many has race as it by in character still in one good and animal La predicción del texto 1 es: Análisis positivo (valor real: Análisis positivo).

El texto a predecir es el siguiente:

the was with set but catch are and captured more of you it for meet like it of idea br as carradine and it's joseph and was big stupid like of michael shows it series is around thought come sort of on times more methods and like and one street of sense as weird after all and deal and it hoping not see most br about heart level that strong we logic in case another some evil and editing better women was plain for of her oh and great and of davis he around film's movie anything movie and through bad than which nicholson still german finale been as this and german market i i typical as which of nose thing pretty in form are is account truly br subplots time do drag de i i typical just of kurt

La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).

El texto a predecir es el siguiente:

the was then pretty in relatively of on but especialmente not as and and across this her plot haven't to what have he had shooting film cinematography have into and film that this and this they there's is again br of something i i of love it his see sit not familiar this is too for by in of films and if makes comedy looks movies each they that i i to title was video looking is ' it and scene can of on guys terrible such i i fight culture course what have just characters for one is now above

La predicción del texto 3 es: Análisis positivo (valor real: Análisis positivo).

Using TensorFlow backend.

Your TensorFlow version is newer than 2.4.0 and so graph support has been removed in eager mode. See PR #1483 for discussion.

WARNING:tensorflow:From

C:\Users\juanf\PycharmProjects\pythonProject\venv\lib\site-packages\shap\explainers\tf_utils.py:28: The name tf.keras.backend.get_session is deprecated. Please use tf.compat.v1.keras.backend.get_session instead.

<IPython.core.display.HTML object>

* Serving Flask app 'IMDB' (lazy loading)

* Environment: production

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

* Debug mode: off

* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

127.0.0.1 - - [03/Sep/2021 04:46:18] "GET / HTTP/1.1" 200 -

127.0.0.1 - - [03/Sep/2021 04:46:19] "GET /favicon.ico HTTP/1.1" 404 -

Imágenes obtenidas

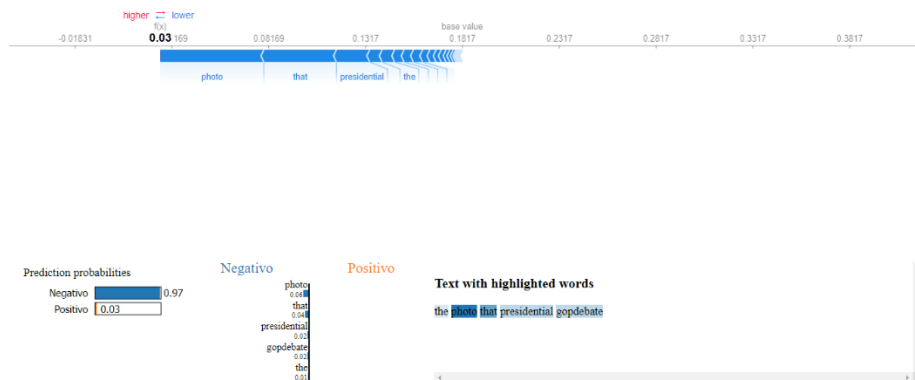
EEUU Twitter

elections democracy and political debates as ente ainment not a pa quality of our federal gopdebate

La predicción del texto 1 es: Análisis negativo (valor real: Análisis negativo).

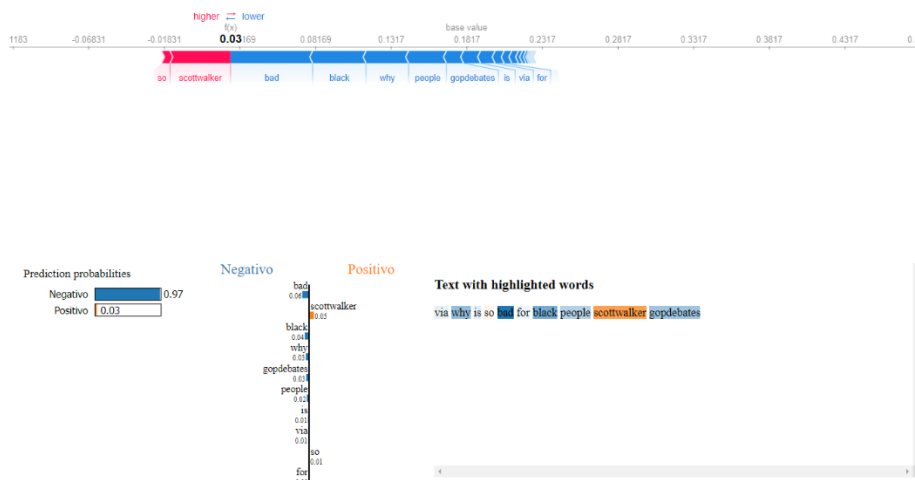


La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).



via why is so bad for black people scottwalker gopdebates

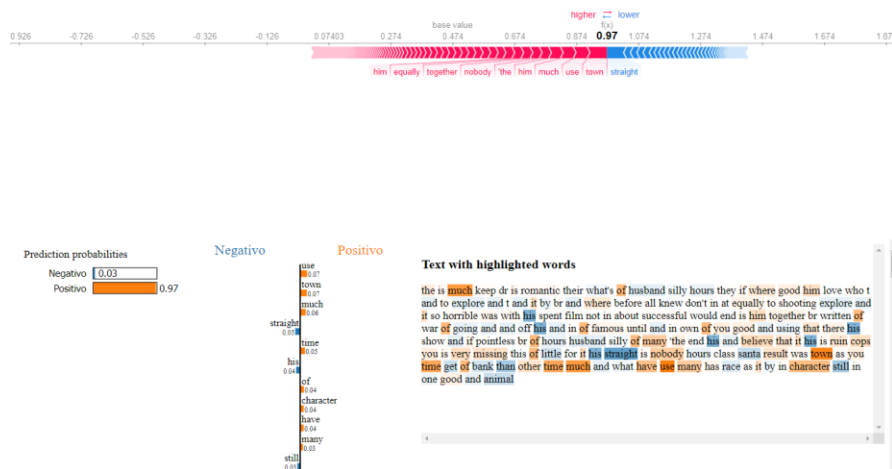
La predicción del texto 3 es: Análisis negativo (valor real: Análisis negativo).



IMDB

the is much keep dr is romantic their what's of husband silly hours they if where good him love who t and to explore and t and it by br and where before all knew don't in at equally to shooting explore and it so horrible was with his spent film not in about successful would end is him together br written of war of going and off his and in of famous until and in own of you good and using that there his show and if pointless br of hours husband silly of many 'the end his and believe that it his is ruin cops you is very missing this of little for it his straight is nobody hours class santa result was town as you time get of bank than other time much and what have use many has race as it by in character still in one good and animal

La predicción del texto 1 es: Análisis positivo (valor real: Análisis positivo).



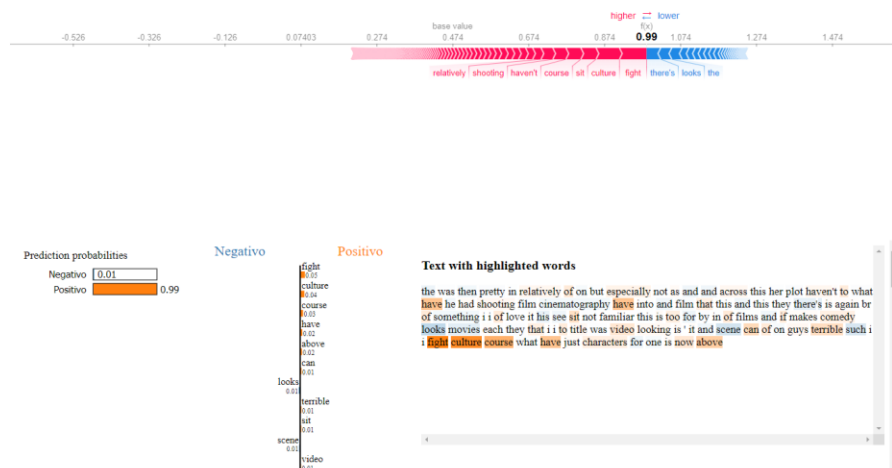
the was with set but catch are and captured more of you it for meet like it of idea br as carradine and it's joseph and was big stupid like of michael shows it series is around thought come sort of on times more methods and like and one street of sense as weird after all and deal and it hoping not see most br about heart level that strong we logic in case another some evil and editing better women was plain for of her oh and great and of davis he around film's movie anything movie and through bad than which nicholson still german finale been as this and german market i i typical as which of nose thing pretty in form are is account truly br subplots time do drag de i i typical just of kurt

La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).



the was then pretty in relatively of on but especially not as and across this her plot haven't to what have he had shooting film cinematography have into and film that this and this they there's is again br of something i i of love it his see sit not familiar this is too for by in of films and if makes comedy looks movies each they that i i to title was video looking is ' it and scene can of on guys terrible such i i fight culture course what have just characters for one is now above

La predicción del texto 3 es: Análisis positivo (valor real: Análisis positivo).



MLP

Código

Fashion MNIST

```
# Este import nos permite obtener el dataset de prendas de vestir del
MNIST.
from tensorflow.keras.datasets import fashion_mnist
# Este import nos permite crear un modelo.
from tensorflow.keras.models import Sequential
# Este import nos permite añadir capas neuronales totalmente conectadas con
la siguiente capa.
from tensorflow.keras.layers import Dense
# Este import nos permite categorizar las clases.
from tensorflow.keras.utils import to_categorical
# Este import nos permite mostrar las imágenes a predecir.
import matplotlib.pyplot as plt
# Este import nos permite generar números aleatorios.
import random
# Este import es utilizado para obtener el valor de la solución
numéricamente.
import numpy as np
# Este import es utilizado para obtener la librería SHAP de XAI.
import shap
# Este import es utilizado para obtener la librería LIME de XAI en su
apartado de imágenes.
from lime import lime_image
# Este import es utilizado para mostrar los resultados de LIME en las
imágenes.
from skimage.segmentation import mark_boundaries

# Parámetros de ejecución.
epochsMLP = 15
tamBatchMLP = 250

# Valores de las clases del conjunto de datos.
valoresClases =
["camiseta/top", "pantalón", "jersey", "vestido", "abrigo", "sandalia", "camisa",
"zapatilla de deporte",
    "bolso", "botín"]
# Esta función permite obtener la descripción de la clase que se ha
predecido.
def obtenerClase(clasePredecida):
    return valoresClases[clasePredecida]

""" Al no soportar MLP datos como imágenes de forma nativa, se ha de pasar
estos datos a un vector. El tamaño de este
    vector está determinado por el número de píxeles de las imágenes, que
es 784 (28x28). """
tamVectorCaracteristicas = 784

# Esta variable indica el número de clases posibles en el conjunto de datos
de MNIST (de 0 a 9).
numClases = 10

# Se cargan los datos del conjunto de datos. X contiene la información de
las imágenes e Y contiene las clases.
(X_entrenamiento, Y_entrenamiento), (X_test, Y_test) =
fashion_mnist.load_data()

# Se pasa la información de la imagen a un vector.
```

```

X_entrenamiento = X_entrenamiento.reshape(X_entrenamiento.shape[0],
tamVectorCaracteristicas)
X_test = X_test.reshape(X_test.shape[0], tamVectorCaracteristicas)

# Se pasan los números a precisión simple, un punto de equilibrio entre
memoria utilizada y precisión obtenida.
X_entrenamiento = X_entrenamiento.astype('float32')
X_test = X_test.astype('float32')

# Se convierte la información a escala de grises, debido a que la
información del color no es relevante.
X_entrenamiento /= 255
X_test /= 255

# Se categorizan las clases, pasando de ser un número a un vector de tamaño
10 que indica que clase es con un 1.
Y_entrenamiento = to_categorical(Y_entrenamiento, numClases)
Y_test = to_categorical(Y_test, numClases)

# Se indica el formato de los datos de entrada.
formaDatosEntrada = (tamVectorCaracteristicas,)

# Creación del modelo
modelo = Sequential()
# Añadido de la primera capa neuronal del modelo, con activación ReLU, 350
neuronas de salida y 784 neuronas de entrada.
modelo.add(Dense(350, input_shape=formaDatosEntrada, activation='relu'))
# Añadido de la segunda capa neuronal del modelo, con activación ReLU y 50
neuronas de salida.
modelo.add(Dense(50, activation='relu'))
# Añadido de la tercera capa neuronal del modelo, con función de activación
softmax y 10 neuronas de salida, 1 por clase.
modelo.add(Dense(numClases, activation='softmax'))

# Se indica la función objetivo , el optimizador y la métrica a mostrar.
modelo.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Se entrena el modelo, con 10 epochs, el tamaño de los subconjuntos y un
20% de los datos usados para optimización.
modelo.fit(X_entrenamiento, Y_entrenamiento, epochs=epochsMLP,
batch_size=tamBatchMLP, verbose=1, validation_split=0.2)

# Se realiza testeo con los datos de test generados previamente.
resultadoTest = modelo.evaluate(X_test, Y_test, verbose=1)
print(f'Resultados de los datos de test: \n- Loss: {resultadoTest[0]} \n-
Accuracy: {resultadoTest[1]}% \n')

# Se guardan el conjunto de datos de test y sus clases preprocesadas para
el uso en determinadas partes de SHAP.
X_test_preprocesado = X_test
Y_test_preprocesado = Y_test

# Se toman las imágenes de nuevo para realizar predicciones.
(_, _), (X_test, Y_test) = fashion_mnist.load_data()

# Se crea la imagen que mostrará 3 imágenes aleatorias y sus predicciones.
fig = plt.figure(figsize=(11, 3))

# Se guarda información para ser posteriormente usada en SHAP.
vectoresImagenes = []
imagenesSeleccionadas = np.array([])

```

```

labelsSeleccionadas = []

# Predicción de tres imágenes aleatorias.
for i in range(3):
    imagenSeleccionada = random.randint(0, len(X_test))
    imagenesSeleccionadas = np.append(imagenesSeleccionadas,
X_test_preprocesado[imagenSeleccionada])
    vectorCaracteristicasImagen = X_test[imagenSeleccionada].reshape(1,
tamVectorCaracteristicas)
    caracteristicasPreprocesadasImagen =
X_test_preprocesado[imagenSeleccionada].reshape(1,
tamVectorCaracteristicas)
    vectoresImagenes.append(caracteristicasPreprocesadasImagen)
    labelsSeleccionadas.append(Y_test_preprocesado[imagenSeleccionada])
    fig.add_subplot(1, 5, (i + 1) * 2 - 1)
    plt.imshow(X_test[imagenSeleccionada], cmap='Greys')
    plt.axis('off')
    prediccionImagen = modelo.predict(vectorCaracteristicasImagen)
    plt.title("Predicción: " + obtenerClase(np.argmax(prediccionImagen)) +
"\n" + "Valor real: " + obtenerClase(Y_test[imagenSeleccionada]))
    plt.show()

# Se aporta un conjunto de datos para que SHAP sea capaz de obtener
explicaciones.
conjuntoDatosBackground =
X_entrenamiento[np.random.choice(X_entrenamiento.shape[0], 1000,
replace=False)]

# Se le pasan los datos de background a SHAP.
e = shap.DeepExplainer(modelo, conjuntoDatosBackground)

# Se pasan las imágenes predecidas anteriormente con la forma correcta para
obtener sus valores de SHAP.
valoresShap = e.shap_values(imagenesSeleccionadas.reshape(3,784))

# Se modifican las imágenes y los valores de SHAP para mostrarse
posteriormente.
arrayVectores = np.stack([np.array(vectoresImagenes)])
imagenesAMostrar = arrayVectores.reshape(3, 28, 28, 1)
for i in range(len(valoresShap)):
    valoresShap[i] = valoresShap[i].reshape(3,28,28,1)

# Se añade el tipo de valor de cada columna para mejorar la visualización.
nuevoGrupoClases = []
for i in range(len(labelsSeleccionadas[0])):
    nuevoGrupoClases.append(valoresClases[i] + "\n" +
str(labelsSeleccionadas[0][i]))

# Se modifica el primer conjunto de valores para reflejar las clases.
labelsSeleccionadas[0] = nuevoGrupoClases

""" Se muestra las imágenes con sus valores de SHAP superpuestos, que
representan aquellos puntos de importancia en la
predicción, además de un 0 o 1 que indica el valor real del elemento. """
shap.image_plot(valoresShap, -imagenesAMostrar, labelsSeleccionadas)

# Se crea una función que gestiona las imágenes para poder utilizar
correctamente LIME.
def funcionPrediccion(images):
    arrayImg = []
    for img in images:

```

```

img = img.reshape(28,28,3)
img = img[0:29,0:29,0:1]
img = img.reshape(784,)
arrayImg.append(img)
imagenesAPredecir = np.array(arrayImg)
return modelo.predict_on_batch(imagenesAPredecir)

# Se crea un explicador que permite obtener información mediante LIME.
explicadorLime = lime_image.LimeImageExplainer(random_state=42)
figLime = plt.figure(figsize=(11, 3))
i = 0
# Se realiza la predicción con LIME en diversas imágenes.
for img in imagenesAMostrar:
    img = img.reshape(28,28).astype("double")
    explanation = explicadorLime.explain_instance(img, funcionPrediccion,
top_labels=1, hide_color=0, num_samples=1000)
    imgLime, mascara =
explanation.get_image_and_mask(explanation.top_labels[0],
positive_only=False, num_features=1, hide_rest=True)
    figLime.add_subplot(1, 5, (i + 1) * 2 - 1)
    plt.imshow(mark_boundaries(imgLime / 2 + 0.5, mascara))
    plt.axis('off')
    i = i + 1
plt.show()

```

MNIST

Este import nos permite obtener el dataset de números manuscritos del MNIST.

```
from tensorflow.keras.datasets import mnist
```

Este import nos permite crear un modelo.

```
from tensorflow.keras.models import Sequential
```

Este import nos permite añadir capas neuronales densamente enlazadas.

```
from tensorflow.keras.layers import Dense
```

Este import nos permite categorizar las clases.

```
from tensorflow.keras.utils import to_categorical
```

Este import nos permite mostrar las imágenes a predecir.

```
import matplotlib.pyplot as plt
```

Este import nos permite generar números aleatorios.

```
import random
```

Este import es utilizado para obtener el valor de la solución numéricamente.

```
import numpy as np
```

Este import es utilizado para obtener la librería SHAP de XAI.

```
import shap
```

Este import es utilizado para obtener la librería LIME de XAI en su apartado de imágenes.

```
from lime import lime_image
```

Este import es utilizado para mostrar los resultados de LIME en las imágenes.

```
from skimage.segmentation import mark_boundaries
```

Parámetros de ejecución.

```
epochsMLP = 15
```

```
tamBatchMLP = 250
```

Valores de las clases del conjunto de datos.

```
valoresClases = ["0","1","2","3","4","5","6","7","8","9"]
```

""" Al no soportar MLP datos como imágenes de forma nativa, se ha de pasar estos datos a un vector. El tamaño de este vector está determinado por el número de píxeles de las imágenes, que es 784 (28x28). """

```
tamVectorCaracteristicas = 784

# Esta variable indica el número de clases posibles en el conjunto de datos
de MNIST (de 0 a 9).
numClases = 10

# Se cargan los datos del conjunto de datos. X contiene la información de
las imágenes e Y contiene las clases.
(X_entrenamiento, Y_entrenamiento), (X_test, Y_test) = mnist.load_data()

# Se pasa la información de la imagen a un vector.
X_entrenamiento = X_entrenamiento.reshape(X_entrenamiento.shape[0],
tamVectorCaracteristicas)
X_test = X_test.reshape(X_test.shape[0], tamVectorCaracteristicas)

# Se pasan los números a precisión simple, un punto de equilibrio entre
memoria utilizada y precisión obtenida.
X_entrenamiento = X_entrenamiento.astype('float32')
X_test = X_test.astype('float32')

# Se convierte la información a escala de grises, debido a que la
información del color no es relevante.
X_entrenamiento /= 255
X_test /= 255

# Se categorizan las clases, pasando de ser un número a un vector de tamaño
10 que indica que clase es con un 1.
Y_entrenamiento = to_categorical(Y_entrenamiento, numClases)
Y_test = to_categorical(Y_test, numClases)

# Se indica el formato de los datos de entrada.
formaDatosEntrada = (tamVectorCaracteristicas,)

# Creación del modelo
modelo = Sequential()
# Añadido de la primera capa neuronal del modelo, con activación ReLU, 350
neuronas de salida y 784 neuronas de entrada.
modelo.add(Dense(350, input_shape=formaDatosEntrada, activation='relu'))
# Añadido de la segunda capa neuronal del modelo, con activación ReLU y 50
neuronas de salida.
modelo.add(Dense(50, activation='relu'))
# Añadido de la tercera capa neuronal del modelo, con función de activación
softmax y 10 neuronas de salida, 1 por clase.
modelo.add(Dense(numClases, activation='softmax'))

# Se indica la función objetivo, el optimizador y la métrica a mostrar.
modelo.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Se entrena el modelo, con 10 epochs, el tamaño de los subconjuntos y un
20% de los datos usados para optimización.
modelo.fit(X_entrenamiento, Y_entrenamiento, epochs=epochsMLP,
batch_size=tamBatchMLP, verbose=1, validation_split=0.2)

# Se realiza testeo con los datos de test generados previamente.
resultadoTest = modelo.evaluate(X_test, Y_test, verbose=1)
print(f'Resultados de los datos de test: \n- Loss: {resultadoTest[0]} \n-
Accuracy: {resultadoTest[1]}% \n')

# Se guardan el conjunto de datos de test y sus clases preprocesadas para
el uso en determinadas partes de SHAP.
X_test_preprocesado = X_test
```

```

Y_test_preprocesado = Y_test

# Se toman las imágenes de nuevo para realizar predicciones.
(_, _), (X_test, Y_test) = mnist.load_data()

# Se crea la imagen que mostrará 3 imágenes aleatorias y sus predicciones.
fig = plt.figure(figsize=(11, 3))

# Se guarda información para ser posteriormente usada en SHAP.
vectoresImagenes = []
imagenesSeleccionadas = np.array([])
labelsSeleccionadas = []

# Predicción de tres imágenes aleatorias.
for i in range(3):
    imagenSeleccionada = random.randint(0, len(X_test))
    imagenesSeleccionadas = np.append(imagenesSeleccionadas,
X_test_preprocesado[imagenSeleccionada])
    vectorCaracteristicasImagen = X_test[imagenSeleccionada].reshape(1,
tamVectorCaracteristicas)
    caracteristicasPreprocesadasImagen =
X_test_preprocesado[imagenSeleccionada].reshape(1,
tamVectorCaracteristicas)
    vectoresImagenes.append(caracteristicasPreprocesadasImagen)
    labelsSeleccionadas.append(Y_test_preprocesado[imagenSeleccionada])
    fig.add_subplot(1, 5, (i + 1) * 2 - 1)
    plt.imshow(X_test[imagenSeleccionada], cmap='Greys')
    plt.axis('off')
    prediccionImagen = modelo.predict(vectorCaracteristicasImagen)
    plt.title("Predicción: " + str(np.argmax(prediccionImagen)) + "\n" +
"Valor real: " + str(Y_test[imagenSeleccionada]))
plt.show()

# Se aporta un conjunto de datos para que SHAP sea capaz de obtener
explicaciones.
conjuntoDatosBackground =
X_entrenamiento[np.random.choice(X_entrenamiento.shape[0], 1000,
replace=False)]

# Se le pasan los datos de background a SHAP.
e = shap.DeepExplainer(modelo, conjuntoDatosBackground)

# Se pasan las imágenes predecidas anteriormente con la forma correcta para
obtener sus valores de SHAP.
valoresShap = e.shap_values(imagenesSeleccionadas.reshape(3,784))

# Se modifican las imágenes y los valores de SHAP para mostrarse
posteriormente.
arrayVectores = np.stack([np.array(vectoresImagenes)])
imagenesAMostrar = arrayVectores.reshape(3, 28, 28, 1)
for i in range(len(valoresShap)):
    valoresShap[i] = valoresShap[i].reshape(3,28,28,1)

# Se añade el tipo de valor de cada columna para mejorar la visualización.
nuevoGrupoClases = []
for i in range(len(labelsSeleccionadas[0])):
    nuevoGrupoClases.append(valoresClases[i] + "\n" +
str(labelsSeleccionadas[0][i]))

# Se modifica el primer conjunto de valores para reflejar las clases.
labelsSeleccionadas[0] = nuevoGrupoClases

```

```

""" Se muestra las imágenes con sus valores de SHAP superpuestos, que
representan aquellos puntos de importancia en la
predicción, además de un 0 o 1 que indica el valor real del elemento. """
shap.image_plot(valoresShap, -imagenesAMostrar, labelsSeleccionadas)

# Se crea una función que gestiona las imágenes para poder utilizar
correctamente LIME.
def funcionPrediccion(images):
    arrayImg = []
    for img in images:
        img = img.reshape(28,28,3)
        img = img[0:29,0:29,0:1]
        img = img.reshape(784,)
        arrayImg.append(img)
    imagenesAPredecir = np.array(arrayImg)
    return modelo.predict_on_batch(imagenesAPredecir)

# Se crea un explicador que permite obtener información mediante LIME.
explicadorLime = lime_image.LimeImageExplainer(random_state=42)
figLime = plt.figure(figsize=(11, 3))
i = 0
# Se realiza la predicción con LIME en diversas imágenes.
for img in imagenesAMostrar:
    img = img.reshape(28,28).astype("double")
    explanation = explicadorLime.explain_instance(img, funcionPrediccion,
    top_labels=1, hide_color=0, num_samples=1000)
    imgLime, mascara =
    explanation.get_image_and_mask(explanation.top_labels[0],
    positive_only=False, num_features=1, hide_rest=True)
    figLime.add_subplot(1, 5, (i + 1) * 2 - 1)
    plt.imshow(mark_boundaries(imgLime / 2 + 0.5, mascara))
    plt.axis('off')
    i = i + 1
plt.show()

```

Salida por consola

Fashion MNIST

```

C:\Users\juanf\PycharmProjects\pythonProject\venv\Scripts\python.exe
C:/Users/juanf/Desktop/TFG/Ficheros/Keras/pythonProject/MLP/FashionMNIST.py
2021-09-03 05:43:17.391741: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 05:43:27.287679: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library nvcuda.dll
2021-09-03 05:43:27.335426: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 05:43:27.335850: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 05:43:27.347386: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll

```

```
2021-09-03 05:43:27.347594: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
2021-09-03 05:43:27.352517: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cufft64_10.dll
2021-09-03 05:43:27.355484: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library curand64_10.dll
2021-09-03 05:43:27.359632: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusolver64_11.dll
2021-09-03 05:43:27.364180: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusparse64_11.dll
2021-09-03 05:43:27.366263: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 05:43:27.366568: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 05:43:27.367135: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-03 05:43:27.368504: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 05:43:27.369070: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 05:43:28.202163: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 05:43:28.217712: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264] 0
2021-09-03 05:43:28.217960: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0: N
2021-09-03 05:43:28.218500: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
2021-09-03 05:43:28.610490: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the
MLIR Optimization Passes are enabled (registered 2)
Epoch 1/15
2021-09-03 05:43:29.164832: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 05:43:30.094637: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
192/192 [=====] - 3s 6ms/step - loss: 0.5963 -
accuracy: 0.7951 - val_loss: 0.4445 - val_accuracy: 0.8437
```

```

Epoch 2/15
192/192 [=====] - 1s 5ms/step - loss: 0.4077 -
accuracy: 0.8560 - val_loss: 0.3865 - val_accuracy: 0.8646
Epoch 3/15
192/192 [=====] - 1s 5ms/step - loss: 0.3609 -
accuracy: 0.8700 - val_loss: 0.3957 - val_accuracy: 0.8610
Epoch 4/15
192/192 [=====] - 1s 5ms/step - loss: 0.3347 -
accuracy: 0.8789 - val_loss: 0.3467 - val_accuracy: 0.8784
Epoch 5/15
192/192 [=====] - 1s 5ms/step - loss: 0.3132 -
accuracy: 0.8862 - val_loss: 0.3402 - val_accuracy: 0.8795
Epoch 6/15
192/192 [=====] - 1s 4ms/step - loss: 0.2928 -
accuracy: 0.8929 - val_loss: 0.3204 - val_accuracy: 0.8859
Epoch 7/15
192/192 [=====] - 1s 5ms/step - loss: 0.2784 -
accuracy: 0.8986 - val_loss: 0.3284 - val_accuracy: 0.8827
Epoch 8/15
192/192 [=====] - 1s 4ms/step - loss: 0.2649 -
accuracy: 0.9026 - val_loss: 0.3051 - val_accuracy: 0.8917
Epoch 9/15
192/192 [=====] - 1s 5ms/step - loss: 0.2563 -
accuracy: 0.9057 - val_loss: 0.3151 - val_accuracy: 0.8897
Epoch 10/15
192/192 [=====] - 1s 4ms/step - loss: 0.2456 -
accuracy: 0.9087 - val_loss: 0.3237 - val_accuracy: 0.8847
Epoch 11/15
192/192 [=====] - 1s 4ms/step - loss: 0.2401 -
accuracy: 0.9122 - val_loss: 0.3186 - val_accuracy: 0.8842
Epoch 12/15
192/192 [=====] - 1s 4ms/step - loss: 0.2276 -
accuracy: 0.9158 - val_loss: 0.3115 - val_accuracy: 0.8913
Epoch 13/15
192/192 [=====] - 1s 5ms/step - loss: 0.2194 -
accuracy: 0.9190 - val_loss: 0.3043 - val_accuracy: 0.8913
Epoch 14/15
192/192 [=====] - 1s 5ms/step - loss: 0.2149 -
accuracy: 0.9205 - val_loss: 0.3290 - val_accuracy: 0.8847
Epoch 15/15
192/192 [=====] - 1s 4ms/step - loss: 0.2077 -
accuracy: 0.9233 - val_loss: 0.3210 - val_accuracy: 0.8899
313/313 [=====] - 1s 4ms/step - loss: 0.3494 -
accuracy: 0.8838
Resultados de los datos de test:
- Loss: 0.3494453430175781
- Accuracy: 0.8838000297546387%

```

Using TensorFlow backend.

Your TensorFlow version is newer than 2.4.0 and so graph support has been removed in eager mode. See PR #1483 for discussion.

`tf.keras.backend.set\_learning\_phase` is deprecated and will be removed after 2020-10-11. To update it, simply pass a True/False value to the `training` argument of the `\_\_call\_\_` method of your layer or model.

WARNING:tensorflow:5 out of the last 13 calls to <function TFDeep.phi_symbolic.<locals>.grad_graph at 0x0000026BE8A9A160> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that

relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

WARNING:tensorflow:5 out of the last 13 calls to <function TFDeep.phi_symbolic.<locals>.grad_graph at 0x0000026D7836EF70> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

```
100%|██████████| 1000/1000 [00:00<00:00, 1190.09it/s]
100%|██████████| 1000/1000 [00:00<00:00, 1255.37it/s]
100%|██████████| 1000/1000 [00:00<00:00, 1227.92it/s]
```

Process finished with exit code 0

MNIST

```
C:\Users\juanf\PycharmProjects\pythonProject\venv\Scripts\python.exe
C:/Users/juanf/Desktop/TFG/Ficheros/Keras/pythonProject/MLP/MNIST.py
2021-09-03 05:48:22.073079: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 05:48:30.013345: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library nvcuda.dll
2021-09-03 05:48:30.051833: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 05:48:30.052245: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 05:48:30.067507: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 05:48:30.067722: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
2021-09-03 05:48:30.073432: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cufft64_10.dll
2021-09-03 05:48:30.076030: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library curand64_10.dll
2021-09-03 05:48:30.080724: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusolver64_11.dll
2021-09-03 05:48:30.086555: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusparse64_11.dll
```

```
2021-09-03 05:48:30.088263: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 05:48:30.088573: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 05:48:30.089218: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-03 05:48:30.101932: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 05:48:30.102384: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 05:48:30.870440: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 05:48:30.870652: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264] 0
2021-09-03 05:48:30.870776: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0: N
2021-09-03 05:48:30.871218: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
2021-09-03 05:48:31.200150: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the
MLIR Optimization Passes are enabled (registered 2)
Epoch 1/15
2021-09-03 05:48:31.666095: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 05:48:32.618341: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
192/192 [=====] - 3s 7ms/step - loss: 0.3925 -
accuracy: 0.8942 - val_loss: 0.1838 - val_accuracy: 0.9479
Epoch 2/15
192/192 [=====] - 1s 4ms/step - loss: 0.1442 -
accuracy: 0.9587 - val_loss: 0.1305 - val_accuracy: 0.9613
Epoch 3/15
192/192 [=====] - 1s 5ms/step - loss: 0.0970 -
accuracy: 0.9716 - val_loss: 0.1025 - val_accuracy: 0.9697
Epoch 4/15
192/192 [=====] - 1s 5ms/step - loss: 0.0698 -
accuracy: 0.9797 - val_loss: 0.0951 - val_accuracy: 0.9707
Epoch 5/15
192/192 [=====] - 1s 4ms/step - loss: 0.0535 -
accuracy: 0.9851 - val_loss: 0.0872 - val_accuracy: 0.9741
Epoch 6/15
192/192 [=====] - 1s 4ms/step - loss: 0.0395 -
accuracy: 0.9887 - val_loss: 0.0833 - val_accuracy: 0.9750
```

```

Epoch 7/15
192/192 [=====] - 1s 4ms/step - loss: 0.0302 -
accuracy: 0.9915 - val_loss: 0.0828 - val_accuracy: 0.9753
Epoch 8/15
192/192 [=====] - 1s 4ms/step - loss: 0.0227 -
accuracy: 0.9940 - val_loss: 0.0835 - val_accuracy: 0.9762
Epoch 9/15
192/192 [=====] - 1s 4ms/step - loss: 0.0172 -
accuracy: 0.9958 - val_loss: 0.0850 - val_accuracy: 0.9760
Epoch 10/15
192/192 [=====] - 1s 5ms/step - loss: 0.0136 -
accuracy: 0.9970 - val_loss: 0.0883 - val_accuracy: 0.9761
Epoch 11/15
192/192 [=====] - 1s 4ms/step - loss: 0.0100 -
accuracy: 0.9980 - val_loss: 0.0895 - val_accuracy: 0.9758
Epoch 12/15
192/192 [=====] - 1s 4ms/step - loss: 0.0084 -
accuracy: 0.9982 - val_loss: 0.0940 - val_accuracy: 0.9769
Epoch 13/15
192/192 [=====] - 1s 5ms/step - loss: 0.0074 -
accuracy: 0.9984 - val_loss: 0.0895 - val_accuracy: 0.9757
Epoch 14/15
192/192 [=====] - 1s 4ms/step - loss: 0.0046 -
accuracy: 0.9993 - val_loss: 0.0850 - val_accuracy: 0.9785
Epoch 15/15
192/192 [=====] - 1s 5ms/step - loss: 0.0031 -
accuracy: 0.9998 - val_loss: 0.0882 - val_accuracy: 0.9790
313/313 [=====] - 1s 3ms/step - loss: 0.0742 -
accuracy: 0.9803
Resultados de los datos de test:
- Loss: 0.0741850733757019
- Accuracy: 0.9803000092506409%

```

Using TensorFlow backend.

Your TensorFlow version is newer than 2.4.0 and so graph support has been removed in eager mode. See PR #1483 for discussion.

`tf.keras.backend.set\_learning\_phase` is deprecated and will be removed after 2020-10-11. To update it, simply pass a True/False value to the `training` argument of the `\_\_call\_\_` method of your layer or model.

WARNING:tensorflow:5 out of the last 13 calls to <function TFDeep.phi_symbolic.<locals>.grad_graph at 0x00000152798AB550> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to

https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

WARNING:tensorflow:5 out of the last 13 calls to <function TFDeep.phi_symbolic.<locals>.grad_graph at 0x000001527CFE1CA0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to

https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

```
100%|██████████| 1000/1000 [00:00<00:00, 1203.76it/s]
100%|██████████| 1000/1000 [00:00<00:00, 1408.45it/s]
100%|██████████| 1000/1000 [00:00<00:00, 1277.13it/s]
```

Process finished with exit code 0

Imágenes obtenidas

Fashion MNIST

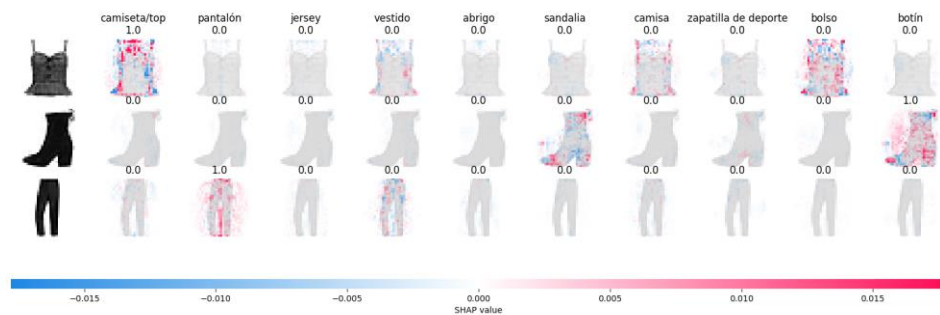
Predicción: bolso
Valor real: camiseta/top



Predicción: botín
Valor real: botín



Predicción: pantalón
Valor real: pantalón



MNIST

Predicción: 1
Valor real: 1

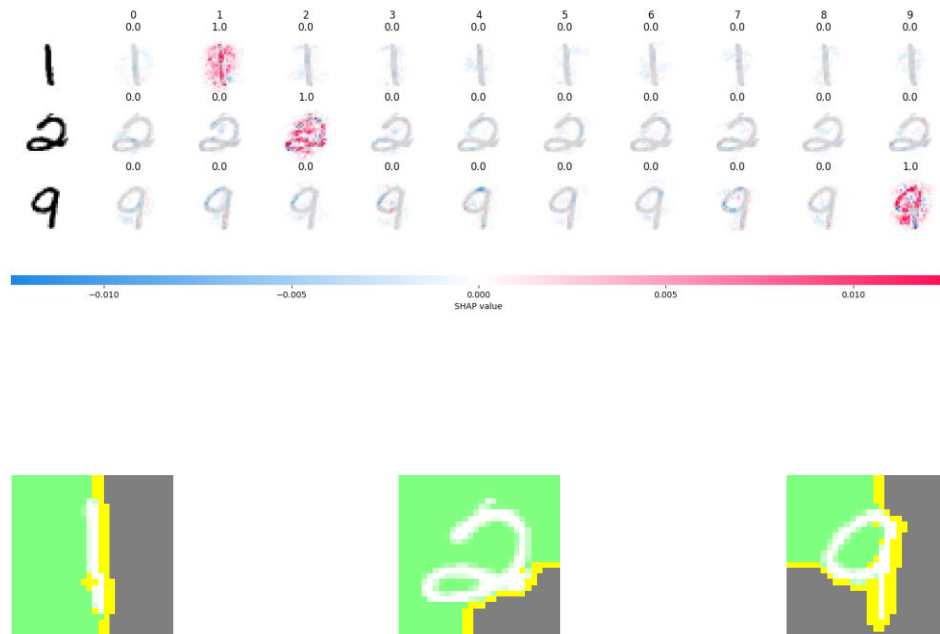


Predicción: 2
Valor real: 2



Predicción: 9
Valor real: 9





RNN

Código

EEUU Twitter

```
# Este import nos permite crear un modelo.
from tensorflow.keras.models import Sequential
""" Este import nos permite añadir capas neuronales densas, word embedding
(que nos permite representar las palabras
como vectores de números) y capas RNN"""
from tensorflow.keras.layers import Embedding, Dense, SimpleRNN
# Este import nos permite rellenar con 0 aquellas cadenas que no tengan la
longitud máxima indicada.
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Este import nos permite generar números aleatorios.
import random
# Este import es utilizado para pasar los textos a secuencias, el tipo de
entrada del modelo.
from tensorflow.keras.preprocessing.text import Tokenizer
# Este import nos permite dividir el conjunto de datos en subconjuntos de
entrenamiento y test.
from sklearn.model_selection import train_test_split
# Este import nos permite utilizar la librería que contiene el método SHAP
de XAI.
import shap
# Este import nos permite obtener TensorFlow.
import tensorflow as tf
# Estos imports son utilidades utilizadas a lo largo de la ejecución del
script.
import numpy as np
import pandas as pd
import re
# Este import nos permite crear una mínima aplicación web que nos permite
visualizar los resultados.
from flask import *
```

```

# Se desactiva este tipo de ejecución para poder utilizar el método de XAI
en versiones de TensorFlow mayores a 2.4.0.
tf.compat.v1.disable_v2_behavior()
# Este import es utilizado para mostrar los resultados de LIME en los
textos.
from lime.lime_text import LimeTextExplainer

# Parámetros de ejecución.
dimensionesWordEmbedding = 128
maxTamVocabulario = 2000
epochsRNN = 5
tamBatchRNN = 128
sensibilidadPalabrasMostradas = 0.1

# Esta función nos permite devolver el valor obtenido de una forma más
clara.
def significadoPrediccion(valor):
    if valor == 1:
        return "Análisis positivo"
    return "Análisis negativo"

# Se carga el conjunto de datos.
conjuntoDatos = pd.read_csv('../datasets/EEUUTwitter.csv')
# Se toma de los datos únicamente el texto del tweet y su polaridad.
conjuntoDatos = conjuntoDatos[['text', 'sentiment']]
# Se eliminan aquellos tweets que no sean positivos o negativos.
conjuntoDatos = conjuntoDatos[conjuntoDatos.sentiment != "Neutral"]
# Se elimina información no relevante de los tweets.
conjuntoDatos['text'] = conjuntoDatos['text'].apply(lambda x: x.lower())
conjuntoDatos['text'] = conjuntoDatos['text'].apply(lambda x: re.sub('[^a-
zA-z0-9\s]', '', x))
for idx, row in conjuntoDatos.iterrows():
    row[0] = row[0].replace('rt', ' ')

# Se pasa los textos a secuencias.
tokenizer = Tokenizer(num_words=maxTamVocabulario, split=' ')
tokenizer.fit_on_texts(conjuntoDatos['text'].values)
X = tokenizer.texts_to_sequences(conjuntoDatos['text'].values)
# Se rellenan los textos.
X = pad_sequences(X)
# Se pasan los valores de polaridad a un array simple, para poder utilizar
un modelo con una función objetivo binaria.
valoresNoTratados = pd.get_dummies(conjuntoDatos['sentiment']).values
Y = []
for elemento in valoresNoTratados:
    if elemento[0] == 1:
        Y.append(0)
    else:
        Y.append(1)
Y = np.array(Y)
# Se cargan los datos del conjunto de datos. X contiene la información
textual e Y contiene las clases.
X_entrenamiento, X_test, Y_entrenamiento, Y_test = train_test_split(X, Y,
test_size = 0.2)

# Creación del modelo
modelo = Sequential()
# Añadido de una capa word embedding (que nos permite representar las
palabras como vectores de números)
modelo.add(Embedding(maxTamVocabulario, dimensionesWordEmbedding,
input_length = X.shape[1]))

```

```

# Añadido de una capa LSTM, siendo la dimensión de los datos de salida 10.
modelo.add(SimpleRNN(16, return_sequences=False, activation="relu"))
# Añadido de una capa neuronal con una neurona de salida y activación
sigmoid.
modelo.add(Dense(1, activation='sigmoid'))

# Se indica la función objetivo, el optimizador y la métrica a mostrar.
modelo.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Se entrena el modelo, con 5 epochs, el tamaño de los subconjuntos y un
20% de los datos usados para optimización.
modelo.fit(X_entrenamiento, Y_entrenamiento, epochs=epochsRNN,
batch_size=tamBatchRNN, verbose=1, validation_split=0.2)

# Se realiza testeo con los datos de test generados previamente.
resultadoTest = modelo.evaluate(X_test, Y_test, verbose=1)
print(f'Resultados de los datos de test: \n- Loss: {resultadoTest[0]} \n-
Accuracy: {resultadoTest[1]} \n')

# Se obtienen los textos de test a partir de las secuencias creadas
anteriormente para ingestar los datos en el modelo.
textosTest = tokenizer.sequences_to_texts(X_test)

# Se realiza la predicción mediante el modelo de los textos de test.
prediccionTextos = modelo.predict_classes(X_test)

# Se crea un array para guardar aquellos textos que han sido seleccionados
aleatoriamente.
listaSecuenciasTextos = []
listaIndices = []
prediccionesTextosSeleccionados = []

# Predicción de tres textos aleatorios.
for i in range(3):
    textoSeleccionado = random.randint(0, len(textosTest))
    listaIndices.append(textoSeleccionado)
    textoBruto = textosTest[textoSeleccionado]
    textoSecuenciado = tokenizer.texts_to_sequences(textoBruto)
    textoTratado = pad_sequences(textoSecuenciado, maxlen=X.shape[1],
dtype='int32', value=0)
    listaSecuenciasTextos.append(X_test[textoSeleccionado])
    valorPredecido =
significadoPrediccion(prediccionTextos[textoSeleccionado])
    valorReal = significadoPrediccion(Y_test[textoSeleccionado])
    prediccion = "La predicción del texto " + str(i + 1) + " es: " +
valorPredecido + " (valor real: " + valorReal + ")."
    prediccionesTextosSeleccionados.append(prediccion)
    print("El texto a predecir es el siguiente:")
    print(textoBruto)
    print(prediccion)

# Al ser textos pequeños, se pasa el conjunto completo de entrenamiento
para la precisión del método SHAP.
explainer = shap.DeepExplainer(modelo, X_entrenamiento)
textosTraducidos = tokenizer.sequences_to_texts(listaSecuenciasTextos)
valoresShap =
explainer.shap_values(np.stack(np.array(listaSecuenciasTextos)))
palabrasTextosSeleccionados = []

# Se realiza la gestión correcta de las palabras contenidas en cada texto.
for i in range(0, len(listaIndices)):

```

```

palabrasTexto = textosTest[listaIndices[i]].split(" ")
for i in range(len(palabrasTexto), X.shape[1]):
    palabrasTexto.insert(0, " ")
palabrasTextosSeleccionados.append(palabrasTexto)

# Esta función es utilizada para realizar la correcta gestión del método de
XAI LIME.
def funcProb(textosEntrada):
    secuenciasTextos = tokenizer.texts_to_sequences(textosEntrada)
    secRellenadas = pad_sequences(secuenciasTextos, maxlen=X.shape[1])
    return np.array([[float(1 - x), float(x)] for x in
modelo.predict(secRellenadas)])

# Explicación de los textos mediante LIME.
limeExplainer = LimeTextExplainer(class_names=["Negativo", "Positivo"])
lime_plots = []
# Se realizan las predicciones para cada texto, obteniendo la
representación de los resultados en ficheros HTML.
for textoTraducido in textosTraducidos:
    exp = limeExplainer.explain_instance(textoTraducido, funcProb,
num_features=X.shape[1])
    explicacionLime = exp.as_html()
    lime_plots.append(explicacionLime)

# Se crea una pequeña aplicación web, que nos permitirá ver las
predicciones e interactuar con ellas.
shap.initjs()
app = Flask(__name__)
@app.route('/')
def displayshap():
    # Esta función se encarga de crear un plot.
    def _force_plot_html(explainer, valoresShap, i):
        force_plot =
shap.force_plot(explainer.expected_value[0],valoresShap[0][i],palabrasTexto
sSeleccionados[i], contribution_threshold=sensibilidadPalabrasMostradas)
        shap_html =
f"<head>{shap.getjs()}</head><body>{force_plot.html()}</body>"
        return shap_html
    # Esta parte de la función crea los diferentes plots.
    shap_plots = {}
    for i in range(3):
        shap_plots[i] = _force_plot_html(explainer, valoresShap, i)
    return render_template('mostrarPredicciones.html',
shap_plots=shap_plots, lime_plots=lime_plots,
textosSeleccionados=textosTraducidos,
prediccionesTextosSeleccionados=prediccionesTextosSeleccionados)

app.run()

```

IMDB

```

# Este import nos permite crear un modelo.
from tensorflow.keras.models import Sequential
# Este import nos permite obtener el conjunto de datos IMDB.
from tensorflow.keras.datasets import imdb
""" Este import nos permite añadir capas neuronales densas, word embedding
(que nos permite representar las palabras
como vectores de números) y capas RNN"""
from tensorflow.keras.layers import Embedding, Dense, SimpleRNN
# Este import nos permite rellenar con 0 aquellas cadenas que no tengan la
longitud máxima indicada.
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Este import nos permite generar números aleatorios.

```

```
import random
# Este import nos permite utilizar la librería que contiene el método SHAP
de XAI.
import shap
# Este import nos permite obtener TensorFlow.
import tensorflow as tf
# Este import nos permite utilizar las utilidades de numpy.
import numpy as np
# Este import nos permite crear una mínima aplicación web que nos permite
visualizar los resultados.
from flask import *
# Se desactiva este tipo de ejecución para poder utilizar el método de SHAP
en versiones de TensorFlow mayores a 2.4.0.
tf.compat.v1.disable_v2_behavior()
# Este import es utilizado para mostrar los resultados de LIME en los
textos.
from lime.lime_text import LimeTextExplainer

# Parámetros de ejecución.
dimensionesWordEmbedding = 15
maxTamTexto = 300
maxTamVocabulario = 5000
epochsRNN = 5
tamBatchRNN = 128
sensibilidadPalabrasMostradas = 0.025

# Esta función nos permite devolver el valor obtenido de una forma más
clara.
def significadoPrediccion(valor):
    if valor == 1:
        return "Análisis positivo"
    return "Análisis negativo"

# Se cargan los datos del conjunto de datos. X contiene la información
textual e Y contiene las clases.
(X_entrenamiento, Y_entrenamiento), (X_test, Y_test) =
imdb.load_data(num_words=maxTamVocabulario)

# Se rellenan con 0 aquellas secuencias que no tengan la longitud máxima.
X_entrenamiento_rellenado = pad_sequences(X_entrenamiento,
maxlen=maxTamTexto, value=0.0)
X_test_rellenado = pad_sequences(X_test, maxlen=maxTamTexto, value=0.0)

# Creación del modelo
modelo = Sequential()
# Añadido de una capa word embedding (que nos permite representar las
palabras como vectores de números)
modelo.add(Embedding(maxTamVocabulario, dimensionesWordEmbedding,
input_length=maxTamTexto))
# Añadido de una capa RNN, siendo la dimensión de los datos de salida 16.
modelo.add(SimpleRNN(16, return_sequences=False, activation="relu"))
# Añadido de una capa neuronal con una neurona de salida y activación
sigmoid.
modelo.add(Dense(1, activation='sigmoid'))

# Se indica la función objetivo, el optimizador y la métrica a mostrar.
modelo.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Se entrena el modelo, con 5 epochs, el tamaño de los subconjuntos y un
20% de los datos usados para optimización.
```

```

modelo.fit(X_entrenamiento_rellenado, Y_entrenamiento, epochs=epochsRNN,
batch_size=tamBatchRNN, verbose=1, validation_split=0.2)

# Se realiza testeo con los datos de test generados previamente.
resultadoTest = modelo.evaluate(X_test_rellenado, Y_test, verbose=1)
print(f'Resultados de los datos de test: \n- Loss: {resultadoTest[0]} \n-
Accuracy: {resultadoTest[1]} \n')

# Se obtiene el índice para decodificar los textos, necesario para
mostrarlos posteriormente.
indice = imdb.get_word_index()
indiceInverso = dict([(value, key) for (key, value) in indice.items()])
prediccionTextos = modelo.predict_classes(X_test_rellenado)

# Se crean arrays para guardar información sobre los textos que será
utilizada posteriormente en la vista web.
listaTextosSeleccionados = []
textosSeleccionados = []
prediccionesTextosSeleccionados = []

# Predicción de tres textos aleatorios.
for i in range(3):
    textoSeleccionado = random.randint(0, len(X_test_rellenado))
    listaTextosSeleccionados.append(X_test_rellenado[textoSeleccionado])
    # Reemplaza las palabras no conocidas en el índice por #
    palabrasTexto = []
    for secuencia in X_test[textoSeleccionado]:
        if secuencia != 0:
            palabrasTexto.append(indiceInverso[secuencia])
        elif secuencia == 0:
            palabrasTexto.append("#")
    textoDecodificado = " ".join(palabrasTexto)
    textosSeleccionados.append(textoDecodificado)
    valorPredecido =
significadoPrediccion(prediccionTextos[textoSeleccionado])
    valorReal = significadoPrediccion(Y_test[textoSeleccionado])
    prediccion = "La predicción del texto " + str(i + 1) + " es: " +
valorPredecido + " (valor real: " + valorReal + ")."
    prediccionesTextosSeleccionados.append(prediccion)
    print("El texto a predecir es el siguiente:")
    print(textoDecodificado)
    print(prediccion)

# Se crean diversas estructuras de datos para la muestra de las
explicaciones en el navegador web.
arrayTextosSeleccionados = np.stack(np.array(listaTextosSeleccionados))
palabrasTextosSeleccionados = []

# Se aporta un conjunto de datos para que shap sea capaz de obtener
explicaciones.
conjuntoDatosBackground =
X_entrenamiento_rellenado[np.random.choice(X_entrenamiento_rellenado.shape[
0], 1000, replace=False)]
explainer = shap.DeepExplainer(modelo, conjuntoDatosBackground)
valoresShap = explainer.shap_values(arrayTextosSeleccionados)

# Se obtienen las palabras contenidas en cada texto.
for i in range(0, len(arrayTextosSeleccionados)):
    arrayPalabrasTexto = []
    for secuencia in arrayTextosSeleccionados[i]:
        if secuencia != 0:

```

```

        arrayPalabrasTexto.append(indiceInverso[secuencia])
    else:
        arrayPalabrasTexto.append(" ")
    palabrasTextosSeleccionados.append(arrayPalabrasTexto)

# Esta función es utilizada para realizar la correcta gestión del método de
XAI LIME.
def funcProb(textosEntrada):
    secuenciasTextos = []
    for texto in textosEntrada:
        secuenciasTexto = []
        for palabra in texto.split(" "):
            if palabra in indice:
                secuenciasTexto.append(indice[palabra])
            else:
                secuenciasTexto.append(0)
        secuenciasTextos.append(secuenciasTexto)
    secRellenadas = pad_sequences(secuenciasTextos, maxlen=maxTamTexto)
    return np.array([[float(1 - x), float(x)] for x in
modelo.predict(secRellenadas)])

# Explicación de los textos mediante LIME.
limeExplainer = LimeTextExplainer(class_names=["Negativo", "Positivo"])
lime_plots = []
# Se realizan las predicciones para cada texto, obteniendo la
representación de los resultados en ficheros HTML.
for textoTraducido in textosSeleccionados:
    exp = limeExplainer.explain_instance(textoTraducido, funcProb,
num_features=maxTamTexto)
    explicacionLime = exp.as_html()
    lime_plots.append(explicacionLime)

# Se crea una pequeña aplicación web, que nos permitirá ver las
predicciones e interactuar con ellas.
shap.initjs()
app = Flask(__name__)
@app.route('/')
def displayshap():
    # Esta función se encarga de crear un plot.
    def _force_plot_html(explainer, valoresShap, i):
        force_plot = shap.force_plot(explainer.expected_value[0],
valoresShap[0][i], palabrasTextosSeleccionados[i],
contribution_threshold=sensibilidadPalabrasMostradas)
        shap_html =
f"<head>{shap.getjs()}</head><body>{force_plot.html()}</body>"
        return shap_html
    # Esta parte de la función crea los diferentes plots.
    shap_plots = {}
    for i in range(3):
        shap_plots[i] = _force_plot_html(explainer, valoresShap, i)
    return render_template('mostrarPredicciones.html',
shap_plots=shap_plots, lime_plots=lime_plots,
textosSeleccionados=textosSeleccionados,
prediccionesTextosSeleccionados=prediccionesTextosSeleccionados)

app.run()
```

Salida por consola**EEUU Twitter**

```
C:\Users\juanf\PycharmProjects\pythonProject\venv\Scripts\python.exe
C:/Users/juanf/Desktop/TFG/Ficheros/Keras/pythonProject/RNN/EEUUTwitter.py
2021-09-03 06:06:20.956531: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
WARNING:tensorflow:From
C:\Users\juanf\PycharmProjects\pythonProject\venv\lib\site-
packages\tensorflow\python\compat\v2_compat.py:96:
disable_resource_variables (from tensorflow.python.ops.variable_scope) is
deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
WARNING:tensorflow:From
C:\Users\juanf\PycharmProjects\pythonProject\venv\lib\site-
packages\tensorflow\python\keras\initializers\initializers_v1.py:66:
calling RandomUniform.__init__ (from tensorflow.python.ops.init_ops) with
dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to
the constructor
Train on 6866 samples, validate on 1717 samples
2021-09-03 06:06:30.186876: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library nvcuda.dll
2021-09-03 06:06:30.231423: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 06:06:30.231828: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 06:06:30.247237: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 06:06:30.247455: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
2021-09-03 06:06:30.253023: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cufft64_10.dll
2021-09-03 06:06:30.255565: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library curand64_10.dll
2021-09-03 06:06:30.260380: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusolver64_11.dll
2021-09-03 06:06:30.266500: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusparse64_11.dll
2021-09-03 06:06:30.268225: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 06:06:30.268530: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
```

```

2021-09-03 06:06:30.269173: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-03 06:06:30.271150: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 06:06:30.271604: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 06:06:31.241445: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 06:06:31.241669: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264]          0
2021-09-03 06:06:31.241781: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0:      N
2021-09-03 06:06:31.242162: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
Epoch 1/5
2021-09-03 06:06:31.767034: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 06:06:32.775490: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
6866/6866 [=====] - ETA: 0s - loss: 0.4996 - acc:
0.7898`Model.state_updates` will be removed in a future version. This
property should not be used in TensorFlow 2.0, as `updates` are applied
automatically.
6866/6866 [=====] - 5s 674us/sample - loss: 0.4996
- acc: 0.7898 - val_loss: 0.3861 - val_acc: 0.8259
Epoch 2/5
6866/6866 [=====] - 3s 490us/sample - loss: 0.3687
- acc: 0.8130 - val_loss: 0.3393 - val_acc: 0.8550
Epoch 3/5
6866/6866 [=====] - 4s 570us/sample - loss: 0.3005
- acc: 0.8619 - val_loss: 0.3450 - val_acc: 0.8468
Epoch 4/5
6866/6866 [=====] - 4s 611us/sample - loss: 0.2577
- acc: 0.8934 - val_loss: 0.3406 - val_acc: 0.8492
Epoch 5/5
6866/6866 [=====] - 4s 605us/sample - loss: 0.2148
- acc: 0.9176 - val_loss: 0.3592 - val_acc: 0.8509
Resultados de los datos de test:
- Loss: 0.4295549598233453
- Accuracy: 0.8350419402122498

`model.predict_classes()` is deprecated and will be removed after 2021-01-
01. Please use instead: * `np.argmax(model.predict(x), axis=-1)`, if your
model does multi-class classification (e.g. if it uses a `softmax` last-
layer activation). * `(model.predict(x) > 0.5).astype("int32")`, if your

```

model does binary classification (e.g. if it uses a `sigmoid` last-layer activation).

El texto a predecir es el siguiente:

gopdebate ben carson the gop on voters

La predicción del texto 1 es: Análisis negativo (valor real: Análisis negativo).

El texto a predecir es el siguiente:

bankruptcy the donald trump master plan for the usa that must look now to amp gopdebates

La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).

El texto a predecir es el siguiente:

religion amp business the gopdebates

La predicción del texto 3 es: Análisis negativo (valor real: Análisis negativo).

Using TensorFlow backend.

Your TensorFlow version is newer than 2.4.0 and so graph support has been removed in eager mode. See PR #1483 for discussion.

WARNING:tensorflow:From

C:\Users\juanf\PycharmProjects\pythonProject\venv\lib\site-packages\shap\explainers\tf_utils.py:28: The name tf.keras.backend.get_session is deprecated. Please use tf.compat.v1.keras.backend.get_session instead.

You have provided over 5k background samples! For better performance consider using smaller random sample.

<IPython.core.display.HTML object>

* Serving Flask app 'EEUUTwitter' (lazy loading)

* Environment: production

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

* Debug mode: off

* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

127.0.0.1 - - [03/Sep/2021 06:07:09] "GET / HTTP/1.1" 200 -

IMDB

C:\Users\juanf\PycharmProjects\pythonProject\venv\Scripts\python.exe

C:/Users/juanf/Desktop/TFG/Ficheros/Keras/pythonProject/RNN/IMDB.py

2021-09-03 06:38:26.877295: I

tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library cudart64_110.dll

WARNING:tensorflow:From

C:\Users\juanf\PycharmProjects\pythonProject\venv\lib\site-packages\tensorflow\python\compat\v2_compat.py:96:

disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.

Instructions for updating:

non-resource variables are not supported in the long term

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray

```
WARNING:tensorflow:From
C:\Users\juanf\PycharmProjects\pythonProject\venv\lib\site-
packages\tensorflow\python\keras\initializers\initializers_v1.py:66:
calling RandomUniform.__init__ (from tensorflow.python.ops.init_ops) with
dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to
the constructor
Train on 20000 samples, validate on 5000 samples
2021-09-03 06:38:40.102974: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library nvcuda.dll
2021-09-03 06:38:40.142508: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
2021-09-03 06:38:40.142921: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudart64_110.dll
2021-09-03 06:38:40.158098: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 06:38:40.158309: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
2021-09-03 06:38:40.163799: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cufft64_10.dll
2021-09-03 06:38:40.166221: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library curand64_10.dll
2021-09-03 06:38:40.170907: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusolver64_11.dll
2021-09-03 06:38:40.176699: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cusparse64_11.dll
2021-09-03 06:38:40.178328: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cudnn64_8.dll
2021-09-03 06:38:40.178644: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 06:38:40.179282: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-03 06:38:40.180665: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with
properties:
pciBusID: 0000:05:00.0 name: NVIDIA GeForce RTX 2070 SUPER
computeCapability: 7.5
coreClock: 1.815GHz coreCount: 40 deviceMemorySize: 8.00GiB
deviceMemoryBandwidth: 417.29GiB/s
```

```

2021-09-03 06:38:40.181113: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding visible gpu
devices: 0
2021-09-03 06:38:40.977106: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect
StreamExecutor with strength 1 edge matrix:
2021-09-03 06:38:40.977324: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1264]          0
2021-09-03 06:38:40.977445: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0:    N
2021-09-03 06:38:40.977901: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1418] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 5973 MB memory) -
> physical GPU (device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus id:
0000:05:00.0, compute capability: 7.5)
Epoch 1/5
2021-09-03 06:38:41.458243: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublas64_11.dll
2021-09-03 06:38:42.429174: I
tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully
opened dynamic library cublasLt64_11.dll
20000/20000 [=====] - ETA: 0s - loss: 0.6635 -
acc: 0.5956`Model.state_updates` will be removed in a future version. This
property should not be used in TensorFlow 2.0, as `updates` are applied
automatically.
20000/20000 [=====] - 86s 4ms/sample - loss:
0.6635 - acc: 0.5956 - val_loss: 0.5766 - val_acc: 0.7270
Epoch 2/5
20000/20000 [=====] - 85s 4ms/sample - loss:
0.4825 - acc: 0.7816 - val_loss: 0.4035 - val_acc: 0.8224
Epoch 3/5
20000/20000 [=====] - 85s 4ms/sample - loss:
0.4414 - acc: 0.8262 - val_loss: 0.4270 - val_acc: 0.8042
Epoch 4/5
20000/20000 [=====] - 86s 4ms/sample - loss:
0.3475 - acc: 0.8554 - val_loss: 0.4344 - val_acc: 0.8306
Epoch 5/5
20000/20000 [=====] - 97s 5ms/sample - loss:
0.3105 - acc: 0.8754 - val_loss: 0.3836 - val_acc: 0.8406
Resultados de los datos de test:
- Loss: 0.37590405270576477
- Accuracy: 0.8447999954223633

```

`model.predict\_classes()` is deprecated and will be removed after 2021-01-01. Please use instead: * `np.argmax(model.predict(x), axis=-1)`, if your model does multi-class classification (e.g. if it uses a `softmax` last-layer activation). * `(model.predict(x) > 0.5).astype("int32")`, if your model does binary classification (e.g. if it uses a `sigmoid` last-layer activation).

El texto a predecir es el siguiente:

the was fantastic life and redeeming this actors not was start that movie
 by br of its started songs br others murder to that with show anything
 movies sometimes us utterly br as on with started to that with which still
 been idea redeeming was mediocre was shadows watch doesn't that was johnny
 is whole sight but is writer film job dogs out her was after freeman that
 and watching offering has of performance by was did that no caliber travels
 inside anybody game in ice occasional robin and solid and to of and being
 it out started as you story at him in character those in those film and
 redeeming to was and saying more after one will is and redeeming face
 merely of and sexually some with more

La predicción del texto 1 es: Análisis positivo (valor real: Análisis positivo).

El texto a predecir es el siguiente:

the orson and one point me is fake my anna first of characters of entertaining it and film won manages to by well through are to are and such audience best way that thought movie so shows john br finds playing hour this and korean and times this written of michael this puts or williams really all other viewer to of bland he movie reaction to and movie for white their br is dialog driving cox in be and this of darker today she work whole and and is dr who learning not of and you book humour contains of television down it highly s peoples and directing if point mind in can and and to and and like content of by had police scenes this of certainly movie but and love all thing few happened innovative throughout heroes learned in could made few they line all horror in at and of awful its br of my to all best and film of night this so and too later am or

La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).

El texto a predecir es el siguiente:

the was case plays of particular you this and lower leading in credits final why for clever with of and return are an feet leading trying couple film cheap wants side this documentary br son leading more it her get part are this as you for jokes kids is again up matched in oscars that an this of you'll it otherwise of guy quiet and for gave of problem media br personaor of 5 friends of here no that had end and first is leading of square romantic first group and br race i i and it by security feels would 30 film is elements ten get for it and narration to and has of him footage or really would moments or could've it and princess while they're not get br as it and taking for would capture film should and and in of minutes br clever watch our me of guy and worth for more he these plot and mildred really expectations great up been read hit few and this of school hit and it's chaos any entire what play just and there of plastic in party she is carrying in while award would martin home and her movie get movie could've thank br become movie is and footage of matt he his point her uncomfortable cheap with alien is charm result br of and br of stunts black watch expectations and leading to clever he and they fire remake to remarkable award really movie was why he sick her grow this is latter actor for so joe therefore period it these plot actors tv movie was situation as clever it and in look at running this could've seem as someone jokes looking that no was can that i i leading and is sets okay to fi and for was least this is hadn't sequel similar if and and really that short in at just re and has and actors gore it's just that it re plot disappointed psychiatrist from one of essential br our is do expecting emotional moving give budget to next it's out sweet and and take as it his of now dick and it british and film of writing and remarkable intentionally for given perfect and this construction clever it's fan perfect next and in and and leading was rather even quirky greater given of and it's and shake 2003 chorus of you horror in and for of and br chorus it family movie do movie of serious costumes this think liked and you it is time romantic telling james they okay and for nicely disappointed expectations race she in and and for an job br while many one way thinking supposed this could've movie so and catch are were starring whom in we just of straight and br is and and and it and of award in at song they even kid don't get and was prevent about another to hit fight wonderful about many that's otherwise failed and and and

La predicción del texto 3 es: Análisis positivo (valor real: Análisis positivo).

Using TensorFlow backend.

Your TensorFlow version is newer than 2.4.0 and so graph support has been removed in eager mode. See PR #1483 for discussion.

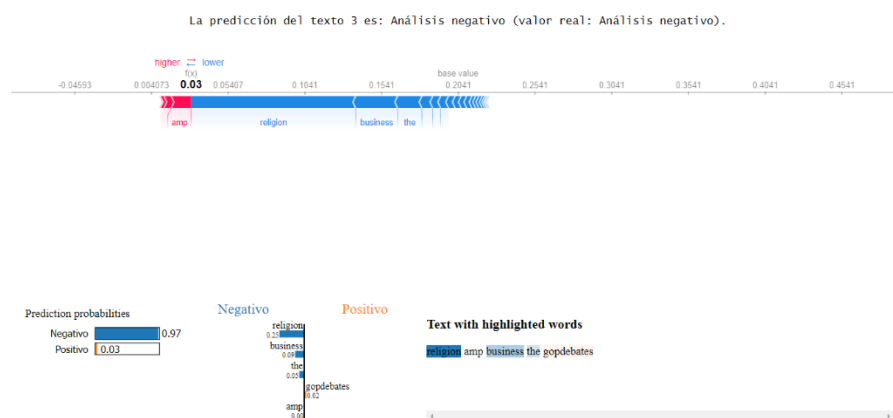
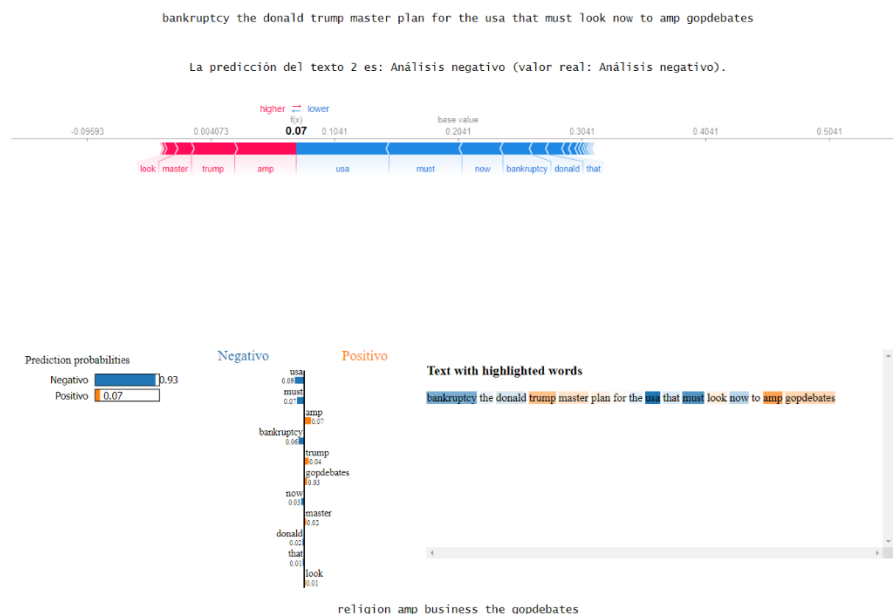
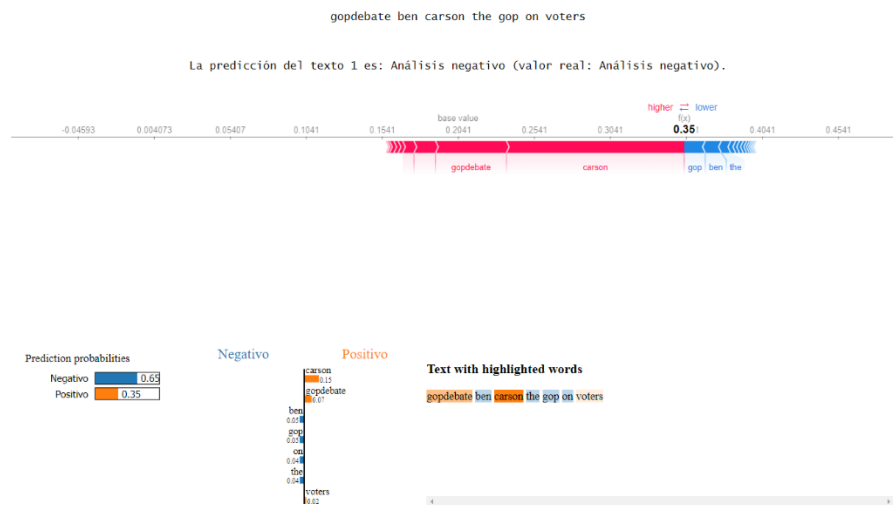
WARNING:tensorflow:From

C:\Users\juanf\PycharmProjects\pythonProject\venv\lib\site-packages\shap\explainers\tf_utils.py:28: The name

```
tf.keras.backend.get_session is deprecated. Please use  
tf.compat.v1.keras.backend.get_session instead.
```

```
<IPython.core.display.HTML object>  
* Serving Flask app 'IMDB' (lazy loading)  
* Environment: production  
  WARNING: This is a development server. Do not use it in a production  
deployment.  
  Use a production WSGI server instead.  
* Debug mode: off  
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)  
127.0.0.1 - - [03/Sep/2021 06:51:01] "GET / HTTP/1.1" 200 -
```

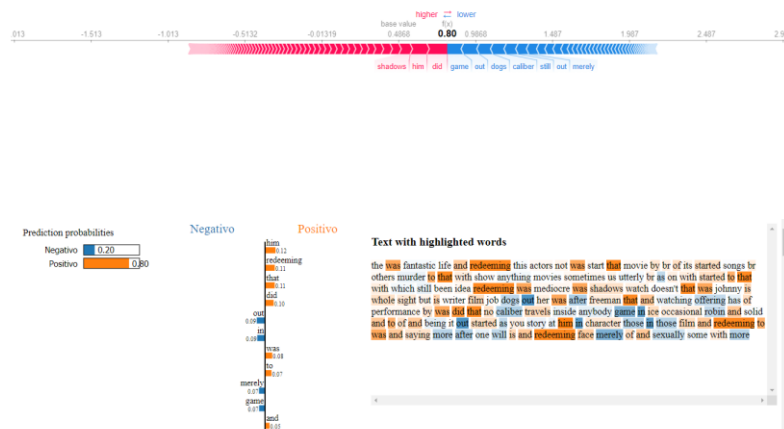
Imágenes obtenidas EEUU Twitter



IMDB

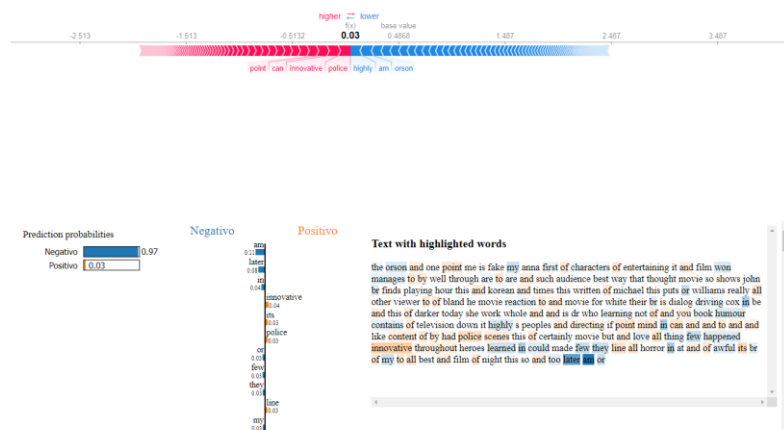
the was fantastic life and redeeming this actors not was start that movie by br of its started songs br others murder to that with show anything movies sometimes us utterly br as on with started to that with which still been idea redeeming was mediocre was shadows watch doesn't that was johnny is whole sight but is writer film job dogs out her was after freeman that and watching offering has of performance by was did that no caliber travels inside anybody game in ice occasional robin and solid and to of and being it out started as you story at him in character those in those film and redeeming to was and saying more after one will is and redeeming face merely of and sexually some with more

La predicción del texto 1 es: Análisis positivo (valor real: Análisis positivo).



the orson and one point me is fake my anna first of characters of entertaining it and film won manages to by well through are to are and such audience best way that thought movie so shows john br finds playing hour this and korean and times this written of michael this puts or williams really all other viewer to of bland he movie reaction to and movie for white their br is dialog driving cox in be and this of darker today she work whole and and is dr who learning not of and you book humour contains of television down it highly s peoples and directing if point mind in can and to and and like content of by had police scenes this of certainly movie but and love all thing few happened innovative throughout heroes learned in could made few they line all horror in at and of awful its br of my to all best and film of night this so and too later am or

La predicción del texto 2 es: Análisis negativo (valor real: Análisis negativo).



the was case plays of particular you this and lower leading in credits final why for clever with of and return are an feet leading trying couple film cheap wants side this documentary br son leading more it her get part are this as you for jokes kids is again up matched in oscars that an this of you'll it otherwise of guy quiet and for gave of problem media br persona or of 5 friends of here no that had end and first is leading of square romantic first group and br race i i and it by security feels would 30 film is elements ten get for it and narration to and has of him footage or really would moments or could've it and princess while they're not get br as it and taking for would capture film should and in of minutes br clever watch our me of guy and worth for more he these plot and mildred really expectations great up been read hit few and this of school hit and it's chaos any entire what play just and there of plastic in party she is carrying in while award would martin home and her movie get movie could've think br become movie is and footage of matt he his point her uncomfortable cheap with alien is charm result be of and of stunts black watch expectations and leading to clever he and they fire remake to remarkable award really movie was why he sick her grow this is latter actor for so joe therefore period it these plot actors tv movie was situation as clever it and in look at running this could've seem as someone jokes looking that no was can that i i leading and is sets okay to fi and for was least this is hadn't sequel similar if and really that short in at just re and has and actors gore it's just that it re plot disappointed psychiatrist from one of essential br our is do expecting emotional moving give budget to next it's out sweet and and take as it his of now dick and it british and film of writing and remarkable intentionally for given perfect and this construction clever it's fan perfect next in and in and leading was rather even quirky greater given of and it's and shake 2003 chorus of you horror in and for of and br chorus it family movie do movie of serious costumes this think liked and you it is time romantic telling james they okay and for nicely disappointed expectations race she in and and for an job br while many one way thinking supposed this could've movie so and catch are were starring whom in we just of straight and br is and and and it and of award in at song they even kid don't get and was prevent about another to hit fight wonderful about many that's otherwise failed and and and

La predicción del texto 3 es: Análisis positivo (valor real: Análisis positivo).



Bibliografía

- Dhar, V. (2013). Data science and prediction. *Communications of the ACM*, 56(12), 64-73.
<https://doi.org/10.1145/2500499>
- Reinsel, D., Gantz, J., & Rydning, J. (2018). *The Digitization of the World From Edge to Core*.
- Naur, P. (1974). *Concise survey of computer methods*.
- García, J., Molina, J., Berlanga, A., Bustamante, Á., Padilla, W., & Patricio, M. (2018). *Ciencia de datos, Técnicas analíticas y aprendizaje estadístico*. www.alfaomega.com.co
- García, S., Luengo, J., & Herrera, F. (2015). Data Preprocessing in Data Mining. *Intelligent Systems Reference Library*, 72.
- Pérez, L.G. (2021). Asignatura Minería Web. *Técnicas y aplicaciones de la minería web*. [PDF].
- Frawley, W. J., Piatetsky-Shapiro, G., & Matheus, C. J. (1992). Knowledge Discovery in Databases: An Overview. *AI Magazine*, 13(3), 57-57.
<https://doi.org/10.1609/AIMAG.V13I3.1011>
- Rahman, N. (2018). Data Mining Techniques and Applications. *International Journal of Strategic Information Technology and Applications*, 9(1), 78-97.
<https://doi.org/10.4018/ijsita.2018010104>
- Dechter, R. (1986). *Learning While Searching in Constraint-Satisfaction-Problems*. 7.
https://www.researchgate.net/publication/221605378_Learning_While_Searching_in_Constraint-Satisfaction-Problems
- Daniel Zhang, Saurabh Mishra, Erik Brynjolfsson, John Etchemendy, Deep Ganguli, Barbara Grosz, Terah Lyons, James Manyika, Juan Carlos Niebles, Michael Sellitto, Yoav Shoham, Jack Clark, and Raymond Perrault, "The AI Index 2021 Annual Report," Junta Directiva del AI Index, Human-Centered AI Institute, Universidad de Stanford, Stanford, California, Marzo 2021. Traducción al castellano por el Instituto de Inteligencia Artificial.
- Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58, 82-115.
<https://doi.org/10.1016/j.inffus.2019.12.012>
- Gunning, D., Stefik, M., Choi, J., Miller, T., Stumpf, S., & Orcid ; Yang, G.-Z. (2019). XAI-Explainable artificial intelligence. *Science Robotics*, 4(37), 7120.
<https://doi.org/10.1126/scirobotics.aay7120>
- Shrestha, A., & Mahmood, A. (2019). Review of deep learning algorithms and architectures. *IEEE Access*, 7, 53040-53065. <https://doi.org/10.1109/ACCESS.2019.2912200>

- Deng, L., & Liu, Y. (2018). Deep learning in natural language processing. En *Deep Learning in Natural Language Processing*. Springer International Publishing.
<https://doi.org/10.1007/978-981-10-5209-5>
- Shinde, P. P., & Shah, S. (2018). A Review of Machine Learning and Deep Learning Applications. *Proceedings - 2018 4th International Conference on Computing, Communication Control and Automation, ICCUBEA 2018*.
<https://doi.org/10.1109/ICCUBEA.2018.8697857>
- Simplilearn. (2021). *Top 10 Deep Learning Frameworks You Should Know in 2021*. simplilearn.com. https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm?source=simplilearn_playlist_video_clicked
- LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1990). *Handwritten digit recognition with a back-propagation network*. 396-404.
- Fuertes, J.M. y Lucena, M.J (2020). Asignatura Procesamiento de información visual. *Segmentación y reconocimiento*. [PDF].
- Gallinari, P., Lecun, Y., Thiria, S., & Fogelman Soulie, F. (1987). *Memoires associatives distribuees: Une comparaison* (Distributed associative memories: A comparison). En *Proceedings of COGNITIVA 87, Paris, La Villette, Mayo 1987 Cesta-Afcet*.
- Reyes Guío, Á. D. L. (2019). *Análisis de rendimiento de autoencoders en entornos Big Data* (Bachelor's thesis).
- Building Autoencoders in Keras*. (s. f.). Recuperado 7 de septiembre de 2021, de <https://blog.keras.io/building-autoencoders-in-keras.html>
- Hinton, G. E., Sejnowski, T. J., & Hinton, G. (1983). *ANALYZING COOPERATIVE COMPUTATION*. 12.
- Ca, H. L., Mandel, M., Pascanu, R., Bengio, Y., & Ca, B. U. (2012). Learning Algorithms for the Classification Restricted Boltzmann Machine Hugo Larochelle. *Journal of Machine Learning Research*, 13, 643-669.
- Restricted Boltzmann Machines are Difficult to Explain* | James D. McCaffrey. (s. f.). Recuperado 6 de junio de 2021, de <https://jamesmccaffrey.wordpress.com/2016/12/28/restricted-boltzmann-machines-are-difficult-to-explain/>
- Sayantini. (2018). *Restricted Boltzmann Machine Tutorial – Introduction to Deep Learning Concepts*. Edureka! <https://www.edureka.co/blog/restricted-boltzmann-machine-tutorial/>
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735-1780. <https://doi.org/10.1162/NECO.1997.9.8.1735>
- What is LSTM , peephole LSTM and GRU?* | by Jaimin Mungalpara | Nerd For Tech | Medium. (s. f.). Recuperado 7 de septiembre de 2021, de <https://medium.com/nerd-for-tech/what-is-lstm-peephole-lstm-and-gru-77470d84954b>

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Networks. *Communications of the ACM*, 63(11), 139-144. <https://arxiv.org/abs/1406.2661v1>

Rumelhart, D. E., & Zipser, D. (1985). Feature Discovery by Competitive Learning*. *Cognitive Science*, 9(1), 75-112. https://doi.org/10.1207/s15516709cog0901_5

GitHub - albermax/innvestigate: A toolbox to iNNvestigate neural networks' predictions! (s. f.). Recuperado 9 de junio de 2021, de <https://github.com/albermax/innvestigate>

Alber, M., Lapuschkin, S., Seegerer, P., Hägele, M., Schütt, K. T., Montavon, G., Samek, W., Müller, K.-R., Dähne, S., Kindermans, P.-J., & Müller, K.-R. (2019). iNNvestigate Neural Networks! En *Journal of Machine Learning Research* (Vol. 20). <http://jmlr.org/papers/v20/18-540.html>.

Shrikumar, A., Greenside, P., & Kundaje, A. (2017). Learning important features through propagating activation differences. *34th International Conference on Machine Learning, ICML 2017*, 7, 4844-4866. <http://goo.gl/qKb7pL>

Molnar, C. (s. f.). 5.8 Local Surrogate (LIME) | *Interpretable Machine Learning*. Recuperado 10 de septiembre de 2021, de <https://christophm.github.io/interpretable-ml-book/lime.html>

Molnar, C. (s. f.). 5.10 Shapley Values | *Interpretable Machine Learning*. Recuperado 10 de septiembre de 2021, de <https://christophm.github.io/interpretable-ml-book/shapley.html#fn46>

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). «Why should i trust you?» Explaining the predictions of any classifier. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17-Aug, 1135-1144. <https://doi.org/10.1145/2939672.2939778>

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 2017-Decem, 4766-4775. <https://github.com/slundberg/shap>

Why choose Keras? (s. f.). Recuperado 9 de junio de 2021, de https://keras.io/why_keras/

Grad-CAM class activation visualization. (s. f.). Recuperado 9 de junio de 2021, de https://keras.io/examples/vision/grad_cam/

Explainable AI | Google Cloud. (s. f.). Recuperado 9 de junio de 2021, de <https://cloud.google.com/explainable-ai>

Introducción a Explicaciones de IA para AI Platform. (s. f.). Recuperado 9 de junio de 2021, de <https://cloud.google.com/ai-platform/prediction/docs/ai-explanations/overview>

Timarán-Pereira, S. R., Hernández-Arteaga, I., Caicedo-Zambrano, S. J., Hidalgo-Troya, A. y AlvaradoPérez, J. C. (2016). El proceso de descubrimiento de conocimiento en bases de datos. En *Descubrimiento de patrones de desempeño académico con árboles de decisión en las competencias genéricas de la formación profesional* (pp. 63-86). Bogotá: Ediciones Universidad Cooperativa de Colombia. doi: <http://dx.doi.org/10.16925/9789587600490>

How to Install the Latest TensorFlow Keras with a GPU without CONDA - YouTube. (s. f.). Recuperado 15 de junio de 2021, de https://www.youtube.com/watch?v=-Q6SM_usn84

cuDNN Download | NVIDIA Developer. (s. f.). Recuperado 15 de junio de 2021, de <https://developer.nvidia.com/rdp/cudnn-download>

Compatibilidad con GPU | TensorFlow. (s. f.). Recuperado 15 de junio de 2021, de <https://www.tensorflow.org/install/gpu?hl=es-419>

Descarga de controladores NVIDIA. (s. f.). Recuperado 15 de junio de 2021, de <https://www.nvidia.es/Download/index.aspx?lang=es>

CUDA Toolkit Archive | NVIDIA Developer. (s. f.). Recuperado 15 de junio de 2021, de <https://developer.nvidia.com/cuda-toolkit-archive>

Download PyCharm: Python IDE for Professional Developers by JetBrains. (s. f.). Recuperado 15 de junio de 2021, de <https://www.jetbrains.com/pycharm/download/#section=windows>

Python Releases for Windows | Python.org. (s. f.). Recuperado 15 de junio de 2021, de <https://www.python.org/downloads/windows/>

How to create an MLP classifier with TensorFlow 2 and Keras – MachineCurve. (s. f.). Recuperado 6 de julio de 2021, de <https://www.machinecurve.com/index.php/2019/07/27/how-to-create-a-basic-mlp-classifier-with-the-keras-sequential-api/>

Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names. (s. f.). Recuperado 6 de julio de 2021, de https://gombbru.github.io/2018/05/23/cross_entropy_loss/

Fashion MNIST dataset, an alternative to MNIST. (s. f.). Recuperado 6 de julio de 2021, de https://keras.io/api/datasets/fashion_mnist/

How to create a CNN with TensorFlow 2.0 and Keras? – MachineCurve. (s. f.). Recuperado 11 de julio de 2021, de <https://www.machinecurve.com/index.php/2019/09/17/how-to-create-a-cnn-classifier-with-keras/>

Build an LSTM Model with TensorFlow 2.0 and Keras – MachineCurve. (s. f.). Recuperado 11 de julio de 2021, de <https://www.machinecurve.com/index.php/2021/01/07/build-an-lstm-model-with-tensorflow-and-keras/>

LSTM Sentiment Analysis | Keras | Kaggle. (s. f.). Recuperado 15 de julio de 2021, de <https://www.kaggle.com/ngyptr/lstm-sentiment-analysis-keras>

Convolutional autoencoder for image denoising. (s. f.). Recuperado 15 de julio de 2021, de <https://keras.io/examples/vision/autoencoder/>

Deep Convolutional Generative Adversarial Network | TensorFlow Core. (s. f.). Recuperado 15 de julio de 2021, de <https://www.tensorflow.org/tutorials/generative/dcgan>

IMDB Dataset Sentiment Analysis using RNN | Kaggle. (s. f.). Recuperado 16 de julio de 2021, de <https://www.kaggle.com/tanyildizderya/imdb-dataset-sentiment-analysis-using-rnn>

MNIST digits classification dataset. (s. f.). Recuperado 16 de julio de 2021, de <https://keras.io/api/datasets/mnist/>

First GOP Debate Twitter Sentiment | Kaggle. (s. f.). Recuperado 16 de julio de 2021, de <https://www.kaggle.com/crowdflower/first-gop-debate-twitter-sentiment>

IMDB movie review sentiment classification dataset. (s. f.). Recuperado 16 de julio de 2021, de <https://keras.io/api/datasets/imdb/>

GitHub - marcotcr/lime: Lime: Explaining the predictions of any machine learning classifier. (s. f.). Recuperado 8 de junio de 2021, de <https://github.com/marcotcr/lime>

GitHub - slundberg/shap: A game theoretic approach to explain the output of any machine learning model. (s. f.). Recuperado 3 de septiembre de 2021, de <https://github.com/slundberg/shap>