



Universidad de Jaén

Escuela Politécnica Superior de Jaén

Redes neuronales adversarias en seguridad informática

Autor: Antonio Mudarra Machuca

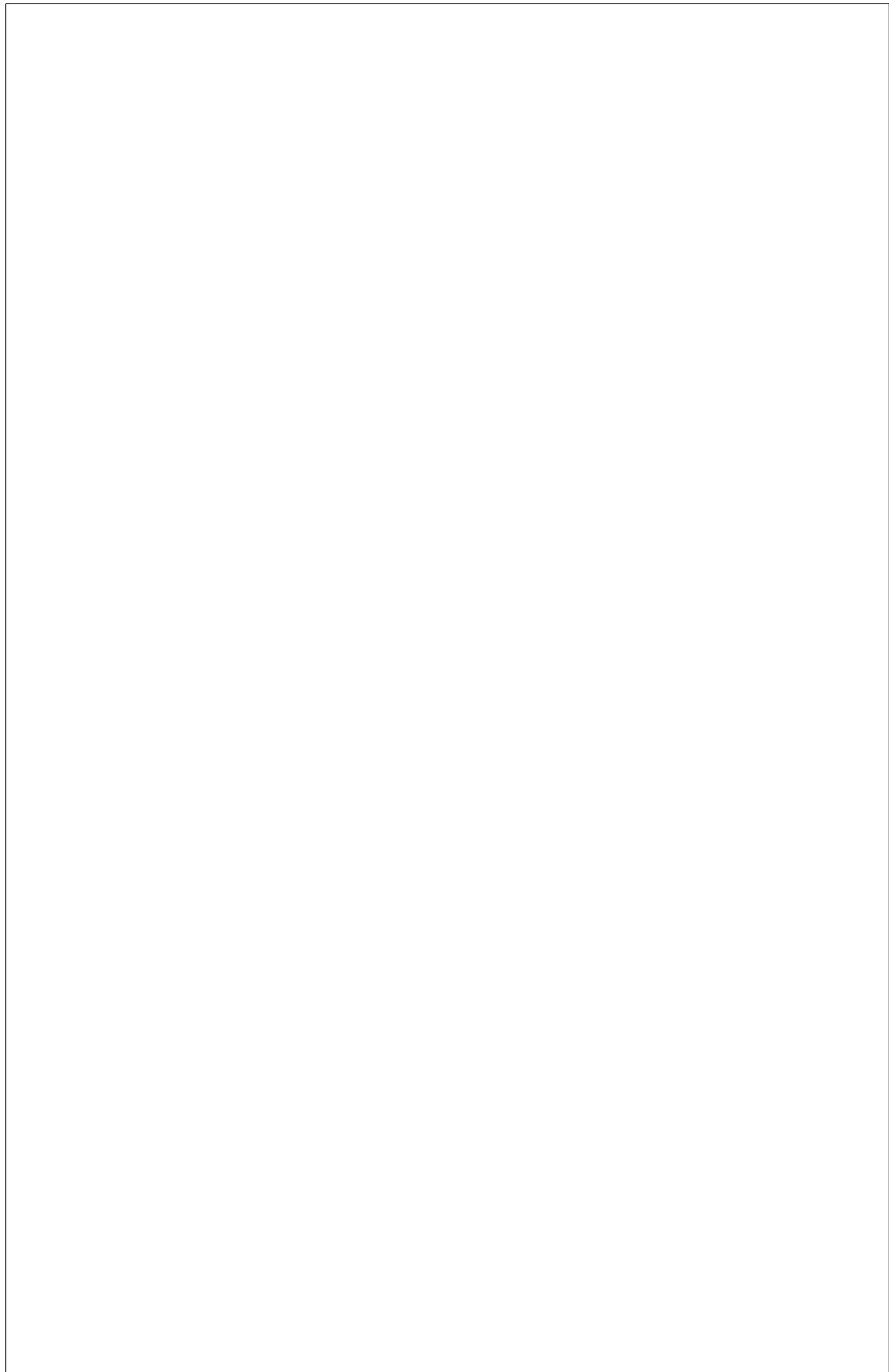
Máster: Seguridad Informática

Directores: Antonio Jesús Rivera Rivas y María José del Jesus Díaz

Departamento del director: Informática

Fecha: diciembre de 2024







UNIVERSIDAD DE JAÉN

D./D^a Antonio Jesús Rivera Rivas y D./D^a María José del Jesus Díaz, tutor(es) del Trabajo Fin de Máster titulado: **Redes neuronales adversarias en seguridad informática**, que presenta Antonio Mudarra Machuca, autoriza(n) su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, diciembre de 2024

El estudiante

Los tutores

Antonio Mudarra Machuca

Antonio Jesús Rivera Rivas

María José del Jesus Díaz

Agradecimientos

Incluir aquí los agradecimientos que se deseen, dedicatoria y cualquier otro texto de carácter personal.

Lista de figuras

1.1. Diagrama Gantt	12
2.1. Ciencia de datos como campo interdisciplinario. Fuente: Elaboración propia.	13
2.2. Retrato de Gottfried Leibniz. Fuente: Wikipedia	16
2.3. Warren Sturgis McCulloch Interview. Fuente: Entrevista en 1969	17
2.4. Los padres de la inteligencia artificial. Fuente: Linkedin	17
2.5. Mark I Perceptron. Fuente: Wikipedia	18
2.6. Kunihiko Fukushima. Fuente: IEICE	18
2.7. Adaptive Systems Research Department at Bell Labs 1989. Fuente: Twitter Yann Lecun	19
2.8. Sepp Hochreiter. Fuente: IDSIA	20
2.9. Ian Goodfellow. Fuente: MIT Technology Review	21
2.10. El proceso de descubrimiento de conocimiento en bases de datos. Fuente: Sección <i>Introduction to Knowledge Discovery and Data Mining</i> [1]	23

2.11. Taxonomía de minería de datos. Fuente: Elaboración propia, inspirado en la sección <i>Introduction to Knowledge Discovery and Data Mining</i> [2]	24
2.12. El <i>Machine Learning</i> como paradigma de programación. Fuente: Elaboración propia.	25
2.13. Clasificación de los algoritmos de aprendizaje automático. Fuente: Elaboración propia.	25
2.14. Partes de una neurona artificial bioinspirada en una neurona biológica. Fuente: Elaboración propia.	27
2.15. Red neuronal artificial. Fuente: Elaboración propia.	28
2.16. Cálculo de los pesos de la red neuronal artificial. Fuente: Elaboración propia.	28
2.17. Problema de clasificación XOR. Fuente: Elaboración propia.	30
2.18. Descenso de gradiente. Fuente: F(x) Data Labs Pvt. Ltd.	31
2.19. Funciones de activación no lineales	34
2.20. Funciones de activación no lineales	35
2.21. Topologías de redes neuronales. Fuente: The neural network zoo	38
2.22. Encoder-Decoder. Fuente: Elaboración propia.	39
2.23. Variable Auto-encoder. Fuente: Elaboración propia, inspirado en el capítulo 2 El auto-encoder variacional [3]	40
2.24. Vector Quantised Variational AutoEncoder (VQ-VAE). Fuente: Neural Discrete Representation Learning [4]	41
2.25. Arquitectura de las redes neuronales generativas adversariales. Fuente: Elaboración propia.	42

2.26. Taxonomía de GANs. Fuente: Elaboración propia, adaptado del artículo “GANs Survey” [5]	45
2.27. Arquitectura Conditional GAN (Conditional GAN (cGAN)) Fuente: Elaboración propia.	46
2.28. Arquitectura Semi-supervised GAN (SGAN) Fuente: Elaboración propia.	47
2.29. Arquitectura Couple GAN (CoGAN) Fuente: Elaboración propia.	48
2.30. Representación CycleGAN	49
2.31. Arquitectura Cycle GAN (CycleGAN) Fuente: Elaboración propia.	51
2.32. Least Square Generative Adversarial Network (LSGAN) Fuente: Least Squares Generative Adversarial Networks [6]	52
2.33. Progressive Generative Adversarial Network (ProGAN) Fuente: ProGAN: Progressive Growing Generative Adversarial Networks	53
2.34. Arquitectura Information Maximizing GAN (InfoGAN). Fuente: Elaboración propia.	56
2.35. Ilustración de la inversión de GAN. Fuente: GAN Inversion: A Survey [7]	57
2.36. Tipos de GAN inversión. Fuente: GAN Inversion: A Survey [7]	58
2.37. Diferencias entre Difusión semántica y Context Encoder	60
2.38. Arquitectura de ART. Fuente: GitHub @Trusted-AI/adversarial-robustness-toolbox	61
2.39. Amenazas en redes neuronales. Fuente: Elaboración propia.	62
2.40. Ataque que realiza CopyCat CNN. Fuente: CopyCat CNN [8]	68
2.41. Ataques y defensas en redes neuronales. Fuente: Elaboración propia.	73

2.42. Amenazas MITRE - ATT & CK, MITRE - ATLAS y STRIDE. Fuente: Elaboración propia. Inspirado en MITRE - ATLAS	75
2.43. Reglamento Europeo Fuente: Elaboración propia, inspirado en [9, 10]	76
4.1. Attack vectors in a generic biometric system Fuente: Exploiting Image Sensor Data in Biometric Systems and Mobile Applications [11]	86
4.2. End to end feature engineering methods Fuente: End-to-end transformational Feature Engineering with CNNs	87
4.3. Clasificación de métodos de extracción y coincidencia de características. Fuente: A survey of feature matching methods [12]	87
4.4. Librerías para el tratamiento de imágenes	88
4.5. Keras	91
4.6. TensorFlow	92
4.7. PyTorch	93
4.8. PyTorch Lightning	95
4.9. Bibliotecas de Python para ciencia de datos	97

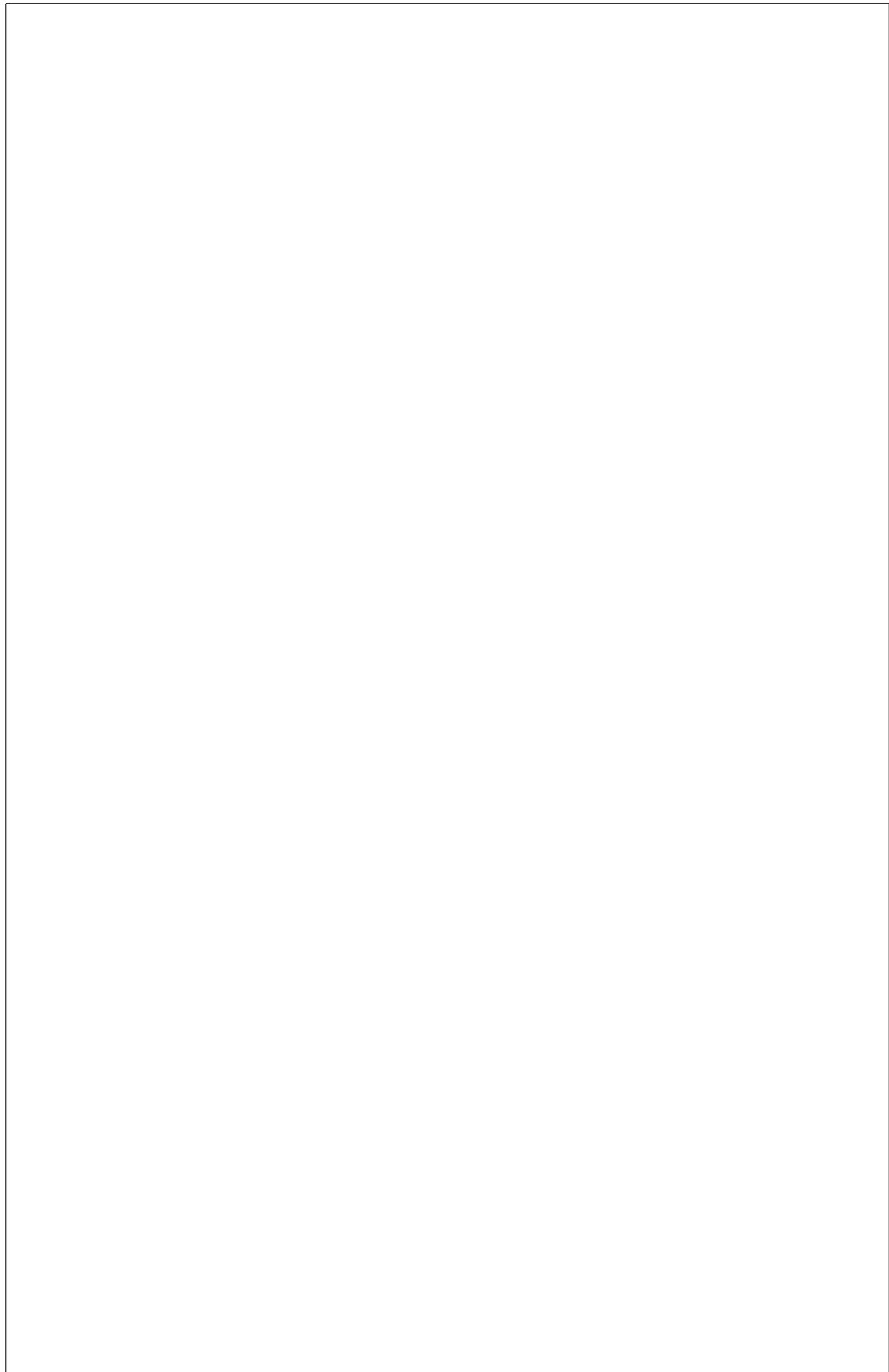
Lista de tablas

1.	Notación números y arrays	1
2.	Notación de índices para vectores, matrices y tensores	1
3.	Notación de vectores, matrices y tensores especiales	2
4.	Notación de operaciones con vectores y matrices	2
5.	Notación de funciones	3
6.	Notación de <i>Machine Learning</i>	4
7.	Notación de <i>GAN</i>	5
8.	Notación probabilidad y estadística	5
2.1.	Hitos de las redes neuronales artificiales	15
2.2.	Lista de ataques de caja blanca	63
2.3.	Lista de ataques de caja negra	64
2.4.	Lista de técnicas de defensa durante el pre procesamiento en Deep Learning	70
2.5.	Lista de técnicas de defensa durante el post procesamiento en Deep Learning	71
2.6.	Lista completa de técnicas de defensa durante el entrenamiento en Deep Learning	71
2.7.	Modelo para identificar amenazas a la seguridad informática	74
2.8.	Clasificación de Amenazas según el Marco MITRE - ATT & CK	74

2.9. Clasificación de Amenazas según el Marco MITRE - ATLAS	74
4.1. Hardware del equipo de desarrollo	85
4.2. Hardware del equipo de entrenamiento	85
4.3. Núcleo de PyTorch	94
4.4. Tecnologías de PyTorch	95
4.5. Librería de PyTorch	95
4.6. Frameworks de deep learning	96

Lista de algoritmos

1. Descenso de gradiente	31
------------------------------------	----



Notación

Notación	Definición
\mathbb{Z}	Números enteros. i.e. $\mathbb{Z} = \{\dots, -2, -1, 0, +1, +2, \dots\}$
\mathbb{Z}^+	Números enteros positivos. i.e. $\mathbb{Z}^+ = \{0, +1, +2, \dots\}$
\mathbb{Z}^{++}	Números enteros positivos sin el cero. i.e. $\mathbb{Z}^{++} = \{+1, +2, \dots\}$
\mathbb{N}	Números naturales
\mathbb{R}	Números reales
\mathbb{C}	Números complejos
R^n	Espacio vectorial n -dimensional de números reales
ϵ	Para cantidades, arbitrariamente pequeñas.

Tabla. 1: Notación números y arrays

Notación	Definición
$\mathbf{a} = [a_i]_{i=1, \dots, n}$	Vectores definidos en minúscula, negrita y cursiva
$\mathbf{A} = [a_{i,j}]_{i=1, \dots, n, j=1, \dots, m}$	Matrices definidas en mayúscula, negrita y cursiva
\mathbf{A}	Tensores definidos en mayúscula, negrita y en estilo sans serif
\mathbf{a}_i	Elemento i del vector \mathbf{a} , empezando el índice por 1
$\mathbf{A}_{i,j}$	Elemento (i, j) de la matriz \mathbf{A}
$\mathbf{A}_{i,:}$	Fila i de la matriz \mathbf{A}
$\mathbf{A}_{:,j}$	Columna j de la matriz \mathbf{A}
$\mathbf{A}_{i,j,k}$	Elemento (i, j, k) 3D del tensor \mathbf{A}
$\mathbf{A}_{:,:,k}$	Desplazamiento 2D del Tensor 3D

Tabla. 2: Notación de índices para vectores, matrices y tensores

Notación	Definición
I_n	Matriz identidad de tamaño $n \times n$
$0_{n,m}$	Matriz de ceros de tamaño $n \times m$
$\underline{0}_n$	Vector de ceros de tamaño n
$1_{n,m}$	Matriz de unos de tamaño $n \times m$
$\underline{1}_n$	Vector de unos de tamaño n
e_i	Vector estándar o vector canónico. i.e. $v_x = (1, 0, 0)$, $v_y = (0, 1, 0)$, $v_z = (0, 0, 1)$

Tabla. 3: Notación de vectores, matrices y tensores especiales

Símbolo	Definición
A^\dagger	Traspuesta de la matriz A
$\text{rk}(A)$	Rango de la matriz A
$\text{tr}(A)$	Traza de la matriz A
$\det(A)$	Determinante de la matriz A
$\text{Im}(\Phi)$	Imagen del mapeo lineal Φ
$\text{ker}(\Phi)$	Núcleo (espacio nulo) de un mapeo lineal Φ
$ \cdot $	Determinante o valor absoluto
$\ x\ _p = \sqrt[p]{\sum_{i=1}^n x_i ^p}$	Norma L^p de x
$\ x\ $	Norma L^2 de x
$\ \underline{x} - \underline{y}\ _q$	Distancia en la misma dimensión entre \underline{x} e \underline{y}
$\underline{x} \odot \underline{m}$	Operación por elementos de los vectores o matrices
$\langle \underline{z}, \underline{y} \rangle = \underline{z}' \underline{y} = \sum_{j=1}^N z_j y_j$	Producto escalar de vectores por columnas $\underline{z}, \underline{y} \in \mathbb{R}^N$

Tabla. 4: Notación de operaciones con vectores y matrices

Notación	Definición
$f : \mathbb{A} \rightarrow \mathbb{B}$ $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$	Una función f con dominio \mathbb{A} y rango \mathbb{B} Gradiente, es un vector que indica la dirección de mayor pendiente de una superficie en un punto dado
$\nabla f(\mathbf{a}) \in \mathbb{R}^n$	Gradiente de la función $f : \mathbb{R}^n \rightarrow \mathbb{R}$ con entrada \mathbf{a}
$\nabla \cdot f = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} + \frac{\partial f}{\partial z}$ $\nabla \times f = \left(\frac{\partial f}{\partial y} - \frac{\partial f}{\partial z}, \frac{\partial f}{\partial z} - \frac{\partial f}{\partial x}, \frac{\partial f}{\partial x} - \frac{\partial f}{\partial y} \right)$	Divergencia Rotación
$f_* = \min_x f(x)$ $x_* \in \arg \min_x f(x)$	El valor de función más pequeño de f El valor x_* (conjunto de valores) que minimiza f

Tabla. 5: Notación de funciones

Notación	Definición
\mathcal{X}	Conjunto de datos, donde $\mathbb{X} \in \mathbb{R}^N$ La dimensión de la matriz de características es $\mathcal{X} = I \times K$
$\mathcal{Y}, y, \text{tag}(x)$	Clases del conjunto de datos \mathbb{X} , esto es, $K = \mathcal{Y} $. i.e. $\mathcal{Y} = \{1, 2, \dots, K\}$
$(x^{(i)}, y^{(i)})$	La tupla elementos i en el conjunto \mathbb{X} (Aprendizaje supervisado)
$x^{(i)}$	Los elementos i en el conjunto \mathbb{X} (Aprendizaje no supervisado)
N	Dimensión del espacio de entrada \mathcal{X}
$\{x_1, x_2, \dots, x_n\}$	Conjunto con n elementos
$T \circ \mathbb{X} = \{x^{(i)}, y^{(i)}\}$	Conjunto de datos de entrenamiento $D \subset \mathbb{X}$
$V \circ \mathbb{X}' = \{x'^{(i)}, y'^{(i)}\}$	Conjunto de datos de validación $H \subset \mathbb{X}$
I	Número de instancias, registros u observaciones
K	Número de atributos, características, entradas o predictores
θ	Parámetros del modelo
η	Tasa de aprendizaje
\hat{y}	Predicción del modelo
\mathcal{L}	Función de perdida
ϕ, φ	Función de activación
$f(\cdot)$	Modelo
$c(\cdot)$	Modelo clasificador
$p(\cdot)$	Modelo predictor

Tabla. 6: Notación de *Machine Learning*

Notación	Definición
z	Ruido
z	Código latente
x'	Dato adversarial
\hat{x}	Instancia generada
y'	Clase objetiva del ejemplo adversario
δ	Perturbación
θ	Parámetros del modelo en la red generadora (G)
ω	Parámetros del modelo en la red discriminadora (D)
I^{LR}	Imagen de baja resolución
I^{HR}	Imagen de alta resolución
ℓ	Funciones Lipschitz
ℓ_2	Funciones Lipschitz
ℓ_∞	Funciones Lipschitz

Tabla. 7: Notación de GAN

Notación	Definición
$Q(x)$	Función de densidad de probabilidad de la variable aleatoria x
$\mathbb{E}_{x \sim Q}$	Esperanza de $f(x)$ con respecto a la distribución de probabilidad de x según Q
$I(X; Y)$	Información mutua, $I(X; Y) = H(X) - H(X Y) = H(Y) - H(Y X)$
$W_p, W_p(F, G)$	Distancia de Wasserstein

Tabla. 8: Notación probabilidad y estadística

Fundamentos matemáticos

Teoremas

Teoremas 0.0.1 (Bayes). *Sean A, B dos eventos, y $P(A|B)$ la probabilidad de A dependiente de B . Entonces.*

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

Teoremas 0.0.2 (Convergencia de perceptrón). *Si el conjunto de patrones de entrenamiento $\{x^1, z^1\}, \{x^2, z^2\}, \dots, \{x^n, z^n\}$ es linealmente separable entonces el Perceptrón simple encuentra una solución en un número finito de iteraciones, es decir, consigue que la salida de la red coincida con la salida deseada para cada uno de los patrones de entrenamiento.*

Teoremas 0.0.3 (Wasserstein). *Sea \mathbb{P}_r una distribución cualquiera y \mathbb{P}_θ la distribución de $g_\theta(Z)$ siendo Z una variable aleatoria con densidad p y g_θ una función que satisface el supuesto.*

Entonces, existe una solución $f: \mathcal{X} \rightarrow \mathbb{R}$ al problema

$$\max_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

y tenemos $\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z \sim p(z)}[\nabla_\theta f(g_\theta(z))]$ cuando ambos términos están bien definidos.

Teoremas 0.0.4 (Lipschitz). *Se dice que una función es continua y uniforme, también llamada función Lipschitz $f: A \rightarrow \mathbb{R}$ cuando existe una constante $M \geq 0$ que verifica:*

$$|f(y) - f(x)| \leq M|y - x| \quad \forall x, y \in A$$

Existe una constante mínima $M_0 \geq 0$ que verifica la desigualdad anterior, M_0 es llamada constante de Lipschitz.

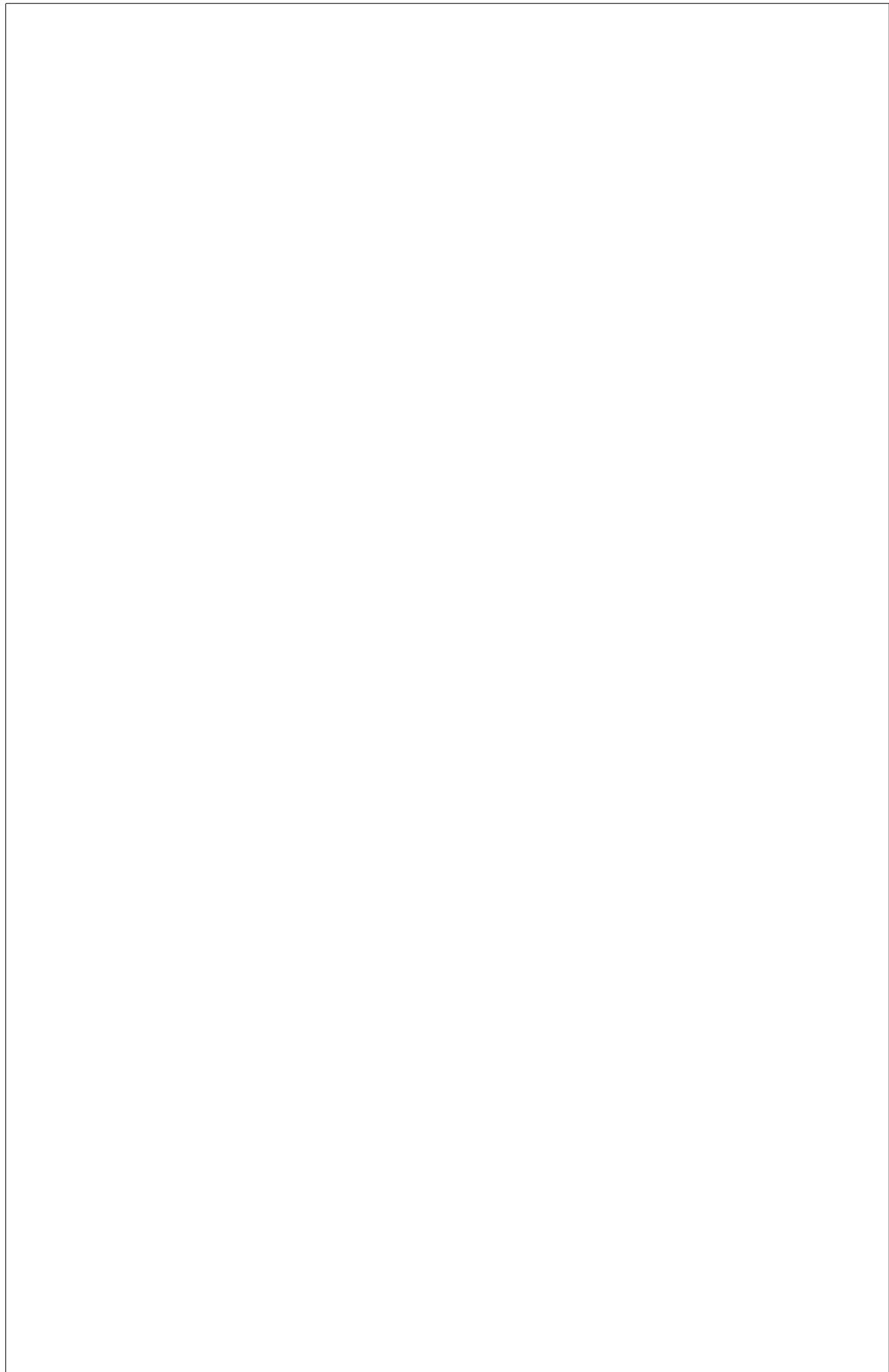
Conceptos previos

Divergencia de Kullback-Leibler

La divergencia de Kullback-Leibler es una medida de la diferencia entre dos valores, en concreto de dos valores de una dos distribuciones de probabilidad. Se le denomina KL o D_{KL} , se puede calcular mediante la ecuación.

$$D_{\text{KL}}(P||Q) = \mathbb{E}_{x \sim P} \left[\log \frac{P(x)}{Q(x)} \right] \quad (1)$$

Esta medida permite conocer el límite inferior de una función que se usa para múltiples funciones de coste como en el auto-encoder variacional.



Capítulo 1

INTRODUCCIÓN

1.1. Introducción

El objetivo principal de este trabajo es el análisis en el campo de la inteligencia artificial () de los distintos tipos de ataques y defensas que se pueden aplicar a un proceso de aprendizaje automático [Machine Learning \(ML\)](#) de las redes neuronales, nos centraremos principalmente en el aprendizaje profundo [Deep Learning \(DL\)](#). Este trabajo tratará principalmente de clasificar y explorar la seguridad que cuentan los modelos actuales. Se han desarrollado una gran cantidad de modelos en los últimos años y existen una gran variedad de modelos muy distintos, aunque podemos clasificar los ataques de todos estos modelos en las siguientes categorías (evadir, envenenar, inferir o extraer).

Debemos comprender que la seguridad repercute en todo el proceso de creación y uso del modelo, desde la recogida de los datos, tratamiento, diseño, creación y ejecución del modelo. Un ataque puede estar dirigido al conjunto de datos con el que se entrenará el modelo, a la arquitectura de la red neuronal, a los pesos, etc. Además, un ataque puede dirigirse a descubrir muestras que produzcan resultados erróneos, por lo que los posibles vectores de ataque son muy variados y complejos.

La idea de hacer robustos los modelos es que las defensas detecten ataques a la vez que se mejora la solidez del aprendizaje, para no cometer fallos al introducir valores anómalos.

Cada uno de estos tipos de ataques tiene su nomenclatura y se debe definirse correctamente, ya que de lo contrario puede ser muy ambiguo el tipo de ataque que se está realizando, el objetivo que busca el ataque y los métodos que están empleando.

Lo que buscaremos en este trabajo será definir, analizar y estructurar los distintos tipos de ataques y sus posibles defensas, con el objetivo final de detectar y defender los modelos para hacerlos más robustos y confiables para la ciudadanía. Además de la creación de una guía que pueda orientar a modelos más seguros y éticos.

1.2. Motivación

En la última década, se han logrado significativos avances en el campo de la inteligencia artificial. Sin embargo, a lo largo de este proceso, como suele ser habitual, se ha descuidado aspectos cruciales relacionados con la seguridad. Esto ha dado lugar a la creación de productos que implementan la inteligencia artificial, pero presentan vulnerabilidades, riesgos potenciales, como redes neuronales poco robustas, filtraciones de datos, incumplimiento normativo, modelos que presentaban respuestas ofensivas, discriminantes ante etnias, etc. Además de presentar poca o ninguna explicabilidad de los resultados que presentan.

Esto lleva a muchas amenazas y riesgos que pueden afectar a empresas u organizaciones que apliquen inteligencia artificial si no hacen una implementación adecuada.

Problemas de seguridad en inteligencia artificial.

- Alineación de la inteligencia artificial.
- Recopilación de datos.
- Bias y discriminación.
- Modelos poco robustos.
- Transparencia y explicabilidad.
- Escala de los modelos.
- Cumplimiento legal y normativo.
- Actualización continua.
- Detección de usos malintencionados.

Esto ha llevado a la creación de regulaciones de la inteligencia artificial que son muy vagas en sus conceptos de implementación. Una primera aproximación fue el libro blanco¹ de la inteligencia artificial en 2018 [13]

En este trabajo propondremos una guía similar a la matriz, [Adversarial Threat](#)

¹Se conoce como libros blancos a los documentos que publican los gobiernos en determinados casos para informar a los órganos legislativos o a la opinión pública con el objetivo de ayudar a los lectores a comprender un tema, resolver o afrontar un problema (por ejemplo diseñando una política gubernamental a largo plazo), o tomar una decisión. [Enlace](#).

[Landscape for Artificial Intelligence Systems \(ATLAS\)](#) con buenas prácticas para la construcción de modelos más robustos y que cumplan con nuestros estándares.

La necesidad de desarrollar un marco de inteligencia artificial fiable para todos los miembros de la Unión Europea llevo a la Comisión Europea a la creación de nuevas normativas con un enfoque basado en el **riesgo**.

Por lo que exploraremos el análisis de seguridad y posibles riesgos que se pueden dar en la sociedad.

1.3. Metodología y planificación del proyecto

Una vez definidos los objetivos del tema de investigación y desarrollo, deberemos definir el alcance para estimar el tiempo requerido y el presupuesto para el proyecto. Recordemos que en la investigación puede ser del tipo exploratorio, buscando nuevos campos para alcanzar nuevos logros técnicos, confirmatoria para validar los resultados obtenidos en otras investigaciones o desarrollos, también puede ser una combinación de los tipos mencionados previamente. En nuestro caso, se tratará de un proyecto híbrido, tanto de exploración como de confirmación.

1.3.1. Metodología

1.3.2. Planificación

Se ha de hacer una planificación estricta de las tareas, objetivos, hitos y dependencias, para ello usaremos los diagramas Gantt² con el objetivo de representar los hitos de esta investigación y del proyecto. Seguiremos una metodología *Scrum* y *Lean* fijando reuniones cada semana para tener un control sobre el avance de la investigación y el desarrollo de este proyecto.

Se ha definido en la Figura 1.1 la lista de tareas, fecha de inicio, duración, fecha de fin y el coste de trabajo usando la técnica de tallas de camisetas.

²Representación gráfica de la evolución de un proyecto. [ASANA](#)

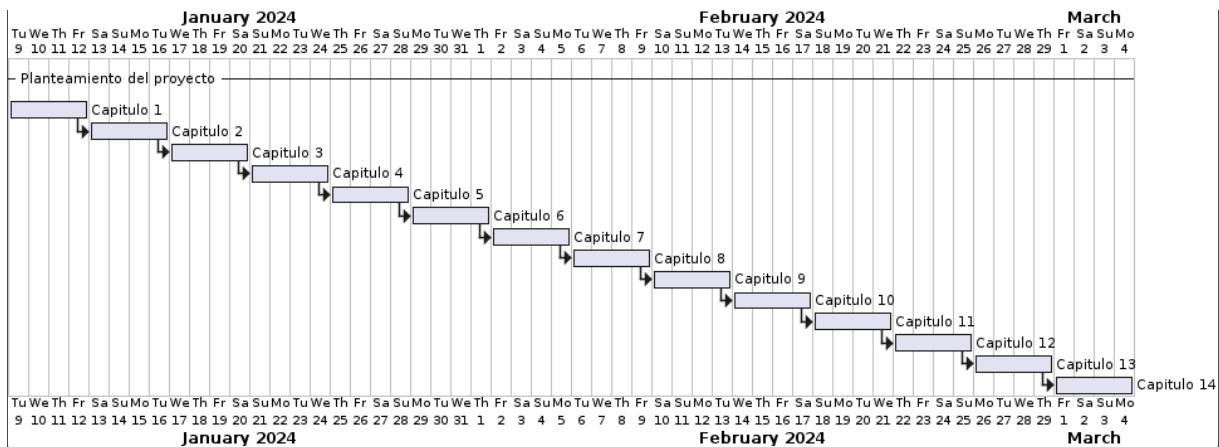


Figura 1.1: Diagrama Gantt

1.3.3. Presupuesto

1.4. Estructura de la memoria

Capítulo 2

ANTECEDENTES Y ESTADO DEL ARTE

2.1. Introducción

El proyecto que desarrollamos se encuentra en una frontera muy difusa de múltiples ramas del conocimiento, siendo muy interdisciplinar, se trata de una revisión de los ataques, seguridad y robustez a las redes neuronales, centrándonos en ataques adversariales. Por lo que debemos explicar que es la ciencia de datos, el proceso [Knowledge Discovery in Databases \(KDD\)](#), la inteligencia artificial generativa y la seguridad informática.

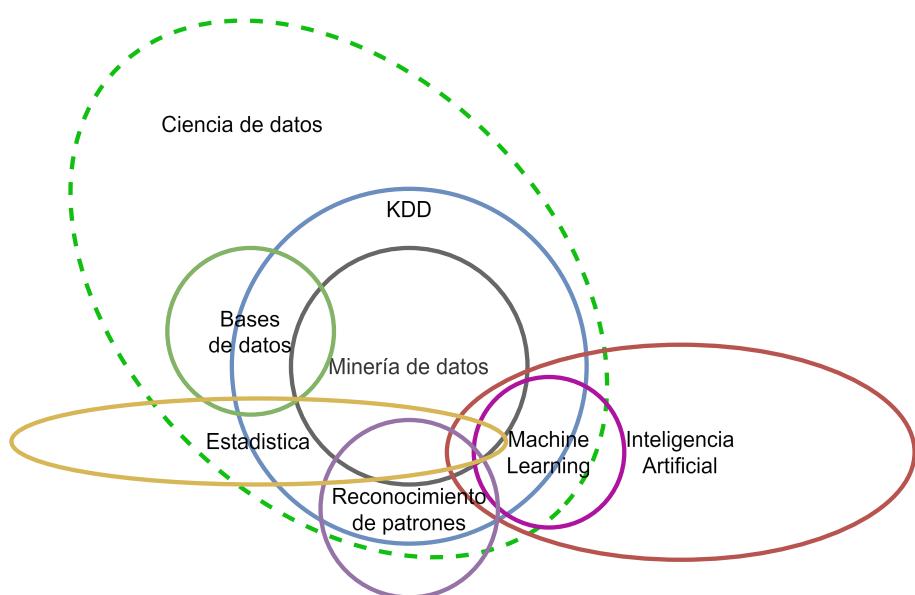


Figura 2.1: Ciencia de datos como campo interdisciplinario.
Fuente: Elaboración propia.

Podemos dividir este trabajo en dos secciones muy relacionadas, la primera la inteligencia artificial y de segundo punto de importancia la seguridad de la información. Desde sus inicios, la inteligencia artificial, aunque con buenos resultados en muchos campos de aplicación, resultaba en grandes fallos de seguridad, fiabilidad y robustez. Por cómo están entrenadas las inteligencias artificiales ([ANN](#)) tiene múltiples puntos de ataque que son susceptibles de ser atacados, los principales son los datos, las arquitecturas o los pesos. Ya que alterando cualquiera de estos componentes de forma se verá enormemente afectada el comportamiento.

2.2. Antecedentes

2.2.1. Historia, línea temporal

A continuación, se describe la línea de temporal con los hitos más relevantes en el desarrollo de la inteligencia artificial, sus avances hasta él [DL](#) y como se ha llegado a las redes neuronales actuales o la inteligencia artificial generativa. Se han adjuntado referencias de las investigaciones, artículos y publicaciones realizadas. [\[14\]](#)

Como podemos observar en la línea temporal de la figura [2.1](#) se realizaron grandes avances en este campo desde principios del siglo XVII, partiendo de las bases matemáticas como la regla de la cadena, pasando por la teoría Hebbiana hasta nuestros días con los modelos de redes neuronales. Haremos un recorrido de los avances más relevantes y su contribución.

Describiremos de forma breve el funcionamiento técnico de los avances más significativos en redes neuronales y visión por computador, para posteriormente explicar las distintas formas de diseñar una red neuronal adversarial generativa.

1676	• The Chain Rule [15] .
1847	• Augustin-Louis Cauchy [16] .
1943	• Threshold Logic Unit (TLU) [17] .
1949	• Teoría Hebbiana .
1958	• Perceptron [18] .
1959-1960	• Adaline y Madaline [18] .
1965	• Multilayer Perceptron (MLP) [19] .
1967-1968	• Deep Learning by Stochastic Gradient Descent [20] .
1980's	• Neuronas Sigmoidales Feedforward neural network (FNN) [21] Backpropagation (BP) [22, 23, 24] .
1985	• Boltzmann Machine [25] .
1987	• Adaptive resonance theory (ART) [26] .
1989	• Convolutional neural networks (CNN) [27] Recurrent neural networks (RNN) [28] .
1990	• Generative Adversarial Networks (GAN) as Game [29] .
1997	• Long short term memory (LSTM) [30, 31] .
2006	• Deep Belief Networks (DBN) [32] Restricted Boltzmann Machine [33] Encoder / Decoder (Auto-encoder) [33] .
2014	• Generative Adversarial Networks (GAN) Moderns [34, 35] .
2018	• Style Generative Adversarial Networks (Style-GAN) [36] .

Tabla. 2.1: Hitos de las redes neuronales artificiales

2.2.2. Historia de la Inteligencia Artificial

Todo surge en 1676 por Gottfried Wilhelm Leibniz [2.2](#) publicó la regla de la cadena del cálculo diferencial, esencial para el análisis matemático, es la esencial para calcular como cambiará la función final si se cambian los pesos de funciones anteriores.

La regla de la cadena es fundamental para técnicas como el descenso de gradiente, propuesto por Augustin-Louis Cauchy en 1847 y utilizado para ajustar iterativamente los pesos de una [NN](#) durante el entrenamiento. Posteriormente en 1805 Adrien-Marie Legendre y Johann Carl Friedrich Gauss desarrollaron [NN](#), matemáticamente eran regresiones lineales muy simples, similares a las redes neuronales lineales simples. Esto lo uso Gauss para redescubrir el planeta enano Ceres.



Figura 2.2: Retrato de Gottfried Leibniz.
Fuente: [Wikipedia](#)

Aunque realmente la historia comienza en 1943 con la investigación de Warren McCulloch y Walter Pitts, publicaron el artículo *A logical calculus of the ideas immanent in nervous activity* [\[17\]](#). Dicho artículo creó distintas ramas de investigación (ordenadores digitales, inteligencia artificial, funcionamiento del perceptrón).



Figura 2.3: Warren Sturgis McCulloch Interview.

Fuente: [Entrevista en 1969](#)

En 1956 en la primera conferencia de inteligencia artificial organizada por la fundación Rochester, se reúnen los investigadores fundadores de los conceptos actuales de la IA (Minsky, McCarthy, Rochester, Shannon), gran parte de la bibliografía se refiere a este punto como el origen y contacto de las redes neuronales artificiales. En dicha conferencia (*Nathaural Rochester*) presento el modelo de una red neuronal que fue el resultado de la investigación desarrollada por el equipo de investigación de IBM.

1956 Dartmouth Conference: The Founding Fathers of AI

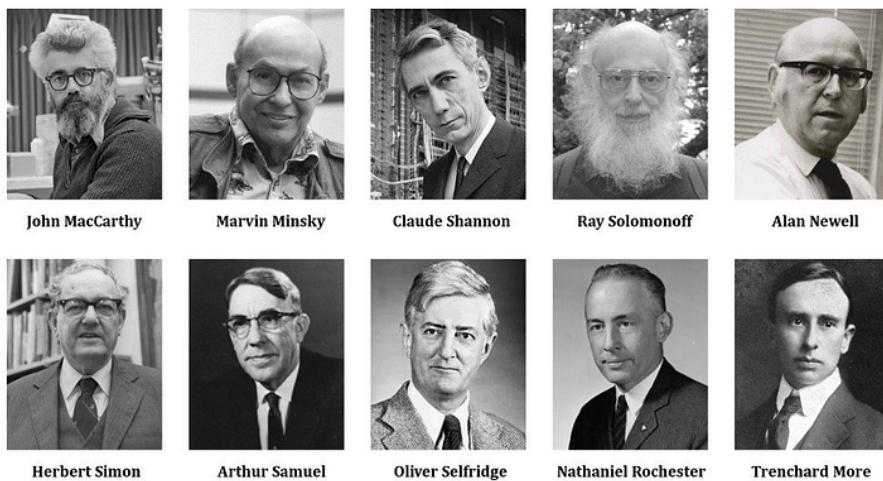


Figura 2.4: Los padres de la inteligencia artificial.

Fuente: [LinkedIn](#)

En 1957 se presenta el “*Perceptron*” por Frank Rosenblatt, dicho elemento es un sistema clasificador de patrones, además contaba con la capacidad de aprender, de ser robusto matemáticamente y poder adaptarse si algún componente se dañaba.

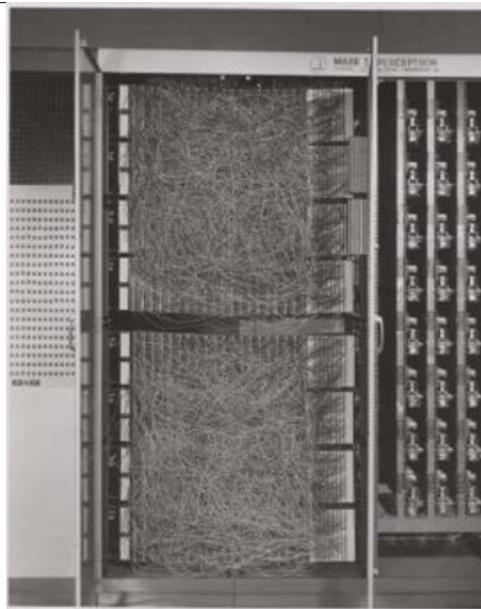


Figura 2.5: Mark I Perceptron.

Fuente: [Wikipedia](#)

El *Perceptron* fue diseñado originalmente para el reconocimiento óptico usando un sistema de 400 fotocélulas en rejilla. Posteriormente, se describió el problema de no-linealidad que presentaban los perceptrones (Problema XOR) [37].

De 1959 a 1960 Bernard Widrow y Ted Hoff desarrollaron “Adaline” y “Madaline” [38] que resolvía el problema de la no-linealidad y que tenía aplicación en el reconocimiento de voz, series temporales, caracteres, etc.

Posteriormente, el *MIT* realizó una investigación matemática muy crítica de todos los problemas que presentaba el *Perceptron* llegando a la conclusión que tenían grandes problemas que no podrían ser resueltos, por lo que en la próxima década (años 60) se redujo drásticamente las investigaciones sobre el campo de las redes neuronales. Esto llevó a uno de los famosos inviernos de la inteligencia artificial (1974 - 1980).



Figura 2.6: Kunihiko Fukushima.

Fuente: [IEICE](#)

Durante la década de los 70 se hacen aportes a la teoría *Hebbiana*, se aportan logros en al análisis y descripción de reglas adaptativas, además de otros aportes al principio de aprendizaje competitivo. En 1979 presentó Kunihiko Fukushima la primera red neuronal **Convolutional Neural Network (CNN)**, la llamó Neocognitron [39], dicho trabajo en un futuro se mejoraría con técnicas de **Backpropagation (BP)**.

En la década de los 80 se realizaron aportes como el algoritmo de **BP** que surgió del artículo de Hopfield [40], esto despertó la curiosidad de muchos investigadores a volver al campo de las redes neuronales. Se realizaron aportes como las redes **Graph Neural Network (GNN)**. La investigación continuó con Stephen Grossberg que realizó aportes derivados de estudios fisiológicos de cómo funcionaban las neuronas y la plasticidad, lo que permitió la creación de reglas y postulados, esto se ve en los trabajos de las redes [26]. La investigación de Hopfield basada en el trabajo de Stephen Grossberg creó un sistema computacional neuronal interconectado que tiende a un mínimo de energía. En 1985 David E. Rumelhart basándose en la investigación realizada por Paul Werbos [23] realizó un análisis experimental del algoritmo **BP** y su aplicación en redes **FNN** [21].

Yann LeCun junto a su equipo, en 1989 crearon la primera aplicación **CNN** con técnicas **BP** dicha aplicación podía reconocer números a partir de imágenes.



Figura 2.7: Adaptive Systems Research Department at Bell Labs 1989.

Fuente: [Twitter Yann Lecun](#)

En la década de los 90 se presentaron múltiples investigaciones y muchos avances en el campo, uno de los más importantes fue la presentación de la primera **GAN** como una curiosidad, ya que se presentó como un duelo entre dos redes neuronales, en un principio fue un generador probabilístico y un predictor con el objetivo de maximizar la

pérdida de cada uno en un juego *minimax*.

En 1991 se presentó el trabajo *Predictability Minimization* [41] dichas técnicas sirvieron de inspiración para el aprendizaje por refuerzo, En marzo de 1991 se hizo una aproximación a los *transformers* con auto atención, lograron separar el conocimiento del control como una máquina clásica, pero de una forma completamente neuronal, además de gestionar actualizaciones de los pesos de forma muy rápida y eficiente.

Durante la década de 1990 las redes neuronales tendían a ser muy sencillas, con pocas capas y no muy complejas por las limitaciones técnicas de la época. Por lo que muchos investigadores propusieron soluciones similares a las redes que permitían una retroalimentación, además de aceptar secuencias de información arbitraria. Otros propusieron soluciones como la jerarquía de autosupervisada que aprende representaciones en distintos niveles de abstracción. Comienzan a proponerse redes similares a las que en un futuro se llamarían **DBN** como un método no supervisado para **FNN**.

En junio de 1991 Sepp Hochreiter Figura 2.8 implementó el primer compresor de redes neuronales, además demostró uno de los principales problemas de las **NN** el llamado problema del desvanecimiento o explosión del gradiente 2.3.5.3 que hacía que el aprendizaje fallará. Un análisis posterior condujo a los investigadores a una primera aproximación , aunque no sería hasta 1997 con la revisión por pares y publicación del artículo *Long short-term memory* [31] que se solucionaría parcialmente el problema.

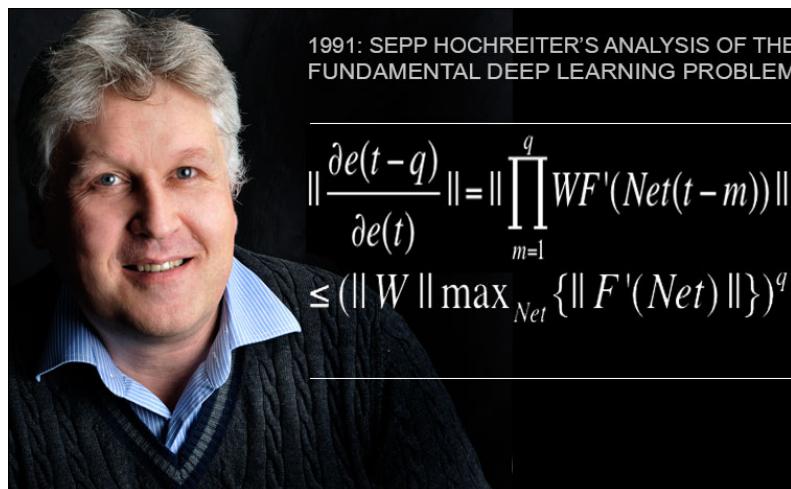


Figura 2.8: Sepp Hochreiter.
Fuente: [IDSIA](#)

Más adelante, en el 2014 Goodfellow Figura 2.9 presentó la primera red neuronal **Generative Adversarial Network (GAN)** pura para la generación de imágenes mediante el enfrentamiento de una red neuronal generativa contra una red neuronal discriminante entrenadas con el mismo conjunto de datos [35]. Durante los próximos años

se realizaron muchos aportes a las redes neuronales generativas, principalmente de paralelización de los cálculos, técnicas de estabilización, generación condiciona, arquitecturas más eficientes, funciones de pérdidas más adecuadas, aplicaciones específicas (cambiar el estilo de pintura), redes apiladas, etc. Fruto de todo ello, NVIDIA en 2018 presentó [Style GAN \(StyleGAN\)](#) [36] un modelo que es capaz de crear imágenes sintéticas de alta calidad y resolución, publicando el código en 2019 con mejoras relevantes a su primera versión.



Figura 2.9: Ian Goodfellow.
Fuente: [MIT Technology Review](#)

2.3. Estado del arte

2.3.1. La ciencia de datos

La ciencia de datos (*Data Science*) es el estudio de los datos con el objetivo de extraer información útil, se usa principalmente para dar información útil a empresas. Es un campo multidisciplinar, ya que combina campos de las matemáticas, estadística e inteligencia artificial para analizar grandes cantidades de datos. Esto tiene como objetivo responder a las siguientes cuestiones: *¿Qué paso?, ¿Por qué pasó?, ¿Qué pasará? O ¿Qué se puede hacer con los resultados?* [42]

La ciencia de datos analiza los datos de distintas formas.

1. **Análisis descriptivo:** examina datos con visualizaciones (gráficos, tablas) para entender eventos pasados o actuales.
2. **Análisis de diagnóstico:** profundiza en los datos para entender las razones detrás del evento. Emplea técnicas como descubrimiento de datos o correlaciones.
3. **Análisis predictivo:** utiliza datos históricos y técnicas como machine learning para hacer predicciones precisas sobre patrones futuros.

4. **Análisis prescriptivo:** busca la mejor respuesta para un resultado esperado. Utiliza técnicas como simulación y redes neuronales para recomendar el mejor curso de acción entre varias alternativas.

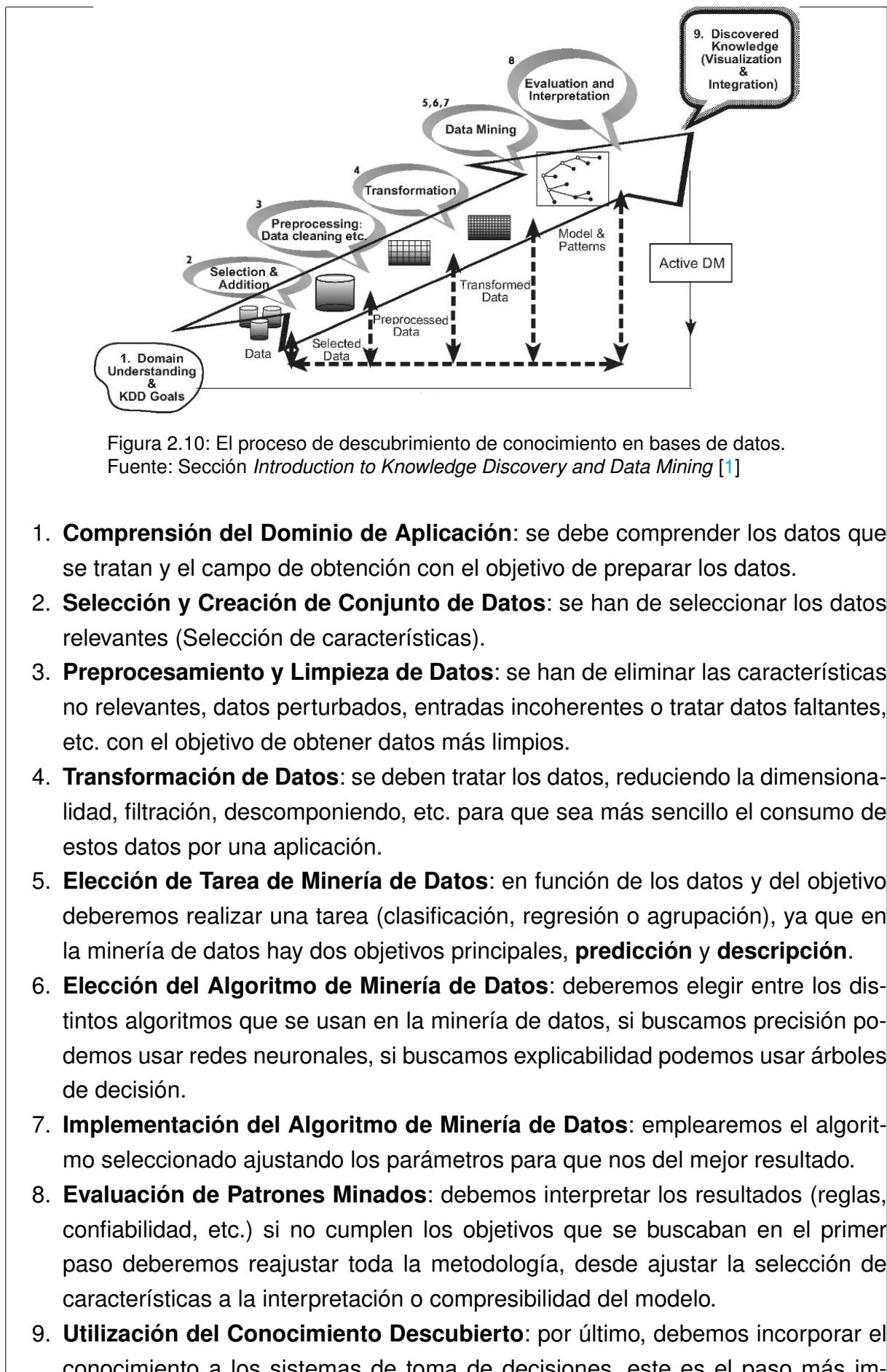
2.3.2. La minería de datos

La minería de datos, es una técnica asistida por computadora, procesa grandes conjuntos de datos para descubrir patrones y relaciones ocultas. Este conocimiento resultante se aplica en la resolución de problemas, análisis de decisiones empresariales, etc. Esta técnica tiene distintas fases para procesar y extraer información útil.

1. Comprender, identificar y definir el alcance del proyecto.
2. Comprender los datos.
3. Depurar datos (limpiar, integrar y dar formato).
4. Modelar datos.
5. Evaluar los resultados.
6. Implementar resultados.

La minería de datos es distinta en función de los datos y el objetivo, la mayoría del estado del arte segmenta la minería en tres tipos, minería de procesos, minería de textos y minería predictiva.

El Descubrimiento de Conocimientos en Bases de Datos **KDD** es un proceso que utiliza algoritmos de minería de datos para explorar y extraer conocimientos útiles de grandes bases de datos. Con el avance tecnológico, se emplean técnicas de inteligencia artificial para este propósito, con el objetivo final de obtener conocimiento de alto nivel a partir de datos de bajo nivel. La Figura 2.10 esquematiza el proceso general del *KDD*.



portante, ya que con él podemos medir los efectos del conocimiento obtenido. Puede suceder que una vez implementado el modelo en sistema de producción pierda eficiencia si las condiciones reales son distintas a las de la creación del modelo.

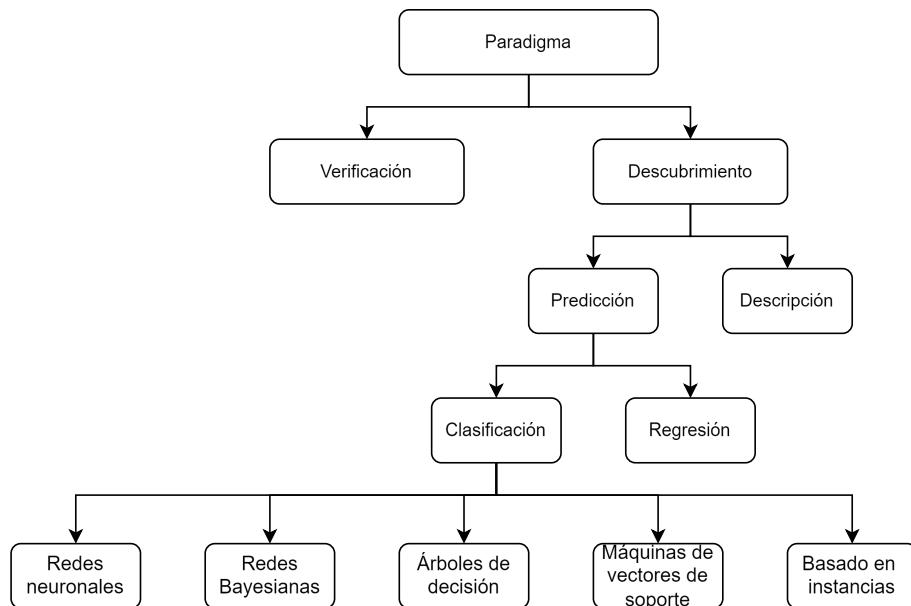


Figura 2.11: Taxonomía de minería de datos.

Fuente: Elaboración propia, inspirado en la sección *Introduction to Knowledge Discovery and Data Mining* [2]

La minería de datos puede estar orientada a la verificación o al descubrimiento de patrones, se debe automatizar su identificación, pudiendo usar un enfoque predictivo o descriptivo. Para el descubrimiento se usa el aprendizaje inductivo, mientras que en el paso de verificación se ha de evaluar las hipótesis externas usando métodos estadísticos clásicos. La terminología clásica del aprendizaje automático está clasificado en supervisado (clasificación y regresión) y no supervisado (agrupamiento), aunque existen muchos más términos en la terminología moderna esto podemos analizarlo más en profundidad en el libro *Data Mining and Knowledge Discovery Handbook* [2].

2.3.3. Aprendizaje automático *Machine Learning*

El *machine learning* es la ciencia o rama de la inteligencia artificial que desarrolla, modelos estadísticos, desarrolla algoritmos que generalizan comportamientos y reconocen patrones. Es decir, hace posible el aprendizaje autónomo de las máquinas para realizar tareas sin la necesidad programar las instrucciones explícitamente. El *machine*

learning busca procesar grandes cantidades de datos e identificar patrones de datos de forma automática.

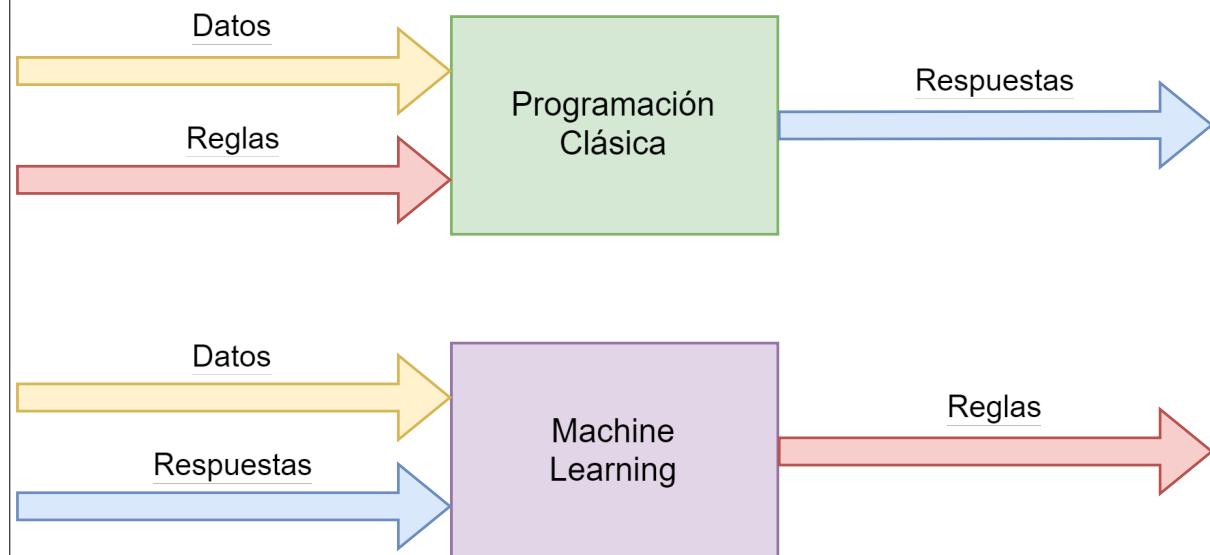


Figura 2.12: El *Machine Learning* como paradigma de programación.
Fuente: Elaboración propia.

Como podemos ver en la Figura 2.12 en el paradigma clásico se necesitaba conocer el dominio de los datos y las reglas en profundidad y programar los descriptores de forma manual para procesar los datos. En el paradigma del *machine learning* permite extraer esas reglas y patrones para procesar nuevos datos a partir de respuestas que conocíamos previamente, estas respuestas previas deben ser extraídas o validadas por expertos.

Una clasificación de las tareas del *machine learning* lo podemos ver en la Figura 2.13, esta clasificación está dividida en función de cómo se realiza el aprendizaje del modelo.

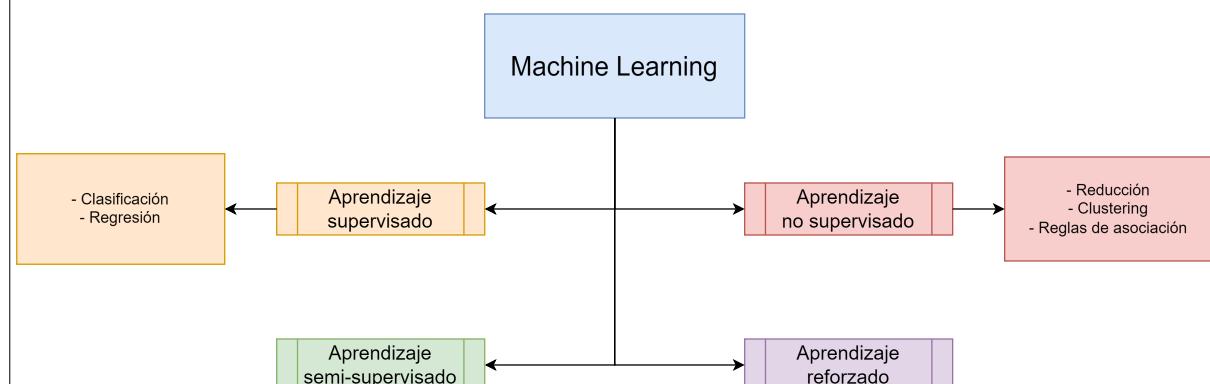


Figura 2.13: Clasificación de los algoritmos de aprendizaje automático.
Fuente: Elaboración propia.

El *machine learning* puede trabajar con datos estructurados y no estructurados,

aunque estos últimos requieren un procesamiento previo para adaptarlos en un formato estructurado.

2.3.4. Aprendizaje profundo

Deep Learning

El *deep learning* es una rama del *machine learning* que usa redes neuronales artificiales para analizar datos no lineales, estas redes imitan el comportamiento del cerebro humano, para esto suelen contar con arquitecturas complejas. Se distingue del *machine learning* por los tipos de datos con los que puede trabajar y por los métodos con los que hace que los modelos aprendan.

Otra de las características que diferencia al *deep learning* del *machine learning* es que elimina parte del procesamiento previo de datos, ya que sus algoritmos pueden ingerir y procesar datos no estructurados.

2.3.5. Redes neuronales artificiales

Artificial Neuronal Networks

El *deep learning* emplea principalmente redes neuronales para reconocer, clasificar y describir con precisión patrones de datos. Estas redes están inspiradas en el funcionamiento del cerebro humano. [43]

Las redes neuronales es un conjunto de neuronas artificiales que están organizadas generalmente por capas. Estas redes tienen una capa de entrada, una capa de salida y múltiples capas ocultas con distintas funciones de activación que ponderan los pesos de cada neurona.

Existen formas muy variadas de componer las capas, a esto se le denomina **topología de red neuronal**, en función del objetivo que se busque, la topología será muy distinta.

2.3.5.1. Elementos de una neurona artificial

Como se ha comentado previamente las neuronas artificiales están inspiradas en las neuronas biológicas del cerebro humano, fueron los investigadores *McCulloch* y *Pitts* [17] los que sentaron las bases de lo que es una neurona artificial, demostrando

que su modelo de red neuronal podía realizar cálculos que se correspondían con la lógica proposicional.

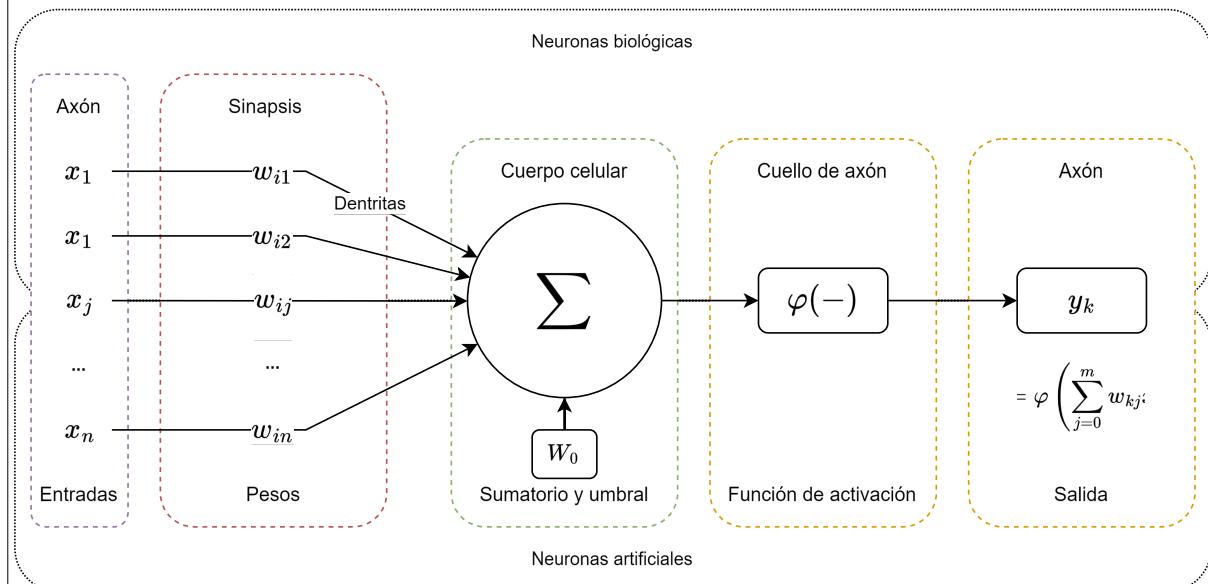


Figura 2.14: Partes de una neurona artificial bioinspirada en una neurona biológica.
Fuente: Elaboración propia.

En la Figura 2.14 podemos ver las distintas partes de una única neurona artificial y sus partes equivalentes con una neurona biológica. El axón opera como la entrada de información a la neurona. La sinapsis como las conexiones con la neurona que son los pesos. El cuerpo celular como la unidad de procesamiento central, donde se pondera la suma de las señales de entrada. El cuello del axón sería nuestra función de activación que se activa cuando supera un umbral. Y por último el axón sería nuestra señal de salida.

2.3.5.2. Funcionamiento de una red neuronal

Las redes neuronales profundas tienen multitud de capas con nodos interconectados, cada capa sobre la capa anterior con el objetivo de optimizar la precisión de una predicción o clasificación. A esta progresión de cálculos se le denomina “propagación hacia delante”.

El proceso de “propagación inversa” es el encargado de calcular errores de precisión, ajustar sesgos y ponderaciones usando algoritmos como [El descenso de gradiente](#).

La capa de entrada es por donde el modelo ingiere los datos, y la capa de salida es donde el modelo responderá con la predicción o clasificación una vez se haya

completado la fase de aprendizaje.

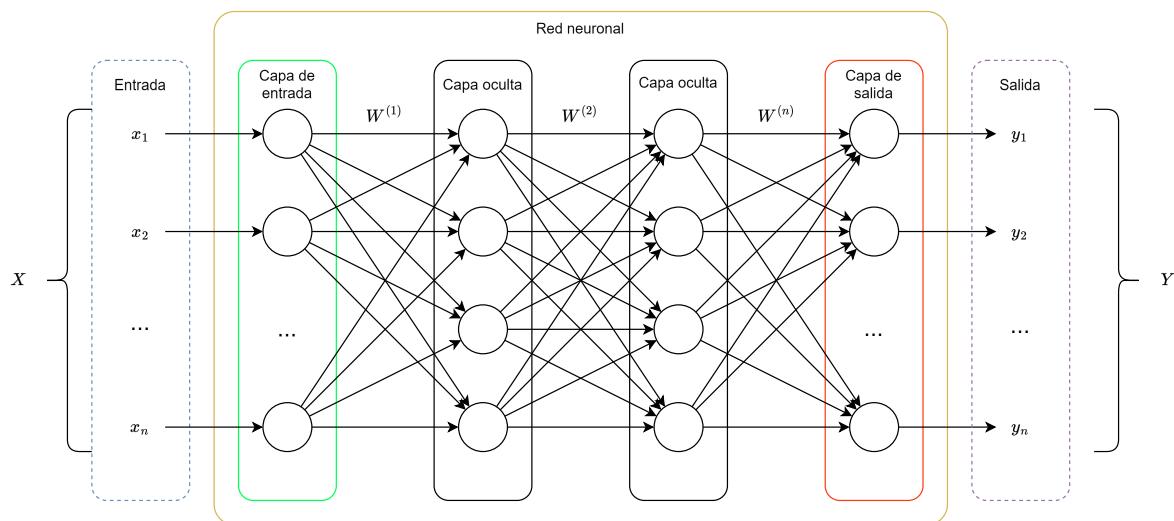


Figura 2.15: Red neuronal artificial.

Fuente: Elaboración propia.

En la Figura 2.15 podemos ver una arquitectura simple de una red neuronal que cuenta con la capa de entrada, la capa de salida y dos capas ocultas.

El vector de entrada $X = (x_1, x_2, \dots, x_n)$ será la información suministrada a nuestra red a través de la capa de entrada, este vector será de tamaño n .

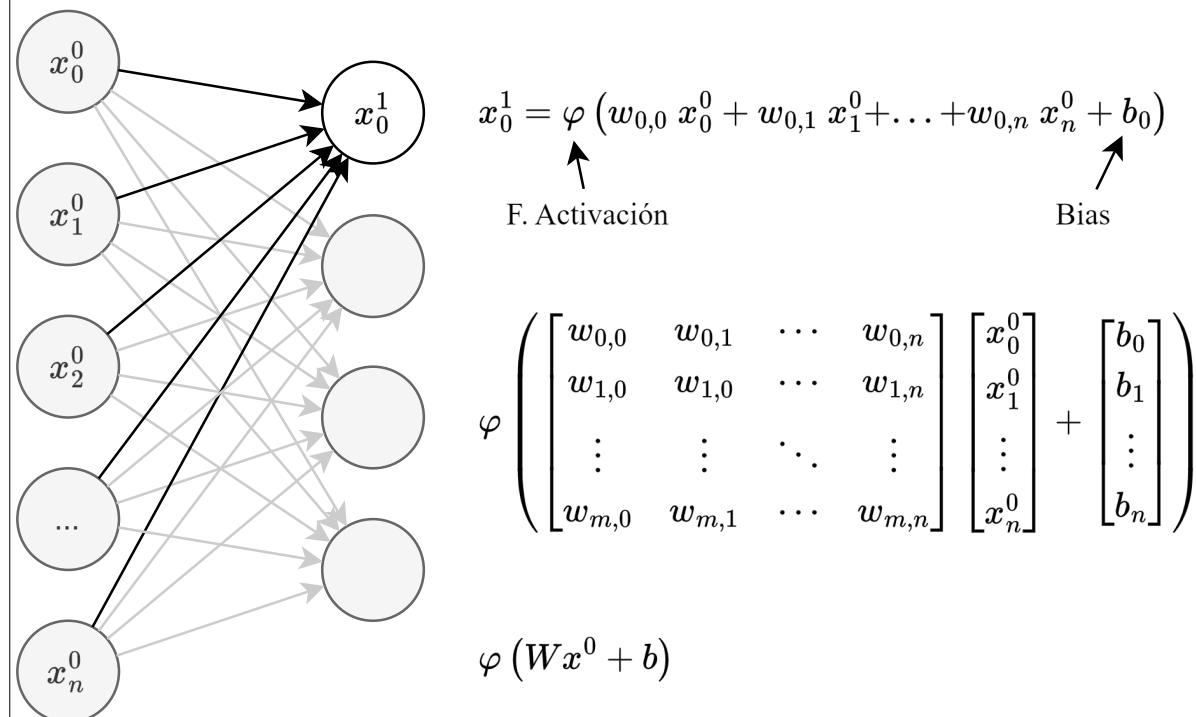


Figura 2.16: Cálculo de los pesos de la red neuronal artificial.

Fuente: Elaboración propia.

1. Función de la red neuronal: $f(x) = Wx + b$
2. Matriz de pesos: $W = [w_{0,n}, w_{1,n}, \dots, w_{m,n}]$
3. Vector de pesos: $w_n \in \mathbb{R}^D$
4. Vector de entrada: $x \in \mathbb{R}^D$
5. Vector del bias: $b \in \mathbb{R}^C$

El perceptrón simple

El Perceptron es una neurona artificial simple llamada también como [Linear Threshold Unit \(LTU\)](#). La [LTU](#) calcula una suma ponderada de los valores de las entradas, $Z = w_1x_1 + w_2x_2 + \dots + w_nx_n$ se aplica el umbral para generar el resultado.

La función umbral común en [LTU](#) es la función Heaviside [2.1](#).

$$heaviside(z) = \begin{cases} 0 & \text{si } z < 0 \\ 1 & \text{si } z \geq 0 \end{cases} \quad sgn(z) = \begin{cases} -1 & \text{si } z < 0 \\ 0 & \text{si } z = 0 \\ +1 & \text{si } z > 0 \end{cases} \quad (2.1)$$

Se puede emplear para una clasificación binaria lineal simple, mediante una combinación lineal de las entradas; si la operación supera el umbral, se obtendrá un resultado binario en un vector.

El algoritmo de entrenamiento del perceptrón se basa en ajustar los pesos de conexión entre las entradas y la neurona de salida para mejorar la precisión de las predicciones. Está inspirado en la regla de Hebb, que sugiere que las conexiones entre neuronas se fortalecen cuando una neurona activa repetidamente a otra.

Primero inicializa los pesos, predice, evalúa su error y actualización de los pesos para en la siguiente iteración minimizar su error.

Rosenblatt demostró que el algoritmo convergería hacia una solución, esto es llamado el Teorema de convergencia del perceptrón [\[44\]](#).

Marvin Minsky y Seymour encontraron una serie de debilidades que los perceptrones son incapaces de resolver, algunas de estas debilidades se pueden resolver apilando varios perceptrones, la [ANN](#) resultante se le denomina [Multilayer Perceptron \(MLP\)](#).

Uno de los problemas clásicos que tenían los perceptrones simples era el problema de clasificación XOR, esto está representado en la Figura [2.17](#). Como tal, el perceptrón es un modelo que puede aprender funciones lineales, pero el problema XOR no es

linealmente separable, lo que significa que no se puede trazar una línea recta para separar las dos clases (A y B) en el espacio de entrada binario de dos dimensiones.

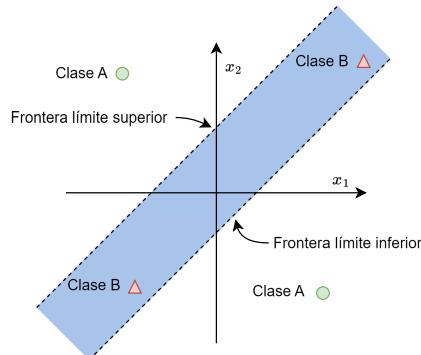


Figura 2.17: Problema de clasificación XOR.

Fuente: Elaboración propia.

El perceptrón multicapa y retropropagación

Fue en 1986 cuando D. E. Rumelhart propuso la retro propagación junto a múltiples capas para resolver el problema de clasificación XOR.

El proceso es el siguiente, por cada instancia de entrenamiento se calcula la salida de cada neurona en la capa consecutiva (paso hacia adelante). A continuación se mide el error de salida de la red (la diferencia entre la salida deseada y la de la red) calculará cuánto contribuyó cada neurona en la última capa oculta al error de cada neurona de salida. Se mide cuantas contribuciones de error provienen de cada neurona en la capa oculta anterior, así sucesivamente hasta que se llega a la capa de entrada (paso hacia atrás). Se mide eficientemente el gradiente de error en todos los pesos de la conexión en la red propagando el gradiente de error hacia atrás. [44]

Este proceso funciona, ya que se cambió la función escalonada de las LTU a una función logística Sigmoid, esto se requería, ya que la función de paso contiene solo segmentos planos y, por tanto, no había gradientes. Esto permitió que el descenso de gradiente pudiera ir progresando lentamente hasta converger. El algoritmo de retroalimentación se puede usar con otras funciones de activación que veremos más adelante.

2.3.5.3. El descenso de gradiente

El método de descenso de gradiente trata de encontrar un mínimo en una función dada, ya sea local o global. El método usa el gradiente negativo $-\nabla f$, con este método

obtenemos la dirección con el descenso máximo en los valores de la función, con esto buscamos encontrar la posición mínima.

En términos simples, el descenso de gradiente va ajustando iterativamente los valores de los parámetros en la dirección opuesta al gradiente de la función de costo. Al moverse en la dirección opuesta al gradiente, el algoritmo busca alcanzar el mínimo local o global de la función de costo.

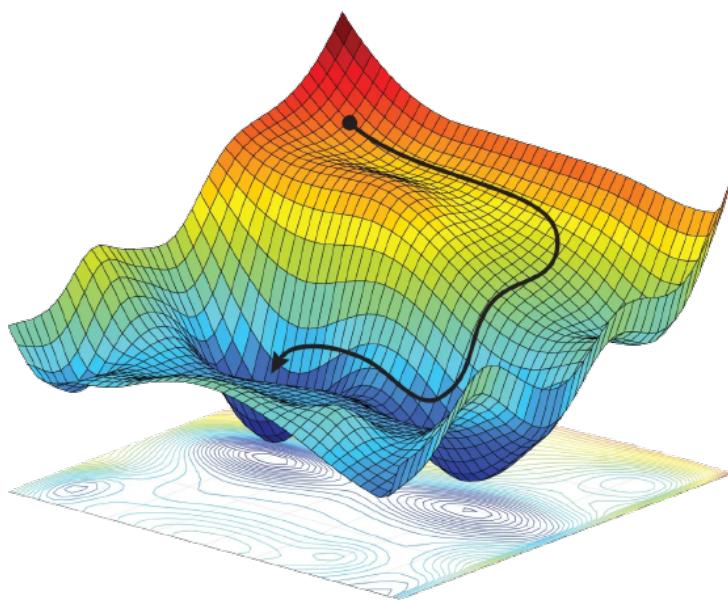


Figura 2.18: Descenso de gradiente.
Fuente: [F\(x\) Data Labs Pvt. Ltd.](#)

Algoritmo 1: Descenso de gradiente

```

1 Seleccionar  $x^{(0)}$ 
2 Establecer  $k \leftarrow 0$ 
3 while  $\|\nabla f(x^{(k)})\| \geq \epsilon$  do
4    $x^{(k+1)} = x^k - t_k - t_k \nabla f(x^{(k)})$ 
5    $k \leftarrow k + 1$ 
6 end
7 return  $x^{(k)}$ 

```

Hay tres variantes del algoritmo de aprendizaje del descenso de gradiente [45].

- **Descenso de gradiente por lotes:** suma el error para cada punto en un conjunto de entrenamiento, actualiza el modelo después de que todos los ejemplos de entrenamiento han sido evaluados, esto es lo que se le conoce como épocas. Es un método eficiente computacionalmente, pero costoso para grandes conjuntos de datos, ya que requiere almacenar en memoria todos los datos, suele producir

un gradiente de error estable y una convergencia, pero a veces ese punto de convergencia no es el ideal y encuentra el mínimo local frente al global.

- **Descenso de gradiente estocástico:** en cada época actualiza los parámetros del ejemplo de entrenamiento, esto reduce la necesidad de memoria, la actualización proporciona velocidad y detalles, aunque requieren un coste computacional mayor. Los gradientes son más ruidosos por sus actualizaciones frecuentes, esto le permite salir de mínimos locales.
- **Descenso de gradiente por mini lotes:** combina aspectos de del descenso de gradiente por lotes como el estocástico. Divide el conjunto de entrenamiento en pequeños lotes y realiza actualizaciones en cada uno de estos lotes. Esto permite tener un equilibrio entre eficiencia computacional y velocidad del descenso de gradiente.

Problema de desvanecimiento y explosión de gradiente

Hablaremos primero del desvanecimiento de gradiente, esto ocurre durante la retro propagación, ya que al movernos hacia atrás el gradiente sigue haciéndose más pequeño, esto hace que las capas anteriores de la red aprendan más lentamente que las posteriores. A medida que el algoritmo itera los cambios se hacen insignificantes, lo que hace que el algoritmo se estanque.

El problema de la explosión de gradiente es lo contrario, el gradiente se hace demasiado grande creando un modelo inestable. Los pesos del modelo crecerán y eventualmente se representarán como NaN.

2.3.5.4. Optimizadores

Los optimizadores son los algoritmos que buscan encontrar la mejor solución para un problema, esto se utilizan en conjunto con el descenso de gradiente para mejorar su eficiencia y rendimiento, generalmente minimizando o maximizando una función objetiva al ajustar sus parámetros. Existen muchos optimizadores que nos permiten explorar de forma eficiente el espacio de posibles soluciones.

Estos son algunos de los optimizadores más famosos.

1. **SGD** (Stochastic Gradient Descent with Momentum): Descenso de gradiente estocástico con momentum.
2. **Adam** (Adaptive Moment Estimation): Descenso de gradiente con tasa de apren-

dizaje adaptativa.

3. **Adamax** (Adaptive Moment Estimation with Infinity Norm): variante de Adam que utiliza la norma infinita en lugar de la norma 2 para calcular y actualizar el término de escala máximo.
4. **AdaGrad** (Adaptive Gradient Algorithm): variante de *SGD* en la que se emplean distintas tasas de aprendizaje teniendo en cuenta el gradiente acumulado en cada una de las variables.
5. **RMSprop** (Root Mean Square Propagation): variante de *AdaGrad* en la que, en lugar de mantener un acumulado los gradientes, se utiliza el concepto de “ventana” para considerar los gradientes más recientes.

2.3.5.5. Funciones de activación

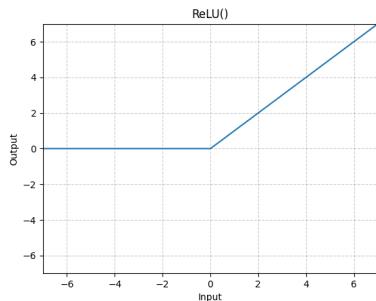
A la salida de una neurona artificial debe existir un filtro o umbral que modifica el resultado previo, esta función puede imponer un umbral que debe pasar para que el valor se transmita a la siguiente capa, esta función se le conoce como función de activación. Esto introduce no linealidad en el modelo, permitiendo que la red aprenda patrones complejos y mejora su capacidad de representación.

Existen muchas funciones de activación que modifican la red neuronal de distintas formas. Se suelen clasificar en dos categorías, funciones de activación de suma ponderada, no lineales u otras.

Existen una gran cantidad de funciones de activación que modifican el descenso de gradiente de forma que puede optimizar o ralentizar el entrenamiento llevando el modelo a una solución óptima o mediocre.

A continuación se muestran las funciones de activación lineales y no lineales más relevantes de la literatura actual, estas funciones y representaciones han sido reconocidas y estructuradas de la documentación de PyTorch [46].

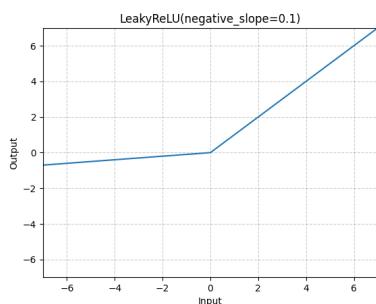
F. Activaciones no lineales - *Non-linear Activations (weighted sum, nonlinearity)*



(a) Función de activación ReLU.
Fuente: [Pytorch torch.nn.ReLU](#)

$$\varphi(x) = \max(0, x)$$

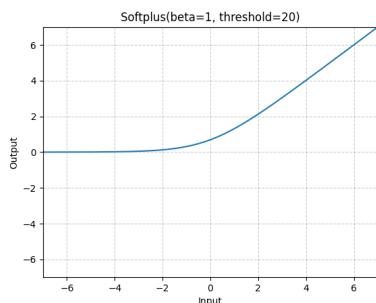
(b) Función de activación ReLU.
Fuente: [Pytorch torch.nn.ReLU](#)



(c) Función de activación LeakyReLU.
Fuente: [Pytorch torch.nn.LeakyReLU](#)

$$\varphi(x) = \max(0, x) + \text{negative_slope} * \min(0, x)$$

(d) Función de activación LeakyReLU.
Fuente: [Pytorch torch.nn.LeakyReLU](#)

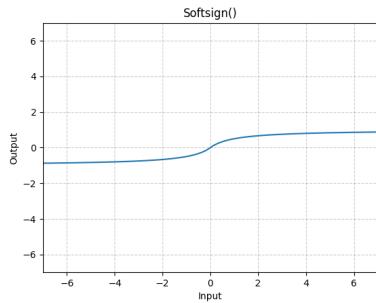


(e) Función de activación Softplus.
Fuente: [Pytorch torch.nn.Softplus](#)

$$\varphi(x) = \frac{1}{\beta} \log(1 + e^{\beta x})$$

(f) Función de activación Softplus.
Fuente: [Pytorch torch.nn.Softplus](#)

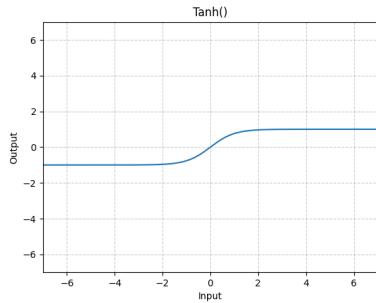
Figura 2.19: Funciones de activación no lineales



(a) Función de activación Softsign.
Fuente: [Pytorch torch.nn.Softsign](#)

$$\varphi(x) = \frac{x}{1 + |x|}$$

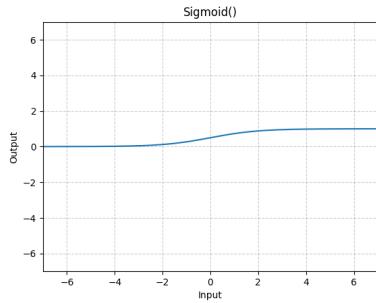
(b) Función de activación Softsign.
Fuente: [Pytorch torch.nn.Softsign](#)



(c) Función de activación Tanh.
Fuente: [Pytorch torch.nn.Tanh](#)

$$\varphi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

(d) Función de activación Tanh.
Fuente: [Pytorch torch.nn.Tanh](#)



(e) Función de activación Sigmoid.
Fuente: [Pytorch torch.nn.Sigmoid](#)

$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

(f) Función de activación Sigmoid.
Fuente: [Pytorch torch.nn.Sigmoid](#)

Figura 2.20: Funciones de activación no lineales

F. Activaciones no lineales - *Non-linear Activations (other)*

1. **Softmax**: Aplica la función Softmax a un Tensor de entrada n -dimensional reescalándolos para que los elementos del Tensor de salida n -dimensional se encuentren en el rango $[0,1]$. [46]

$$\varphi(x_i) = \frac{e^{x_i}}{\sum_j e^x_j} \quad (2.2)$$

2.3.5.6. Tipos de capas

Existen una gran variedad de capas con distintos propósitos, permiten organizar y estructurar el funcionamiento de la red durante el proceso de aprendizaje. Cada capa influye con un propósito específico, contribuyendo de manera única al funcionamiento global de la red.

Algunas de las características de los tipos de capas más famosas son las siguientes.

- **Capas densas y dispersas (Denses and Sparses Layers):**

- **Capas densas:** Capas que conectan todas las neuronas de una capa con todas las de la capa siguiente.

Utilizadas en [ANN](#) tradicionales para tareas de clasificación y regresión.

- **Capas dispersas:** Capas que contienen conexiones dispersas entre neuronas, lo que implica que no todas las neuronas están conectadas entre sí.

Se utilizan para reducir la complejidad computacional y el consumo de memoria en grandes modelos.

- **Capas lineales (Linear Layers):**

- Estas capas generalmente se combinan con funciones de activación no lineales para introducir no linealidad en la red neuronal.
 - Tanto `nn.Linear` como `nn.Bilinear` realizan transformaciones lineales de las entradas ponderando los pesos. `nn.Identity` no realiza ninguna transformación, y `nn.LazyLinear` puede realizar transformaciones lineales perezosas.
 - Proporcionan flexibilidad al permitir ajustar sus parámetros según las necesidades del modelo o la tarea.

- **Capas Convolucionales (Convolutional Layers):**

- Aplican filtros o convoluciones a la entrada para detectar patrones como bordes, texturas y formas.
 - Se utilizan principalmente para procesar datos de imágenes y extraer características relevantes.
 - Son la base del funcionamiento en tareas de clasificación de imágenes o de segmentación de imágenes.

- **Capas de Pooling (Pooling Layers):**

- Comprimen la dimensionalidad de las características extraídas de las capas convolucionales. Se usan entre capas convolucionales.
 - Existen múltiples variantes como `nn.MaxPool1d` o `nn.AvgPool1d` que seleccionan los valores más significativos de una región dada.

- Ayudan a reducir el overfitting, simplificar la representación de características, reducir el coste computacional, etc.
- **Capas de Padding (Paddings Layers):**
 - Se usan para ajustar las dimensiones de los datos de entrada, permiten después aplicar operaciones.
 - Añade ceros o constantes alrededor de los bordes de regiones para ajustar el tamaño a posteriores operaciones.
 - Previenen la pérdida de información en los bordes.
 - **Capas de Normalización (Normalization Layers):**
 - Realizan operaciones sobre los mapas de activación, se utilizan para estabilizar y acelerar el entrenamiento de la red.
 - Algunas de las capas relevantes son `nn.BatchNorm1d` y `nn.LayerNorm`, estas permiten aplicar una normalización por lotes o por capas respectivamente.
 - Ayudan a mantener la distribución de activaciones más consistentes durante el entrenamiento.
 - **Capas de Dropout (Dropout Layers):**
 - Se desactivan aleatoriamente un porcentaje de neuronas en la capa, esto evita dependencias en el entrenamiento, además de generalizar mejor.
 - Es una técnica de regularización que reduce el sobreajuste, se usan para entrenamientos más estables eliminando dependencias.
 - **Capas Recurrentes (Recurrent Layers):**
 - Diseñadas para manejar datos secuenciales o de series temporales.
 - Incorporan conexiones cíclicas que les permiten propagar información de un paso de tiempo a otro, las más usadas son `nn.RNN`, `nn.LSTM` y `nn.GRU`.
 - Las capas recurrentes tienen memoria y contexto considerando los estados previos, son útiles para traducción de idiomas, procesamiento del lenguaje natural y reconocimiento de voz.

2.3.5.7. Topologías de redes neuronales

La topología o arquitectura de una red neuronal nos referimos a la organización y disposición de las neuronas en la red, formando capas. Estos parámetros fundamentales determinan cómo se estructura la red. Se suele considerar el número de capas, número de neuronas por capa, tipo de conexiones (unidireccionales o recurrentes).

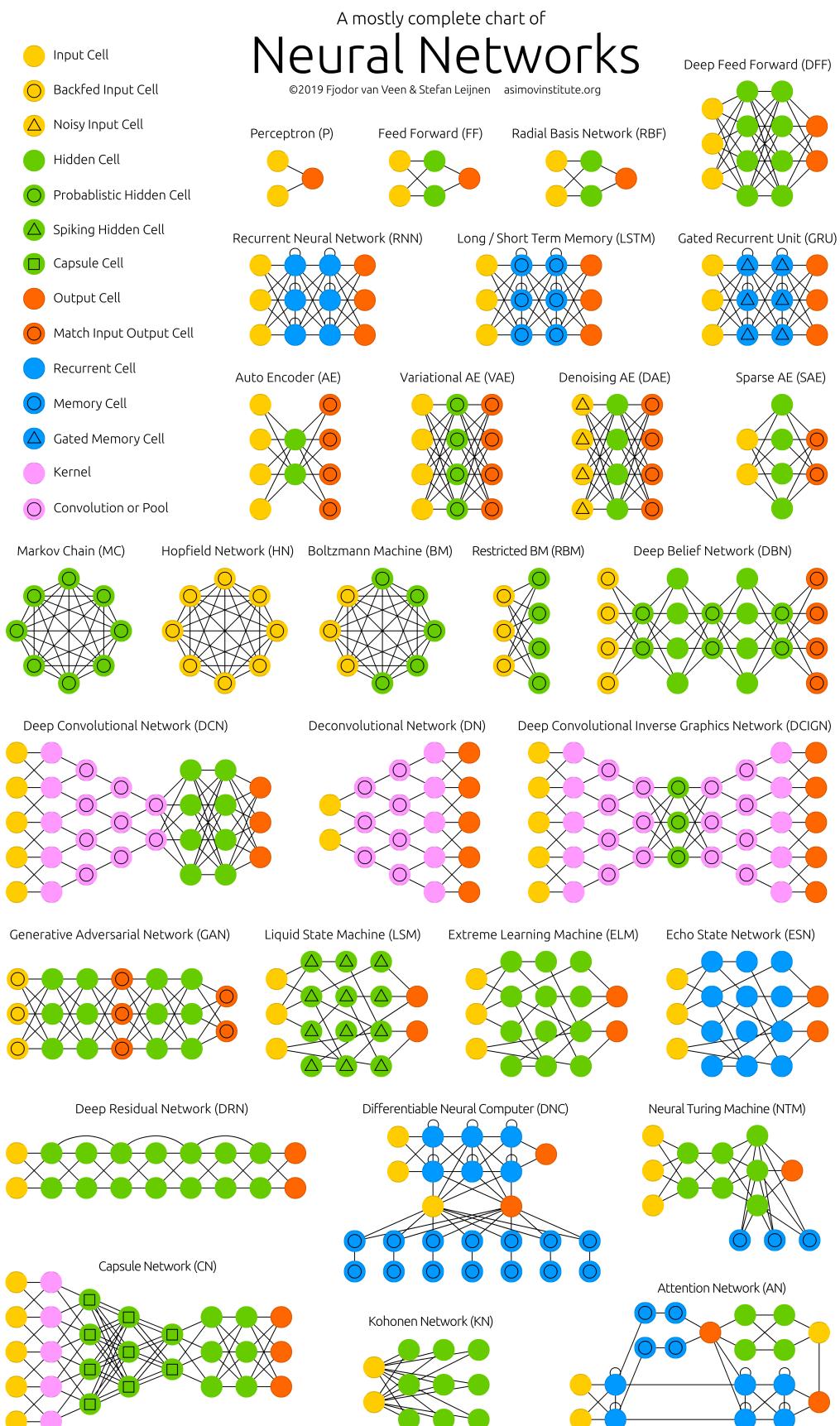


Figura 2.21: Topologías de redes neuronales.
Fuente: The neural network zoo

2.3.6. Autocodificador Variacional

Variational Autoencoders (VAE)

Los **Variational Autoencoder (VAE)** son complejos, ya que se basan en el teorema de **Bayes**, en la divergencia de Kullback-Leibler, junto con los **Autoencoder (AE)** que son un tipo de arquitectura de red neuronal **NN** que dan como salida el mismo tipo de dato que el dato de entrada. Estos modelos comprimen en un espacio latente los datos de entrada para después descomprimir la información.

Los **AE** son sencillos, pero no funcionan bien para la generación de instancias, pues lo que queremos es generar variaciones de la entrada.

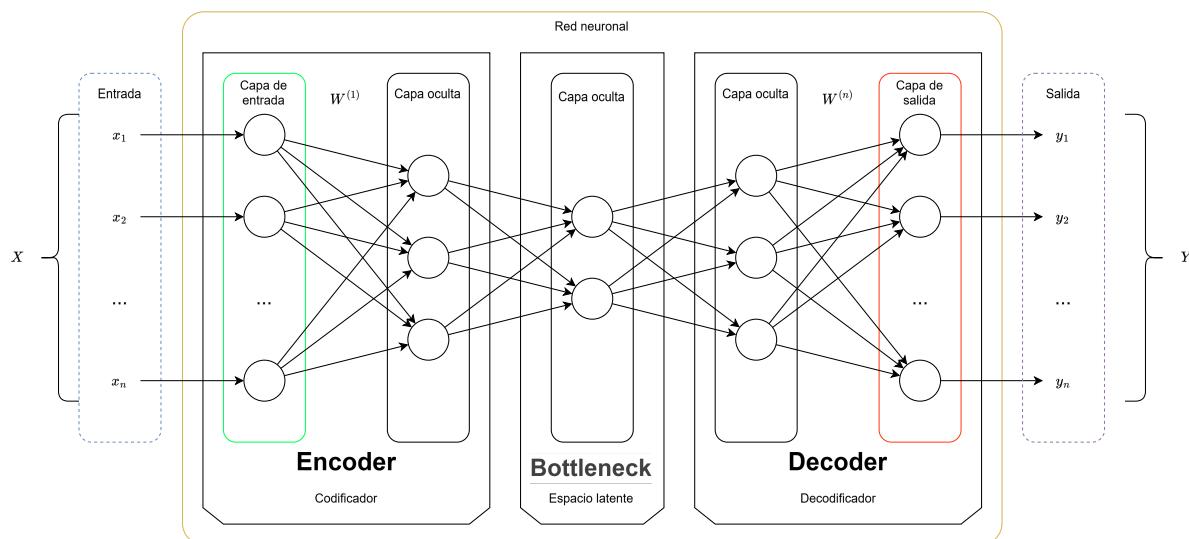


Figura 2.22: Encoder-Decoder.
Fuente: Elaboración propia.

Los modelos **VAE** dan una reconstrucción del dato de entrada, tienen múltiples funcionalidades, reducción de la dimensionalidad, compresión de datos, permiten eliminar el ruido, detectar anomalías, generar datos, aprendizaje de representación, traducción de imágenes y relleno de imágenes, etc.

Los **VAE** son modelos predictivos que ha demostrado ser exitosos para aprender de forma no supervisada distribuciones complejas de datos e iterar sobre el espacio latente. Permiten generar instancias complejas nuevas como números manuscritos, rostros humanos o segmentar imágenes [47].

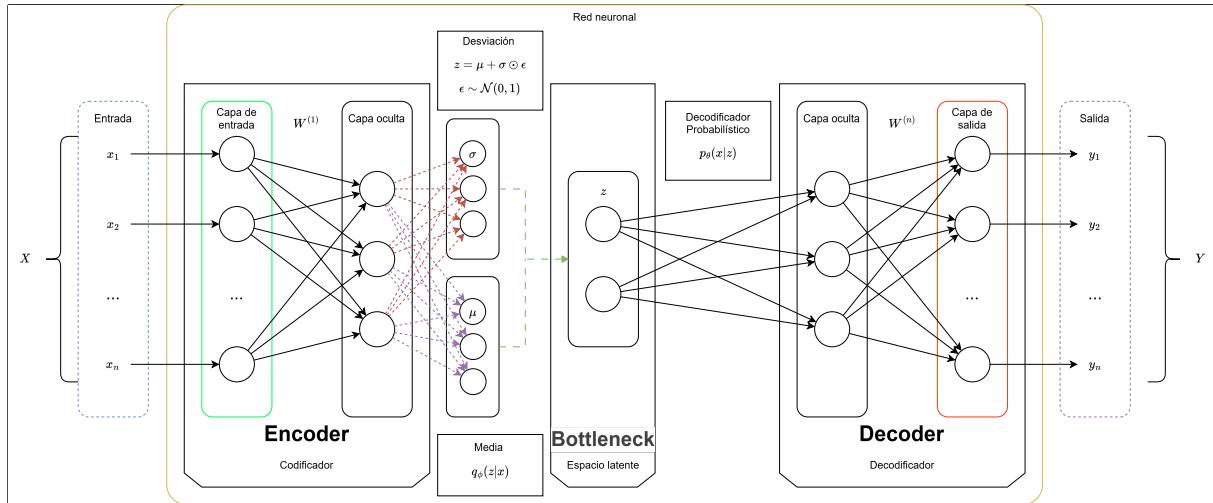


Figura 2.23: Variable Auto-encoder.

Fuente: Elaboración propia, inspirado en el capítulo 2 El auto-encoder variacional [3]

Los **AE** presentan problemas, estos los intentan solucionar **VAE**, como vemos en las figuras 2.22 y 2.23, los **AE** agrupan la codificación en grupos distintos y los **VAE** son más continuos, los **VAE** se mapean a una distribución, esto es gracias a que en el codificador de los **VAE** se usan dos vectores, un vector es la mediana μ que indica donde debería estar la codificación en el espacio latente y otro vector es la desviación estándar σ que representa el área alrededor de ese punto. Según como avanza el entrenamiento, el decodificador aprende los puntos y los vectores que los rodean. [48]

El entrenamiento y generación de instancias con **VAE** es el siguiente:

- Se eligen dos instancias entre las que se quiere generar.
- Se introducen las dos instancias en el codificador y se obtiene un vector latente para cada instancia.
- Elige varios vectores intermedios que se encuentre entre los vectores latentes de cada instancia.
- Se toman los vectores intermedios y se pasan por el decodificador del para generar las instancias intermedias entre las dos instancias iniciales.

Los **VAE** presentan distintos problemas, uno de los principales es que solo aprenden una representación latente continua, para ello aparecen los **Vector Quantised Variational AutoEncoder (VQ-VAE)** que estos sí pueden aprender representaciones latentes discretas, realizando reconstrucciones más nítidas. Los *auto encoders* cuantizados añaden una lista de vectores gestionada por un índice, mediante la distancia euclíadiana se entrena el decodificador.

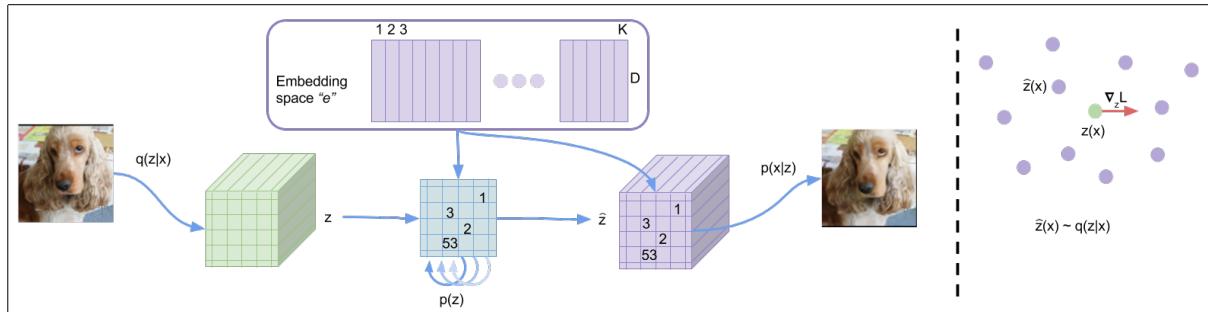


Figura 2.24: Vector Quantised Variational AutoEncoder (VQ-VAE).
Fuente: Neural Discrete Representation Learning [4]

2.3.7. Redes generativas adversariales *Generative Adversarial Networks (GAN)*

Las primeras redes neuronales adversariales se presentaron en la década de los 90 como una curiosidad, no fue hasta 2014 con *Ian Goodfellow* que creó y propuso la primera **GAN**. Su arquitectura estaba formada por dos redes neuronales artificiales, una “generadora (G)” y otra “discriminadora (D)”. La red G se encarga de generar instancias del mismo dominio que el conjunto de datos, mientras que la red D es la encargada de aceptar o rechazar si los datos generados por la red G son reales o falsos. Ambas redes se entrena conjuntamente de manera que G minimiza las detecciones de D , y a su vez D detecta las instancias generadas por G [49].

Las redes generativas pueden contar con lo que se denomina ruido o *latent space*. El *latent space* es una representación abstracta de las características de los datos de entrada que el generador utiliza para crear nuevos ejemplos que se asemejen con las instancias del conjunto de entrenamiento. El objetivo de nuestras redes generadoras es que aprendan a mapear puntos de este espacio latente a puntos del espacio pertenecientes al conjunto de entrenamiento.

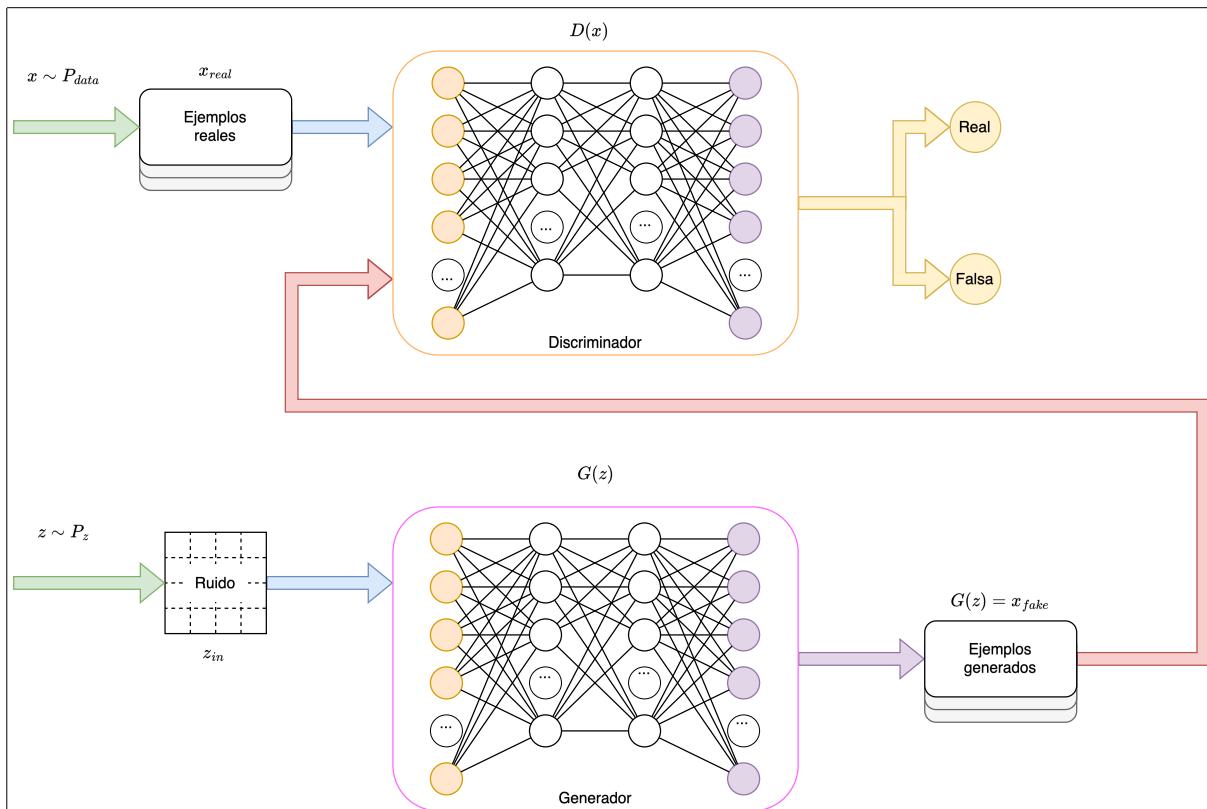


Figura 2.25: Arquitectura de las redes neuronales generativas adversariales.
Fuente: Elaboración propia.

Una **GAN** básica se entrena de forma similar a un algoritmo “mín máx”, esto está representado en la ecuación 2.3. Con este proceso hacemos que la red G mejore la generación de instancias haciéndola más parecidas, aunque también puede generar instancias irreales, ya que, aunque las instancias sintéticas superan la validación del discriminador, no son físicamente viables en un contexto real.

Como podemos observar en la ecuación 2.3 está compuesta por dos secciones similares que luego se suman, estas partes representan las dos redes neuronales compitiendo y la pérdida (*loss*) de ambas, en función de cómo modifiquemos los distintos componentes y estructuras de las redes neuronales la **GAN** se comportara de una forma u otra, estas modificaciones las veremos en las siguientes secciones.

$$\min_G \max_D \mathbb{E}_{x \sim P_{\text{data}}(x)} [\log (D(x | y))] + \mathbb{E}_{z \sim P_{\text{noise}}(z)} [\log (1 - D(G(z | y)))] \quad (2.3)$$

Esto está representado en la Figura 2.25, donde podemos observar las distintas partes de una red **GAN**. Además, podemos observar la distribución de probabilidad aleatoria representada como P_z , P_{noise} . El objetivo de la red es la optimización de la

distribución de probabilidad de la red generativa P_x, P_{data} sea similar a $G(z)$, es decir, $G(z) \sim P_g$.

$$\min_G \max_D = \mathbb{E}_{x \sim P_{\text{data}}} [\log(D(x))] + \mathbb{E}_{z \sim P_{\text{noise}}(z)} [\log(1 - D(G(z)))] \quad (2.4)$$

La función objetiva está definida por la ecuación 2.5, nos referimos con θ a los parámetros de la red G y con ω a los parámetros de la red D . La esperanza de $f(x)$ con respecto a la distribución de probabilidad de x según Q es lo que se denota como $\mathbb{E}_{x \sim Q}$.

$$\min_{\theta} \max_{\omega} = \mathbb{E}_{x \sim Q} [\log(D_{\omega}(x))] + \mathbb{E}_{z \sim P_{\theta}} [\log(1 - D_{\theta}(G_{\theta}(z)))] \quad (2.5)$$

Para que la red D funcione deberá recibir tantos datos originales x_{real} como datos generados por G , x_{fake} , G se optimiza al mismo tiempo para que D no pueda detectar los datos generados por G , este proceso está representado en las ecuaciones 2.6 y 2.7.

Las redes GAN originales produce una probabilidad de que la imagen de entrada proceda de la distribución generadora de datos. Usaban una red *feed-forward* con función final sigmoidea.

En la ecuación 2.6 podemos observar la función de pérdida de la red discriminadora (D), la sección $\log(D(x))$ es la encargada de dar la probabilidad de que está clasificando correctamente la instancia generada por la red G , maximizando la sección $\log(1 - D(G(z)))$ conseguimos etiquetar correctamente la instancia falsa generada por la red G .

$$\nabla_{\omega_D} \frac{1}{m} \sum_{i=1}^m [\log(D(x^{(i)})) + \log(1 - D(G(z^{(i)})))] \quad (2.6)$$

En la ecuación 2.7 podemos observar la función de pérdida de la red generadora (G), la pérdida del generador se calcula a partir de la clasificación (Real, Falsa) del discriminador. Se recompensa si consigue engañar al discriminador y se le penaliza en caso contrario.

$$\nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^{(i)}))) \quad (2.7)$$

2.3.7.1. Usos y aplicaciones

Las redes **GAN** se han usado principalmente para la generación de imágenes, aunque puede generar instancias sintéticas de muchos dominios.

Algunas de las tareas que se pueden realizar con una red **GAN**

- **Síntesis de imágenes:** permite crear una imagen a partir de unos datos de entrada.
- **Imagen a imagen:** permite “traducir” una imagen, es decir, transformar una imagen de un estilo a otro.
- **Escalado de imágenes:** aumentar la resolución de una imagen sin perder calidad, generando datos faltantes para no perder detalles.
- **Eliminación de ruido:** limpiar imágenes de baja calidad, son capaces de mejorar la claridad y reconstruir artefactos
- **Detección de manipulación de la cámara:**
- **Codificación de vídeo:** mejorar la eficiencia de compresión de video mediante la generación de cuadros intermedios o reducción de ruido.

2.3.7.2. Taxonomía de GANs

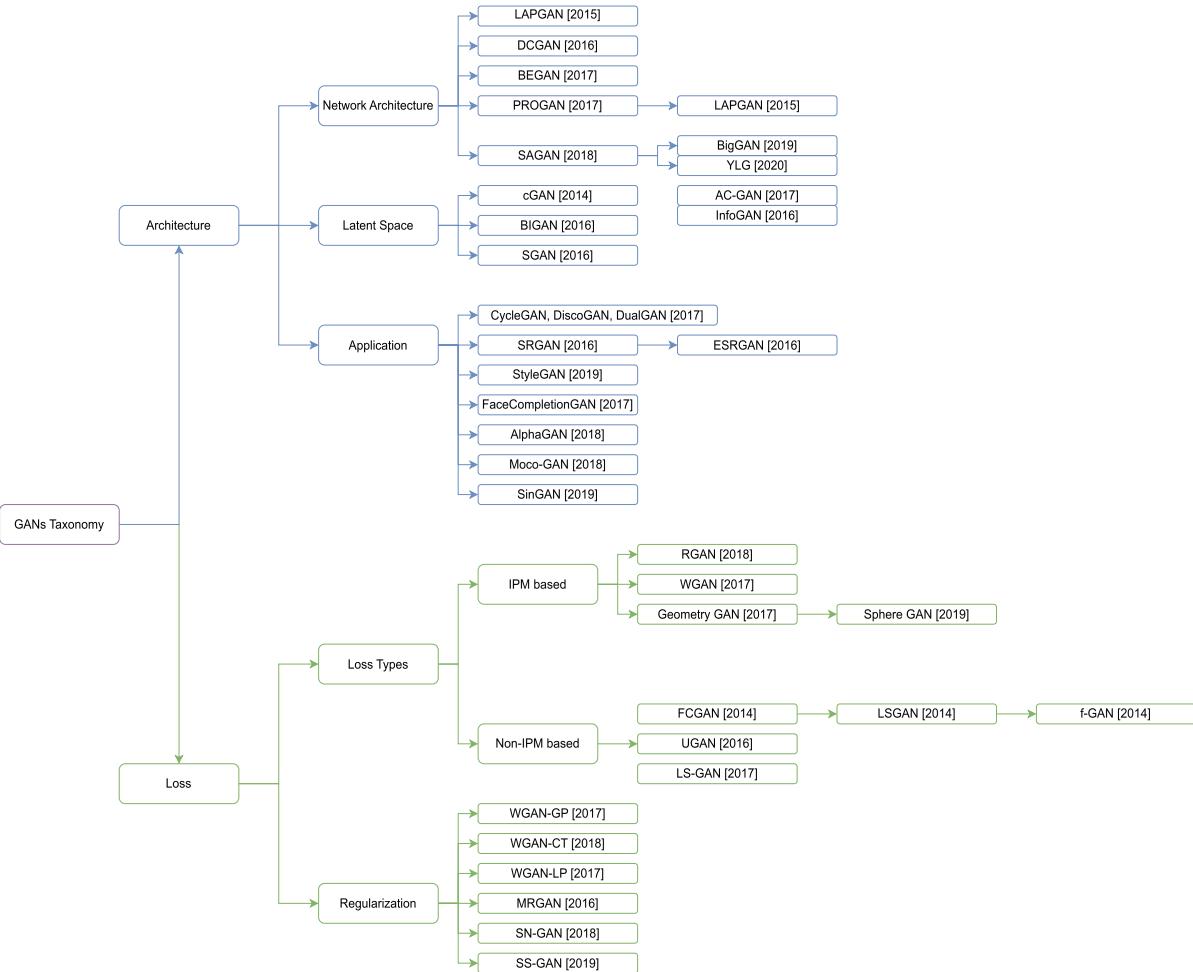


Figura 2.26: Taxonomía de GANs.

Fuente: Elaboración propia, adaptado del artículo “GANs Survey” [5]

Conditional Generative Adversarial Network (cGAN) [2014]

Las redes **cGAN** [50, 51, 52, 53] o GAN condicional fue una de las primeras derivaciones que se crearon, modifica la arquitectura alterando el ruido introducido, permitiendo condicionar la red con información adicional como etiquetas de clase. Esto permite que la red aprenda a generar instancias de una clase dada.

En esta arquitectura se modifica tanto la red generadora G como la red discriminadora D , de forma que ambas deben conocer la clase de la instancia.

$$\min_G \max_D = \mathbb{E}_{x \sim P_{\text{data}}(x)} [\log (D(x | y))] + \mathbb{E}_{z \sim P_{\text{noise}}(z)} [\log (D(G(z | y)))] \quad (2.8)$$

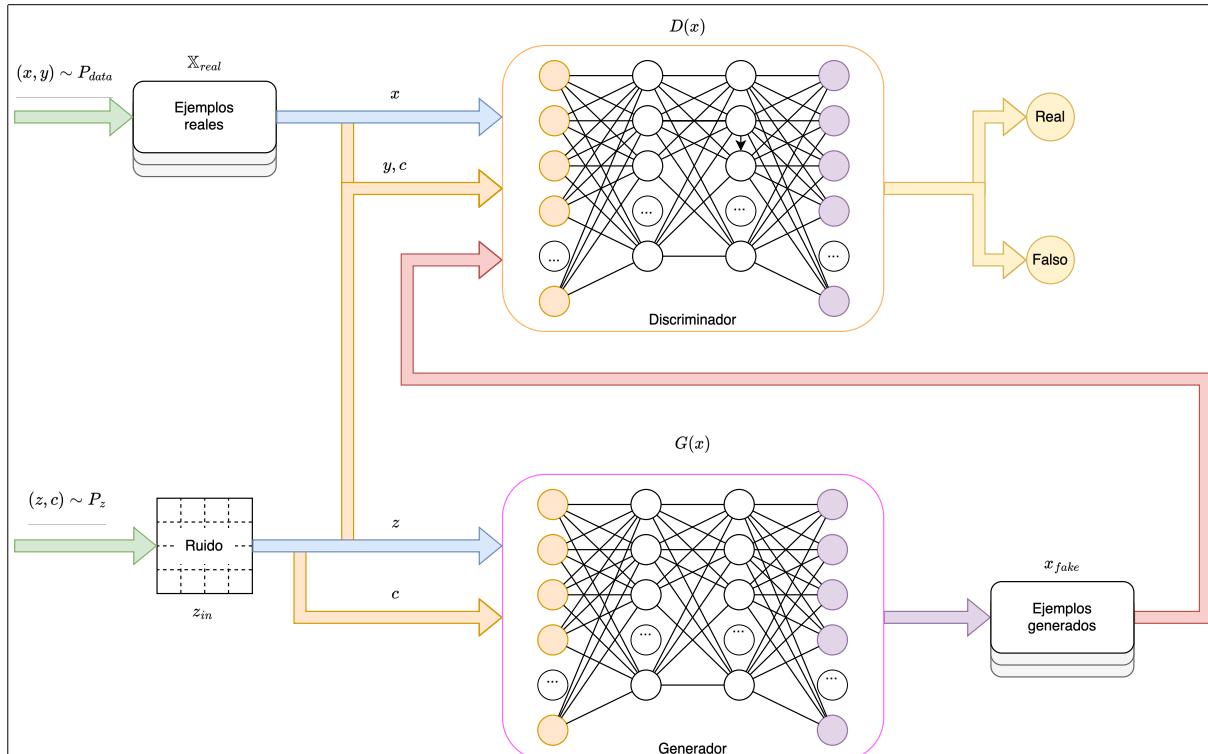


Figura 2.27: Arquitectura Conditional GAN (cGAN)
Fuente: Elaboración propia.

Deep Convolutional Generative Adversarial Network (DCGAN) [2015]

Las redes **Deep Convolutional GAN (DCGAN)** [54, 55, 56, 57] son una de las redes generativas más famosas y usadas dentro de su ámbito por las ventajas de rendimiento, además estas redes generan detalles de alta resolución partiendo de puntos espacialmente locales de un mapa de características reducido.

El discriminador es un clasificador de imágenes basado en **CNN**, frente al generador que a partir de una semilla (ruido) generara imágenes apilando distintas capas.

Las **DCGAN** alteran las capas en D cambiando cualquier capa de agrupación (*pooling layers*) por convoluciones escalonadas y en la red G las agrupaciones por convoluciones escalonadas fraccionadas, usan **Batchnorm** en ambas redes, eliminan capas ocultas de tipo *fully-connected*, usan funciones de activación **ReLU** en el generador para todas las capas excepto en la capa de salida que usan **Tanh** [45].

Semi-supervised Generative Adversarial Network (SGAN) [2016]

En las redes **Semi supervised GAN (SGAN)** [58], la red generadora G como la red discriminadora D cuentan con datos semi-supervisados, esto difiere de una **GAN** al modificar la red discriminadora con un clasificador. Aprende a clasificar las muestras en diferentes clases en función de las etiquetas dadas.

El uso de estos datos semi-supervisados y de un clasificador en la red discriminadora D , mejoran los resultados de ambas redes. Además, permite contar con un volumen mayor de datos.

Se ha demostrado que las redes **SGAN** mejoran el rendimiento de la clasificación en conjuntos de datos restringidos.

En la Figura 2.28 podemos observar la arquitectura de la red usando datos etiquetados, además del clasificador final de la red D .

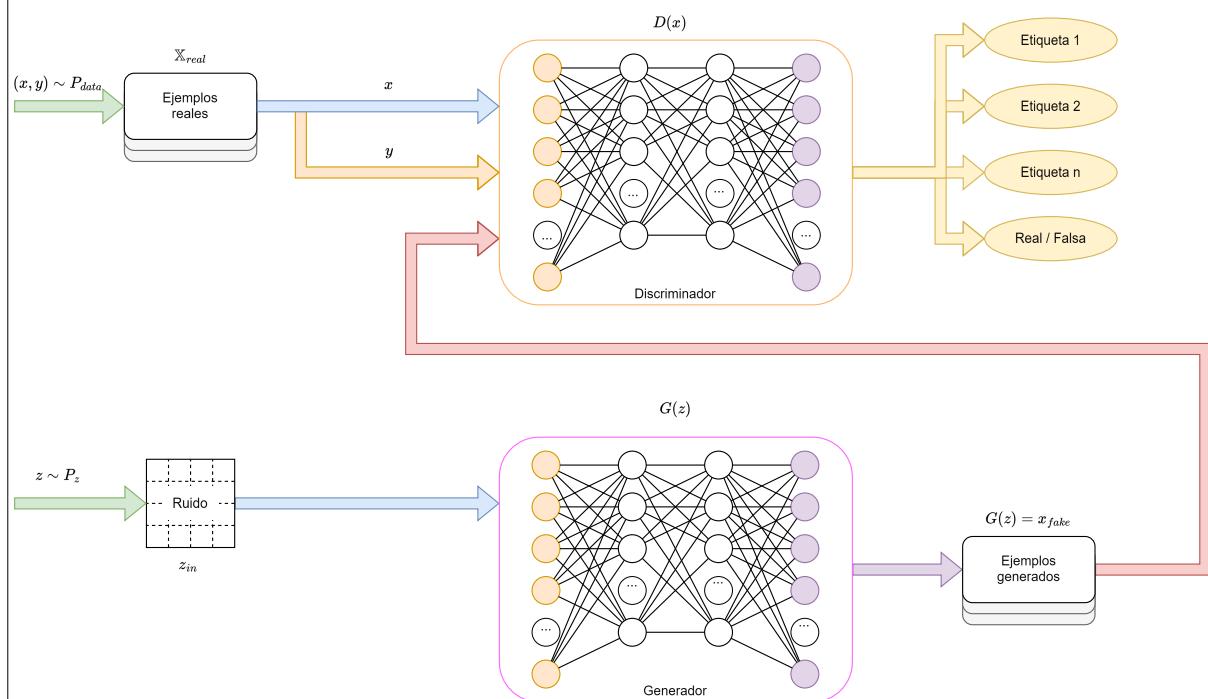


Figura 2.28: Arquitectura Semi-supervised GAN (SGAN)
Fuente: Elaboración propia.

Couple Generative Adversarial Network (CoGAN) [2016]

Las redes **Couple GAN (CoGAN)** [59] son dos **GAN** que comparten ciertos pesos de las capas, esto permite distribuir de forma conjunta instancias de varios dominios sin necesidad de incluir tuplas.

$$\begin{aligned} \min_{G_1, G_2} \max_{D_1, D_2} = & \mathbb{E}_{x_1 \sim P_{x_1}} [\log (D_1(x_1))] + \mathbb{E}_{z \sim P_{\text{noise}}(z)} [\log (D_1(G_1(z)))] \\ & + \mathbb{E}_{x_2 \sim P_{x_2}} [\log (D_2(x_2))] + \mathbb{E}_{z \sim P_{\text{noise}}(z)} [\log (D_2(G_2(z)))] \end{aligned} \quad (2.9)$$

En la Figura 2.29 podemos observar el funcionamiento básico en una iteración de esta arquitectura, lo más relevante es el trabajo conjunto de ambas redes por sus pesos compartidos, permitiendo que sea un entrenamiento mucho más rápido.

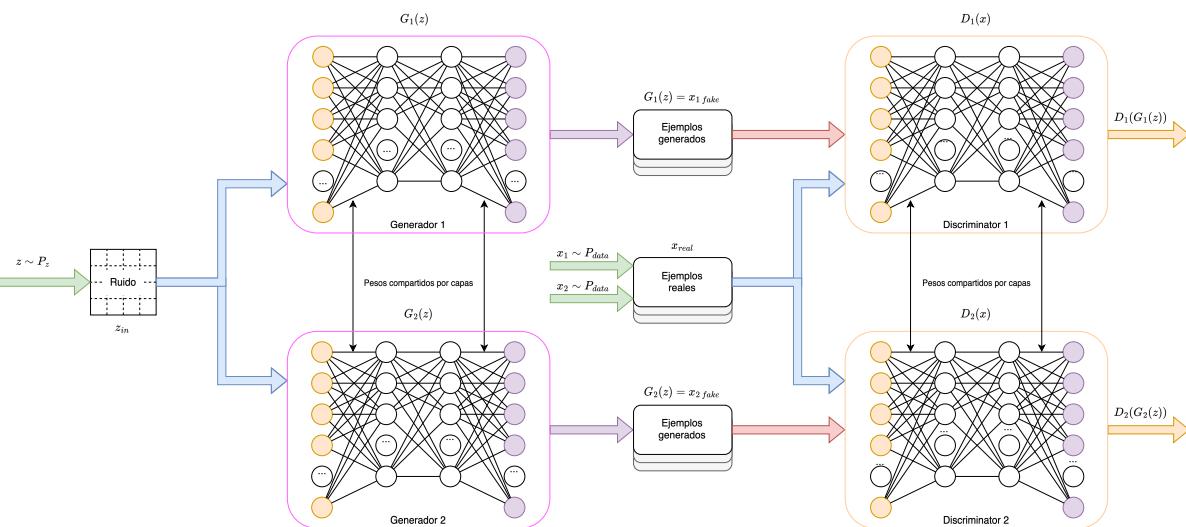


Figura 2.29: Arquitectura Couple GAN (CoGAN)
Fuente: Elaboración propia.

Patch Generative Adversarial Network (PatchGAN) [2016]

Patch GAN (PatchGAN) [60] es una arquitectura en la red discriminadora D , se emplea en tareas de traducción de imagen por ejemplo en redes **Pixel to Pixel (Pix2Pix)**, la idea principal es simple, segmentar la imagen en parches ($N \times N$) y discriminar (real o falso) cada uno de esos parches por separado, se centra en analizar las estadísticas de estilo locales y la estructura de alta frecuencia. Estos discriminadores han demostrado que son eficaces a la hora de generar resultados nítidos y de reducir artefactos. Estos discriminadores se suelen usar en conjunto con redes generadoras **G Convolutional Networks (U-Net)** [61] para tareas de traducción de imágenes.

A continuación se explica de forma simplificada las ecuaciones de una red **PatchGAN**, debemos recordar que estas redes son **ConvNET PatchGAN Discriminator**

Se ha demostrado que **PatchGAN** reduce los artefactos [60], además de que se puede usar en multitud de otras redes como un complemento que mejora la velocidad

de entrenamiento y resultados. En el estudio

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z} [\|y - G(x, z)\|_1] \quad (2.10)$$

Cycle Generative Adversarial Network (CycleGAN) [2017]

Las redes [Cycle GAN \(CycleGAN\)](#) [62, 63, 64] se componen por dos generadores y dos discriminadores, esta red es similar a las redes [Pix2Pix](#) [65], ya que lo que hacen es “traducir” una instancia a otro dominio, estas redes que a su vez deriva de las redes cGAN.

El funcionamiento de estas redes es el siguiente, el primer generador $G_{X \rightarrow Y}$, toma la imagen x del dominio X y la asigna a \hat{y} , se comprueba con el discriminador D_Y que \hat{y} sea similar a las instancias reales pertenecientes al dominio Y , este discriminador clasificará si la instancia es una muestra real perteneciente al dominio Y o es una instancia generada por $G_{X \rightarrow Y}$.

Estas redes permiten extraer las características de un dominio e incorporar esas características a otro dominio con las ventajas de que no necesita de datos de entrenamiento emparejados.

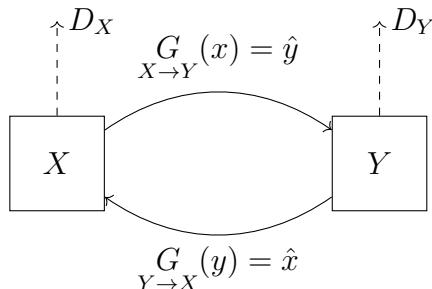


Figura 2.30: Representación CycleGAN

Función de mapeo para el paso de $G_{X \rightarrow Y}$ y el discriminador D_Y 2.11

$$\begin{aligned} \mathcal{L}_{GAN}(G_{X \rightarrow Y}, D_Y, X, Y) = & \mathbb{E}_{x \sim P_{\text{data}}(x)} \left[\log \left(1 - D_Y(G_{X \rightarrow Y}(x)) \right) \right] \\ & + \mathbb{E}_{y \sim P_{\text{data}}(y)} \left[\log (D_Y(y)) \right] \end{aligned} \quad (2.11)$$

Función de mapeo para el paso de $G_{Y \rightarrow X}$ y el discriminador D_X 2.12

$$\begin{aligned}\mathcal{L}_{GAN}(\underset{Y \rightarrow X}{G}, D_X, Y, X) = & \mathbb{E}_{x \sim P_{\text{data}}(x)} \left[\log \left(1 - D_X(\underset{Y \rightarrow X}{G}(x)) \right) \right] \\ & + \mathbb{E}_{y \sim P_{\text{data}}(y)} \left[\log \left(D_X(y) \right) \right]\end{aligned}\quad (2.12)$$

Cada uno de estos generadores $\underset{X \rightarrow Y}{G}$ y $\underset{Y \rightarrow X}{G}$ transforman los datos de un dominio a otro, pasando por las siguientes tres fases, **codificación** usando capas convolucionales, **transformación** usando bloques *ResNet* y por último la **descodificación** usando capas convolucionales inversas y una última capa convolucional [66].

A estas dos métricas se le añade la pérdida del ciclo 2.13 esta fórmula se refiere a la consistencia de las transformaciones, si tomamos una instancia del dominio X y la pasamos al dominio Y , si volvemos a pasarla al dominio X debe ser consistente.

$$\begin{aligned}\mathcal{L}_{cyc}(\underset{X \rightarrow Y}{G}, \underset{Y \rightarrow X}{G}) = & \mathbb{E}_{x \sim P_{\text{data}}(x)} \left[\text{MAE} \left(\underset{Y \rightarrow X}{G}(\underset{X \rightarrow Y}{G}(x)) - x \right) \right] \\ & + \mathbb{E}_{y \sim P_{\text{data}}(y)} \left[\text{MAE} \left(\underset{X \rightarrow Y}{G}(\underset{Y \rightarrow X}{G}(y)) - y \right) \right]\end{aligned}\quad (2.13)$$

En este punto tenemos tres perdidas, a la que podemos añadir la pérdida de la identidad 2.14 para mejorar el resultado, esto significa que el generador para datos de un dominio debe dar resultados similares a su dominio, es decir $\underset{X \rightarrow Y}{G}(x) \approx \hat{x}$ y $\underset{Y \rightarrow X}{G}(y) \approx \hat{y}$, añadir esta pérdida ayuda a preservar las características generales del dominio de origen (colores y tinte en caso imágenes).

$$\begin{aligned}\mathcal{L}_{identity}(\underset{X \rightarrow Y}{G}, \underset{Y \rightarrow X}{G}) = & \mathbb{E}_{x \sim P_{\text{data}}(x)} \left[\text{MAE} \left(\underset{X \rightarrow Y}{G}(x) - x \right) \right] \\ & + \mathbb{E}_{y \sim P_{\text{data}}(y)} \left[\text{MAE} \left(\underset{Y \rightarrow X}{G}(y) - y \right) \right]\end{aligned}\quad (2.14)$$

- La pérdida adversarial de $\underset{X \rightarrow Y}{G}$ contra D_Y .
- La pérdida adversarial de $\underset{Y \rightarrow X}{G}$ contra D_X .
- La pérdida del ciclo para el generador $\underset{X \rightarrow Y}{G}$ y $\underset{Y \rightarrow X}{G}$.
- La pérdida de identidad $\underset{X \rightarrow Y}{G}(x) \approx \hat{x}$ y $\underset{Y \rightarrow X}{G}(y) \approx \hat{y}$

$$\begin{aligned}
 \mathcal{L}_{X \rightarrow Y, Y \rightarrow X}(G, D_X, D_Y) = & \mathcal{L}_{GAN} \left(G_{X \rightarrow Y}, D_Y, X, Y \right) \\
 & + \mathcal{L}_{GAN} \left(G_{Y \rightarrow X}, D_X, Y, X \right) \\
 & + \lambda \mathcal{L}_{cycle} \left(G_{X \rightarrow Y}, G_{Y \rightarrow X} \right) \\
 & + 0,5\lambda \mathcal{L}_{identity} \left(G_{X \rightarrow Y}, G_{Y \rightarrow X} \right)
 \end{aligned} \tag{2.15}$$

Esta ecuación 2.15 es una generalización de cómo funcionan las CycleGAN, en cada caso se deberá implementar de forma distinta. En la figura 2.31 podemos observar estos dos generadores y los dos discriminadores funcionando para cada uno de los ciclos de entrenamiento.

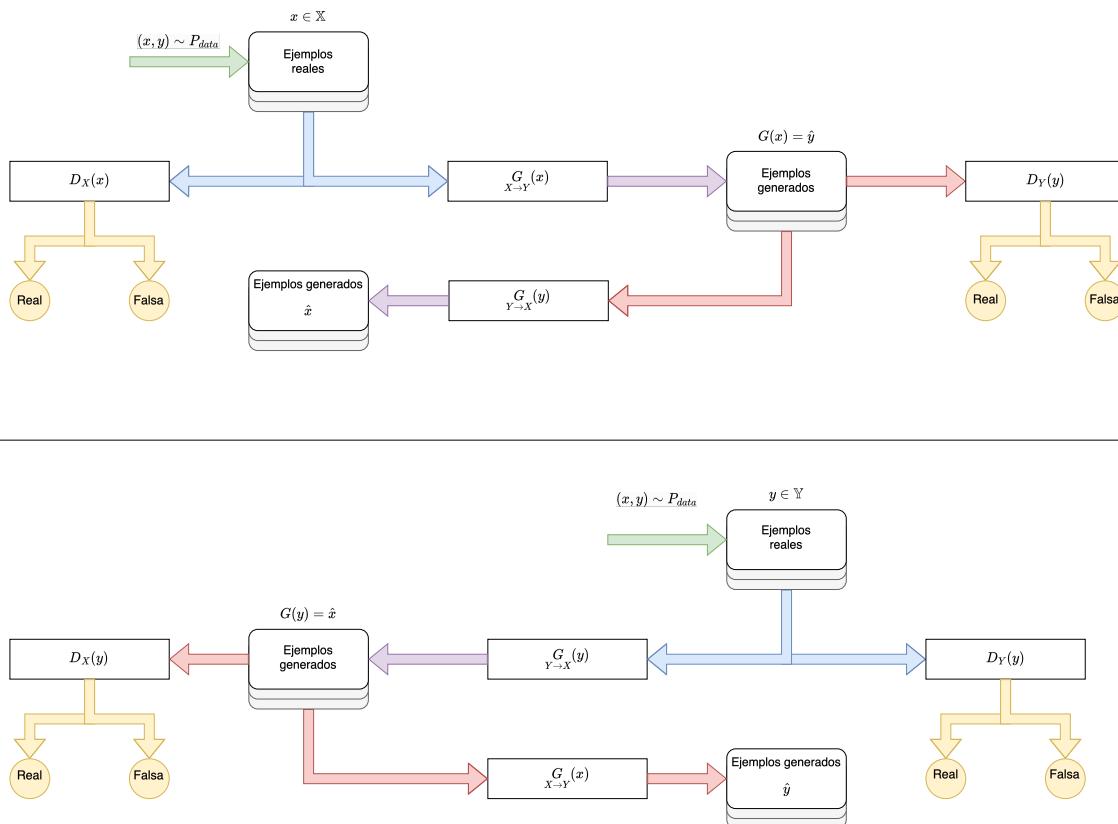


Figura 2.31: Arquitectura Cycle GAN (CycleGAN)
 Fuente: Elaboración propia.

Least Square Generative Adversarial Network (LSGAN) [2016]

Las redes Least Square GAN (LSGAN) [6] o redes generativas de mínimos cuadrados, en estas arquitecturas se modifica la función de pérdida del discriminador, esto

mejora la calidad de las instancias generadas así como el proceso de aprendizaje.

A continuación en la ecuación 2.16 se muestra las funciones objetivo para LSGAN, debemos tener en cuenta que a son las etiquetas para datos falsos, b son datos reales y c indica el valor que la red G envía a la red D para que los valide como verdaderos.

$$\begin{aligned} \min_D &= \frac{1}{2} \mathbb{E}_{x \sim P_{\text{data}}(x)} [(D(x) - b)^2] + \frac{1}{2} \mathbb{E}_{z \sim P_{\text{noise}}(z)} [(D(G(z)) - a)^2] \\ \min_G &= \frac{1}{2} \mathbb{E}_{z \sim P_{\text{noise}}(z)} [(D(G(z)) - c)^2] \end{aligned} \quad (2.16)$$

En la Figura 2.32b podemos observar el límite de decisión de una GAN clásica, frente a la Figura 2.32c que representa el límite de decisión de una LSGAN

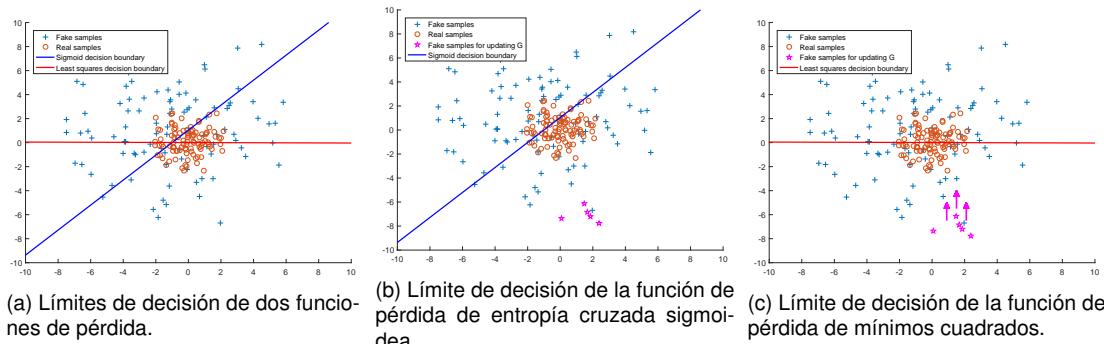


Figura 2.32: Least Square Generative Adversarial Network (LSGAN)

Fuente: Least Squares Generative Adversarial Networks [6]

Progressive Generative Adversarial Network (ProGAN) [2017]

Las redes **Progressive GAN (ProGAN)** o **Progressive Growing GAN (PGGAN)** [67] están basadas en las redes **DCGAN** donde tanto G como D empiezan con un entrenamiento de imágenes en baja resolución y va aumentando progresivamente la resolución, la idea es hacer que los generadores como los discriminadores vayan aprendiendo progresivamente.

Primero las redes aprenden imágenes de baja resolución 4×4 , después 8×8 , 16×16 , hasta llegar a la resolución, 1024×1024 como vemos en el ejemplo de la Figura 2.33a

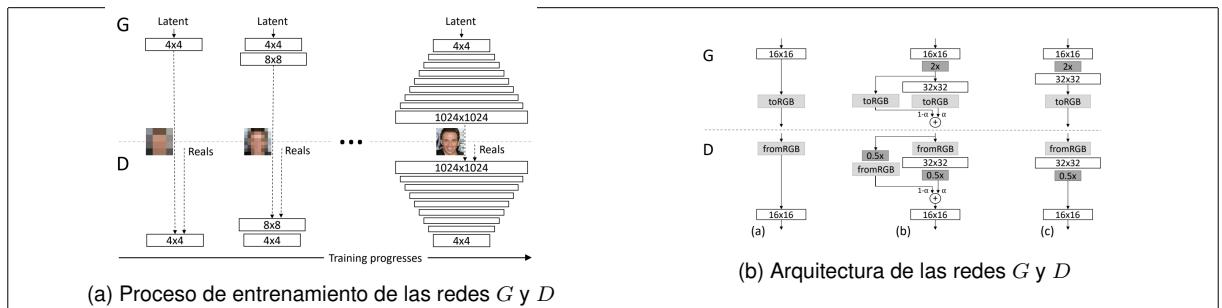


Figura 2.33: Progressive Generative Adversarial Network (ProGAN)

Fuente: [ProGAN: Progressive Growing Generative Adversarial Networks](#)

Super Resolution Generative Adversarial Network (SRGAN) [2016]

Las redes [Super Resolution GAN \(SRGAN\)](#) [68] son una variante de las redes [DCGAN](#). Esta red aprende a mapear imágenes de baja resolución a imágenes de alta resolución, utiliza la calidad perceptual en lugar de solo el [Peak Signal-to-Noise Ratio \(PSNR\)](#), estas redes son las que mejores resultados están dando, además de que las abstracciones que aprende no se limitan únicamente al dominio de su entrenamiento sino a la abstracción de formas y patrones en imágenes, estos modelos son sencillos de entrenar y dan buenos resultados.

$$\begin{aligned} \min_{\theta_G} \max_{\omega_D} & \mathbb{E}_{I^{HR} \sim P_{train}(I^{HR})} \left[\log (D_{\omega_D}(I^{HR})) \right] \\ & + \mathbb{E}_{I^{LR} \sim P_G(I^{LR})} \left[\log (1 - D_{\omega_D}(G_{\theta_G}(I^{LR}))) \right] \end{aligned} \quad (2.17)$$

Enhanced Super Resolution Generative Adversarial Network (ESRGAN) [2016]

Las redes [Enhanced Super Resolution GAN \(ESRGAN\)](#) [69, 68] son una mejora de las [SRGAN](#), ESRGAN utiliza características antes de la activación para la pérdida perceptual. Esto mejora la consistencia de brillo y la recuperación de texturas. Además, añaden la interpolación de redes para equilibrar la calidad visual y el [PSNR](#).

Los investigadores *Xintao Wang, Ke Yu, Shixiang Wu, ...*[69] demostraron que las redes [ESRGAN](#) cuentan con un rendimiento superior a [SRGAN](#) con un entrenamiento sencillo, contrario a lo que creían los investigadores *Christian Ledig, Lucas Theis, ...*[68] modelos más profundos son más difíciles de entrenar.

Style Generative Adversarial Network (StyleGAN) [2018]

Las redes [StyleGAN](#) [70, 71], es un proyecto desarrollando por NVIDIA, el principal uso de la red es la generación de rostros, actualmente se encuentra en su tercera versión [72], esta versión cuenta con importantes mejoras de rendimiento, estabilidad, interpolación, entrelazamiento y reducción del *aliasing*. Además, la tercera versión cuenta con la posibilidad de generación de vídeo.

La arquitectura [StyleGAN](#) [73], está basada en las redes [PGGAN](#) que permiten el entrenamiento progresivo, aumentando gradualmente la resolución de las instancias, se introduce la mezcla de estilos de forma distinta a [CycleGAN](#), ya que hace un mezclado de los vectores latentes, también añade métodos de inversión con el objetivo de reconstruir vectores latentes de determinadas instancias para manipular editar las instancias en el espacio latente.

Self Attention Generative Adversarial Network (SAGAN) [2018]

En las redes [Self-Attention GAN \(SAGAN\)](#) [74] permiten modelar dependencias de largo alcance usando técnicas de atención para tareas de generación de imágenes. Estas redes permiten generar más detalles usando pistas del mapa de características, la red D puede comprobar que los rasgos detallados de las instancias generadas son coherentes entre sí.

Wasserstein Generative Adversarial Network (WGAN) [2017]

Las redes [Wasserstein GAN \(WGAN\)](#) [75, 76, 77] se basan en la arquitectura clásica de una red [GAN](#), pero modifican el discriminador con la intención de que los modelos converjan, esta arquitectura utiliza la distancia de Wasserstein¹ como función de perdida, esto busca mejorar la estabilidad del entrenamiento y la calidad de las instancias generadas, además evita problemas de colapso de modo².

¹La distancia de Wasserstein mide cuánto “movimiento” se necesita para transformar una distribución en otra. En términos simples, es como medir la cantidad de trabajo necesario para mover tierra de un lugar a otro.

²Durante el entrenamiento, un generador puede “colapsar” a una configuración en la que siempre produce las mismas salidas.

Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) [2017]

Las redes [Wasserstein GAN Gradient Penalty \(WGAN-GP\)](#) [78] basadas en la arquitectura de las redes [WGAN](#), añaden un factor de penalización, mejorando considerablemente la estabilidad del modelo.

Al agregar esta penalización de gradiente, mejora la suavidad en la función de pérdida lo que ayuda a la estabilidad, la penalización, también se recortan los pesos esto ayuda a mantener controlado la continuidad de *Lipschitz* en el discriminador con lo que el modelo converge a una solución más realista.

Information Maximizing Generative Adversarial Network (InfoGAN) [2016]

Las redes [Information Maximizing GAN \(InfoGAN\)](#) [79], introduce un discriminador adicional llamado Q o red de reconocimiento para controlar las características de las instancias, es la responsable de estimar los códigos latentes a partir de las instancias generadas. La función objetivo de las redes [InfoGAN](#) es la regularización de la función objetivo de las [cGAN](#). Esto permite controlar la red neuronal de forma que la podamos condicionar a que genere instancias de un dominio y que las instancias generadas contengan ciertas características.

La arquitectura lo logra usando la red Q con dos estructuras, una para controlar las variables discretas y otra para las variables continuas del ruido latente, estas variables se utilizan para el cálculo de la perdida. Con esto logra controlar que las instancias generadas sean válidas y coherentes con las clases y características.

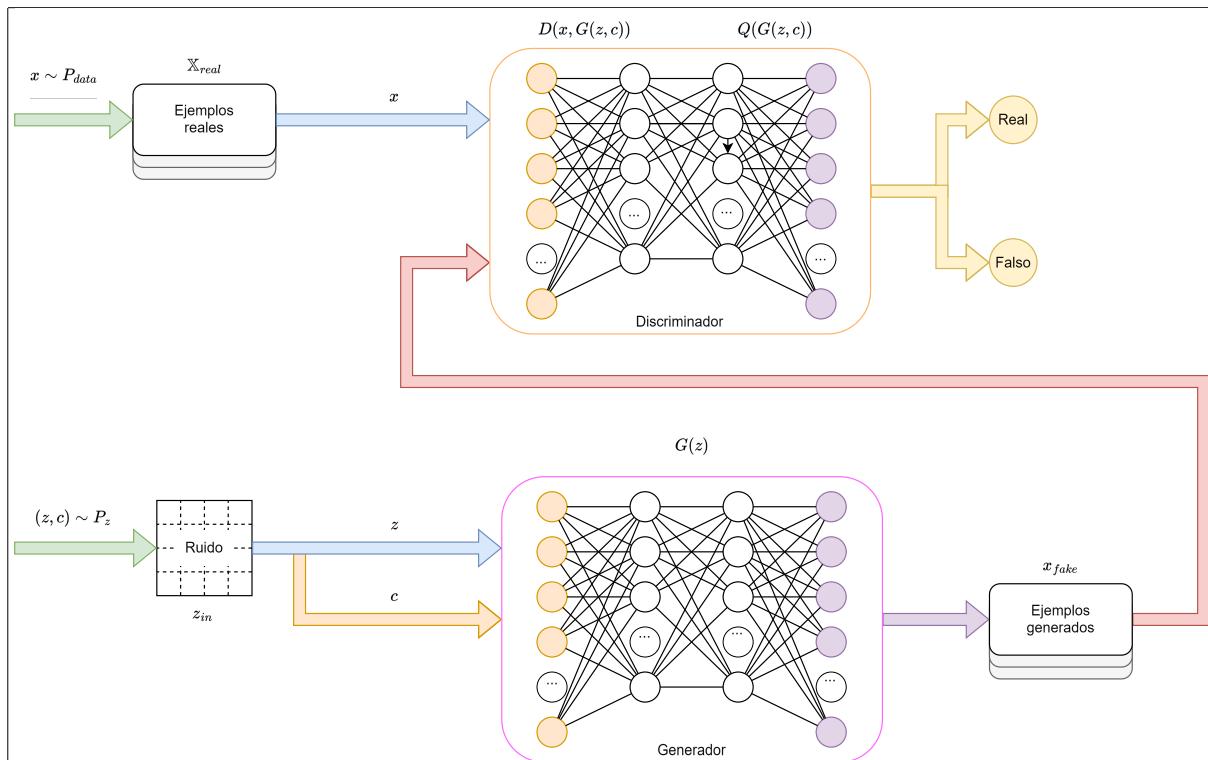


Figura 2.34: Arquitectura Information Maximizing GAN (InfoGAN).
Fuente: Elaboración propia.

La arquitectura de **InfoGAN** diferencia entre las redes G , D y Q , aunque D y Q comparten la misma arquitectura de red, solo se diferencia en las unidades de salida en la última capa, esto hace que el coste de entrenamiento del modelo sea levemente superior a una red **GAN** clásica, pero con importantes mejoras.

$$\begin{aligned} \min_{G, Q} \max_D V_{InfoGAN}(D, G, Q) &= V(G, D) - \lambda I(G, Q) \\ \min_G \max_D V_{InfoGAN}(D, G) &= V(G, D) - \lambda I(c; G(z, c)) \end{aligned} \quad (2.18)$$

$I(c; x)$ es la información mutua y mide cuanto se sabe de c si conocemos x , es decir si la instancia de x es completamente irrelevante a c , entonces el discriminador indicara $I(c; x) = 0$.

$$\begin{aligned} I(c; G(z, c)) &= H(c) - H(c | G(z, c)) \\ &= H(c) - \mathbb{E}_{x \sim G(z, c)} \left[\mathbb{E}_{c' \sim P_{c \sim x}} [\log P(c' | x)] \right] \end{aligned} \quad (2.19)$$

En el artículo *Interpreting the Latent Space of GANs for Semantic Face Editing*

[80] se propone un método para la interpretación semántica codificada en el espacio latente, esto permite controlar las características de las instancias generadas.

2.3.7.3. Inversión de GANs

Como se especifica en el artículo [7] el generador aprende patrones en el espacio de distribución. Este espacio N – dimensional es el código latente con las características del dominio, se le denomina z . La inversión de **GAN** proporciona una relación del espacio latente para obtener el código latente del dominio.

El objetivo de la inversión **GAN** es el de obtener el vector latente para cualquier instancia determinada. La obtención de este vector latente proporciona flexibilidad para realizar manipulación de instancias reales en vez de solo poder manipular instancias generadas.

La **GAN** Inversión tiene múltiples aplicaciones, destaca en la manipulación, generación, restauración e interpolación de instancias. Así como en la reconstrucción 3D, comprensión de las instancias, aprendizaje multimodal o imágenes médicas [7].

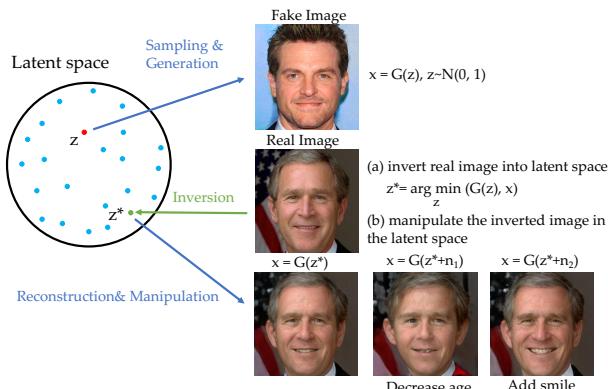


Figura 2.35: Ilustración de la inversión de GAN.
Fuente: GAN Inversion: A Survey [7]

Este problema está definido formalmente 2.20 como una función objetivo.

$$z^* = \arg \min_z \ell(G(z), x), \quad (2.20)$$

A continuación, en la Figura 2.36 se presentan el flujo para la extracción del vector del código latente de los distintos métodos que han propuesto en el artículo [7].

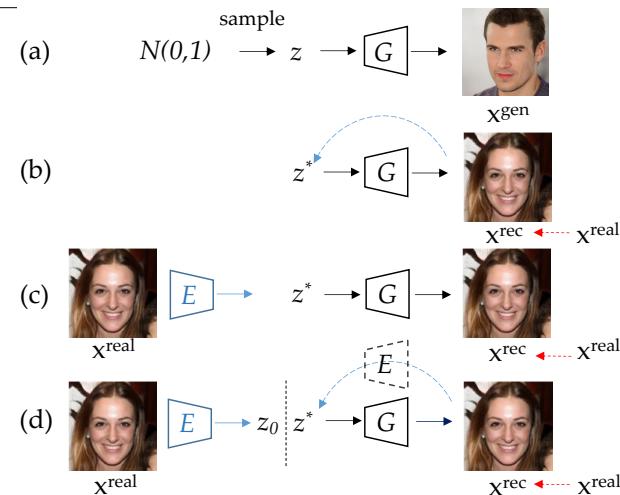


Figura 2.36: Tipos de GAN inversión.

Fuente: GAN Inversion: A Survey [7]

En la sección **a** de la Figura 2.36 podemos observar una **GAN** clásica sin ningún método de inversión.

En la sección **b** de la Figura 2.36 podemos ya ver el método *Optimization-based* que optimiza el vector latente z para reconstruir una imagen x^{rec} cercana a la instancia real x . La función objetivo es la definida en la ecuación 2.21.

$$z^* = \arg \min_z \ell(x, G(z; \theta)) \quad (2.21)$$

En la sección **c** de la Figura 2.36 podemos observar el método *Learning-based* que incorpora un módulo codificador llamado E que se entrena utilizando instancias generadas por G con sus vectores latentes. Es decir, el codificador E intenta generar un vector latente denominado z_n para la instancia x_n tal que cuando el vector latente se pase a la red G . La función objetivo es la definida en la ecuación 2.22.

$$\theta_E^* = \arg \min_{\theta_E} \sum_n \mathcal{L}(G(E(x_n; \theta_E)), x_n) \quad (2.22)$$

En la sección **d** de la Figura 2.36 podemos observar el método *Hybrid-based* que incorpora los enfoques *Optimization* y *Learning*. Durante la inferencia, la instancia real x se pasa al codificador E con el que se obtiene z , este valor será la inicialización z_0 del vector latente de optimización para reducir la distancia entre la instancia real x y la instancia reconstruida x^{rec} .

Aplicaciones de la inversión de GANs

Algunos investigadores han estudiado la inversión de GANs para diseccionar GANs y comprender sus abstracciones internas, esto también sirve para entender qué características las GANs no aprenden.

Interpolación de instancias: Una posibilidad es usar una interpolación lineal en el espacio latente, con ello podemos fusionar dos vectores latentes en el espacio latente, esto logra una transición suave

$$z = z_1(\lambda) + z_2(1 - \lambda) \text{ donde } \lambda \in (0, 1) \quad (2.23)$$

Nos referimos como z al vector latente de la interpolación de las dos instancias z_1 y z_2 y α es el factor de paso

Manipulación de instancias: Mediante una transformación lineal en el espacio latente, se puede lograr una manipulación de las instancias, esto se logra transformando el código latente de la imagen en direcciones semánticas específicas.

$$\begin{aligned} x' &= G(z') \\ z' &= z + \alpha n \end{aligned} \quad (2.24)$$

Nos referimos como z' y x' al código latente manipulado de la instancia, α es el factor de paso y n es la dirección semántica específica del espacio latente.

Difusión semántica: De forma similar a *Context Encoder* se hace un recorte de la instancia para fusionar junto con otra instancia recortada y difuminar la fusión de ambas instancias, el objetivo es que la instancia generada conserve las características de la imagen destino.

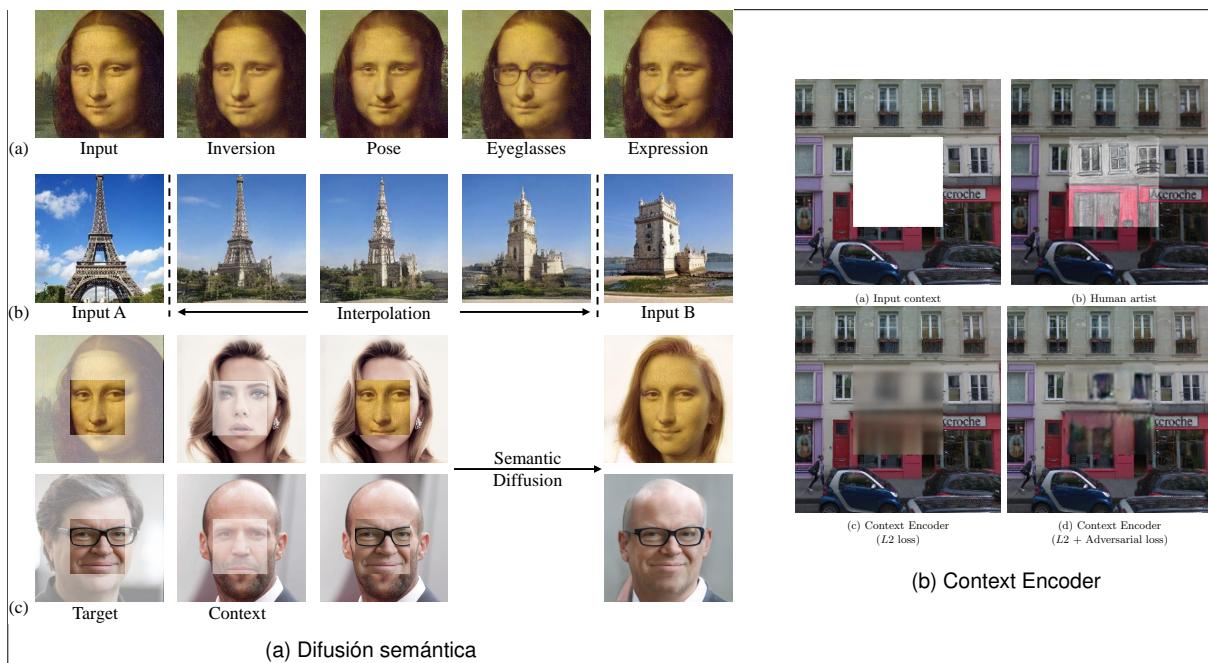


Figura 2.37: Diferencias entre Difusión semántica y Context Encoder

2.3.8. La seguridad de las redes neuronales

El estado del arte de la seguridad informática es muy complejo, ya que los delincuentes informáticos adoptan nuevas técnicas y tácticas para eludir las medidas de seguridad, esto hace que las amenazas informáticas estén en constante evolución, también amenaza a las tecnologías que emplean la inteligencia artificial y redes neuronales.

A continuación, se explicarán las distintas amenazas que tiene una red neuronal y como puede ser atacada o implementar defensas, posteriormente analizaremos el estado del arte de la inteligencia artificial que se emplea para mejorar la seguridad informática de los dispositivos y de los usuarios.

La librería ART [81], es una librería de Python especializada en la seguridad del ML, ART proporciona herramientas que permiten a los desarrolladores e investigadores evaluar modelos de aprendizaje automático, la librería se especializa en amenazas de evasión, encantamiento, extracción e inferencia. La librería consta con 6 módulos específicos para ataque, defensas, estimadores, evaluaciones, métricas y preprocesamiento. Esto podemos verlo en la Figura 2.38.

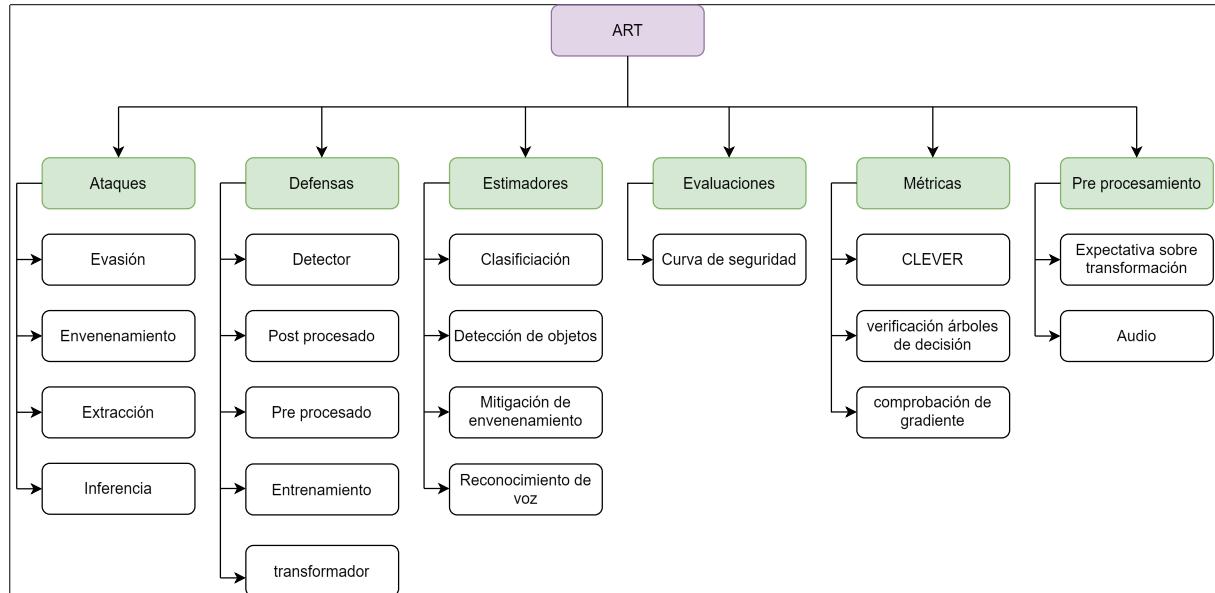


Figura 2.38: Arquitectura de ART.

Fuente: [GitHub @Trusted-AI/adversarial-robustness-toolbox](https://github.com/Trusted-AI/adversarial-robustness-toolbox)

Los modelos profundos ofrecen mejoras notables, aunque sin las consideraciones de seguridad pueden ser que nuestros modelos se vean amenazados por la injerencia de terceros

Como podemos observar en la Figura 2.39 existen multitud de amenazas, existen cuatro formas de clasificar las amenazas, esta clasificación es en función de cómo se altera el comportamiento del modelo.

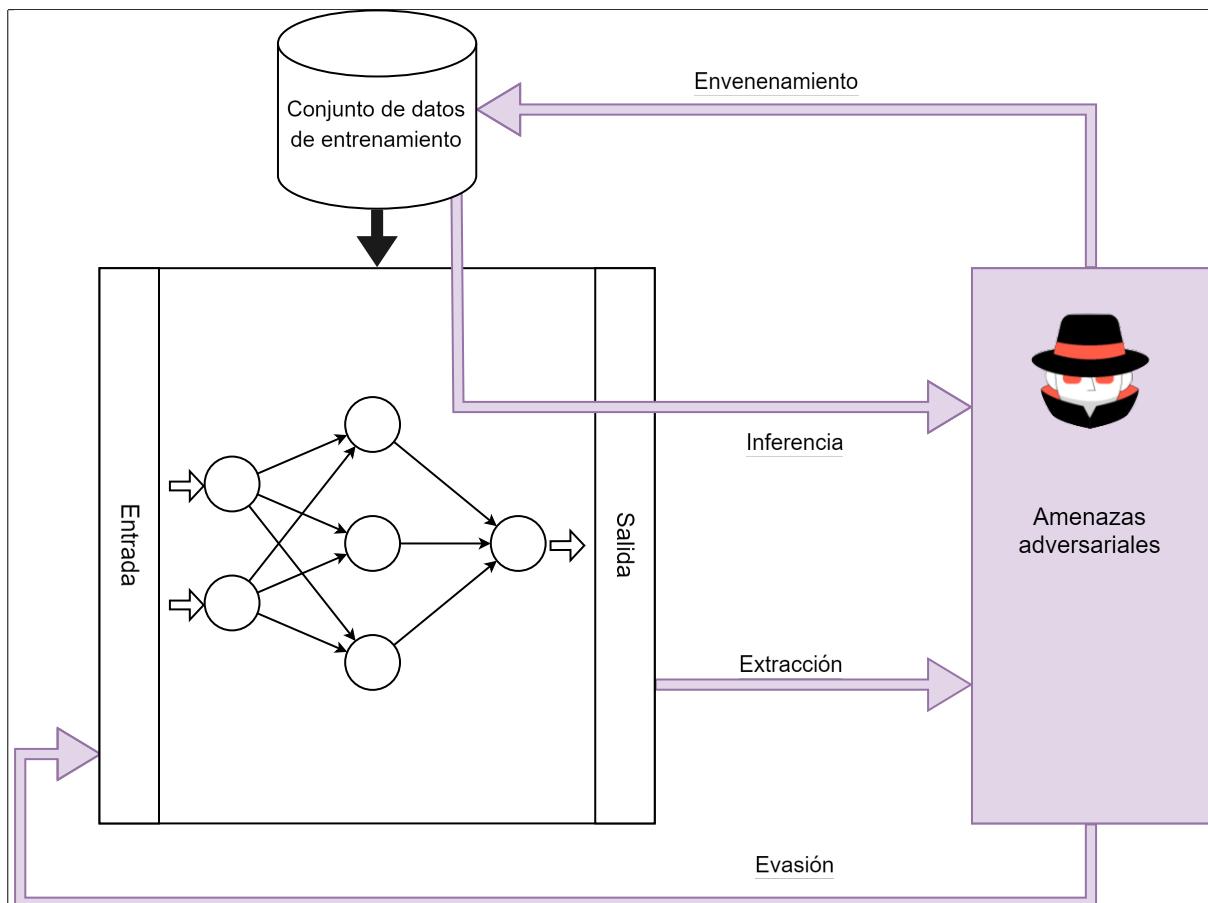


Figura 2.39: Amenazas en redes neuronales.
Fuente: Elaboración propia.

2.3.9. Ataques en *deep learning*

Los modelos de inteligencia artificial presentan una serie de ventajas muy interesantes, sobre todo para un atacante, ya que si se conoce el modelo que emplea puede ser viable la perturbación del modelo con el objetivo de alterar el comportamiento o los resultados.

A continuación se muestra los posibles vectores de ataque que se pueden ejecutar sobre los modelos, se trata de una recopilación de todos los artículos científicos más relevantes del momento.

2.3.9.1. Evasión

Busca explotar las vulnerabilidades de la red neuronal para inducir errores en la capacidad de clasificar o de predecir. Se envía información con perturbaciones mínimas que alteren notablemente la respuesta del modelo.

2.3.9.1.1 White Box

Se conoce todo sobre el modelo, arquitectura, pesos, umbrales, formato de entrada de datos y salida, etc. [82]

Uno de los primeros ataques de evasión que se publicaron fue el de [Fast Gradient Method \(FGM\)](#) de *Ian J. Goodfellow* en 2014 [83]

Tabla. 2.2: Lista de ataques de caja blanca

Título	Referencias
Auto Projected Gradient Descent (Auto-PGD)	(Croce and Hein, 2020)
Shadow Attack	(Ghiasi et al., 2020)
Wasserstein Attack	(Wong et al., 2020)
PE Malware Attacks	(Suciu et al., 2018), (Demetrio et al., 2020), (Demetrio et al., 2019)
Imperceptible, Robust, and Targeted Adversarial Examples for Automatic Speech Recognition	(Qin et al., 2019)
Brendel & Bethge Attack	(Brendel et al., 2019)
Targeted Universal Adversarial Perturbations	(Hirano and Takemoto, 2019)
Audio Adversarial Examples: Targeted Attacks on Speech-to-Text	(Carlini and Wagner, 2018)
High Confidence Low Uncertainty (HCLU) Attack	(Grosse et al., 2018)
Iterative Frame Saliency	(Inkawhich et al., 2018)
DPatch	(Liu et al., 2018)
Robust DPatch	(Liu et al., 2018), (Lee and Kolter, 2019)
ShapeShifter	(Chen et al., 2018)
Projected Gradient Descent (PGD)	(Madry et al., 2017)
NewtonFool	(Jang et al., 2017)
Elastic Net	(Chen et al., 2017)
Adversarial Patch	(Brown et al., 2017)
Decision Tree Attack	(Papernot et al., 2016)
Carlini & Wagner (C&W) ℓ_2 and ℓ_∞ attack	(Carlini and Wagner, 2016)

Continuado en la próxima página

Tabla. 2.2: Lista de ataques de caja blanca (Continuado)

Basic Iterative Method (BIM)	(Kurakin et al., 2016)
Jacobian Saliency Map	(Papernot et al., 2016)
Universal Perturbation	(Moosavi-Dezfooli et al., 2016)
Feature Adversaries	(Sabour et al., 2016)
DeepFool	(Moosavi-Dezfooli et al., 2015)
Virtual Adversarial Method	(Miyato et al., 2015)
Fast Gradient Method	(Goodfellow et al., 2014)

2.3.9.1.2 Black Box

Se desconoce el modelo, arquitectura, pesos, umbrales, pero se conoce el formato de entrada de datos y la respuesta del modelo. [82]

Título	Referencias
Square Attack	(Andriushchenko et al., 2020)
HopSkipJump Attack	(Chen et al., 2019)
Threshold Attack	(Vargas et al., 2019)
Pixel Attack	(Vargas et al., 2019), (Su et al., 2019)
Simple Black-box Adversarial (SimBA)	(Guo et al., 2019)
Spatial Transformation	(Engstrom et al., 2017)
Query-efficient Black-box	(Ilyas et al., 2017)
Zeroth Order Optimisation (ZOO)	(Chen et al., 2017)
Decision-based/Boundary Attack	(Brendel et al., 2018)
Geometric Decision-based Attack (GeoDA)	(Rahmati et al., 2020)

Tabla. 2.3: Lista de ataques de caja negra

2.3.9.2. Envenenamiento

Los atacantes intentan manipular los datos de entrenamiento con el objetivo de influir en el resultado del aprendizaje del modelo, esto pueden hacerlo desde distintas etapas.

Estos ataques inyectan datos de entrenamiento especialmente diseñados que au-

mentan el error. Los ataques de envenenamiento se centran en el hecho de que la mayoría de los algoritmos de aprendizaje asumen que sus datos tienen una distribución natural, esto puede no cumplirse en entornos que no cumplan las medidas de seguridad. [84]

Existen los ataques de puerta trasera, es decir, modelos profundos que tienen un comportamiento normal, pero que dado un dato o una secuencia de entrada dispara un comportamiento malicioso, esto puede hacer que el modelo tenga un comportamiento normal en la mayoría de casos, pero que dada determinada secuencia puede deteriorar enormemente el rendimiento del modelo. [85]

Estos ataques de puerta trasera logran muy buenos resultados sesgando el conjunto de datos de forma muy limitada, consiguen una tasa de error del 92 % con una tasa de envenenamiento de solo el 5 % [86].

En el artículo [87] trata de como evitar la detección de los ataques de puerta trasera. En el artículo se diseña un algoritmo de entrenamiento adversario adaptativo que optimiza la función de pérdida original del modelo y también maximiza la indistinguibilidad de las representaciones ocultas de datos envenenados y datos limpios.

En la sección Defensas en Deep Learning 2.3.10 entraremos en detalle de como nos podemos defender a estos ataques, es decir, detectar los ataques de puerta trasera.

El artículo [88] de 2019 tratan de ocultar el disparador de puerta trasera. En propuestas previas, los datos envenenados o los disparadores se podían identificar mediante una inspección visual, en el artículo proponen una forma novedosa de ataque de puerta trasera donde los datos envenenados parecen naturales con etiquetas correctas y el disparador está oculto en los datos envenenados, con esto logra mantener el ataque oculto hasta el momento de la prueba.

Los autores del artículo [89] en 2020 proponen un ataque de envenenamiento de etiqueta limpia, escalable y transferible contra aprendizaje por transferencia, en el artículo crean imágenes envenenadas con su centro cerca de la imagen objetiva en el espacio de características. Cuentan con mejoras de rendimiento de un factor de por 12 y un 26.75 % en el aprendizaje de transferencia.

En artículos previos se analizan ataques a modelos predicción clásicos, en el artículo [90] se analizan ataques de envenenamiento a **GAN** usando **Deep Generative Models (DGMs)** corruptos que generan datos regulares, pero dado determinado patrón o distribución de activación generará datos objetivos. En el artículo analizan estos ataques para **GAN** y **VAE**. Estos ataques fusionan ataques sigilosos y de fidelidad.

En el artículo [91] de 2018 muestran un método basado en optimización que genera instancias envenenadas, demuestran que con solo una instancia envenenada pueden sesgar el clasificador si se usa el aprendizaje por transferencia.

Por ejemplo, un atacante podría agregar una imagen aparentemente inofensiva (que esté correctamente etiquetada) a un conjunto de entrenamiento para un motor de reconocimiento facial y controlar la identidad de una persona elegida en el momento de la prueba.

El mismo artículo describe un ataque sencillo y efectivo para seguir un clasificador, para ello usan múltiples imágenes envenenadas, aproximadamente unas 50 instancias envenenadas para un entrenamiento.

En los artículos previos se demuestran que para determinados conjuntos de datos pequeños es posible realizar ataques de envenenamiento de distintas formas tanto antes como después del entrenamiento, pero una de las limitaciones de esos ataques es el tamaño de los conjuntos de datos, ya que para modelos con conjuntos de datos muy grandes el coste computacional sería enorme.

En el artículo [92] de 2020 abordan como solventar el problema del coste computacional de ataques de envenenamiento, el mecanismo central del nuevo ataque es igualar la dirección del gradiente de los ejemplos maliciosos. Este fue el primer artículo que demostraba que los ataques de envenenamiento pueden funcionar, lo demostró usando el conjunto de datos *ImageNet* envenenado, además demostró limitaciones en estrategias defensivas. El artículo concluye que los ataques de envenenamiento es una amenaza real en sistemas reales a gran escala.

El artículo [93] de 2022 da un gran avance en ataques a modelos de detección de objetos, en trabajos previos se analizan ataques de envenenamiento a modelos de clasificación o de generación. Estos ataques son muy peligrosos, ya que la conducción autónoma actual que usa tecnología artificial de detección de objeto puede ser vulnerable, el artículo propone cuatro tipos de ataques de puerta trasera, en el mismo artículo se desarrollan métricas apropiadas para evaluar cada tipo de ataque.

- **Object Generation Attack (OGA):** ataque de generación de objetos, un disparador puede generar falsamente un objeto de la clase objetivo.
- **Regional Misclassification Attack (RMA):** ataque de clasificación errónea regional, un disparador puede cambiar la predicción de un objeto circundante a la clase objetivo.

- **Global Misclassification Attack (GMA)**: ataque de clasificación errónea global, un único desencadenante puede cambiar las predicciones de todos los objetos en una imagen a la clase objetiva.
- **Object Disappearance Attack (ODA)**: ataque de desaparición de objetos, un disparador puede hacer que el detector no detecte el objeto de la clase objetivo.

Realizaron experimentos con dos modelos típicos de detección de objetos, Faster-RCNN [94] y YOLOv3 [95] con distintos conjuntos de datos. Demuestran que un ajuste correcto no puede eliminar la puerta trasera oculta. Por último, proponen un método de detección basado en entropía y que es en tiempo de ejecución.

2.3.9.3. Extracción

En este tipo de ataques se busca extraer un funcionamiento interno equivalente de la red neuronal, se puede considerar un ataque de ingeniería inversa.

En el artículo [96] se clasifican los ataques de extracción de modelos en torno a dos objetivos, **precisión**, es decir, un buen rendimiento en la tarea que fue diseñada y **fidelidad**, es decir, que las respuestas sean consistentes con el modelo víctima, es decir los más similares, incluso cuando se le presentan datos no vistos previamente.

Realizan un ataque de extracción basado en el aprendizaje supervisando el entrenamiento del modelo extraído. Explican las limitaciones de estos ataques que impiden que los modelos extraídos sean de alta fidelidad.

Demuestran que los ataques de extracción son viables para distintos tamaños de conjuntos de datos.

En el artículo [8] continúa con las investigaciones previas de que las **CNN** son vulnerables a ataques de ejemplos adversarios. Demostraron que se puede crear una **CNN** imitadora consultando un modelo remoto como una caja negra con datos aleatorios no etiquetados extrayendo sus etiquetas del modelo víctima con imágenes aleatorias. Posteriormente, con las imágenes aleatorias y las etiquetas extraídas pueden imitar el comportamiento de los pesos del modelo víctima.

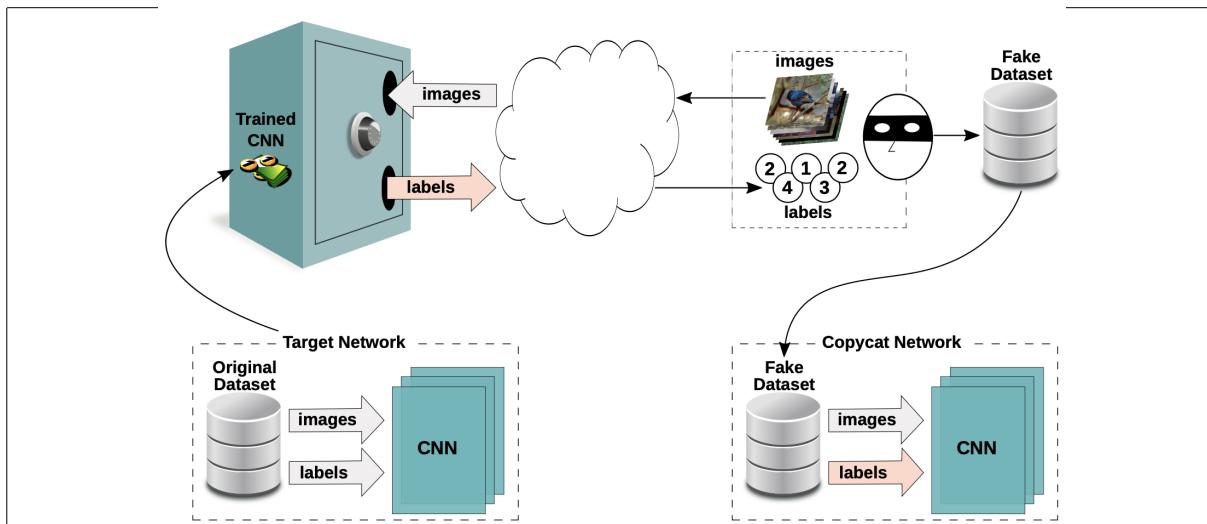


Figura 2.40: Ataque que realiza CopyCat CNN.

Fuente: CopyCat CNN [8]

En el artículo también demostraron que se podía extraer al menos el 93.7 % del rendimiento de los modelos víctima con datos de dominio no problemáticos y al menos el 98.6 % del rendimiento si se usan datos del dominio del problema. Además, se logró extraer el 97.3 % del modelo de *Microsoft Azure Emotion* usando su API.

2.3.9.4. Inferencia

En el artículo [97] se analizan los ataques de inferencia usando árboles de decisión y un conjunto de datos con información médica de unos individuos, estos ataques permiten extraer información sensible o privada sobre los datos de entrenamiento o sobre las instancias (individuos) a partir de un modelo dé [ML](#).

En el artículo se menciona los ataques de inversión de modelo, dicho ataque implica que un atacante abusa del acceso al modelo para extraer información sensible, en concreto habla sobre extraer información genómica sensible de pacientes.

Además, también menciona que se puede extraer los valores de confianza revelados junto con las predicciones, es decir, el nivel de confianza de las predicciones, menciona que se puede usar este dato para inferir información sensible sobre los individuos.

En el artículo [98] se mencionan los ataques de inferencia de membresía, estos ataques permiten extraer si una instancia se utilizó para entrenar el modelo. Esto es porque explotan la confianza anormal de los modelos cuando se utilizan instancias del entrenamiento en el modelo entrando. Estos ataques no se pueden usar si el atacante

solo obtiene información de la etiqueta y no de la medida de confianza.

En el artículo presentan ataques de membresía usando únicamente etiquetas sin los valores de confianza que estudios anteriores si necesitaban. Esto lo logran evaluando las etiquetas predichas por el modelo en función de determinadas perturbaciones de una instancia.

Ciertas defensas contra estos ataques es la ocultación o alteración del nivel de confianza, en el artículo señalan que los ataques de inferencia de membresía por etiqueta anulan las defensas. Uno de los descubrimientos más relevantes de dicha investigación es que las únicas estrategias de defensas que previenen con éxito todos los ataques son los modelos de entrenamiento con privacidad diferencial y regularización L_2 .

2.3.10. Defensas en *deep learning*

Se han hecho grandes avances en el estudio de los modelos de inteligencia artificial, se han investigado los posibles ataques y también como defender los modelos frente a alteraciones externas o alteraciones internas.

En las siguientes subsecciones reduciremos la explicación a los artículos más relevantes que se pueden aprovechar en este proyecto.

2.3.10.1. Pre procesamiento

Estas son algunas de las técnicas de defensa que se pueden aplicar a los modelos antes de su inserción o inferencia.

En el artículo [99] se presenta la técnica *InverseGAN*, con el objetivo de resolver el problema de inferencia en redes *GAN*, creando una red de codificación, aunque no consiguen en la práctica la inversión del código latente si lo logran en un estudio teórico, además demuestran que *InverseGAN* supera a otros métodos de defensa.

En el artículo *DefenseGAN* [100] se presenta una técnica para defender a los modelos de clasificación, en esta técnica se modela una distribución de imágenes sin perturbaciones y en la inferencia encuentran una versión cercana de la imagen sin cambios adversariales, esta imagen es la que se introduce en el clasificador.

Como los métodos anteriores existen muchos más, aquí se presenta una tabla con

los artículos.

Tabla. 2.4: Lista de técnicas de defensa durante el pre procesamiento en Deep Learning

Técnica	Referencias
InverseGAN	An Lin et al., 2019
DefenseGAN	Samangouei et al., 2018
Video Compression	Carlini et al., 2019
Resampling	Yang et al., 2019
Thermometer Encoding	Buckman et al., 2018
MP3 Compression	Carlini, N. and Wagner, D., 2018
Total Variance Minimization	Guo et al., 2018
PixelDefend	Song et al., 2017
Gaussian Data Augmentation	Zantedeschi et al., 2017
Feature Squeezing	Xu et al., 2017
Spatial Smoothing	Xu et al., 2017
JPEG Compression	Dziugaite et al., 2016
Label Smoothing	Warde-Farley and Goodfellow, 2016
Virtual Adversarial Training	Miyato et al., 2015
Cutout	DeVries et al., 2017
Mixup	Zhang et al., 2017
CutMix	Yun et al., 2019

2.3.10.2. Post procesamiento

Las técnicas post procesamiento se implementan después de que el modelo ha realizado su predicción y están diseñadas para mejorar la robustez frente a ataques adversariales.

En estos artículos se plantean defensas frente a extracción de datos, en *Random Noise* [101] se propone un método híbrido para la defensa que combina técnicas clásicas, limitar consultas, monitoreo de patrones de uso, etc. A otros más específicos,

como introducir ruido en las respuestas o reducir los modelos para disminuir los datos sensibles que se pueden extraer, también recomiendan técnicas como *data-dependent randomization*.

Técnica	Referencias
Reverse Sigmoid	Lee et al., 2018
Random Noise	Chandrasekaran et al., 2018
Class Labels	Tramer et al., 2016, Chandrasekaran et al., 2018
High Confidence	Tramer et al., 2016
Rounding	Tramer et al., 2016

Tabla. 2.5: Lista de técnicas de defensa durante el post procesamiento en Deep Learning

2.3.10.3. Entrenamiento

En el proceso de entrenamiento es donde se realizan las técnicas más curiosas y variadas para defender a los modelos.

El artículo [\[102\]](#) presenta un método para certificar robustez en grandes redes neuronales, dentro de ciertos límites. Este método utiliza varias transformaciones para abstraer la información, equilibrando eficiencia y precisión.

Técnica	Referencias
General Adversarial Training	Szegedy et al., 2013
Madry's Protocol	Madry et al., 2017
Fast Is Better Than Free	Wong et al., 2020
Certified Adversarial Training	Mirman et al., 2018 [102]
Interval Bound Propagation	Gowal et al., 2018
DP-InstaHide	Borgnia et al., 2021

Tabla. 2.6: Lista completa de técnicas de defensa durante el entrenamiento en Deep Learning

2.3.11. Métricas en *deep learning*

Existen múltiples métricas para evaluar las redes [ANN](#), entre ellas destacan *Classification Accuracy*, *Logarithmic Loss*, *Confusion Matrix*, etc. Existen también las mé-

tricas para validar la seguridad de los modelos, estas son las que nos interesan a nosotros, métricas de robustez, certificación y verificación.

2.3.11.1. Robustez

En el artículo [103] se analiza el estado de las métricas de robustez y su poca aparición en publicaciones, es por ello que plantean un nuevo enfoque usando la estimación de la constante local de Lipschitz para un análisis de robustez, proponen el uso de la Teoría de Valores Extremos para una evaluación eficiente. Los investigadores la han definido como [Cross Lipschitz Extreme Value for nEwork Robustness \(CLEVER\)](#) independiente de los ataques y computacionalmente eficiente para redes neuronales grandes, dando buenos resultados como las pruebas de robustez indicada por las normas ℓ_2 e ℓ_∞ . Se considera CLEVER como la primera métrica de robustez independiente de ataques aplicable a cualquier red neuronal clasificadora.

2.3.11.2. Certificación

Randomized Smoothing [104] es la técnica que se describe en el artículo de 2019, se usa para mejorar la robustez de un clasificador, esta técnica analiza las perturbaciones adversas, en concreto en la norma ℓ_2 , demostrando una garantía estricta de robustez en norma ℓ_2 , esto significa que el clasificador seguirá funcionando correctamente incluso si se introducen perturbaciones en los datos de entrada dentro de unos límites. Los investigadores destacan resultados usando modelos de [ImageNet](#) con una precisión certificada del 49 %.

2.3.11.3. Verificación

En el artículo [105] de 2019 se aborda la verificación de la robustez en modelos basados en árboles, árboles de decisión, bosques aleatorios, [Gradient Boosted Decision Trees \(GBDT\)](#). Proponen un algoritmo de múltiples niveles, algoritmo eficiente que proporciona límites estrictos y permite mejoras iterativas, este algoritmo mejora de forma significativa la velocidad frente a enfoques previos basados en programación lineal.

2.3.12. Redes neuronales en *Red team* y *Blue team*

Un buen símil entre la seguridad informática clásica y la seguridad en redes neuronales son los conceptos de equipos de ataque y defensa (“*red team*” y “*blue team*”).

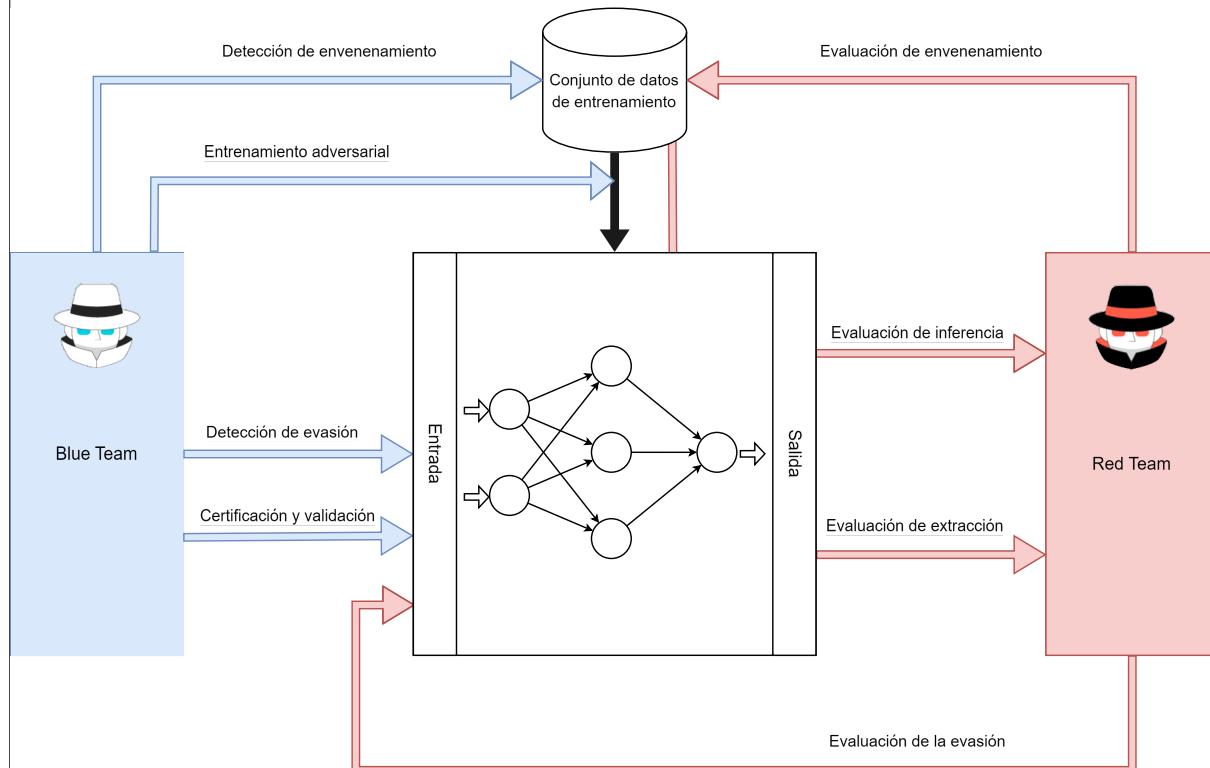


Figura 2.41: Ataques y defensas en redes neuronales.
Fuente: Elaboración propia.

La idea detrás del aprendizaje automático es la de poder predecir modelos predictivos, para ello se usan conjuntos de datos para el entrenamiento.

Los conjuntos de datos se han de procesar, ya que suelen contener mucho ruido, es decir, información muy poco valiosa, errónea o, por el contrario, una alta dimensionalidad de los datos que puede ser contraproducente por no poder reproducir esas medidas o por la poca información que aportan.

Los ataques al aprendizaje automático o profundo suelen estar dirigidos a las distintas etapas de creación y uso de un modelo.

- Alteración de los datos de entrada.
- Alteración del proceso de aprendizaje.
- Extracción de los datos de entrenamiento.
- Bloqueo del modelo.

Durante el transcurso del trabajo discutiremos los siguientes ataques y defensas de la sección [La seguridad de las redes neuronales](#) que se pueden ser vulnerables en los modelos.

- Envenenamiento de datos.
- Puerta trasera.
- Extracción de datos
- Ingeniería inversa.
- Evasión.

Podemos clasificar las amenazas de la seguridad informática con los siguientes sistemas, [STRIDE³](#), con la matriz [MITRE](#) o [ATLAS](#).

Spoofing	Tampering	Repudiation
Information disclosure	Denial of service	Elevation of privilege

Tabla. 2.7: Modelo para identificar amenazas a la seguridad informática

Reconnaissance	Resource Development	Initial Access	Execution
Persistence	Privilege Escalation	Defense Evasion	Collection
Discovery	Lateral Movement	Credential Access	Impact
Exfiltration	Command and Control		

Tabla. 2.8: Clasificación de Amenazas según el Marco MITRE - ATT & CK

Como hemos visto previamente en las tablas [2.7](#) y [2.8](#) las amenazas son ataques clásicos que pueden sufrir los sistemas informáticos, mientras que en la tabla [2.9](#) se modifican las amenazas de sistemas informáticos que implementen modelos de [IA](#)

Reconnaissance	Resource Development	Initial Access	ML Model Access
Execution	Persistence	Privilege Escalation	Defense Evasion
Credential Access	Discovery	Collection	ML Attack Staging
Exfiltration	Impact		

Tabla. 2.9: Clasificación de Amenazas según el Marco MITRE - ATLAS

³Modelado de amenazas de [microsoft](#)

MITRE - ATT & CK	MITRE - ATLAS	STRIDE
Reconnaissance	Reconnaissance	Spoofing
Resource Development	Resource Development	Tampering
Initial Access	Initial Access	Repudiation
Execution	ML Model Access	Information Disclosure
Persistence	Execution	Denial of Service
Privilege Escalation	Persistence	Elevation of Privilege
Defense Evasion	Privilege Escalation	
Credential Access	Defense Evasion	
Discovery	Credential Access	
Lateral Movement	Discovery	
Collection	Collection	
Command and Control	ML Attack Staging	
Exfiltration	Exfiltration	
Impact	Impact	

Figura 2.42: Amenazas MITRE - ATT & CK, MITRE - ATLAS y STRIDE.

Fuente: Elaboración propia. Inspirado en [MITRE - ATLAS](#)

2.4. Normativa y estándares

2.4.1. El Reglamento Europeo de Inteligencia Artificial

A partir de 2022 se creó el primer reglamento a nivel Europeo del uso de la Inteligencia artificial. El reglamento establece una jerarquía de riesgos en función del uso de la **IA** y sobre las categorías detectadas se establecen unas obligaciones DIGITAL [9].

Mediante un análisis de riesgos se han clasificado un conjunto de familias de sistemas de **IA** que pueden considerarse de alto riesgo respecto a un riesgo a la salud, seguridad o derechos fundamentales.

El reglamento detalla sistemas de identificación biométrica, infraestructuras críticas, selección y promoción de personal, usos en fronteras o usados por las Fuerzas y Cuerpos de Seguridad del Estado o la Administración de Justicia. También se indica que la lista no es fija y puede ser modificada por un acto delegado.

Los productos que ya cuenten con una normativa armonizada por la **UE** deberán ser sometidos a una evaluación de conformidad. También se definen una autoridad de supervisión de mercado como autoridades nacionales competentes.

REGLAMENTO EUROPEO INTELIGENCIA ARTIFICIAL



Figura 2.43: Reglamento Europeo
Fuente: Elaboración propia, inspirado en [9, 10]

El reglamento define el siguiente esquema para la certificación de productos, una autoridad notificante habilita a organismos de evaluación de conformidad para hacer las evaluaciones de conformidad en materia de **IA** a productos que quieran comercializar o poner en funcionamiento. Los productos con regulación deben ser sujetos a la supervisión designada.

Para el caso de sistemas de identificación biométrica utilizados por Fuerzas y Cuerpos de Seguridad del Estado, migración y administración de justicia, las autoridades de supervisión serán las encargadas de supervisar las actividades de seguridad, migración y asilo, en su defecto será la agencia de protección de datos.

El Comité orientará sobre el reglamento, elaborará guías y establecerá las reglas básicas para elaborar *sandboxes*. Los *sandboxes* permitirán desarrollar y probar sistemas de inteligencia artificial en un entorno regulado. Esto facilitará la elaboración de directrices y reglas más precisas y aplicables, asegurando que las innovaciones en in-

teligencia artificial se desplieguen de manera segura, ética y conforme a la legislación Europea.

2.4.1.1. Sistemas de Inteligencia artificial prohibidos

Se prohíben los siguientes sistemas de la [Inteligencia Artificial \(IA\)](#).

- Técnicas subliminales que distorsionen el comportamiento de una persona.
- Explotación de vulnerabilidades de un grupo específico de personas por su edad, discapacidad o situación social o económica.
- Elaboración de perfiles de personas según su comportamiento, creando un “báremo social” que pueda resultar en un trato desproporcionadamente desfavorable.
- Identificación biométrica en tiempo real en lugares accesibles al público por parte de las fuerzas y cuerpos de seguridad, exceptuando muchos casos.

El reglamento especifica que la lista puede ser modificada por la Comisión.

2.4.1.2. Sistemas de Inteligencia artificial de Alto Riesgo (HRAIS)

Se definen como sistema de alto riesgo los que cumplan alguna de estas tareas.

- Los sistemas que ya cuenten con una legislación Europea Armonizada y estén sujetos a evaluación de conformidad.
- Sistemas de identificación biométrica.
- Gestión de infraestructuras críticas.
- Educación y formación profesional.
- Selección de personal y gestión de las relaciones laborales.
- Gestión del acceso de las personas a servicios esenciales, públicos y privados.
- Actividades de fuerzas y cuerpos de seguridad.
- Migración, asilo y control de fronteras.
- Administración de justicia y procesos democráticos.

Los sistemas de alto riesgo deben cumplir con un nivel adecuado de las siguientes exigencias.

- Gestión de riesgos.
- Gobernanza y gestión de los datos.
- Documentación técnica actualizada.

- Registros de actividad del sistema.
- Información a los usuarios.
- Supervisados por una o varias personas.
- Precisión, robustez y seguridad informática.

2.4.1.3. Obligaciones de transparencia de ciertos sistemas de Inteligencia artificial

Los proveedores de sistemas deben informar a las personas que están interactuando con un sistema de [Inteligencia Artificial \(IA\)](#), excepto cuando se usen para la persecución del crimen.

También se deberá informar a las personas de los sistemas *deep fakes* excepto en persecución del crimen, contenidos evidentemente creativos, satíricos o ficticios.

2.4.1.4. Medidas de apoyo a la innovación

Las autoridades nacionales pueden crear *sandboxes* regulatorios para desarrollar, entrenar, probar y validar sistemas de [Inteligencia Artificial \(IA\)](#) bajo su guía, supervisión y soporte.

2.4.1.5. Gobernanza

El reglamento designa dos entidades para la gobernanza, el Comité Europeo de Inteligencia Artificial y las Autoridades nacionales competentes.

El Comité Europeo de Inteligencia Artificial, con un representante de cada [EEMM](#) y un grupo permanente de interesados. Sus tareas incluyen, recolección de información técnica y de las mejores prácticas, armonización de pruebas de conformidad, cooperación con organizaciones de expertos en servicios digitales, publicar recomendaciones sobre especificaciones, estándares y guías. Aconsejar a la Comisión en líneas maestras para la implantación del reglamento, etc.

La Autoridad nacional competente deberá designar una autoridad notificadora y una autoridad de supervisión, estas deben ser objetivas e imparciales y deben ser dotadas de los recursos adecuados.

2.4.1.6. Monitorización post-comercialización, compartición de información y supervisión de mercado

Los proveedores de sistemas de alto riesgo deben contar con planes de monitorización apropiados a los riesgos detectados. En caso de cualquier incidente, los proveedores deberán informar a las autoridades de supervisión y a otras autoridades encargadas.

Sé informarán a la Comisión sobre las actividades relacionadas con el Reglamento. Los [HRAIS](#) serán supervisados según la normativa correspondiente: los financieros por el supervisor financiero, los sistemas biométricos por fuerzas de seguridad o la agencia de protección de datos, y las instituciones de la Unión por el Supervisor Europeo de Protección de Datos. Los Estados miembros deben coordinar a las autoridades nacionales de supervisión.

Las autoridades de mercado pueden acceder a datos, código y documentación de los [HRAIS](#) para verificar el cumplimiento, otorgar o denegar permisos para pruebas, y suspender pruebas ante incidentes serios.

Las autoridades de mercado evaluarán sistemas que tengan riesgos y exigir correcciones. Se puede prohibir o restringir sistemas no corregidos y exigir medidas para los que presentan algún riesgo elevado. Se crearán centros de pruebas y un banco de expertos en [Inteligencia Artificial \(IA\)](#) para apoyar y asesorar.

2.4.1.7. Códigos de conducta

Los códigos de conducta deben ser redactados por los Estados miembros y por la comisión, estos fomentarán la aplicación voluntaria de estos códigos a los sistemas que no sean de alto riesgo.

2.4.1.8. Confidencialidad y sanciones

El reglamento define una tabla de sanciones en función de la falta y la entidad agresora, estas sanciones pueden ser modificadas, además de que se pueden aplicar superiores e inferiores, valorando las circunstancias y el proveedor.

2.5. Aplicaciones de las redes neuronales en la seguridad informática

Como hemos tratado previamente, existen distintas aplicaciones que se le puede dar a las redes neuronales para segurizar el funcionamiento de un sistema o proteger a un usuario frente ataques maliciosos.

En la actualidad existen muchos tipos de ataque informáticos, entre estos destacan malware, phishing, spam, man in the middle, denegación de servicio, inyección SQL, DNS *Tunneling*, *botnets*, o vulnerabilidades *zero-day* [106], estos son solo una clasificación rápida de todos los tipos de ataques que se pueden sufrir.

La [Inteligencia Artificial \(IA\)](#) ofrece numerosas ventajas en la recopilación de información, toma de decisiones y autonomía de sistemas, podemos usar modelos de inteligencia artificial para mejorar la seguridad clásica de los sistemas de información. Esto permite mejorar la eficiencia de los métodos de seguridad empresarial, seguridad ciudadana, etc.

Estos campos han impulsado una carrera armamentística entre defensores y atacantes, además del desarrollo de códigos éticos y regulaciones internacionales en el ámbito militar.

Algunas herramientas que han llegado al ámbito empresarial para mejorar sus sistemas de seguridad, destacan:

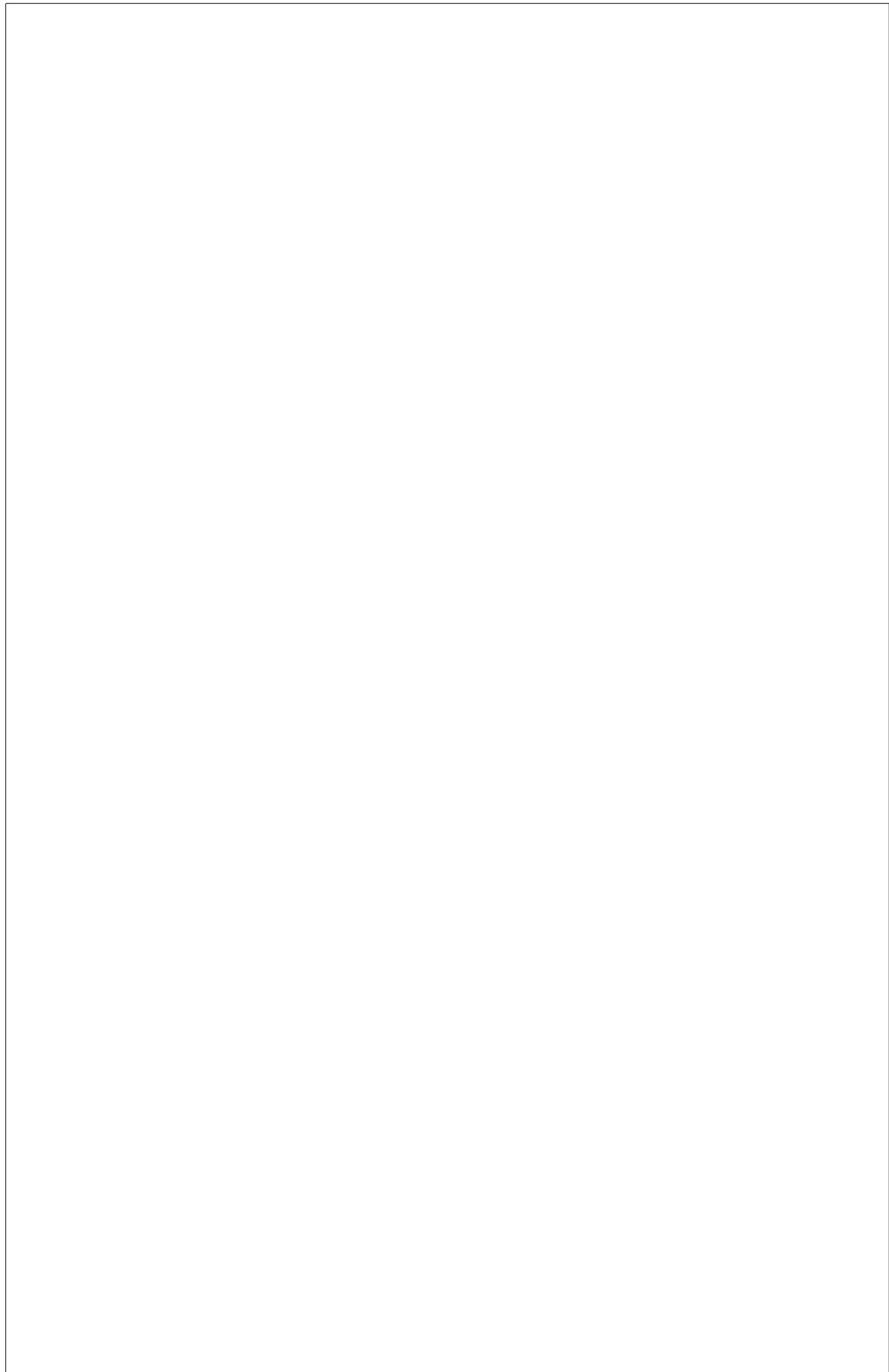
[Cylance](#) Es una de las principales empresas que proporcionan herramientas de seguridad informática que emplean inteligencia artificial.

- CylancePROTECT: plataforma de protección de puntos finales, especializada en identificar y neutralizar malware. Mediante el análisis de archivos y la ejecución de modelos predictivos. Es versátil ya que emplea los modelos predictivos para anticipar el comportamiento de malware desconocido.
- CylanceOPTICS: herramienta de detección y respuesta, permite la visualización de las etapas de un ataque, facilita la respuesta rápida y automatizadas basadas en reglas a las amenazas identificando, minimiza el impacto sin bloquear el resto de sistemas no comprometidos.

[Fortinet](#) Es una de las empresas con más recorrido en la seguridad informática, especializada en operaciones TI. Cuenta con FortiAI, que es un dispositivo hardware y software para empresas, este cuenta con millones de funciones de malware entrena-

das, detecta y analiza amenazas en tiempo real, además de manejar un gran volumen de tráfico de red en tiempo real. Estas soluciones todo en uno vienen acompañadas con decenas de soluciones adicionales. Asume tareas de [Intrusion Detection System \(IDS\)](#), [Security information and event management \(SIEM\)](#) y [Security orchestration, automation, and response \(SOAR\)](#).

TODO:



Capítulo 3

OBJETIVOS

En este capítulo explicaremos brevemente los objetivos que se quieren abordar en el proyecto, describiremos los objetivos principales como los específicos y su finalidad.

3.1. Objetivos principales

El objetivo principal de este proyecto es el de realizar un estudio de las redes neuronales adversarias en el campo de la seguridad informática y su aplicación en la seguridad informática.

Realizar un estudio de la aplicación de redes neuronales en seguridad informática y analizar vectores de ataque a modelos de redes neuronales. Especialmente nos vamos a centrar en las redes neuronales adversarias y modelos generativos como son [GAN](#) y [VAE](#), aunque existen otros modelos generativos, acotaremos la revisión bibliográfica a las [GAN](#) únicamente y mencionaremos otros modelos con características similares.

Se adaptará y ejecutará métodos de redes neuronales adversarias en el área de seguridad, específicamente se ha planteado la temática de generar un modelo que sea capaz de evadir las medidas de seguridad clásicas de biometría. Es decir, usaremos una red neuronal adversaria para generar instancias falsas que sean capaces de evadir parcial o totalmente un sistema biométrico. El objetivo de esto es buscar si es posible evadir estas medidas de seguridad. Además, se deberá evaluar, comparar y analizar los resultados.

Se ha desarrollado un estudio de posibles vectores de ataque a algoritmos de

biometría empleando estas arquitecturas y de como podemos optimizar la generación de estas instancias. Esto con el objetivo de generar instancias que puedan llegar a dar falsos positivos en sistemas de autenticación o identificación biométrica.

El objetivo de este proyecto es el de analizar si la seguridad informática puede ser comprometida desde distintos puntos, desde vulnerar un modelo que tenía fines legítimos o desde un atacante que utilice un modelo con fines maliciosos.

3.2. Objetivos específicos

- **Búsqueda de una visión general de la ciencia de datos actual:** Estudio del proceso KDD, aprendizaje profundo, redes neuronales, modelos y tareas. Nos enfocaremos en las redes neuronales adversarias aplicadas a la seguridad informática, centrándonos en redes generativas adversarias ([GAN](#)) y brevemente en los ([VAE](#)). Además, se analizará el estado normativo de estas aplicaciones dentro de la Unión Europea.
- **Análisis de vectores de ataque en redes neuronales.** Se analizarán los vectores de ataques específicos en la cadena de suministro que pueden ser comprometidos dentro de nuestro campo de estudio.
- **Implementar un modelo de red neuronal generativa adversaria que genere instancias sintéticas.** Con el fin de analizar la capacidad de las redes neuronales para evadir la seguridad, se creará un modelo con la tarea de generar instancias falsas que pueda provocar falsos positivos en sistemas de seguridad biométrica.
- **Optimizar la generación y eficiencia de instancias sintéticas.** Se buscará mejorar las técnicas de generación de instancias sintéticas, se explorará técnicas de optimización.
- **Evaluación del modelo mediante experimentos para extraer métricas.** Se deberá evaluar los resultados obtenidos por el modelo, analizando las métricas del modelo y analizando la evaluación de las instancias frente a algoritmos de extracción y coincidencia de características.
- **Documentación y guía de instalación para la creación del modelo.** Se redactará la documentación de uso e instalación del modelo y las herramientas usadas durante el proyecto. Estos recursos estarán disponibles en el repositorio oficial del proyecto.

Capítulo 4

MATERIALES Y MÉTODOS

4.1. Equipos para la experimentación

Para el desarrollo de este proyecto se han usado distintos equipos, principalmente un ordenador para el desarrollo y un servidor para el entrenamiento.

Estos son los componentes empleados en el desarrollo del proyecto.

CPU	AMD Ryzen 9 7900X3D 4.4GHz/5.6GHz
RAM	2 x (Corsair Vengeance DDR5 6000MHz 32GB 16GB CL30 Memoria Dual AMD EXPO e Intel XMP)
GPU	Gigabyte GeForce RTX 4060 EAGLE OC 8GB GDDR6 DLSS3
Disco duro	Samsung 990 PRO 2TB SSD PCIe 4.0 NVMe M.2
Placa base	MSI MAG X670E TOMAHAWK WIFI

Tabla. 4.1: Hardware del equipo de desarrollo

Producto	RS720-E9-RS8-G
CPU	2 x (Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz)
RAM	4 x (Samsung M393A8G40AB2-CVF DDR4-2933 64GB)
GPU	NVIDIA GA100 [A100 PCIe 80GB]
Disco duro	NVMe SSD 980 PRO

Tabla. 4.2: Hardware del equipo de entrenamiento

4.2. Extracción y coincidencia de características

El objetivo de extracción y coincidencia es el de descubrir una o varias características en dos instancias que permitan reconocer e identificar de forma inequívoca, existe multitud de métodos en [Computer Vision \(CV\)](#) para detectar puntos de interés. En esta sección discutiremos los métodos y herramientas que podemos emplear usando de ejemplo las librerías de OpenCV y Scikit-Image.

Existen cientos de aplicaciones en la sociedad y en el mundo industrial, médico, automovilístico, militar, etc. Se incluyen tareas de reconstrucción de imágenes, imágenes 3D, mapeo, estabilización de vídeo, etc.

Uno de los grandes retos es automatizar robots para que reconozcan y puedan interactuar con el entorno. Para ello se debe reconstruir en 3D el entorno para determinar distancias entre objetos, este es uno de los problemas más difíciles, ya que se deben tener en cuenta puntos de vista, iluminación, oclusión, características de los sensores, etc. Por esta razón, estos métodos deben ser robustos ante problemas de oclusión, rotación, escala, iluminación, deformación, etc. [\[12\]](#)

En estudios similares se hace mención de los distintos vectores de ataque que pueden sufrir estos sistemas biométricos, en la denominada “cadena de suministro”. [\[11\]](#) como podemos observar en la Figura 4.1.

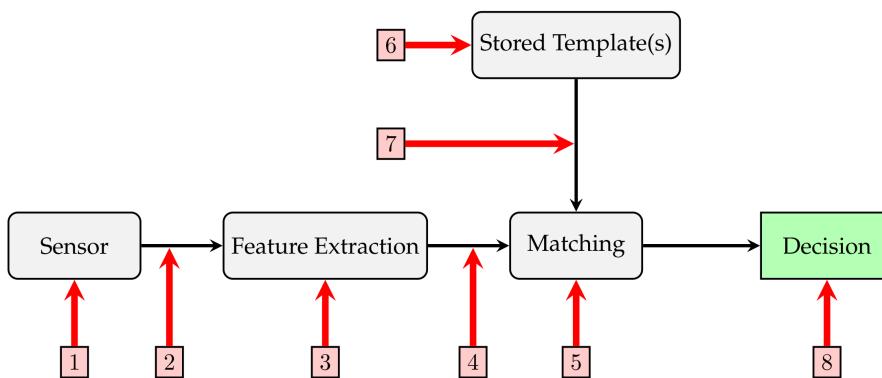


Figura 4.1: Attack vectors in a generic biometric system

Fuente: Exploiting Image Sensor Data in Biometric Systems and Mobile Applications [\[11\]](#)

Actualmente, se clasifican los algoritmos en métodos basados en **regiones** o métodos basados en **características**. En los métodos por regiones usan la comparación directa, estos emplean una ventana deslizante que comparará ambas regiones. Los métodos basados en características ofrecen más flexibilidad y estabilidad, estos métod-

dos busca detectar estructuras (esquinas, bordes, formas), con el objetivo de generar descriptores robustos. Se debe tomar las muestras validando de que sean útiles, que no haya un sobre muestreo. Estos métodos dan buenos resultados, pero a costa de un coste computacional muy elevado cuando la resolución de imágenes o vídeos aumenta.

La extracción de características debe ser robusta para su posterior comparación, los métodos por **Artificial Neural Network (ANN)** usan capas convolucionales, estas capas generarán descriptores basados en las coordenadas de las características y sus relaciones de vecindad. También existen los extractores *end-to-end* que permiten generar los descriptores directamente desde imágenes.

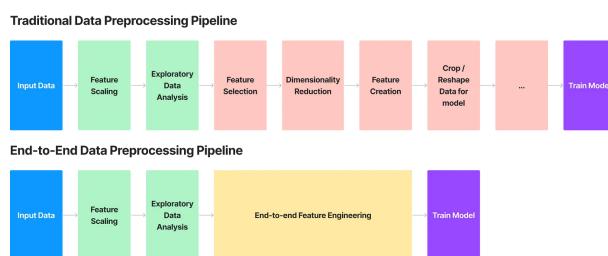


Figura 4.2: End to end feature engineering methods

Fuente: [End-to-end transformational Feature Engineering with CNNs](#)

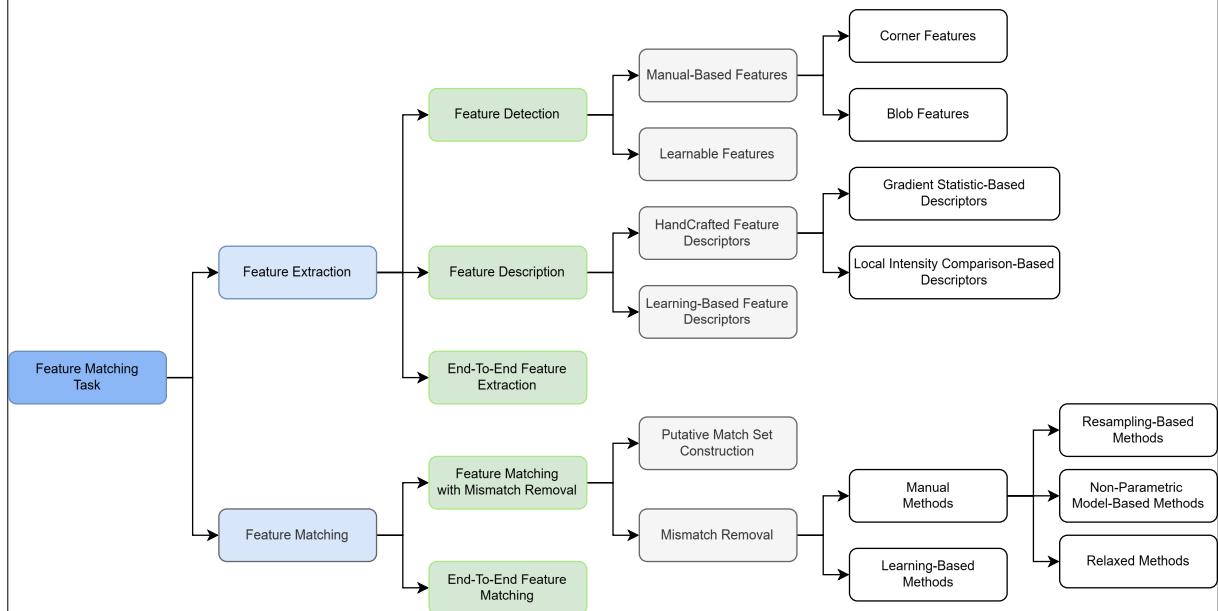


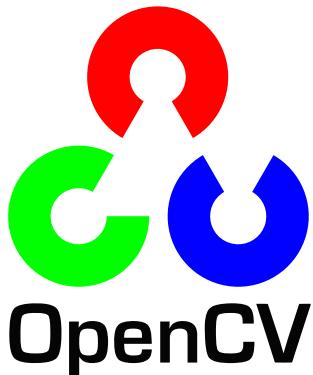
Figura 4.3: Clasificación de métodos de extracción y coincidencia de características.

Fuente: [A survey of feature matching methods \[12\]](#)

Como podemos ver en la Figura 4.3 existen una gran variedad de métodos y para las distintas tareas dentro de la extracción y coincidencia en **Computer Vision (CV)**, a continuación se listan los algoritmos clásicos más relevantes en la literatura actual.

Posteriormente, se identificarán algunos de los métodos de redes neuronales más relevantes en la literatura que permiten también extraer y comparar puntos de interés.

Algunas de las herramientas que permite usar estos algoritmos y métodos son las librerías de `OpenCV` y `scikit-image`, estas librerías no implementa todos los métodos que requerimos, pero sí una gran cantidad de ellos.



(a) Librería OpenCV



(b) Librería scikit-image

Figura 4.4: Librerías para el tratamiento de imágenes

Expicaremos algunos de los métodos más relevantes y usados en [Computer Vision \(CV\)](#) para la detección de puntos de referencia y sus descriptores. Estos métodos permiten identificar de forma eficiente distintas huellas o identificadores biométricos.

4.2.1. Algoritmos de detección de puntos de interés

■ Bordes:

- Operador Roberts cross (1965) [107, 108]
- Robinson (1977) [109]
- Canny (1986) [110]
- Deriche (1987) [111]
- Differential (1998) [112]
- STAR Detector (2008) [113]
- Operador Prewitt (2013) [114]
- Operador Sobel (2014) [115]

■ Esquinas:

- Harris operator (1988) [116]
- Shi-Tomasi (1994) [117]

¹OpenCV Python cuenta con una implementación experimental.

²Algoritmo patentado hasta el año 2020

³Algoritmo patentado hasta el año 2028

- FAST (1994) [118, 119]
- **Blobs:**
 - SIFT² (1999) [120]
 - AKAZE (2011) [121]
- **Crestas:**
 - Hough transform (1972) [122]
- **Regiones:**
 - MSER (2006) [123]

4.2.2. Algoritmos de cálculo de descriptores de puntos de interés

- **Orientación de gradientes:**
 - SIFT² (1999) [120]
 - HOG (2005) [124]
 - GLOH (2005) [125]
 - SURF³ (2006) [126, 127]
 - CSIFT (2006) [128]
 - DAISY¹ (2009) [129]
 - RootSIFT (2012) [130]
- **Binarios:**
 - BRIEF¹ (2010) [131]
 - AKAZE (2011) [121]
 - BRISK (2011) [132]
 - ORB (2011) [133]
 - LATCH (2016) [134]

4.2.3. Algoritmos de búsqueda de descriptores

- **Búsqueda lineal:**
 - Brute-Force Matcher [135]
- **Búsqueda aproximada:**
 - FLANN (2012) [135, 136]

En la literatura se mencionan principalmente dos métodos de búsqueda de descriptores, método por fuerza bruta o búsqueda aproximada (FLANN [135, 136]).

Estos métodos presentan ventajas y desventajas, la principal ventaja es que son sencillos y muy eficientes, la principal desventaja el coste computacional cuando la

extracción ha dado un gran número de puntos de interés. Actualmente, ya existen métodos que solucionan el problema de alta densidad en la coincidencia de descriptores, como es el caso de [FDD](#) [137].

4.2.4. Redes neuronales para la detección y coincidencia

A continuación se explican otros métodos que son relevantes, ya que usan redes neuronales y métodos de inteligencia artificial para extraer y comprobar la coincidencia de instancias biométricas.

Estos son los métodos *end-to-end* mencionados previamente que permite extraer directamente descriptores desde una imagen usando redes neuronales.

- [SuperPoint](#) [138] es un modelo basado en redes neuronales, este es de los más relevantes en esta tarea que, ya que permite detectar y describir puntos de interés.
- [KeyNet](#) [139] es otro de los modelos que permite detectar puntos de interés robustos mediante filtros [CNN](#).
- [D2-Net](#) [140] es uno de los trabajos más recientes, dicha red permite detectar y describir puntos de interés, aunque en este caso ellos invierten el proceso y describen para posteriormente detectar los puntos de interés. Es robusto y fiable en instancias con cambios de iluminación.

4.2.5. Otros algoritmos o métodos

4.2.5.1. Iterative Closest Point Algorithm (ICP)

Es un algoritmo iterativo de puntos más cercanos ([Iterative Closest Point \(ICP\)](#)), que consiste en transformar iterativamente un conjunto de características para que coincidan mejor con otro, minimizando una métrica de error basada normalmente en una distancia (distancia euclidiana) [141].

Existen algunas implementaciones externas de este algoritmo, en nuestro caso partiremos de esta implementación que usa [opencv](#). [abreheret/icp-opencv](#)

SIFT (Scale Invariant Feature Transform)

Es uno de los métodos más relevantes en [CV](#), este permite detectar puntos característicos en una imagen y luego describirlos mediante un histograma orientado de gradientes. [\[120\]](#)

El algoritmo cuenta con dos partes, la extracción de puntos de interés y la descripción de la región del punto de interés. La detección de puntos de interés detecta mediante esquinas y mediante *blobs*.

- **Localidad**
- **Distinción**
- **Cantidad**
- **Eficiencia**
- **Extensibilidad**

Una de las grandes desventajas de este algoritmo es su velocidad, además estaba patentado hasta el año 2020, lo que hizo que no se pudiera usar en productos comerciales, pero sí en investigación.

4.3. Análisis de frameworks de Deep Learning

Encontrar un *framework* que cubra nuestras necesidades es una tarea compleja, ya que hay una gran variedad de estos que aun siendo respaldados sus desarrollos por grandes empresas son abandonados y puede no cumplir con nuestras

En la siguiente sección se muestran y se explican las características de los principales *frameworks* de desarrollo de modelos de redes neuronales profundas más relevante del momento.

4.3.1. Keras



Figura 4.5: Keras

Keras [142] es uno de los primeros *frameworks* en la creación y ejecución de modelos de redes neuronales profundas, publicada en 2015, bajo la licencia [MIT](#), es de código abierto, soporta múltiples entornos de ejecución como [TensorFlow](#) [143] o [CNTK](#) [144].

Escrita en [python](#) el proyecto está estructurado en módulos independientes que suelen trabajar en conjunto, cuenta con soporte para capas neuronales, funciones de perdida, optimizadores, métricas, etc. Además, también permite extender estos módulos para adaptarlos a tus propias modificaciones.

El objetivo principal de Keras es el de ser una interfaz intuitiva en vez de un framework completo, ya que utiliza se implementó el núcleo de [tensorflow](#).

Características principales:

- Interfaz sencilla para crear redes neuronales
- Definida por módulos que permiten agregar funcionalidades.
- Sencilla de extender sus características.

Ventajas

- Fácil de usar para principiantes
- Rápido de aprender para usuarios expertos
- Es la base de otros frameworks

Desventajas

- Menos funciones avanzadas que otros frameworks
- Más restringido que frameworks avanzados

4.3.2. TensorFlow



Figura 4.6: TensorFlow

TensorFlow [143] es el *framework* del grupo de investigación de inteligencia artificial de Google, es de código abierto y cuenta con la licencia [APACHE](#). Es uno de los más completos, cuenta con un amplio ecosistema, ya que se enfoca en proyectos empresariales. Proporciona una [Application Programming Interface \(API\)](#) de [python](#) así como para los lenguajes de C++, Haskell, Java, Go y Rust, existen otras implementaciones de terceros de la [API](#) de TensorFlow.

Además, se expande mediante implementaciones de alto rendimiento para dispositivos de bajos recursos, como es el caso de tensorflow.js, este permite ejecutar modelos ligeros directamente en un navegador web, ofrece distintos **backend**. La **API** es flexible y permite el uso de funciones de Keras.

Características principales:

- Soporta un gran conjunto de herramientas y librerías para la construcción de redes neuronales, tanto convolucionales como recurrentes.
- Soporte para distintos tipos de hardware, tanto **Central Processing Unit (CPU)**, **Graphics Processing Unit (GPU)**, **Tensor Processing Unit (TPU)**

Ventajas

- Multiplataforma
- Gran soporte a muchas tareas
- Soporte y documentación
- Compatible con uso industrial

Desventajas

- Curva de aprendizaje alta
- Grafo estático poco eficiente ante cambios

4.3.3. PyTorch



Figura 4.7: PyTorch

PyTorch [145] es uno de los *frameworks* actuales más relevantes para el desarrollo de modelos de inteligencia artificial, es de código abierto y es desarrollada principalmente por *Facebook's AI Research lab* (FAIR), cuenta con una licencia libre **BSD**, fue lanzado en 2016.

Cuenta con soporte para Windows, Linux y MacOS, cuenta con soporte para gestores de paquetes de `conda` y `pip`, además PyTorch distribuye su librería para otros sistemas (C++) mediante LibTorch, da soporte para tarjetas gráficas NVIDIA con tecnología CUDA [11.8, 12.1, 12.4], tarjetas gráficas AMD ROCm [6.2] o **CPU**.

Sus principales ventajas es que es la fusión de varios proyectos con otras organi-

zaciones, por lo que existe una compatibilidad entre estos *Frameworks*. Es compatible con otras librerías de ciencias de datos, *numpy*, *scipy*, *panda*.

Soporta la ejecución y entrenamiento en [GPU](#) y [TPU](#) y la distribución entre múltiples tarjetas gráficas, cuenta con una [API](#) disponible para otros lenguajes, además de modelos pre-entrenados, conjuntos de datos o ejemplos de arquitecturas comunes en la literatura actual.

PyTorch cuenta con un gran ecosistema, estos son otros proyectos que expanden la librería y permite realizar, desde visualización de los procesos, entrenamientos o resultados, exportar los modelos a otros entornos de ejecución, etc.

Características principales:

- PyTorch trabaja con un grafo dinámico haciendo que el desarrollo y experimentación sea más rápido.
- Programación imperativa permite construir modelos más sencillos.
- Soporte para [GPU](#), [TPU](#), etc.

Ventajas

- Fácil para principiantes
- Adaptable para desarrollo e investigación
- Compatible con grafo dinámico

Desventajas

- Menos funciones que TensorFlow
- Menos opciones para despliegue

Módulo	Descripción
torch	Biblioteca principal para el cálculo numérico y la manipulación de tensores.
torch.nn	Proporciona herramientas para construir redes neuronales y capas.
torch.autograd	Soporte para la diferenciación automática para la optimización del entrenamiento.
torch.optim	Optimizadores, como SGD y Adam, para ajustar los parámetros de los modelos.
torch.utils.data	Herramientas para el manejo de datos y su carga durante el entrenamiento.

Tabla. 4.3: Núcleo de PyTorch

Tecnologías	Descripción
torch.cuda	Permite el uso de GPUs NVIDIA para acelerar los cálculos de tensores
torch.onnx	Herramientas para exportar modelos de PyTorch al formato ONNX para interoperabilidad con otros frameworks

Tabla. 4.4: Tecnologías de PyTorch

Librería	Descripción
torchvision	Conjunto de bibliotecas para tareas de visión por computadora, con modelos preentrenados y utilidades
torchaudio	Biblioteca para trabajar con datos de audio, con soporte para diversas operaciones de audio
torchtext	Biblioteca para la carga, preprocesamiento y uso de datos de texto en tareas de NLP

Tabla. 4.5: Librería de PyTorch



Figura 4.8: PyTorch Lightning

PyTorch Lightning [146] es un framework que expande y simplifica PyTorch, este permite simplificar el código de PyTorch a una arquitectura de objetos con el objetivo de realizar módulos más reutilizables.

Este framework permite simplificar y abstraer el código, controla la gestión de memoria y de dispositivos, permite crear y pruebas o depurar de forma más sencilla.

Cuenta con una extensa API, cuenta con gestión de modelos en dispositivos *Accelerators*, ejecución de funciones por eventos mediante *Callbacks*, un *Command Line Interface (CLI)* para la gestión por terminal, gestión de logs, control del entrenamiento mediante la clase Trainer, gestión automática de hiperparámetros mediante Tuner y muchas más utilidades.

4.3.5. Otras alternativas, frameworks de deep learning

Se han analizado muchos frameworks que cumplen parcialmente nuestras necesidades, algunas de las alternativas son las siguientes. Muchos de estos proyectos siguen en desarrollo y aumentando sus capacidades, aunque otros o han llegado a su objetivo o han sido abandonados parcial o totalmente.

Framework	Especialización	Empresa	Estado
MXNet	Generalista	F. APACHE y Amazon	En desarrollo
CAFFE	Visión por computador	Berkeley AI Research	Abandonado
CNTK	Generalista	Microsoft	Abandonado
LangChain	LLMs	LangChain, Inc	En desarrollo
OpenNN	Generalista	Artelnics	Abandonado
Deeplearning4j	Generalista	Konduit and contributors	En desarrollo
Chainer	Generalista	Seiya and contributors	Abandonado

Tabla. 4.6: Frameworks de deep learning

4.4. Bibliotecas para ciencia de datos

Existe una gran variedad de bibliotecas y librerías que ofrecen soluciones a los principales problemas en el campo de la ciencia de datos y de la inteligencia artificial. A continuación mostramos las herramientas más usadas por la comunidad científica y cuáles son sus características.

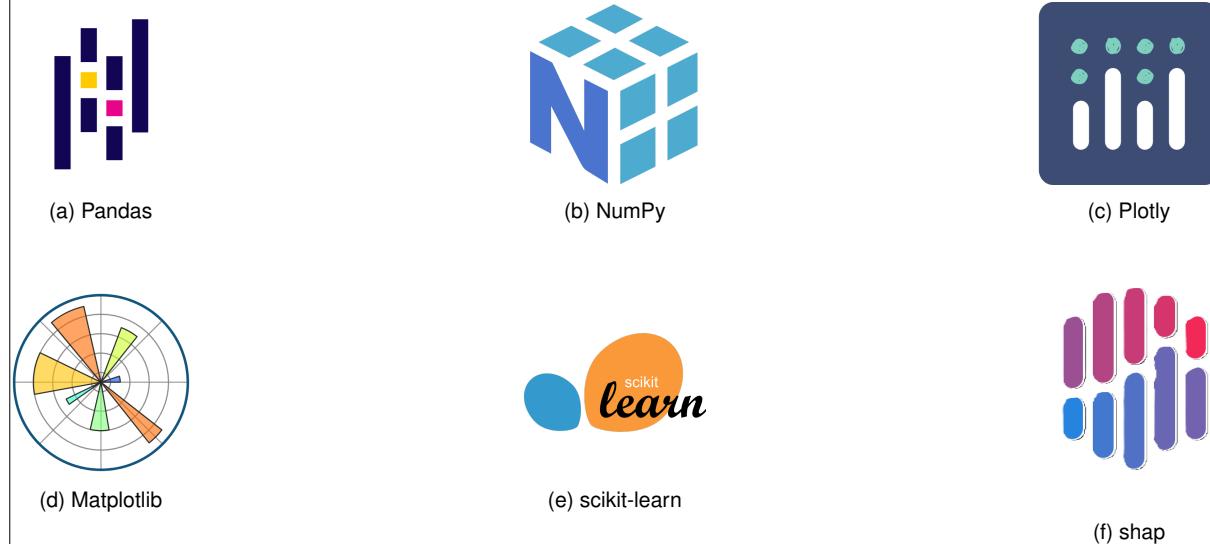


Figura 4.9: Bibliotecas de Python para ciencia de datos

4.4.1. Pandas

Pandas es una de las bibliotecas más importantes en *Data Science*, facilita el manejo de grandes volúmenes de datos, simplifica la lectura y escritura de múltiples fuentes. Esto lo realiza creando una estructura de datos tipo tabla con columnas llamadas **Series** de datos y operando sobre matrices.

Alguna de las características más relevantes de la librería son:

- **Gestión de datos:** permite el tratamiento adecuado de valores faltantes.
- **Agrupación y ordenación:** facilita el agrupamiento y la ordenación de datos.
- **Manejo de series temporales:** incluye herramientas para manipular y analizar datos con índices de tiempo.
- **Mezcla y fusión de datos:** permite combinar conjuntos de datos de diversas fuentes.
- **Edición de estructura:** es posible agregar, eliminar o modificar filas y columnas de forma eficiente.

- **Interfaz de entrada y salida:** proporciona funciones para leer y escribir datos en múltiples formatos.

4.4.2. Numpy

NumPy (Numerical Python) es una de las bibliotecas principales para el cálculo científico, gestiona grandes volúmenes de datos, operar con vectores y matrices multidimensionales, cuenta con cientos de funciones matemáticas de alto rendimiento, esta librería es la base en la que se apoyan otras bibliotecas.

Alguna de las características más relevantes de `numpy` son:

- **Datos multidimensionales:** proporciona la estructura `ndarray`, que permite almacenar y manipular datos en forma de vectores o matrices de múltiples dimensiones de forma muy eficiente.
- **Alto rendimiento:** está implementado en C con una alta eficiencia.
- **Compatibilidad:** es compatible con pandas, SciPy, Matplotlib, etc.
- **Manipulación de datos:** permite reestructurar, aplanar y cambiar la forma, facilita el procesamiento de datos.
- **Operaciones matemáticas avanzadas:** cuenta con una amplia variedad de funciones matemáticas.

4.4.3. Plotly

Plotly es una de las bibliotecas por referencia de generación de gráficos para la visualización de datos, genera gráficos dinámicos, interactivos, compatible con una gran variedad de bibliotecas, permite exportar los gráficos a distintos tipos de formatos, incluso se integra en aplicaciones web.

Cuenta con gráficos avanzados como contornos, mapas de calor, soporta gráficos de series temporales, gráficos de velas y cascada. Cuenta con visualización geo espacial. Plotly es útil para inteligencia artificial con gráficos de regresión y representación de curvas ROC, en bioinformática, con gráficos de volcanes y Manhattan, etc.

4.4.4. **Mathplotlib**

Mathplotlib es una biblioteca para la generación de gráficas, se especializa en ciencia de datos, estadística e inteligencia artificial, en este caso solo soporta python y al igual que **plotly** cuenta con una gran variedad de gráficos, es compatible con otras herramientas, permite extender sus funciones como con otros frameworks como **Seaborn**. Además, permite exportar en múltiples formatos, una de las pocas desventajas es que no cuenta con soporte para gráficas interactivas en web.

Aunque es mucho más completo y cuenta con muchas más características que **plotly**, puede ser más complejo, pues la personalización requiere un aprendizaje profundo de la biblioteca.

4.4.5. **Scikit-learn**

Scikit-learn es la biblioteca para python especializada en inteligencia artificial, machine learning y minería de datos. Cuenta con una gran variedad de funciones, algoritmos, métricas e implementaciones de arquitecturas de modelos de inteligencia artificial.

Es una de las bibliotecas principales para la investigación y cuenta con un gran apoyo por parte de la comunidad y una buena documentación.

Existe una derivación de la biblioteca que se especializa en el análisis y procesamiento de imágenes. **scikit-image** cuenta con muchas tareas de manipulación de color, canales, transformaciones geométricas, filtrado, restauración, segmentación de objetos, etc.

4.4.6. **Adversarial Robustness Toolbox (ART)**

Adversarial Robustness Toolbox es una de las librerías de python para la seguridad del *machine learning*, esta librería es muy completa y cuenta con el apoyo de cientos de investigadores a nivel mundial, pensada para el análisis de modelos de inteligencia artificial, implementa los algoritmos, métodos y técnicas con los que los investigadores logran evadir o atacar los modelos.

Su complejidad es alta, pues se requiere de un estudio profundo de como funcionan los modelos, además de comprender el funcionamiento de la técnica que se quiere

realizar.

Esta librería está apoyada por el grupo de investigadores de IBM, que mantiene y desarrolla el proyecto, investigadores pueden aportar sus descubrimientos implementándolos en la librería.

4.4.7. Shap

SHAP es una de las bibliotecas para python especializada en la explicabilidad de modelos de inteligencia artificial, se usa para entender los resultados de los modelos de redes neuronales o de los modelos random forest, etc. La biblioteca permite la visualización de características, análisis globales, análisis de las predicciones, detectar sesgos, etc.

4.4.8. Otras librerías

Existen una gran variedad de herramientas que puede ser útiles para la investigación científica, desarrollo e implementación de modelos.

- AzureML
- Scipy
- Seaborn
- Imbalance Learning
- statsmodels
- NLTK
- LightGBM
- XGBoost
- AIX360

Glosario

AE Es un tipo de red neuronal que permite sacar un tipo de dato del mismo tipo de entrada. [39, 40](#)

AI La inteligencia artificial (IA), también conocida por su nombre inglés, Artificial Intelligence (AI), es una tecnología que trata de realizar las tareas y tomar las decisiones empresariales de forma automática y autónoma, aprendiendo de forma continua. [\[147\], 9](#)

APACHE La licencia de software Apache 2.0 permite a los usuarios utilizar, modificar y distribuir el software, manteniendo la declaración de derechos de autor y una renuncia de responsabilidad. Proporciona la concesión de patentes.. [92](#)

API Es un conjunto de protocolos que permiten a distintos componentes de software comunicarse y transferir datos. [92–94](#)

ART Modelo de red neuronal que equilibra la adaptación a nueva información con la estabilidad frente a patrones familiares. [19](#)

ATLAS MITRE ATLAS es un framework de trabajo orientado para la seguridad informática en el uso de inteligencia artificial (IA). [10, 74](#)

backend Es el motor que realiza los cálculos, como multiplicaciones de matrices, operaciones de convolución, y otras tareas intensivas en computación. [93](#)

BP Método utilizado en redes neuronales para calcular el gradiente y el cálculo de los pesos, es una abreviatura de “propagación de errores hacia atrás”. [19](#)

BSD La licencia de software BSD se origina para los productos UNIX de *Berkeley Software Distribution*. [93](#)

CLEVER . [72](#)

CLI Permite interactuar con un sistema operativo o software a través de comandos de texto. [95](#)

CPU . [93](#)

DGM . [65](#)

DL *Deep learning (DL)*, también conocido como aprendizaje profundo, es un tipo de machine learning que se estructura inspirándose en el cerebro humano y sus

redes neuronales. El aprendizaje profundo procesa datos para detectar objetos, reconocer conversaciones, traducir idiomas y tomar decisiones. Al ser un tipo de machine learning, esta tecnología sirve para que la inteligencia artificial aprenda de forma continua [148]. 9, 14

FGM . 63

GBDT Los árboles de decisión impulsados por gradientes son una técnica para optimizar el valor predictivo de un modelo a través de pasos sucesivos en el proceso de aprendizaje. Cada iteración del árbol de decisión implica ajustar los valores de los coeficientes, ponderaciones o sesgos aplicados a cada una de las variables de entrada que se utilizan para predecir el valor objetivo, con el objetivo de minimizar la función de pérdida. 72

GMA . 67

GNN Es una clase de redes neuronales especializada en el procesamiento de datos que se puedan representar como grafos. 19

GPU . 93, 94

IDS Sistema de detección de intrusos, herramienta de seguridad diseñada para monitorizar el tráfico de red con el objetivo de identificar y detener posibles amenazas.. 81

ImageNet Gran base de datos de imágenes, diseñada para su uso en la investigación de software de reconocimiento de objetos visuales. 72

KDD Es el proceso utilizado para extraer de forma eficiente y automática **información útil** a partir de grandes volúmenes de datos. 13, 22

LSTM Memoria larga a corto plazo. 20

LTU La unidad de umbral lineal es una neurona artificial muy simple cuya salida es el sumatorio de la entrada total umbralizada. Es decir, una LTU con umbral T calcula la suma ponderada de sus entradas y, a continuación, emite **0** si esta suma es inferior a T y **1** si la suma es superior a T. Las LTU constituyen la base de los perceptrones [149]. 29, 30

MIT La licencia de software MIT se origina en el Instituto Tecnológico de Massachusetts, es permisiva y cuenta con la características de *copyleft*. 92

MITRE La abreviatura de (Tácticas, Técnicas y Conocimiento Amplio de Enemigos) es un marco de trabajo para la evaluación de la seguridad en las organizaciones. <https://attack.mitre.org/>. 74

ML El machine learning es la tecnología que permite que un sistema aprenda de forma continua. El sistema recibe un input, un humano reacciona y, así, la próxima vez que el sistema reciba ese input, sabrá cómo actuar sin necesidad de acudir al humano. [150]. 9, 60, 68

MLP El Perceptrón de múltiples capas es una red neuronal artificial formada por capas local o totalmente conectadas. [29](#)

ODA . [67](#)

OGA . [66](#)

Pix2Pix *Pixel to Pixel (Pix2Pix)*, es la abreviatura para tareas de traducción de imágenes. [48, 49](#)

RMA . [66](#)

RNN Es una clase de redes neuronales en la que las conexiones entre nodos forman un grafo dirigido a lo largo de una secuencia iterativa temporal. A diferencia de las FNN, las RNN puede usar sus pesos internos para procesar secuencias de entrada. [20](#)

SIEM Es el término para denominar las soluciones empresariales que abordan puntos vulnerables antes de afectar las operaciones empresariales. [81](#)

SOAR Es un conjunto de funciones que se utilizan para proteger los sistemas de TI de amenazas mediante la organización, automatización y respuesta de seguridad. [81](#)

STRIDE Son las siglas de las amenazas de *Spoofing, Tampering, Repudiation, Information, Denial, Elevation* que violan las propiedades de **Autenticidad, Integridad, No Repudio, Información, Disponibilidad y Autorización** respectivamente. [74](#)

TPU . [93, 94](#)

U-Net U-Net son las siglas de “red en forma de U”. Se trata de una arquitectura de codificador-decodificador con conexiones de salto entre las capas reflejadas de las pilas de codificadores y decodificadores. [48](#)

VAE Un codificador automático variacional es un tipo de modelo generativo basado en probabilidad. [39, 40, 65, 83, 84](#)

VQ-VAE Un codificador automático variacional es un tipo de modelo generativo basado en probabilidad. [40](#)

Siglas

ANN Artificial Neural Network. [14](#), [29](#), [36](#), [71](#), [87](#)

BP Backpropagation. [19](#)

cGAN Conditional GAN. [V](#), [45](#), [46](#), [49](#), [55](#)

CNN Convolutional Neural Network. [19](#), [46](#), [67](#), [90](#)

CoGAN Couple GAN. [47](#)

ConvNET Convolutional Neural Networks. [48](#)

CV Computer Vision. [86](#)–[88](#), [91](#)

CycleGAN Cycle GAN. [49](#), [51](#), [54](#)

DBN Deep Belief Networks. [20](#)

DCGAN Deep Convolutional GAN. [46](#), [52](#), [53](#)

EEMM Estado Miembro. [78](#)

ESRGAN Enhanced Super Resolution GAN. [53](#)

FDD Fixed-length Dense Descriptor. [90](#)

FNN Feedforward Neural Network. [19](#), [20](#)

GAN Generative Adversarial Network. [19](#), [20](#), [41](#)–[44](#), [47](#), [52](#), [54](#), [56](#)–[58](#), [65](#), [69](#), [83](#), [84](#)

HRAIS Sistemas de inteligencia artificial de alto riesgo. [79](#)

IA Inteligencia Artificial. [74](#)–[80](#)

ICP Iterative Closest Point. [90](#)

InfoGAN Information Maximizing GAN. [55](#), [56](#)

LSGAN Least Square GAN. [51](#), [52](#)

NN Neural Network. [16](#), [20](#), [39](#)

PatchGAN Patch GAN. [48](#)

PGGAN Progressive Growing GAN. [52](#), [54](#)

ProGAN Progressive GAN. [52](#)

PSNR Peak Signal-to-Noise Ratio. [53](#)

SAGAN Self-Attention GAN. [54](#)

SGAN Semi supervised GAN. [47](#)

SRGAN Super Resolution GAN. [53](#)

StyleGAN Style GAN. [21](#), [54](#)

UE Unión Europea. [76](#)

WGAN Wasserstein GAN. [54](#), [55](#)

WGAN-GP Wasserstein GAN Gradient Penalty. [55](#)

Bibliografía

- [1] Lior Rokach and Oded Maimon. *Data mining and knowledge discovery handbook*. Springer New York, 2010.
- [2] Oded Maimon and Lior Rokach. *Data mining and knowledge discovery handbook*, volume 2. Springer, 2005.
- [3] César Pérez Curiel. Análisis del espacio latente en el auto-encoder variacional. [Online; accessed 10-September-2024], Julio 2022.
- [4] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, 2018. URL <https://arxiv.org/abs/1711.00937>.
- [5] Zhengwei Wang, Qi She, and Tomas Ward. Generative adversarial networks in computer vision: A survey and taxonomy. *ACM Computing Surveys*, 54:1–38, 02 2021. doi: 10.1145/3439723.
- [6] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2017.
- [7] Weihao Xia, Yulun Zhang, Yujiu Yang, Jing-Hao Xue, Bolei Zhou, and Ming-Hsuan Yang. Gan inversion: A survey, 2022.
- [8] Jacson Rodrigues Correia-Silva, Rodrigo F. Berriel, Claudine Badue, Alberto F. de Souza, and Thiago Oliveira-Santos. Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2018. doi: 10.1109/ijcnn.2018.8489592. URL <http://dx.doi.org/10.1109/IJCNN.2018.8489592>.
- [9] MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL. El reglamento europeo de ia, en resumen, 2022. https://portal.mineco.gob.es/es-es/digitalizacionIA/sandbox-IA/Documents/20220919_Resumen_detallado_Reglamento_IA.pdf.

- [10] Una regulación europea de la inteligencia artificial adecuada y sin fronteras, 2022. [://www.telefonica.com/es/sala-comunicacion/blog/](http://www.telefonica.com/es/sala-comunicacion/blog/).
- [11] Luca Debiasi. *Exploiting Image Sensor Data in Biometric Systems and Mobile Applications*. PhD thesis, Department of Computer Sciences, University of Salzburg, Austria, March 2020.
- [12] Qian Huang, Xiaotong Guo, Yiming Wang, Huashan Sun, and Lijie Yang. A survey of feature matching methods. *IET Image Processing*, 18, 02 2024. doi: 10.1049/ipr2.13032.
- [13] Contenido y Tecnologías (Comisión Europea) Comisión Europea, Dirección General de Redes de Comunicación. Libro blanco sobre la inteligencia artificial - un enfoque europeo orientado a la excelencia y la confianza. Technical report, Comisión Europea, 2020.
- [14] Juergen Schmidhuber. Annotated history of modern ai and deep learning, 2022.
- [15] Gottfried Wilhelm Leibniz. *The early mathematical manuscripts of Leibniz*. Courier Corporation, 2012.
- [16] Claude Lemaréchal. Cauchy and the gradient method. *Doc Math Extra*, 251 (254):10, 2012.
- [17] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.
- [18] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [19] Eric B Baum. On the capabilities of multilayer perceptrons. *Journal of complexity*, 4(3):193–215, 1988.
- [20] Walter J Karplus. 1967 index ieee transactions on electronic computers vol. ec-16. *IEEE Transactions on Electronic Computers*, EC-16(6):913–932, 1967.
- [21] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning internal representations by error propagation, 1985.
- [22] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Cornell Aeronautical Laboratory. Report no. VG-1196-G-8. Spartan Books, 1962. URL
<https://books.google.es/books?id=7FhRAAAAMAAJ>.
- [23] P J Werbos. Applications of advances in nonlinear sensitivity analysis, Jan 1982.

- [24] Yann LeCun. A learning scheme for asymmetric threshold networks. *Proceedings of COGNITIVA*, 85(537):599–604, 1985.
- [25] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985. ISSN 0364-0213. doi: [https://doi.org/10.1016/S0364-0213\(85\)80012-4](https://doi.org/10.1016/S0364-0213(85)80012-4). URL <https://www.sciencedirect.com/science/article/pii/S0364021385800124>.
- [26] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1):23–63, 1987.
- [27] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- [28] Jürgen Schmidhuber. Habilitation thesis: System modeling and optimization. *ff demonstrates credit assignment across the equivalent of*, 1:150, 1993.
- [29] Jürgen Schmidhuber and AI Blog. Unsupervised neural networks fight in a minimax game. *Juergen Schmidhuber's AI Blog*, 1990s.
- [30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997. URL <https://api.semanticscholar.org/CorpusID:1915014>.
- [31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [32] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.
- [33] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.
- [34] Jean-Arcady Meyer and Stewart W. Wilson. *A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers*, pages 222–227. The MIT Press, 1991.
- [35] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.
- [36] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

- [37] J. C. Cuevas-Tello. Apuntes de redes neuronales artificiales, 2018.
- [38] Bernard Widrow. An adaptive “adaline” neuron using chemical “memistors”, 1553-1552, 1960.
- [39] K Fukushima. Neural network model for pattern recognition mechanism not affected by position deviation-neocognitron. *Trans.(A) IECE, Japan*, pages 658–665, 1979.
- [40] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.
- [41] J urgen Schmidhuber. Learning factorial codes by predictability minimization (compact version of tr cu-cs-565-91). *Juergen Schmidhuber's AI Blog*, 1991.
- [42] AWS. ¿qué es la ciencia de datos?
<https://aws.amazon.com/es/what-is/data-science/>, 2023. [Online; accessed 31-January-2024].
- [43] IBM. ¿qué es el deep learning?
<https://www.ibm.com/es-es/topics/deep-learning>, 2023. [Online; accessed 1-February-2024].
- [44] A. Géron. *Neural Networks and Deep Learning*. O'Reilly, 2018. URL <https://books.google.es/books?id=5pm6tQEACAAJ>.
- [45] IBM. ¿qué es el descenso de gradiente?
<https://www.ibm.com/mx-es/topics/gradient-descent>, 2023. [Online; accessed 10-February-2024].
- [46] Pytorch. <https://github.com/pytorch/pytorch>, 2024.
- [47] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. URL <https://arxiv.org/abs/1312.6114>.
- [48] @alihamzaaliasia. Autoencoder — vq-vae y el comienzo de la generacion de imagenes como stable diffusion y dalle. (con codigo funcional).
<https://medium.com/@alihamzaaliasia/be49bbf9dca2>, 2024. [Online; accessed 10-September-2024].
- [49] Jordi de la Torre. Redes generativas adversarias (gan) fundamentos teóricos y aplicaciones. *arXiv preprint arXiv:2302.09346*, 2023.
- [50] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

- [51] Yufeng Zheng, Yunkai Zhang, and Zeyu Zheng. Continuous conditional generative adversarial networks (cgan) with generator regularization, 2021.
- [52] Xiaomin Li, Anne Hee Hiong Ngu, and Vangelis Metsis. Tts-cgan: A transformer time-series conditional gan for biosignal data augmentation, 2022.
- [53] Saul Dobilas Towards Data Science. cgan: Conditional generative adversarial network — how to gain control over gan outputs. [cGAN: Conditional Generative Adversarial Network - How to Gain Control Over GAN Outputs](#), 2022.
- [54] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.
- [55] Jairo Viola, YangQuan Chen, and Jing Wang. Faultface: Deep convolutional generative adversarial network (dcgan) based ball-bearing failure detection method, 2020.
- [56] Süleyman Aslan, Uğur Gündükay, B. Uğur Töreyin, and A. Enis Çetin. Deep convolutional generative adversarial networks based flame detection in video, 2019.
- [57] J. D. Curtó, I. C. Zarza, Fernando de la Torre, Irwin King, and Michael R. Lyu. High-resolution deep convolutional generative adversarial networks, 2020.
- [58] Augustus Odena. Semi-supervised learning with generative adversarial networks, 2016.
- [59] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks, 2016.
- [60] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.
- [61] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.
- [62] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.
- [63] Caleb Eunho Lee. An overview on cyclegan — unpaired image-to-image translation. <https://calebelee05.medium.com/7a1c1024473e>, 2021. [Online; accessed 3-March-2024].
- [64] Sebastian Theiler. The beauty of cyclegan.
<https://medium.com/analytics-vidhya/c51c153493b8>, 2019. URL
<https://medium.com/analytics-vidhya/c51c153493b8>. [Online; accessed 6-March-2024].

- [65] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.
- [66] Archit Rathore Hardik Bansal. Understanding and implementing cyclegan in tensorflow. <https://hardikbansal.github.io/CycleGANBlog/>, 2019. URL <https://hardikbansal.github.io/CycleGANBlog/>. [Online; accessed 7-March-2024].
- [67] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018.
- [68] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2017.
- [69] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, and Xiaoou Tang. Esrgan: Enhanced super-resolution generative adversarial networks, 2018.
- [70] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.
- [71] Axel Sauer, Tero Karras, Samuli Laine, Andreas Geiger, and Timo Aila. Stylegan-t: Unlocking the power of gans for fast large-scale text-to-image synthesis, 2023.
- [72] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks, 2021.
- [73] Andrew Melnik, Maksim Miasayedzenkau, Dzianis Makarovets, Dzianis Pirshuk, Eren Akbulut, Dennis Holzmann, Tarek Renusch, Gustav Reichert, and Helge Ritter. Face generation and editing with stylegan: A survey, 2023.
- [74] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks, 2018.
- [75] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.
- [76] Lilian Weng. From gan to wgan. [lilianweng.github.io](https://lilianweng.github.io/posts/2017-08-20-gan/), 2017. URL <https://lilianweng.github.io/posts/2017-08-20-gan/>.
- [77] Lilian Weng. From gan to wgan, 2019.
- [78] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

- [79] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets, 2016.
- [80] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing, 2020.
- [81] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox v1.2.0. *CoRR*, 1807.01069, 2018. URL <https://arxiv.org/pdf/1807.01069>.
- [82] Will Schroeder. Learning machine learning part 3: Attacking black box models. posts.specterops.io/learning-machine-learning-part-3, 2022. [Online; accessed 30-January-2024].
- [83] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.
- [84] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines, 2013.
- [85] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain, 2019.
- [86] Yize Cheng, Wenbin Hu, and Minhao Cheng. Attacking by aligning: Clean-label backdoor attacks on object detection, 2023.
- [87] Te Juin Lester Tan and Reza Shokri. Bypassing backdoor detection algorithms in deep learning, 2020.
- [88] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks, 2019.
- [89] Hojjat Aghakhani, Dongyu Meng, Yu-Xiang Wang, Christopher Kruegel, and Giovanni Vigna. Bullseye polytope: A scalable clean-label poisoning attack with improved transferability, 2021.
- [90] Ambrish Rawat, Killian Levacher, and Mathieu Sinn. The devil is in the gan: Backdoor attacks and defenses in deep generative models, 2022.
- [91] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks, 2018.

- [92] Jonas Geiping, Liam Fowl, W. Ronny Huang, Wojciech Czaja, Gavin Taylor, Michael Moeller, and Tom Goldstein. Witches' brew: Industrial scale data poisoning via gradient matching, 2021.
- [93] Shih-Han Chan, Yinpeng Dong, Jun Zhu, Xiaolu Zhang, and Jun Zhou. Baddet: Backdoor attacks on object detection, 2022.
- [94] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.
- [95] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.
- [96] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks, 2020.
- [97] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1322–1333, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450338325. doi: 10.1145/2810103.2813677. URL <https://doi.org/10.1145/2810103.2813677>.
- [98] Christopher A. Choquette-Choo, Florian Tramer, Nicholas Carlini, and Nicolas Papernot. Label-only membership inference attacks, 2021.
- [99] Wei-An Lin, Yogesh Balaji, Pouya Samangouei, and Rama Chellappa. Invert and defend: Model-based approximate inversion of generative adversarial networks for secure inference, 2019. URL <https://arxiv.org/abs/1911.10291>.
- [100] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models, 2018. URL <https://arxiv.org/abs/1805.06605>.
- [101] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. Exploring connections between active learning and model extraction, 2019. URL <https://arxiv.org/abs/1811.02054>.
- [102] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning (ICML)*, 2018. URL <https://www.icml.cc/Conferences/2018/Schedule?showEvent=2477>.

- [103] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach, 2018.
- [104] Jeremy M Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified adversarial robustness via randomized smoothing, 2019.
- [105] Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane Boning, and Cho-Jui Hsieh. Robustness verification of tree-based models, 2019.
- [106] cisco. What is a cyberattack? - most common types - cisco.
<https://www.cisco.com/c/en/us/products/security/common-cyberattacks.html#~types-of-cyber-attacks>, 2022. [Online; Accessed on 05/29/2024].
- [107] L Roberts. Machine perception of 3-d solids, optical and electro-optical information processing, 1965.
- [108] Sigurd Angenent, Eric Pichon, and Allen Tannenbaum. Mathematical methods in medical image processing. *Bulletin of the American mathematical society*, 43(3):365–396, 2006.
- [109] Güner S Robinson. Edge detection by compass gradient masks. *Computer graphics and image processing*, 6(5):492–501, 1977.
- [110] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, 0(6):679–698, 1986.
- [111] Rachid Deriche. Using canny’s criteria to derive a recursively implemented optimal edge detector. *International journal of computer vision*, 1(2):167–187, 1987.
- [112] Tony Lindeberg. Edge detection and ridge detection with automatic scale selection. *International journal of computer vision*, 30:117–156, 1998.
- [113] Motilal Agrawal, Kurt Konolige, and Morten Rufus Blas. Censure: Center surround extrema for realtime feature detection and matching. In *European conference on computer vision*, pages 102–115. Springer, 2008.
- [114] Jules R Dim and Tamio Takamura. Alternative approach for satellite cloud classification: edge gradient application. *Advances in Meteorology*, 2013(1):584816, 2013.
- [115] Irwin Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

- [116] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, 1988.
- [117] Jianbo Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994. doi: 10.1109/CVPR.1994.323794.
- [118] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1508–1515. ieee, 2005.
- [119] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pages 430–443. Springer, 2006.
- [120] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999. doi: 10.1109/ICCV.1999.790410.
- [121] Pablo F Alcantarilla and T Solutions. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Trans. Patt. Anal. Mach. Intell.*, 34(7):1281–1298, 2011.
- [122] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.
- [123] Michael Donoser and Horst Bischof. Efficient maximally stable extremal region (mser) tracking. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 1, pages 553–560. ieee, 2006.
- [124] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. ieee, 2005.
- [125] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 27(10):1615–1630, 2005.
- [126] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pages 404–417. Springer, 2006.

- [127] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008. ISSN 1077-3142. doi: <https://doi.org/10.1016/j.cviu.2007.09.014>. URL <https://www.sciencedirect.com/science/article/pii/S1077314207001555>. Similarity Matching in Computer Vision and Multimedia.
- [128] Alaa E Abdel-Hakim and Aly A Farag. Csift: A sift descriptor with color invariant characteristics. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 2, pages 1978–1983. ieee, 2006.
- [129] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE transactions on pattern analysis and machine intelligence*, 32(5):815–830, 2009.
- [130] Relja Arandjelović and Andrew Zisserman. Three things everyone should know to improve object retrieval. In *2012 IEEE conference on computer vision and pattern recognition*, pages 2911–2918. IEEE, 2012.
- [131] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5–11, 2010, Proceedings, Part IV 11*, pages 778–792. Springer, 2010.
- [132] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *2011 International conference on computer vision*, pages 2548–2555. ieee, 2011.
- [133] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. ieee, 2011.
- [134] Benjamin S. Riggan, Nathan J. Short, Shuowen Hu, Afsane Ghasemi, Simon Denman, Sridha Sridharan, Clinton Fookes, Xiang Yu, Jianchao Yang, Linjie Luo, Wilmot Li, Jonathan Brandt, Dimitris N. Metaxas, David Chan, Mohammad H. Mahoor, Tejas I. Dhamecha, Praneet Sharma, Richa Singh, Mayank Vatsa, Afzel Noore, Jun-Cheng Chen, Vishal M. Patel, Rama Chellappa, Soumyadip Sengupta, Carlos Domingo Castillo, and David W. Jacobs. 2016 ieee winter conference on applications of computer vision, wacv 2016, lake placid, ny, usa, march 7-10, 2016. In *IEEE Workshop/Winter Conference on Applications of Computer Vision*, 2016. URL <https://api.semanticscholar.org/CorpusID:42536071>.

- [135] OpenCV. Basics of brute-force matcher.
https://docs.opencv.org/4.10.0/dc/dc3/tutorial_py_matcher.html, 2024.
[Online; accessed 30-September-2024].
- [136] Relja Arandjelovic. Three things everyone should know to improve object retrieval. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, page 2911–2918, USA, 2012. IEEE Computer Society. ISBN 9781467312264.
- [137] Zhiyu Pan, Yongjie Duan, Jianjiang Feng, and Jie Zhou. Fixed-length dense descriptor for efficient fingerprint matching, 2024. URL
<https://arxiv.org/abs/2311.18576>.
- [138] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *CVPR Deep Learning for Visual SLAM Workshop*, 2018. URL
<http://arxiv.org/abs/1712.07629>.
- [139] Axel Barroso-Laguna, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Key.Net: Keypoint Detection by Handcrafted and Learned CNN Filters. In *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision*, 2019.
- [140] Mihai Dusmanu, Ignacio Rocco, Tomas Pajdla, Marc Pollefeys, Josef Sivic, Akihiko Torii, and Torsten Sattler. D2-Net: A Trainable CNN for Joint Detection and Description of Local Features. In *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.
- [141] Imane Salhi, Martyna Poreba, Erwan Piriou, Gouet-Brunet Valerie, and Maroun Ojail. Multimodal localization for embedded systems: A survey. *Multimodal Scene Understanding: Algorithms, Applications and Deep Learning*, pages 199–278, 1 2019. doi: 10.1016/B978-0-12-817358-9.00014-7.
- [142] Francois Chollet et al. Keras, 2015. URL
<https://github.com/fchollet/keras>.
- [143] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals,

Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow, Large-scale machine learning on heterogeneous systems, November 2015.

- [144] Frank Seide and Amit Agarwal. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 2135, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2945397. URL <https://doi.org/10.1145/2939672.2945397>.
- [145] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024. doi: 10.1145/3620665.3640366. URL <https://pytorch.org/assets/pytorch2-2.pdf>.
- [146] William Falcon and The PyTorch Lightning team. PyTorch Lightning, March 2019. URL <https://github.com/Lightning-AI/lightning>.
- [147] ticportal. Inteligencia artificial para software empresarial. <https://www.ticportal.es/glosario-tic/inteligencia-artificial-software-empresarial>, 2022. [Online; accessed 30-January-2024].
- [148] ticportal. Deep learning: ¿se pueden programar las máquinas para pensar como humanos? <https://www.ticportal.es/glosario-tic/deep-learning-dl>, 2023. [Online; accessed 30-January-2024].
- [149] Bill Wilson. The machine learning dictionary. <https://www.cse.unsw.edu.au/~billw/mldict.html>, 2012.

- [150] ticportal. Machine learning: ¿cómo aprenden las máquinas a trabajar por su cuenta? <https://www.ticportal.es/glosario-tic/machine-learning>, 2022. [Online; accessed 30-January-2024].
- [151] SE Fahlman. Proceedings of the 1988 connectionist models summer school. In *Faster-learning variations on back-propagation: An empirical study*, 1988.
- [152] Juergen Schmidhuber. Generative adversarial networks are special cases of artificial curiosity (1990) and also closely related to predictability minimization (1991), 2020.
- [153] El señor de las gan: el hombre que dio imaginación a las máquinas, 2014. URL <https://www.technologyreview.es/s/10016/el-señor-de-las-gan-el-hombre-que-dio-imaginacion-las-maquinas>.
- [154] J. M. Child. The manuscripts of leibniz on his discovery of the differential calculus. part ii (continued). *The Monist*, 27(3):411–454, 1917. ISSN 00269662. URL <http://www.jstor.org/stable/27900650>.
- [155] Will Schroeder. Learning machine learning part 1: Introduction and revoke-obfuscation. posts.specterops.io/learning-machine-learning-part-1, 2022. [Online; accessed 30-January-2024].
- [156] Will Schroeder. Learning machine learning part 2: Attacking white box models. posts.specterops.io/learning-machine-learning-part-2, 2022. [Online; accessed 30-January-2024].
- [157] Jing Lin, Long Dang, Mohamed Rahouti, and Kaiqi Xiong. MI attack models: Adversarial attacks and data poisoning attacks, 2021.
- [158] Bill Wilson. The ai dictionary.
<https://www.cse.unsw.edu.au/~billw/aidict.html>, 2012.
- [159] Mahmoud Afifi. 11k hands: gender recognition and biometric identification using a large dataset of hand images. *Multimedia Tools and Applications*, 2019. doi: 10.1007/s11042-019-7424-8. URL <https://doi.org/10.1007/s11042-019-7424-8>.
- [160] Wenxiu Diao, Feng Zhang, Jiande Sun, Yinghui Xing, Kai Zhang, and Lorenzo Bruzzone. Zergan: Zero-reference gan for fusion of multispectral and panchromatic images. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8195–8209, 2023. doi: 10.1109/TNNLS.2021.3137373.

- [161] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, 8(2):179–187, 1962. doi: 10.1109/TIT.1962.1057692.
- [162] Rafael M. Luque-Baena, David Elizondo, Ezequiel López-Rubio, Esteban J. Palomo, and Tim Watson. Assessment of geometric features for individual identification and verification in biometric hand systems. *Expert Systems with Applications*, 40(9):3580–3594, 2013. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2012.12.065>. URL <https://www.sciencedirect.com/science/article/pii/S0957417412013061>.
- [163] Redacción KeepCoding. Visualización de activaciones y filtros en red convolucional. <https://keepcoding.io/>, 2022. [Online; accessed 20-February-2024].
- [164] Jie Li, Yongli Ren, and Ke Deng. Fairgan: Gans-based fairness-aware learning for recommendations with implicit feedback. In *Proceedings of the ACM Web Conference 2022*, pages 297–307, 2022.
- [165] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. On convergence and stability of gans, 2017.
- [166] Haichao Shi, Jing Dong, Wei Wang, Yinlong Qian, and Xiaoyu Zhang. Ssgan: Secure steganography based on generative adversarial networks, 2018.
- [167] Jiqing Wu, Zhiwu Huang, Janine Thoma, Dinesh Acharya, and Luc Van Gool. Wasserstein divergence for gans, 2018.
- [168] Zhipeng Cai, Zuobin Xiong, Honghui Xu, Peng Wang, Wei Li, and Yi Pan. Generative adversarial networks: A survey toward private and secure applications. *ACM Comput. Surv.*, 54(6), jul 2021. ISSN 0360-0300. doi: 10.1145/3459992. URL <https://doi.org/10.1145/3459992>.
- [169] Guillermo Iglesias, Edgar Talavera, and Alberto Díaz-Álvarez. A survey on gans for computer vision: Recent research, analysis and taxonomy. *Computer Science Review*, 48:100553, May 2023. ISSN 1574-0137. doi: 10.1016/j.cosrev.2023.100553. URL <http://dx.doi.org/10.1016/j.cosrev.2023.100553>.
- [170] Sertis. Gan inversion: A brief walkthrough — part i. <https://sertiscorp.medium.com/gan-inversion-a-brief-walkthrough-part-i-bc2ee1b73253>, 11 2021. [Online; accessed 12-March-2024].

- [171] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks, 2020.
- [172] M. Brown, R. Szeliski, and Simon Winder. Multi-image matching using multi-scale oriented patches. In *Multi-image matching using multi-scale oriented patches*, volume 1, pages 510– 517 vol. 1, 07 2005. ISBN 0-7695-2372-2. doi: 10.1109/CVPR.2005.235.
- [173] Hans Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, Carnegie Mellon University, Pittsburgh, PA, September 1980.
- [174] Menghan Li, Bin Huang, and Guohui Tian. A comprehensive survey on 3d face recognition methods. *Engineering Applications of Artificial Intelligence*, 110: 104669, 2022. ISSN 0952-1976. doi: <https://doi.org/10.1016/j.engappai.2022.104669>. URL <https://www.sciencedirect.com/science/article/pii/S0952197622000057>.
- [175] Anastasiya Mishchuk, Dmytro Mishkin, Filip Radenovic, and Jiri Matas. Working hard to know your neighbor’s margins: Local descriptor learning loss. In *Proceedings of NeurIPS*, dec 2017.
- [176] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020. doi: 10.1038/s41586-020-2649-2.
- [177] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.
- [178] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.
- [179] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuel Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.