UNIVERSIDAD REY JUAN CARLOS

MÁSTER EN DATA SCIENCE

TRABAJO FIN DE MÁSTER


# Análisis Convolucional de la Obra de Francisco de Goya

Pablo Olmos Martínez

Director

Víctor C. Aceña Gil

Madrid, 2023

Resumen

Este trabajo tiene como objetivo analizar y agrupar la obra del famoso artista español Francisco de Goya utilizando técnicas de aprendizaje automático. Para llevar a cabo esta investigación, se realizó un web scraping de la plataforma WikiArt para obtener un conjunto de imágenes de la obra de Goya. A continuación, se extrajeron características de las imágenes utilizando una red neuronal convolucional llamada ResNet18. Una vez obtenidas estas características, se crearon clústers utilizando un algoritmo de agrupamiento y se realizó una visualización de estos. Los resultados muestran que es posible utilizar técnicas de aprendizaje automático para analizar y agrupar la obra de Goya de manera efectiva, proporcionando una nueva perspectiva y una mayor comprensión de su trabajo.

Palabras Clave: Francisco de Goya, Aprendizaje automático, Red neuronal convolucional, ResNet18, Web scraping, Análisis de imágenes, Agrupamiento, Visualización.

Abstract

This work aims to analyze and group the work of the famous Spanish artist Francisco de Goya using machine learning techniques. To carry out this research, a web scraping of the WikiArt platform was carried out to obtain a set of images of Goya's work. Then, Features of the images were extracted using a convolutional neural network called ResNet18. Once these Features were obtained, clusters were created using a clustering algorithm and a visualization of them was performed. The results show that it is possible to use machine learning techniques to effectively analyze and group Goya's work, providing a new perspective and greater understanding of his work.

Key Words: Francisco de Goya, Machine Learning, Convolutional Neural Network, ResNet18, Web Scraping, Image Analysis, Clustering, Visualization.

## Índice General

## Índice de figuras

## Índice de Tablas



## Índice de Ecuaciones

# 1. Introducción y objetivos.

En este capítulo, se proporcionará una visión detallada del contexto en el que se desarrolla este trabajo. En primer lugar, se presentará una breve introducción a los modelos de redes neuronales convolucionales, algoritmos de clasificación, gráficos y visualización de datos. A continuación, se plantearán los objetivos del trabajo y se revisará la estructura de trabajo que se seguirá a lo largo del TFM, indicando de forma esquemática las tecnologías utilizadas y los objetivos de estas.

Las redes neuronales convolucionales (*CNN*, por sus siglas en inglés) han experimentado un gran avance y desarrollo en el análisis y reconocimiento de imágenes en los últimos años, y su aplicación va desde sistemas de vigilancia y reconocimiento facial hasta aplicaciones médicas preventivas.

El objetivo de este TFM es el estudio e implementación de una *CNN* que permita reconocer y clasificar la obra pictórica de Francisco de Goya disponible en webs públicas [1] Por lo tanto, se centrará en la consecución de los siguientes objetivos:

- Poner en contexto los modelos de aprendizaje profundo y aprendizaje automático, así como el uso de distintos algoritmos en el desarrollo del proyecto, como el web scraping, *t-sne*, *clustering*, gráficos y visualización web.
- Presentar los experimentos realizados y exponer las visualizaciones correspondientes a dichos experimentos.
- Proporcionar conclusiones obtenidas a lo largo de los procesos llevados a cabo en los puntos anteriores y plantear futuros trabajos.

En nuestro desarrollo, nos hemos basado en el trabajo de Google, que ofrece un recorrido visual en tres dimensiones basado en *t-sne* y clusterizado en puntos de interés.[2] . La obra del famoso artista español Francisco de Goya es una fuente inagotable de inspiración y análisis para el mundo artístico y cultural. Su estilo y técnica únicos han sido estudiados y admirados durante siglos, y su trabajo sigue siendo relevante y significativo en la actualidad.

El uso de técnicas de aprendizaje automático en el análisis de la obra de Goya ofrece la oportunidad de profundizar en la comprensión y valoración de su trabajo de una manera nueva y diferente. Las redes neuronales convolucionales, en particular, han demostrado ser una herramienta eficaz para el análisis y la clasificación de imágenes, por lo que su uso en este contexto puede proporcionar resultados interesantes y significativos. Además, la creación de clústers y la visualización de la obra de Goya mediante técnicas de aprendizaje automático puede proporcionar una nueva perspectiva y una mayor comprensión de su trabajo. Esto puede ser de gran utilidad tanto para expertos en el campo como para aquellos que deseen conocer más sobre la obra de Goya.

En primer lugar, el uso de tecnología y técnicas de aprendizaje automático en el análisis de la obra de Goya puede resultar atractivo y motivador para un público joven, ya que proporciona un enfoque innovador y moderno para su valoración. Además, el uso de gráficos y visualizaciones para mostrar los resultados del análisis puede hacer que el proceso sea más comprensible y atractivo para un público no especializado.

Este trabajo puede ser utilizado como una herramienta pedagógica para enseñar a jóvenes estudiantes sobre técnicas de aprendizaje automático y análisis de imágenes. La realización de proyectos de este tipo puede ser una forma efectiva de motivar a los estudiantes y proporcionarles una oportunidad de aplicar lo que han aprendido en un contexto práctico y significativo. Además, este trabajo también puede ayudar en la difusión de la obra de Goya para un público joven y no especializado, ayudando a aumentar su conocimiento y aprecio por el artista y su legado. En resumen, el uso de técnicas de aprendizaje automático en el análisis de la obra de Francisco de Goya no solo proporciona una nueva perspectiva y una mayor comprensión de su trabajo, sino que también puede ser utilizado como una herramienta pedagógica y para difusión de su obra.

En resumen, la motivación detrás de este TFM es explorar el uso de técnicas de aprendizaje automático para analizar y agrupar la obra de Goya de manera efectiva, proporcionando una nueva perspectiva y una mayor comprensión de su trabajo a través de la visualización de los resultados obtenidos.

# 2. Estado del Arte.

En este Capítulo, se realiza un resumen del estado actual sobre el Aprendizaje Automático (ML, por sus siglas en inglés) y el Aprendizaje Profundo (DL, por sus siglas en inglés), todos los algoritmos empleados a lo largo del TFM.

## 2.1. Aprendizaje Automático & Aprendizaje Profundo.

El ML es un campo de las ciencias de la computación y una rama de las Inteligencia Artificial (IA) cuyo principal objetivo es desarrollar técnicas que permitan a las computadoras aprender.

Tipos de Aprendizaje Automático:

- Aprendizaje Supervisado.
  - Regresión.
  - Clasificación.
- Aprendizaje No Supervisado.
  - Clustering
  - Reducción de Dimensiones
- Aprendizaje Semi-Supervisado.
- Aprendizaje por Refuerzo.

El aprendizaje Supervisado en ML es aquel que se aplica cuando cada dato o conjunto de datos está asociado a una etiqueta, el No supervisado utiliza información No etiquetada. “El algoritmo más empleado en ML no Supervisado es el *Clustering*, cuyo objetivo es agrupar muestras, así como la reducción de la dimensionalidad [28].”

“El DL es un campo del ML englobado a su vez en la Inteligencia Artificial donde en contraste con el enfoque tradicional del ML las técnicas de Aprendizaje Profundo se encargan de extraer por sí solas, las características (Características) más relevantes para solucionar un problema, así como las transformaciones para la salida esperada [29].”

En la ilustración 1 mostramos la comparativa entre el aprendizaje automático y el aprendizaje profundo en términos de diferencias de diseño y estructura.

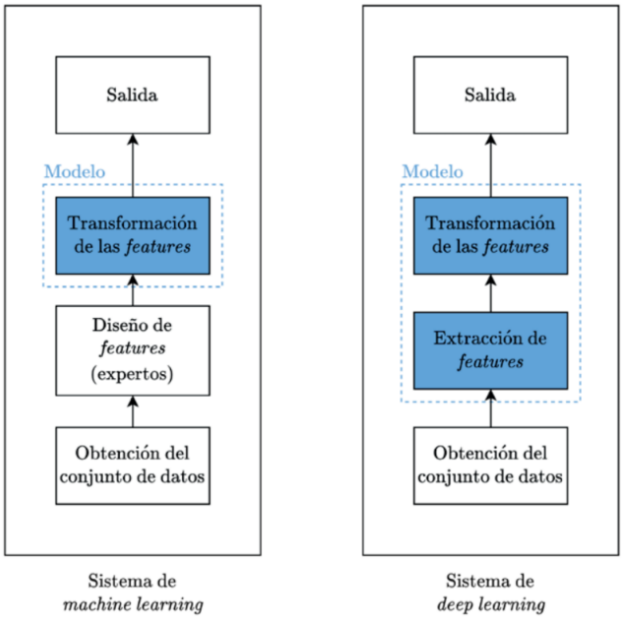



**Ilustración 1. Comparativa Aprendizaje Automático y Profundo**

## 2.2. WebScraping.

Es una técnica empleada para extraer información de los sitios webs y basado en el Modelo Cliente Servidor.

Para el desarrollo del módulo de *web scraping*, se ha realizado un estudio de las diferentes páginas web de arte que contienen la información necesaria para el cumplimiento del proyecto. La web está en *HTML*, lenguaje informático que permite la creación y estructuración de secciones, párrafos y enlaces mediante etiquetas y atributos.

Antes de la implementación del desarrollo realizamos un análisis sobre la legalidad de obtener información de estos sitios web en relación con el fin del proyecto. Una vez que decidimos la web sobre la que íbamos a extraer las pinturas y que la legalidad fue validada desarrollamos las implementaciones a desarrollar:

- Primera, peticiones al navegador (Google Chrome) basado en programación *HTTP* de los servidores web, que nos permite extraer la información de la petición que vayamos a realizar.
- Segunda, una vez extraída la información, navegaríamos a través de un árbol que sería el objeto resultante después de parsear el documento *HTML* de entrada, lo cual nos permite interactuar con todos los elementos de una Web. Podemos diferenciar fundamentalmente: o El Árbol, que representa un objeto resultante de *parsear* el documento *HTML* de entrada. Nos permite navegar a través del objeto *Tag*, este objeto se corresponde con una etiqueta *HTML*. Podemos acceder a sus atributos tratando al objeto como si fuera un diccionario. Un Objetos *Tag* puede contener otros objetos *Tag* o *NavigableString*.

Los desarrollos correspondientes a esta parte se han llevado a cabo utilizando el software libre Python 3.9, seleccionado principalmente por el amplio número de librerías de las que dispone. Concretamente se han utilizado *BeautifulSoup* y *Request* [4] :

- La librería *Request* nos permite enviar peticiones *HTTP* de manera sencilla obteniendo todo el código *HTML* de la página web a la que se ha lanzado la petición *HTTP*.
- La Librería *BeautifulSoup* nos permite la extracción de datos de archivos *HTML*. La función principal de la librería es analizar la estructura del archivo y la creación de un árbol de contenido por el que podemos navegar y extraer toda la información que necesitemos de una forma rápida y sencilla.

## 2.3. Tratamiento de Imágenes.

### 2.3.1. Funciones Convexas y Funciones No Convexas.

"El Error Cuadrático Medio (*MSE, Mean Squared Error*) es una función de coste comúnmente utilizada en la regresión lineal. Se calcula como la media de la diferencia entre lo que predice el modelo y los valores reales, elevada al cuadrado. Es una función convexa, lo que significa que tiene un único punto de mínimo global y siempre se asegura de que el modelo se ajuste a los datos de forma óptima [5]."

En la ilustración 2 mostramos un ejemplo de un modelo de regresión lineal simple y cómo se utiliza para entender las funciones de coste. Esta ilustración puede mostrar cómo se ajusta el modelo a los datos de entrenamiento y cómo se utiliza la función de coste para medir la precisión del modelo.

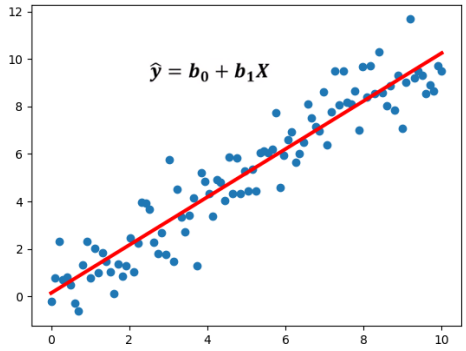



**Ilustración 2. Regresión Lineal Simple**

**[Ilustración2]**

Mostramos la Ecuación de la función del Error Cuadrático Medio (ECM) y cómo se utiliza para medir la precisión del modelo. Esta ilustración puede mostrar cómo se calcula el ECM, utilizando la diferencia entre los valores predichos por el modelo y los valores reales, y cómo se utiliza para minimizar el error y ajustar el modelo a los datos de entrenamiento.

$$ECM = \frac{1}{n} \sum_{i=1}^{n} (\widehat{Yi} - Yi)^2$$

**Ecuación 1. Error Cuadrático Medio**

Una función de coste convexa es una función matemática cuya curva es siempre cóncava hacia abajo, lo que significa que tiene un único mínimo global. Esto hace que sea más fácil encontrar el valor óptimo de los parámetros al optimizar la función.

En el contexto del aprendizaje automático, una función de coste convexa se utiliza para medir el error o "coste" entre los valores predichos por un modelo y los valores reales. La optimización de la función de coste se refiere al proceso de encontrar los valores óptimos de los parámetros del modelo que minimizan el coste.

Para minimizar la función de coste calculamos las derivadas parciales y las igualamos a cero y podemos afirmar que en el caso de encontrar un mínimo este será el mínimo global de la función.

En la ilustración 4 mostramos una función de coste convexa que nos ayuda a entender cómo una función convexa tiene un único punto de mínimo global, lo que significa que siempre se asegura de que el modelo se ajuste a los datos de forma óptima.

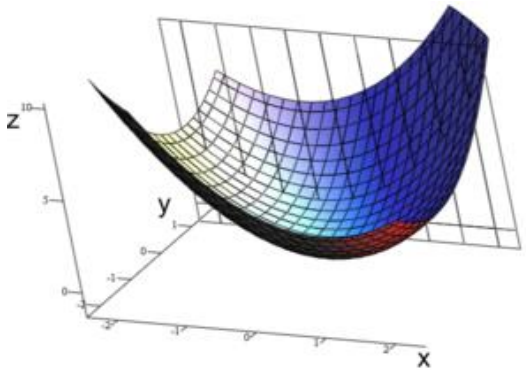



**Ilustración 3. Función de Coste Convexa**

**[Ilustración3]**

Las funciones no convexas son aquellas que tienen múltiples puntos mínimos, en lugar de un único mínimo global. Esto se debe a que la curva de la función no es siempre cóncava hacia abajo, sino que puede tener varios puntos donde la pendiente es cero.

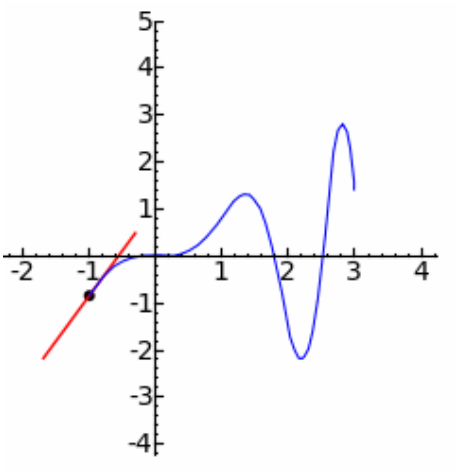



**Ilustración 4. Función de Coste No Convexa**

**[Ilustración4]**

Para encontrar los mínimos de una función no convexa, se utilizan técnicas de optimización numérica, como el gradiente descendiente, que consiste en iterativamente ir actualizando los valores de los parámetros para encontrar el punto donde la función alcanza su mínimo. Sin embargo, debido a la naturaleza no convexa de estas funciones, es posible que el algoritmo se quede atrapado en un mínimo local en lugar de alcanzar el mínimo global. Existen varios métodos para tratar de evitar este problema, como el uso de técnicas de optimización estocástica, que agrega cierto ruido al proceso de optimización para evitar quedarse atrapado en un mínimo local.

"Lo que hacemos con la derivada es igualar la pendiente a cero para resolver donde la pendiente es nula f '(x) = 0 En las funciones no convexas podemos encontrarnos con múltiples puntos mínimos y por tanto múltiples ecuaciones que resolver, máximos, mínimos, puntos sillas [6]."

En la ilustración 6 mostramos una función de coste no convexa 3d que nos ayuda a entender como una función no convexa tiene varios puntos de mínimos locales, lo que significa que el modelo no se ajusta necesariamente a los datos de forma óptima y puede llegar a un punto de optimización no deseado.

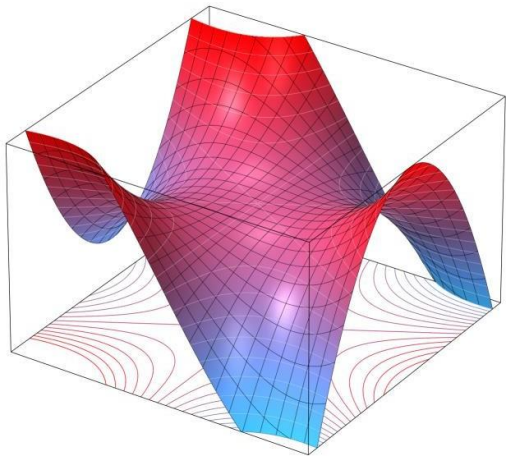


**Ilustración 5. Función de Coste No Convexa 3D**

**[Ilustración5]**

### 2.3.2. Gradient Descent.

Si aumentamos el número de parámetros en nuestro modelo la complejidad computacional se eleva notablemente por lo que necesitamos métodos iterativos que optimicen el cálculo de la función de coste como es el *Gradient Descent*.

Pretendemos calcular gradiente de la función de coste, es decir, la dirección y sentido en el que dicha función aumenta. Por lo que debemos tomar el sentido contrario, sentido donde la función se minimiza y continuar avanzando en ese sentido.

Para calcular el Gradient Descent:

- Iniciamos de un punto aleatorio.
- Calculamos la derivada de la función en dicho punto. Al tener n dimensiones deberán ser derivadas parciales para cada parámetro. Cada uno de estos valores nos dirá la pendiente en el eje de dicho parámetro.
- Todas estas derivadas parciales forman un vector hacia donde la pendiente asciende. Este vector es conocido como el Gradiente.
- Como nuestro objetivo es descender ya que buscamos minimizar la función de coste.
- Esto nos daría un nuevo conjunto de parámetros, posicionándonos en una nueva posición dentro de nuestra función de coste.
- Iteraremos Múltiples veces hasta que detengamos el algoritmo, normalmente dónde ya no suponga una variación notable del coste.
- Tasa de Aprendizaje, cuando avanzamos en cada paso. O cómo afecta el gradiente a la actualización de nuestros parámetros en cada paso.

Consideraciones del Tasa de Aprendizaje (*LR*):

- “Si escogemos un *LR* muy grande convergemos antes hacia la solución, pero corrigiendo el riesgo de la saltarnos mínimos. Pudiendo no converger.

- Si escogemos un *LR* muy pequeño nos aseguramos de converger siempre, pero con un coste de tiempo muy grande [7].”

En la ilustración 7 mostramos el algoritmo de descenso del gradiente (*Gradient Descent Algorithm*). Este algoritmo es uno de los métodos comunes utilizados para optimizar una función de coste y ajustar un modelo a los datos de entrenamiento. La ilustración nos ayuda a entender cómo el algoritmo busca el mínimo de la función de coste, moviéndose en la dirección opuesta al gradiente de la función.

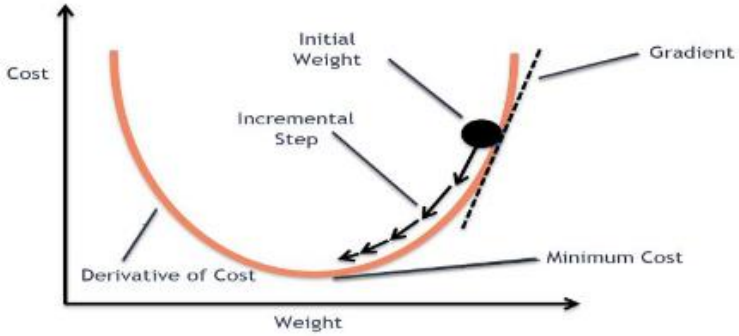



**Ilustración 6. Algoritmo Gradient Descent**

**[Ilustración6]**

### 2.3.3. Redes Neuronales. Función de Activación & Backpropagation.

Una red neuronal artificial es un sistema de conexión de modelos computacionales pertenecientes al Aprendizaje Profundo, donde la clave es cómo están interconectadas las neuronas, llamadas nodos, que cumplen una determinada función en el sistema.

Veamos los tipos de nodos:

- Nodos de entrada: son aquellos que se encargan de recibir los datos desde el exterior. Es la información que la red neuronal tiene que procesar.
- Nodos ocultos: son las capas intermedias de la red neuronal, encargadas de recibir los datos captados por los nodos de entrada y de transmitirlos a lo largo de toda la red neuronal para ser procesados.
- Nodos de salida: son los que reciben la información una vez procesada por las capas intermedias, suelen transformar y procesar esta información para transmitirla.

En la ilustración 8 mostramos el concepto de una red neuronal artificial. Esta ilustración nos ayuda a entender cómo una red neuronal está compuesta por capas de nodos (neuronas) conectados entre sí, y cómo estas capas procesan y transmiten información.

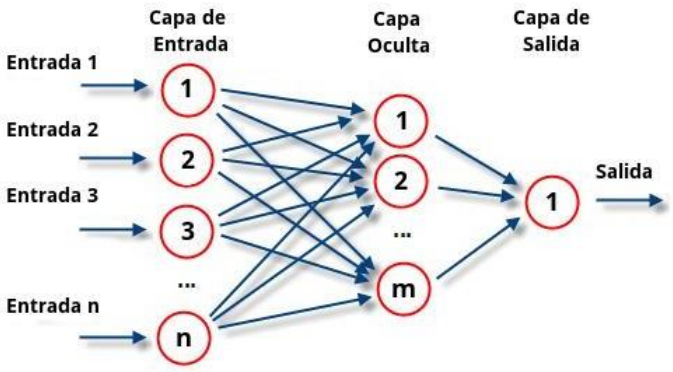



**Ilustración 7. Redes Neuronales Artificiales**

**[Ilustración7]**

### 2.3.3.1. Función de Activación.

La función de activación de un nodo define la salida de un nodo dada una entrada o un conjunto de entradas, con el objetivo de que la red pueda aprender relaciones complejas en los datos. La elección esta función tiene un gran impacto en la capacidad y rendimiento de la red y podemos utilizar diferentes funciones de activación en distintas partes el modelo.

Por lo general las capas intermedias suelen emplear la misma función de activación, suele ser distinto de la capa de salida que está condicionada al tipo de aprendizaje de la red.

Principales características:

- Estas funciones diferenciables, para el cálculo en varias variables del concepto más simple de función derivable, principalmente para el uso del *BackPropagation*.
- Nuestras redes serán entrenadas empleando el *Gradient Descent* y *BackPropagation*, el gradiente de cada capa es afectado por el anterior.
- Debe ser simétrica en Zero, para que los Gradientes no se desplacen en ninguna dirección en particular [8].

Algunas de las más comunes son *ReLU*:

En la ilustración 9 mostramos la función de activación *ReLU (Rectified Linear Unit).* Es una función comúnmente utilizada en las redes neuronales artificiales para aplicar una no-linealidad al modelo. La ilustración nos ayuda a entender cómo la función *ReLU* es 0 para valores negativos de entrada y el valor de entrada para valores positivos, lo que ayuda a evitar el problema del gradiente desvaneciente en las redes neuronales. También puede mostrar otras funciones de activación como la función sigmoide o la función tangente hiperbólica para hacer una comparación.

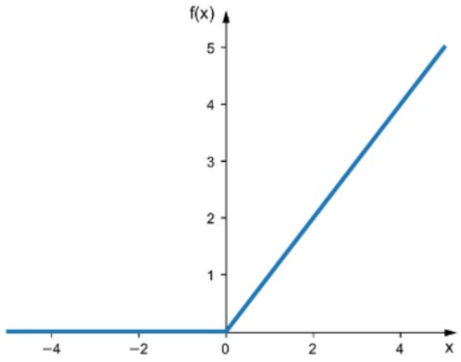



**Ilustración 8. Función de Activación RELU**

**[Ilustración8]**

### 2.3.3.2. BackPropagation.

Cuando trabajamos con redes neuronales buscamos ajustar los pesos de cada neurona de tal manera que se minimice el error de la función de coste. Mediante el algoritmo de backpropagation podemos corregir el aprendizaje en cada una de las capas intermedias con el fin de mejorar la calidad de la red neuronal.

Empleamos *BackPropagation* para calcular los diferentes gradientes de la función de coste global, y así poder usar el algoritmo del descenso del gradiente para ajustar los parámetros de la red.

Características [9] :

- Proceso de aprendizaje supervisado.
- Nuestro principal objetivo del entrenamiento es reducir el error o la diferencia entre la predicción y la salida real.
- Actualizamos los pesos utilizando el descenso de gradiente. Calcula el gradiente de la función de error con respecto a los pesos de la red neuronal.

En la ilustración 10 mostramos el algoritmo de *backpropagation*. Este algoritmo es uno de los métodos utilizados para entrenar redes neuronales artificiales. La ilustración muestra cómo el algoritmo utiliza el proceso de propagación hacia atrás (*backpropagation*) para calcular el gradiente de la función de coste en cada capa de la red neuronal.

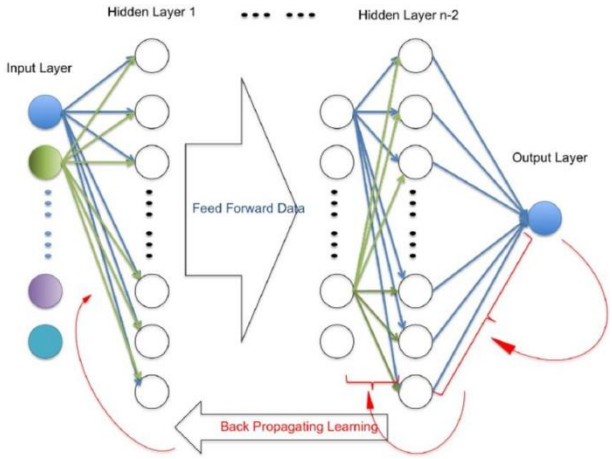



**Ilustración 9. Backpropagation**

**[Ilustración9]**

### 2.3.4. Redes Neuronales Convolucionales.

Las *CNN* es un tipo de Red Neuronal Artificial con aprendizaje supervisado que procesa sus capas para identificar distintas características en el tratamiento de imágenes de entrada que permite su reconocimiento y clasificación.

Pasos Necesarios previos [10] :

- Trabajaremos con imágenes pixeladas siguiendo un esquema *RGB*.
- Una imagen digital en color en esquema *RGB* está formada por una matriz tridimensional (x*y*z) donde:
  - XY corresponden con la anchura y la altura de la imagen.
  - Z corresponde al color de la imagen, rojo verde o azul.
- Por lo que un paso previo necesario para el tratamiento de imágenes es normalizarlas, los colores de los píxeles tienen un valor entre 0-255.

Mediante la normalización obtendremos valores entre 0 y 1 para todo el conjunto de imágenes.

En la Ilustración 11 mostramos el concepto de los planos *RGB* en el proceso de normalización de imágenes en una red neuronal Convolucional (*CNN*). Esta ilustración puede mostrar cómo las imágenes son representadas en términos de tres planos *RGB* (rojo, verde y azul) y cómo cada plano contiene información sobre un color específico.

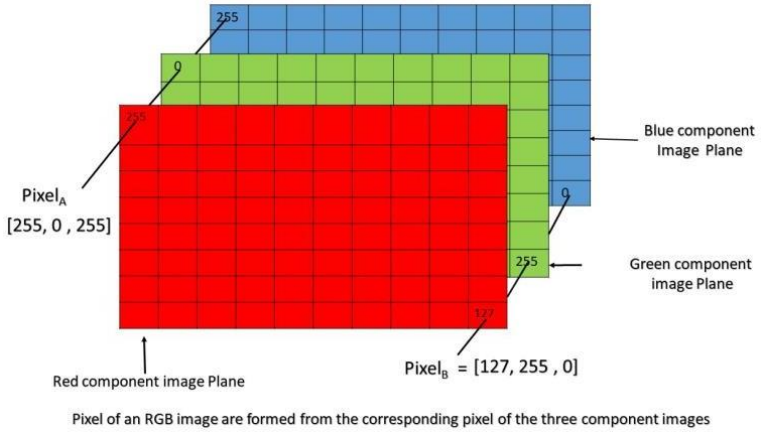



**Ilustración 10. Planos RGB**

**[Ilustración10]**

Convoluciones.

"Una *CNN* consta de un conjunto finito de capas de convolución que se encargan de extraer características de las imágenes [10]."

- Las primeras Capas de Convolucionales serán de un nivel de abstracción menor, aumentando en función del número de capas que se vayan introduciendo.
- Para realizar la extracción de los Características de cada Imagen empleamos filtros de un tamaño inferior al tamaño de la imagen. La imagen de salida luego pasará por otro filtro llamado pooling, el cual se encarga de disminuir el tamaño espacial de la imagen.

En la ilustración 12 mostramos el proceso de Convolución en una red neuronal Convolucional (*CNN*). La ilustración nos ayuda a entender cómo se utilizan los filtros o *kernels* para extraer características de las imágenes y cómo se aplican estos filtros en una ventana móvil sobre la imagen para generar una nueva imagen con menos dimensiones llamada mapa de características.

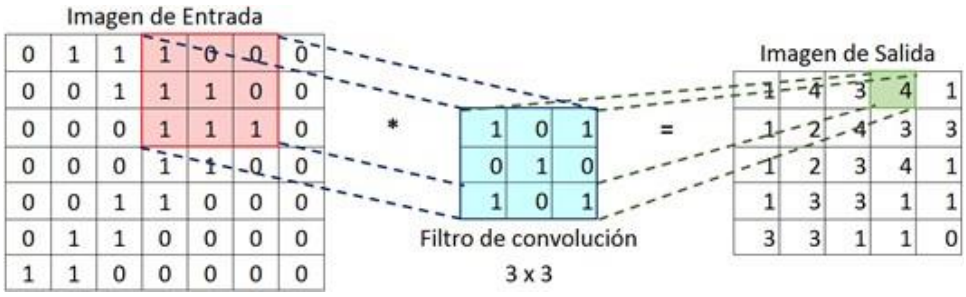



**Ilustración 11. Proceso de Convolución**

**[Imagen11]**

- Convolucional Layer [14], Es el componente básico de la *CNN* y procesará la salida de las neuronas, en llamadas "regiones locales de entrada", es decir píxeles cercanos de la imagen de entrada y aplicando kernels que realizan productos matriciales con las imágenes de entrada (inicialmente tomará valores aleatorios, mediante *Backpropagation* se irá corrigiendo).

En la ilustración 13 mostramos el concepto de una capa Convolucional en una red neuronal Convolucional (*CNN*). Esta ilustración nos ayuda a entender cómo la capa Convolucional se aplica varias veces a la imagen de entrada utilizando diferentes filtros cada vez, lo que permite que la red neuronal aprenda características cada vez más complejas y abstractas de las imágenes.

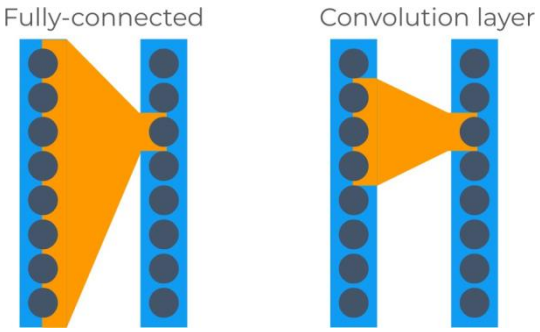



**Ilustración 12. Convolutional Layer**

**[Ilustración12]**

- Subsampling con *MaxPooling* y *AveragePooling*. "El objetivo es generar muestras de la imagen de entrada reduciendo su tamaño y el coste computacional reduciendo el número de parámetros a entrenar y reducir el *Overfitting* [13]."

En la ilustración 14 mostramos los conceptos de *MaxPool* y *AveragePool* en una red neuronal Convolucional (*CNN*). Esta ilustración nos ayuda a entender cómo *MaxPool* y *AveragePool* son dos operaciones de reducción de dimensionalidad utilizadas en las *CNN* para reducir el tamaño de las imágenes y reducir el número de parámetros en el modelo. *MaxPool* consiste en dividir la imagen en una cuadrícula y seleccionar el valor máximo de cada cuadrado de la cuadrícula, creando una imagen de menor resolución con características más importantes. *AveragePool* es similar a *MaxPool* pero en lugar de seleccionar el valor máximo, se selecciona el valor promedio de cada cuadrado de la cuadrícula. La ilustración puede mostrar cómo estas operaciones ayudan a reducir el overfitting y aumentar el rendimiento del modelo.

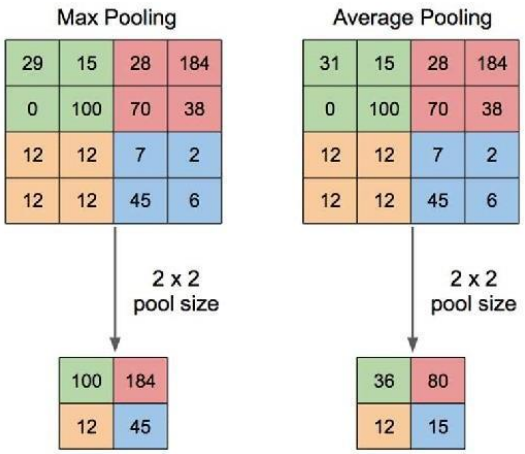



**Ilustración 13. MaxPool & AveragePool**

**[Imagen13]**

- Proceso Iterativo *CNN* [12] :
  - Tratamiento de Normalización de Imágenes *RGB*.
  - Capa de Entrada de la que obtenemos los Características Maps de las imágenes de entrada o *Subsampling* Mediante *MaxPooling* o *AveragePooling*.
  - Función de Activación *ReLU*, cuyo principal cometido es transformar los valores negativos a cero y el tratamiento de funciones no Lineales. El uso de esta función supera el "*the vanishing gradient problem*" y hace que los modelos aprendan mucho más rápido.
  - *Fully Connected* (FC): Capas al final de la arquitectura las cuales reciben la información de las capas anteriores aplica una función de activación que clasifica las imágenes.
  - El algoritmo actualiza los pesos (o coeficientes de redes) Mediante *Backpropagation* del gradiente del error (algoritmo de descenso de gradiente que hemos visto en pasos anteriores).

En la ilustración 15 mostramos el concepto de las capas de una red neuronal residual (*ResNet*) de 18 capas. *ResNet* es una arquitectura de red neuronal que se utiliza para resolver problemas de aprendizaje profundo y tiene una estructura de capas residuales. Estas capas residuales ayudan a resolver el problema del gradiente desvaneciente, que es común en redes profundas.

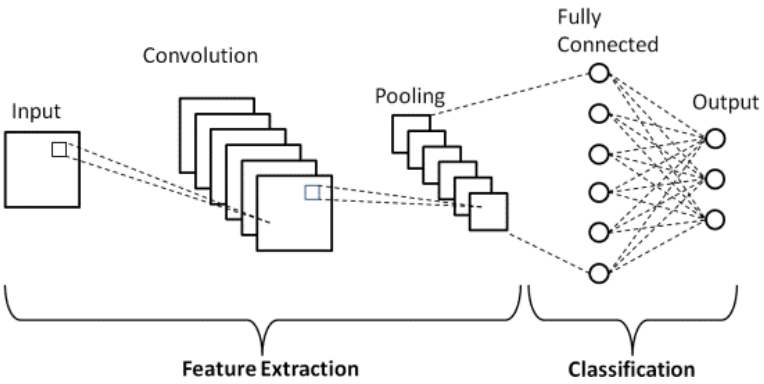



**Ilustración 14. Capas *ResNet18***

**[Ilustración14]**

### 2.3.5. *RestNet18* y " *the vanishing gradient problem*".

Las redes neuronales Residuales, como hemos visto anteriormente en toda red neuronal, la derivada o gradiente del error cometido debe ser transmitido de las últimas a las primeras capas, mediante un proceso conocido como *backpropagation*. Cuando estamos trabajando con redes muy profundas, con un nivel de capas intermedias muy alto, el gradiente del error es muy pequeño lo que eleva los costes computacionales de forma considerable, el gradiente tiende a cero.

La forma que planteamos para solucionar este problema, que es conocido como "*the vanishing gradient problem*" [15] , es saltar conexiones entre capas, estas son

conocidas como "*skip connections*" [16] mediante las cuales podemos construir Redes de múltiples capas sin que se degrade el desempeño computacional.

En la ilustración 16 mostramos el concepto de las conexiones saltadas (*Skip Connections*) en una red neuronal residual (*ResNet18*). *Skip Connections* es una técnica utilizada en las arquitecturas de redes neuronales residuales para conectar la entrada de una capa con la salida de una capa anterior, permitiendo que el gradiente fluya directamente a través de varias capas, lo que ayuda a evitar el problema del gradiente desvaneciente.

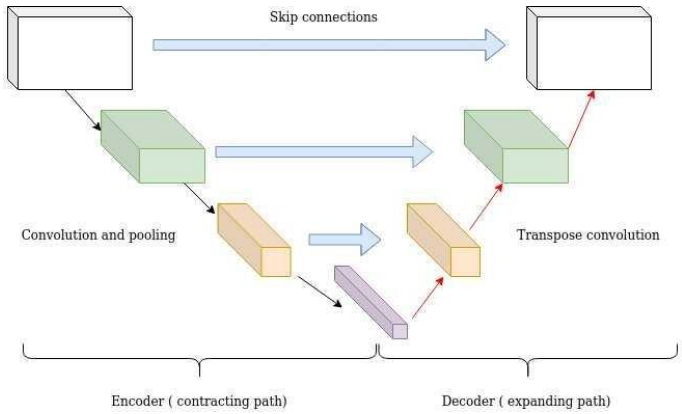



**Ilustración 15. *SkipConnections***

**[Ilustración15]**

Características de *RESNET*:

- El gradiente de la función de pérdida puede retroceder directamente hacia la entrada a través de "*skip connections*" y por lo tanto sufre una menor degradación.
- Al final de cada bloque residual se mezcla información que ha pasado por las capas de convolución con otra que no lo ha hecho potenciando el rendimiento de los sistemas de reconocimiento visual.

En la ilustración 17 mostramos el rendimiento de una red neuronal en función del número de capas. Esta ilustración nos ayuda a cómo el rendimiento de la red neuronal mejora a medida que aumenta el número de capas, hasta alcanzar un punto en el que el rendimiento no mejora significativamente con el aumento adicional de capas.

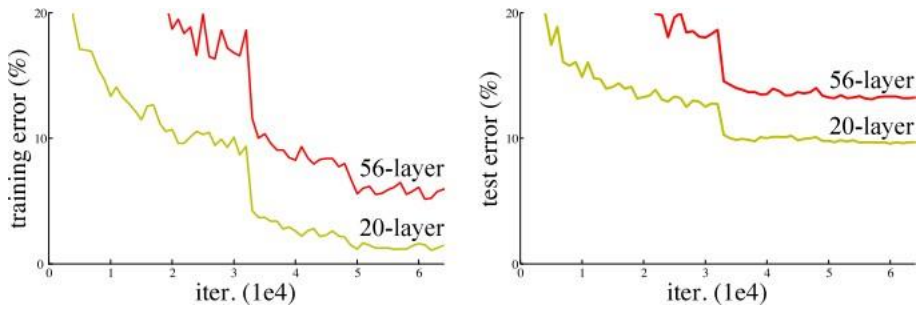



**Ilustración 16. Rendimientos con menor número de capas**

**[Ilustración16]**

## 2.4. Reducción de la Dimensionalidad.

Hablamos de reducción de la dimensionalidad como el proceso donde disminuimos el número de variables aleatorias a tratar principalmente transformando espacios de alta dimensionalidad en otros de dimensión menor y cuyos ventajes principales son:

- Reducción del espacio y tiempo requerido.
- Eliminación de la Multicolinealidad, indispensable en ciertos algoritmos de ML.
- Facilitar la visualización de los Datos.

En función de la transformación de los datos estas reducciones pueden ser:

- Lineales, ejemplo *PCA*.
- No lineales, ejemplo *t-sne*.

### 2.4.1. Análisis de Componentes Principales. *PCA*.

Es un algoritmo no supervisado de reducción de la dimensionalidad perdiendo la menor cantidad de varianza posible. Cada nueva dimensión o componente principal tiene las siguientes características [17] :

- Es una combinación lineal de las variables originales.
- Los componentes Principales son los autovectores de la matriz de correlaciones.
- La primera componente será la que mayor varianza recoja, la segunda la siguiente y así sucesivamente.
- Herramienta muy útil para la visualización de Datos.

### 2.4.2. TSNE.

Es un algoritmo no supervisado de reducción de la dimensionalidad, pero en vez de emplear distancias clásicas como Euclídea o *Gower*, utiliza probabilidades condicionales de similitud entre puntos, también conocido como *Stochastic Neighbor Embedding* [18] [19].

Las principales características son:

- Algoritmo no Lineal.
- Es reducir un espacio de alta dimensionalidad a uno de dimensión inferior, normalmente 2d, encontrando una representación fiel. *Embedding*.
- Muy útil para el "*crowding problem*" que implica "*la maldición de la dimensión*" y básicamente en nuestro caso afecta ya que al aumentar el número de dimensiones la distancia al vecino más próximo aumenta.

Pasos Algoritmo:

- Paso 1: Comienza convirtiendo la distancia entre los puntos de datos en medidas de probabilidad condicionales que representan similitud entre los datos. Hay que prestar

atención a las colas que son estrechas y pueden acumular mucha relación de puntos.

- Paso 2: La idea intuitiva es realizar asignaciones de baja dimensión que representen distribuciones de probabilidad similares, aquí nos podemos encontrar con "*crowding problem*" debido a las "*colas cortas*" de las distribuciones Gaussianas. Para subsanar este problema y que los puntos tengan una "cola más larga" l *t-sne* usa una distribución t-student con un grado de libertad. La optimización de esta distribución t-student se realiza mediante una función *Gradient Descent* que intuitivamente representa la fuerza y la atracción repulsión entre dos puntos. Gradiente positivo implica atracción y, al contrario. Este "*push-and-pull*" hace que los puntos se asienten en espacio de baja dimensionalidad.
- Paso 3: Las *t-sne* optimizan directamente a través de la función de Coste Gradiente que es como sabemos son Funciones de Coste No Convexas que pueden dar problemas con los mínimos locales. *Stochastic Neighbor* implica que no está cerrada la frontera de los puntos que son vecinos contra los puntos que no lo son permitiendo al algoritmo tener en cuenta la estructura local como la global. (esto lo realizaremos con el parámetro *perplexity*).

*Perplexity* puede interpretarse como una medida suave del número efectivo de vecinos. El rendimiento de *t-sne* es bastante robusto a los cambios en la perplejidad, y los valores típicos están entre 5 y 50.

*Interaction*, no hay un número fijo de pasos que produzca un resultado estable. Diferentes conjuntos de datos pueden requerir diferentes números de iteraciones para converger. Los autores adviertes que si vemos un gráfico de *t-sne* con extrañas formas "*pellizcadas"*, lo más probable es que el proceso se haya detenido demasiado pronto.

En la ilustración 18 mostramos el concepto de *t-sne (t-Distributed Stochastic Neighbor Embedding)* en un conjunto de datos *MNIST*. *t-sne* es un algoritmo de reducción de dimensionalidad utilizado para visualizar datos en alta dimensionalidad en un espacio bidimensional o tridimensional. *t-sne*

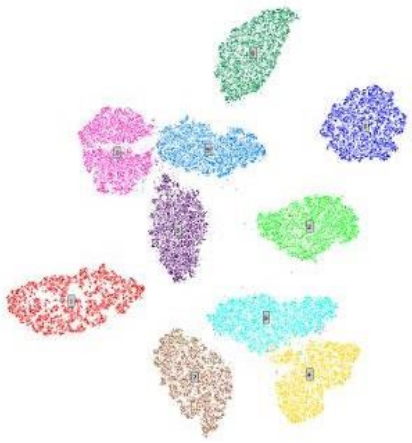


**Ilustración 17*. t-sne de Mnist***

**[Ilustración17]**

## 2.5. Clustering y Grafos.

Dentro de los diferentes modelos de aprendizaje no supervisado, que son aquellos que no necesitan datos etiquetados en el proceso de entrenamiento (no existe una referencia que replicar) y cuyo objetivo es encontrar patrones de datos similares entre sí, el análisis clúster es uno de los más populares y empleados.

Se basan en la posibilidad de agrupar a los individuos en función de sus similaridades empleando normalmente diferentes métricas de distancia y a partir de ella calcular una medida de similitud entre pares.

Los más populares son:

- *Kmeans*, basado en centroides.
- Clúster Jerárquico.
- *DBSCAN*, basado en densidades.

### 2.5.1. KMeans Clustering.

"El *KMeans* es un algoritmo de clasificación no supervisada que agrupa (*Clustering*) los diferentes ítems a estudio en función de sus diferentes características. Pretendemos resolver un problema de optimización, siendo la función para optimizar (minimizar) la suma de las distancias cuadráticas de cada objeto al centroide de su clúster [20] [21]."

En la siguiente fórmula mostramos la función a minimizar en el algoritmo de *Kmeans*.

$$d(x,y)^2 = \sum_{j=1}^{m}(xj - yj)^2$$

**Ecuación 2. Función Para Minimizar en el Algoritmo *Kmeans***

Pasos:

- Paso 1: una vez escogido el número de grupos, k, se establecen k centroides en el espacio de los datos, por ejemplo, escogiéndolos aleatoriamente.
- Paso 2, asignamos a cada ítem a su centroide más cercano.
- Paso 3, se actualiza la posición del centroide de cada grupo tomando como nuevo centroide la posición del promedio de los ítems pertenecientes a dicho grupo. Iteramos los pasos 2 y 3 hasta que los centroides no se muevan en el espacio Vectorial.

Para determinar el número óptimo de Clústeres para ello, vamos a realizar varias ejecuciones con una k diferente (desde 1 clúster hasta n) y representaremos en un gráfico la distancia media de cada punto hasta su centroide. La idea es que a medida que vamos aumentando la cantidad de centroides, la distancia media de los puntos al centroide irá disminuyendo cada vez menos.

En la ilustración 20 mostramos el concepto del número óptimo de clústers en el algoritmo de *Kmeans* utilizando el método *KElbow* que es una técnica utilizada para determinar el número óptimo de clústers en el algoritmo de *Kmeans*.

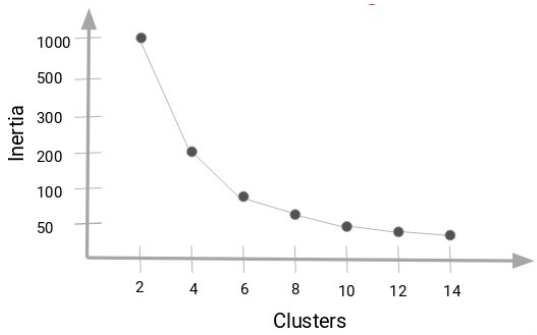



**Ilustración 18. Número óptimo de *Clúster KElbow***

**[Ilustración18]**

Características de los Clústeres:

- Homogeneidad intragrupo.
- Heterogeneidad extragrupo.

La teoría de grafos aplicada a análisis de grupos, personas o items relacionados o unidos por un factor (puede ser amistad, lazos en común, similaridades en los relativo a una o varias variables) y que nos permite de forma gráfica representar estas relaciones y analizar los diferentes patrones de relación.

### 2.5.2. Grafos.

"La teoría de grafos es la base matemática sobre la que se construye el estudio de las redes complejas. Un grafo es una representación gráfica de un conjunto de puntos (también nodos) unidos por líneas (aristas), que representan elementos e interacciones entre los mismos [22] [23]."

Nosotros trabajaremos con los Multigrafo, aquellos que aceptan más de una arista entre dos vértices. Podemos distinguir distintos grupos de Grafos:

- Grafo No dirigido, en los cuales se ha añadido una orientación a las aristas, representada gráficamente por una flecha.
- Grafo etiquetado, en los cuales se ha añadido un peso a las aristas o un etiquetado a los vértices.

En la ilustración 21 mostramos el concepto de un grafo etiquetado que es una estructura de datos en la que cada nodo o arista tiene asociada una etiqueta o rótulo. Estas etiquetas pueden ser utilizadas para describir atributos de los nodos o aristas.

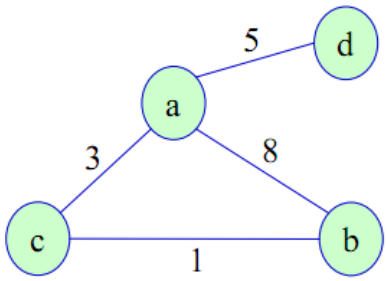



**Ilustración 19.Grafo Etiquetado**

**[Ilustración19]**

## 2.6. Visualización Dinámica.

Campo del estudio del Aprendizaje Automático entre otros cuya finalidad es la representación de datos en formato gráfico, nos permite detectar y comprender:

- Tendencias.
- Valores atípicos.
- Patrones de Datos.

La visualización de grafos es la representación de un conjunto completo donde podemos observar la posición de los vértices y la orientación de las aristas que dan forma al grafo. La librería *D3JS* está diseñada para la visualización de datos de distinta índole, siguiendo los estándares web. Funciona bajo una combinación de *SVG, Canvas y HTML* y permite la interacción con los gráficos generados.

Esta librería mejora la calidad de la representación del grafo, pero al ser una herramienta que se emplea en la visualización de múltiples tipos de datos requiere un conocimiento y know-how importante para su utilización.

En la ilustración 22 mostramos una visualización interactiva gráfica en 2D utilizando la librería *JavaScript 3DJS* que permite crear visualizaciones interactivas de grafos.

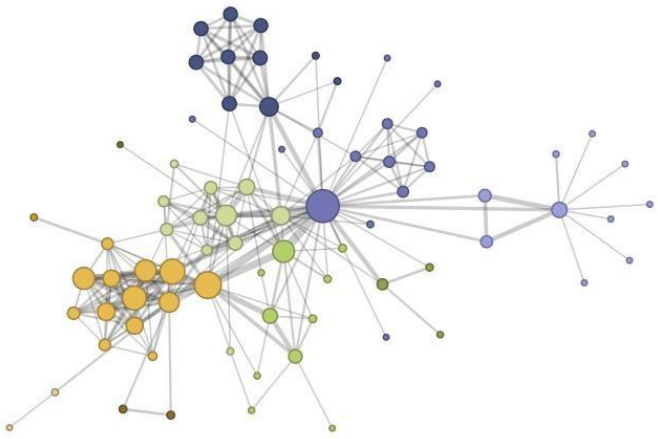


**Ilustración 20. Visualización Grafo 3DJS**

**[Ilustración20]**

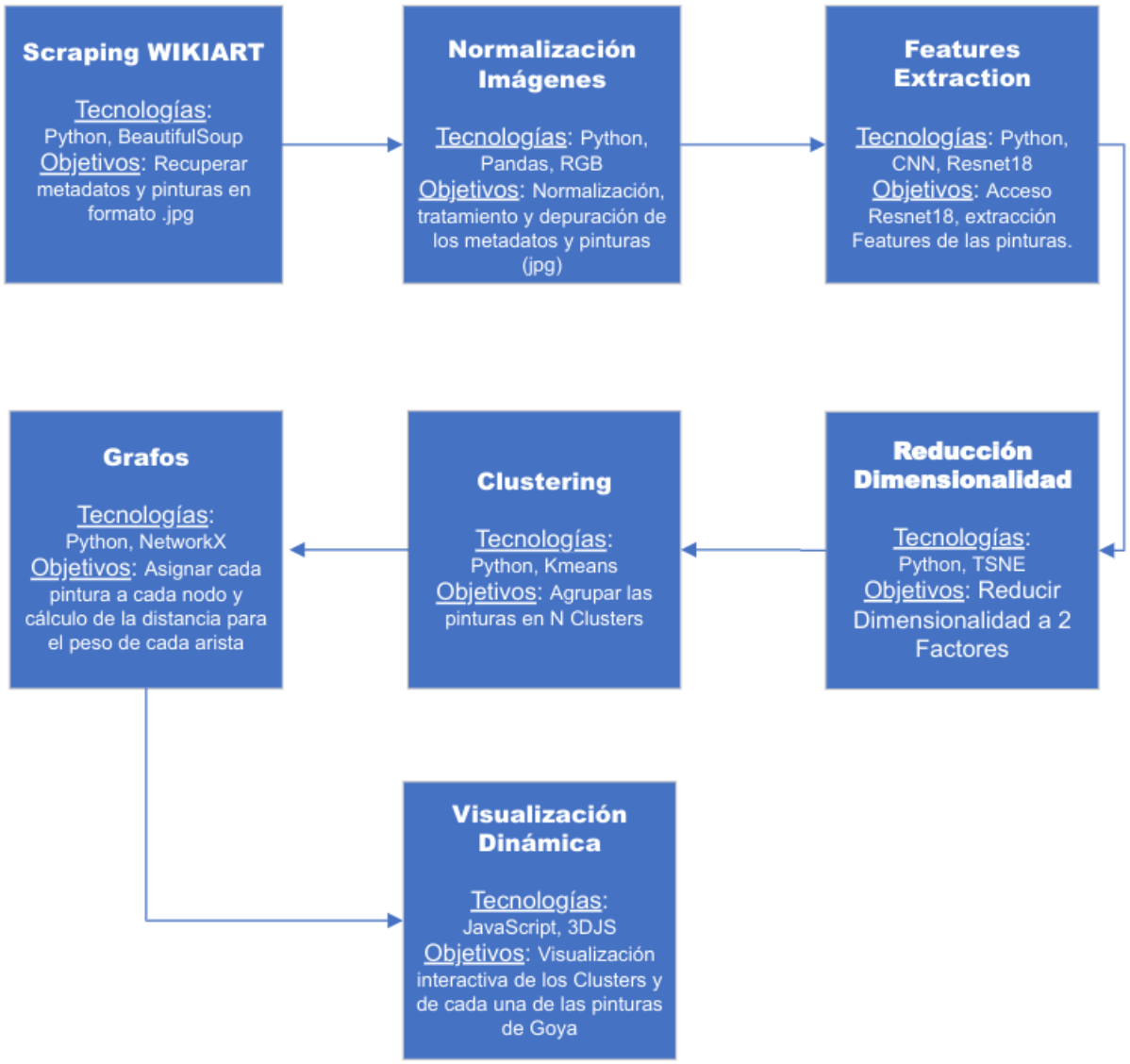
# 3. Experimentos y Resultados.

En este capítulo mostraremos los experimentos llevados a cabo, desde el proceso inicial de captura de datos hasta la evaluación final y la implementación visual de los resultados obtenidos.

Para resolver el problema de clasificación de la obra de Francisco de Goya basado en las capas convolucionales de las *CNN* emplearemos secuencialmente distintos algoritmos con diferentes ajustes en los hiperparámetros a fin de mejorar sus prestaciones.

## 3.1. Sistema Teórico.

### 3.1.1. Flujo del Proyecto.




### 3.1.2. Desarrollo Teórico del Proyecto.

El objetivo de este TFM es analizar y agrupar la obra de Francisco de Goya utilizando técnicas de aprendizaje automático. Para lograr este objetivo, hemos seguido los siguientes pasos:

1. Scrapeo de la web *WikiArt*: En primer lugar, hemos utilizado técnicas de web scraping en *Python* para descargar las imágenes de las pinturas de Goya disponibles en la web *WikiArt*. Esto nos ha permitido obtener un conjunto de datos de imágenes que hemos utilizado para entrenar y evaluar nuestro modelo de redes neuronales.

2. Tratamiento y Normalización de las imágenes: Una vez que hemos descargado las imágenes, hemos normalizado cada una de ellas utilizando la tecnología *RGB*. Esto ha sido necesario para ajustar el formato de las imágenes y hacerlas compatibles con nuestro modelo de redes neuronales.

3. Extracción de características: A continuación, hemos utilizado la red *ResNet18* para extraer características de cada una de las imágenes normalizadas. Esto nos ha permitido obtener un conjunto de datos de características que hemos utilizado para entrenar y evaluar nuestro modelo de clasificación.

4. Utilizando la red ResNet18, hemos extraído características de cada imagen y luego hemos utilizado el algoritmo *t-sne* (t-Distributed Stochastic Neighbor Embedding) para reducir la dimensión de nuestros datos de características. Específicamente, hemos seleccionado dos dimensiones del espacio reducido por *t-sne* para analizar y agrupar nuestras imágenes. Esto nos ha permitido visualizar los datos en un plano y analizar su estructura y distribución de forma más intuitiva.

5. Una vez que hemos utilizado el algoritmo *t-sne* para reducir la dimensión de nuestro conjunto de datos de características extraídas de las pinturas de Francisco de Goya y visualizar los datos en un plano, hemos utilizado el algoritmo de clustering *KMeans* para agrupar nuestras imágenes en diferentes clústers. *KMeans* es un algoritmo de clustering iterativo que divide un conjunto de n puntos en k grupos basándose en la distancia Euclídea. En cada iteración, *KMeans* asigna cada punto al clúster cuyo centroide esté más cerca y luego recalcula los centroides de cada clúster como la media de los puntos asignados a ese clúster.

   Este proceso se repite hasta que los centroides de los clústers no cambien significativamente o se alcance el número máximo de iteraciones. Para utilizar *KMeans* en nuestro conjunto de datos, hemos especificado el número de clústers que deseamos obtener y hemos utilizado las coordenadas de *t-sne* de cada punto como entrada para el algoritmo.

   Una vez que *KMeans* ha finalizado, hemos obtenido k clústers y hemos asignado a cada pintura a un clúster en función de su pertenencia. Con estos pasos, hemos podido utilizar el algoritmo *KMeans* para agrupar las pinturas de Francisco de Goya en diferentes clústers y obtener una mejor comprensión de la estructura y distribución de nuestro conjunto de datos.

6. Una vez que hemos utilizado el algoritmo *KMeans* para agrupar nuestras pinturas de Francisco de Goya en diferentes clústers, hemos aplicado la teoría de grafos para analizar y agrupar las pinturas utilizando técnicas de aprendizaje automático. La teoría de grafos nos permite representar un conjunto de datos como una red de nodos y aristas, donde cada nodo representa una pintura y cada arista representa una relación entre dos pinturas.

   Primero hemos asignado un nodo a cada una de las pinturas y segundo hemos calculado el peso de la arista como la inversa de la distancia euclídea entre los dos vectores *t-sne1* y *t-sne2*.

   De esta manera si la distancia entre las dos pinturas es muy alta, el valor del peso de la arista será muy pequeño y al contrario si la distancia entre las dos pinturas es muy baja, el peso de la arista entre las pinturas será muy alto.

Con estos pasos, hemos podido utilizar la teoría de grafos para analizar y agrupar la obra de Francisco de Goya utilizando técnicas de aprendizaje automático, asignando a cada pintura un nodo y el peso de cada arista en función de la relación que exista entre las pinturas.

7. Para finalizar nuestro TFM, hemos empleado la teoría de grafos y la visualización en 2D para mostrar la obra de Francisco de Goya de manera dinámica y atractiva.

   Utilizando los nodos representativos de cada pintura y las aristas que representan las relaciones entre ellas, hemos creado una red de datos que podemos visualizar en 2D utilizando la herramienta *3DJS*.

   Esto nos ha permitido mostrar de forma interactiva la estructura y distribución de nuestro conjunto de datos y hacer una exploración más profunda de la obra de Francisco de Goya. Con esta visualización en 2D, hemos podido acercar la obra de Goya a un público más amplio y ofrecer una experiencia pedagógica y atractiva para todos los interesados en la pintura.

   Además, hemos podido profundizar en el análisis de nuestros datos y obtener una mejor comprensión de la estructura y distribución de la obra de Goya utilizando técnicas de aprendizaje automático y la teoría de grafos.

## 3.2. Sistema Práctico.

### 3.2.1. Scrapeo Web WIKIART.

Nuestro primer objetivo fue encontrar una web pública de acceso libre todo un banco de imágenes e información de las pinturas de Goya, investigamos en repositorios como las webs de los museos:

- Museo del Prado, Louvre entre otros.
- Repositorios web como bing_image_downloader, Librería de *Python* la cual permite la descarga de imágenes.

Ambos tipos de fuentes no almacenaban la información que buscábamos de forma adecuada, sino desestructura y repetida. Tras una búsqueda bastante exhaustiva en internet encontramos el repositorio de información *WikiArt* el cual nos brindaba información pública y estructurada de cientos de pintores y obras.

Después de comprobar la accesibilidad desarrollamos un *script* de *Python* para el scrapeo de las obras de Francisco de Goya De este trabajo obtenemos varias fuentes de datos:

- 388 imágenes en formato jpg de la obra de Francisco de Goya, esta información la mantendremos en una carpeta dentro de nuestro equipo que compartiremos comprimidas.
- Generaremos un *DataFrame* (*DF*) en *Pandas Python* con la información detallada de la imagen que nos ayudará en la visualización de la pintura y en la comprensión de aspectos de esta.

- Guardaremos este *DF* en formato pickle.

Después de ese desarrollo disponemos de casi 400 pinturas en formato jpg, el siguiente paso para el tratamiento es conocer cómo son estas pinturas y tratar de normalizarlas siguiendo un esquema que nos permita trabajar de forma homogénea. Procederemos a realizar una normalización de las imágenes.

### 3.2.2. Tratamiento y Normalización de las Imágenes.

Nuestro siguiente paso en el proceso será realizar una normalización del conjunto de 388 pinturas para ello y gracias a la librería *OpenCV* [24] y *Pytorch* [25] conseguiremos homogeneizar el formato de las imágenes normalizándolas según el esquema *RGB* anteriormente mencionado, con los siguientes parámetros (mostraremos unas imágenes a modo de ejemplo que nos ayude a visualizar el proceso):

- **Contraste**, Brillo de las imágenes de entrada parámetro a Homogeneizar.

En la ilustración 23 consistiría en dos imágenes, una antes y otra después de haber aplicado la parametrización para corregir el contraste.


**Ilustración 21. Parametrización Imagen 1**

- **Tamaño de la imagen**, Cambiar el tamaño de una imagen significa cambiar las dimensiones de esta, ya sea solo el ancho, solo el alto o cambiar ambos. Además, la relación de aspecto de la imagen original podría conservarse en la imagen redimensionada. Para cambiar el tamaño de una imagen, *OpenCV* proporciona la función *cv2.resize().*

En la ilustración 24 consistiría en dos imágenes, una antes y otra después de haber aplicado la parametrización para estandarizar el tamaño de todas las imágenes.

Ilustración 22. Parametrización Imagen 2

Intensidad de los colores, *RGB* es el formato de colores empleado por *Pytorch* , sabemos que una imagen es un conjunto de pixeles distribuidos en forma de cuadrícula.

Cada píxel a su vez está conformado por otros componentes, que combinados forman los colores que podemos ver en una imagen. Está formado por tres componentes que calculan la media y la d desviación estándar de la imagen para los tres canales y usaremos estos valores para la normalización de las imágenes.

Podemos emplear la media y el std de la red Imagenet como ejemplo

- mean = [0,485, 0,456, 0,406]
- Std = [0,229, 0,224, 0,225]

En la ilustración 25 consistiría en dos imágenes, una antes y otra después de haber aplicado la parametrización de colores empleada por *Pytorch*.

Ilustración 23. Parametrización Imagen 3

Vemos que la resolución no es la más adecuada para nuestro data set, por lo que iremos probando hasta dar con:

- mean = [0.1, 0.1, 0.1]
- Std = [0.9, 0.9, 0.9]

En la ilustración 26 consistiría en dos imágenes, una antes y otra después de haber aplicado la parametrización más adecuada a nuestro dataset de imágenes.


**Ilustración 24. Parametrización Imagen 4**

Los parámetros que emplearemos para homogeneizar las imágenes son:

- Nivel Contraste y Brillo de la Imagen 256.
- Tamaño de las imágenes 512x512.
- Tensor *RGB*:
  - mean = [0.1, 0.1, 0.1]
  - Std = [0.9, 0.9, 0.9]

Todos estos parámetros son fundamentales para los siguientes pasos ya que determinan la dimensión de las neuronas de entrada en la Red Neuronal Convolucional.

- 512x512x3 -> 786.432 Neuronas de Input.
- 388 pinturas.

### 3.2.3. Características Extraction de RESNET18.

Una vez normalizadas las 388 pinturas nuestro siguiente paso será conectarnos a una de las *CNN* recursivas públicas ya entrenadas, entre las más populares y accesibles está la *ResNet18* que es una red recursiva, lo cual nos permite alcanzar una mayor profundidad efectiva, a diferencia de las no recursivas donde las primeras capas casi no se especializan y son muy similares independientemente del caso a resolver. Tal y como hemos visto anteriormente cuando estamos trabajando con redes muy profundas, con un nivel de capas intermedias muy alto, el gradiente del error es muy pequeño lo que eleva los costes computacionales de forma considerable, esquivándolo mediante las “skip connections” y mejorando notablemente el desempeño computacional. A diferencia de otras redes como *DenseNet* [26] que nos ofrece un mayor número de capas finales pero el rendimiento computacional es inferior.

En nuestros experimentos planteamos trabajar con dos redes neuronales Convolucionales previamente entrenadas de acceso público:

- *DenseNet121*, con 1024 Características en la capa Convolucional *AveragePool*.

- *ResNet18*, con 512 Características en la capa Convolucional *AveragePool*.

Decidimos trabajar con la red *ResNet18* por:

- A nivel computacional mucho más rápido la extracción de las Características ya que existen menos capas debido a su arquitectura más reducida por las "*skip connections*" para evitar " *the vanishing gradient problem*", por lo que el número de Características en *Resnet18* es la mitad que *DenseNet121*.

Una vez normalizadas las imágenes y mediante la librería *Pytorch* accederemos a la *CNN Resnet18* para acceder a una de las capas convolucionales que nos facilitarán información sobre las características de las imágenes de la Red. A partir de la parametrización anterior tenemos:

- 512x512x3 -> 786.432 Neuronas de Input.

Seleccionaremos la siguiente capa Convolucional:

- *AveragePooling* la cual nos proporciona 512 Características.

Visualización distribuciones *Feature Image1 & Feature Image512*.

En la ilustración 27 mostramos un histograma de la distribución de una característica específica (*Feature1*) de nuestro dataset de imágenes en la red neuronal *ResNet18*.

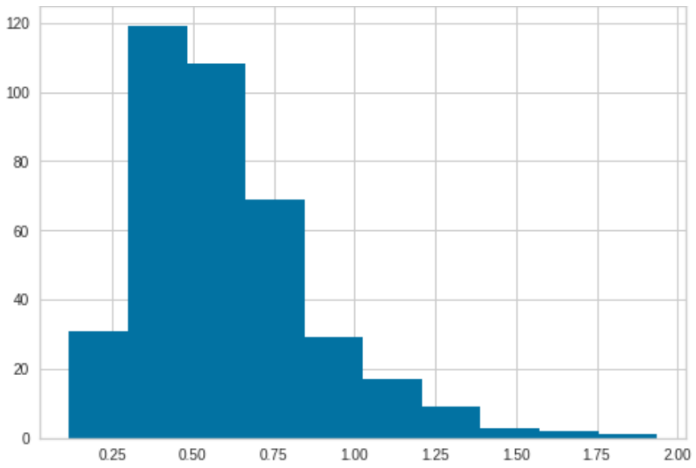



**Ilustración 25. Distribución Característica 1**

En la ilustración 28 mostramos un histograma de la distribución de una característica específica (*Feature512*) de nuestro *dataset* de imágenes en la red neuronal *ResNet18*.

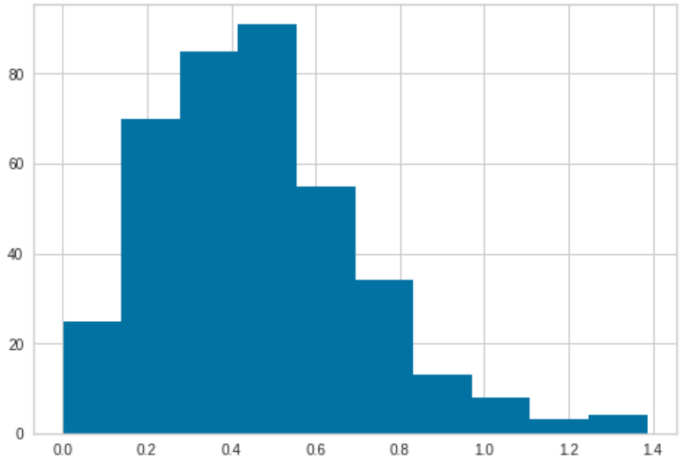



**Ilustración 26. Distribución Característica 512**

De esta manera hemos conseguido extraer las Características de nuestra fuente de imágenes que concatenaremos con nuestro *DF* original con las 512 variables extraídas de *ResNet18* y los datos de la información del *Scrapeo* de la Pintura.

Ahora disponemos de toda la información de las pinturas en 512 Variables Numéricas Positivas. Nuestro siguiente objetivo será reducir la dimensionalidad, y para ello proponemos los siguientes algoritmos:

- *PCA*, reducción de dimensionalidad lineal, pretendiendo preservar la estructura global de los datos, no parametrizable y sensible a los valores atípicos. Nos basamos en decidir cuanta varianza seleccionamos.
- *t-sne*, reducción no lineal de la dimensionalidad, intenta preservar la estructura local de los datos, es tratable con hiperparámetros al contrario que el *PCA* y es poco sensible a los datos atípicos.

El *t-sne* encontrará dependencias lineales y no lineales, al contrario que el *PCA* que sólo encontrará las lineales, siendo además el *t-sne* manipulable a través de hiperparámetros y poco sensible a los valores atípicos.

Con el objetivo de reducir la dimensionalidad de nuestro *DF*, emplearemos el algoritmo *t-sne* para pasar las 512 Características a un plano de 2 dimensiones. Replicaremos varios experimentos modificando los hiperparámetros e interpretando la visualización de los datos:

En la tabla1 mostramos la variación de los hiperparámetros de los experimentos de *t-sne*.

| **Experimentos** | Experimento1 | Experimento2 | Experimento3 |
|---|---|---|---|
| *Early Exaggeration* | 100 | 30 | 30 |
| *Learning Rate* | 5 | 5 | 500 |
| *Preplexity* | 30 | 30 | 20 |
| *Iteractions* | 500 | 2000 | 2000 |

**Tabla 1. Experimentos t-sne**

En la ilustración 30 mostramos los resultados visuales del experimento 1 utilizando *t-sne*.

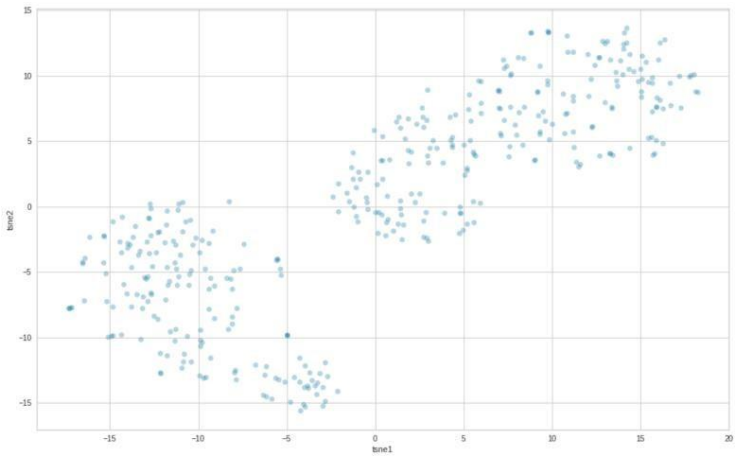



**Ilustración 27. Visualización *t-sne* Experimento 1**

En este primer experimento vemos que existen dos grandes grupos de datos, pero mucho tamaño y con mucha dispersión. Debemos ajustar los hiperparámetros.

En la ilustración 31 mostramos los resultados visuales del experimento 2 utilizando *t-sne*.

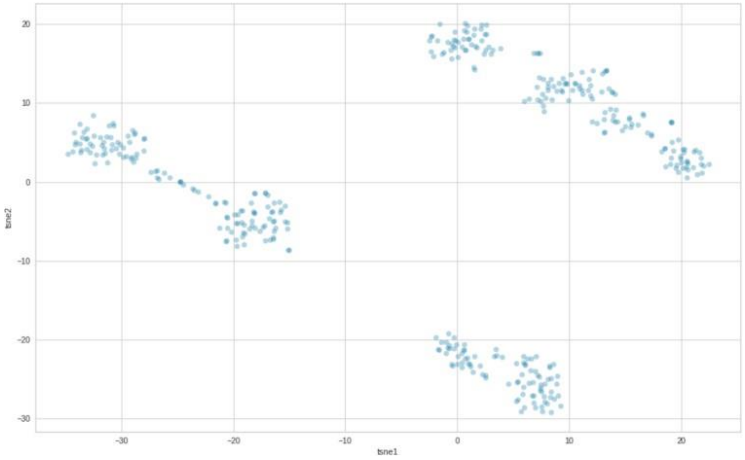



**Ilustración 28. Visualización *t-sne* Experimento 2**

Vemos con el segundo experimento que mejora la visualización, tres grupos todavía muy dispersos y achatados. Tal y como identifican los autores del algoritmo deberíamos aumentar el tamaño las iteraciones y el *LR*.

En la ilustración 32 mostramos los resultados visuales del experimento 3 utilizando *t-sne*.

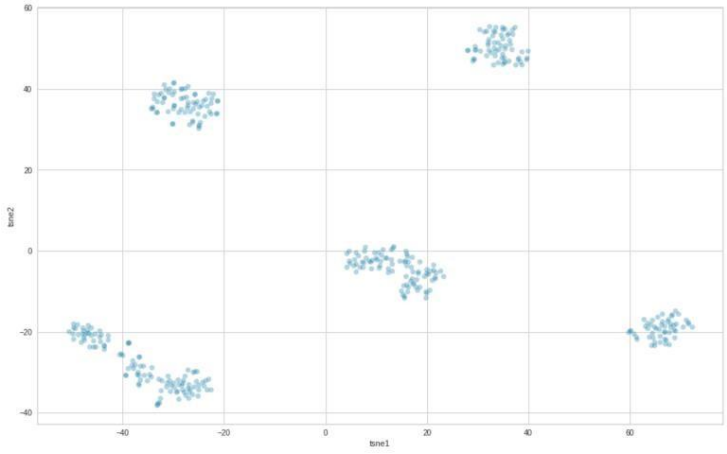



**Ilustración 29. Visualización t-sne Experimento 3**

Seleccionaremos los resultados de experimento 3, con ello:

- Reducimos la dimensionalidad a 2d.
- Intuitivamente identificamos 5 grupos o Clústeres diferenciados que emplearemos como parámetro k en el algoritmo *KMeans*.
- Uno de ellos quizá está un poco achatado de más, el de la esquina inferior izquierda, pero en general están bastante diferenciados unos de otros.

### 3.2.4. Clustering KMeans.

Intuitivamente hemos detectado 5 grupos de pinturas dentro del plano 2d generado a través del *t-sne*. Para confirmar formalmente estos grupos a través de un algoritmo de Clustering seleccionaremos el *Kmeans*. Para inicializar *KMeans* necesitamos identificar el número K de clúster que el algoritmo debe identificar. Intuitivamente gracias al *t-sne* detectamos 5 clúster. Vamos a contrastarlo con el valor del *Score Elbow.*

En la ilustración 33 mostramos la distribución de los puntajes de un algoritmo de clustering utilizando el método *Elbow* (*Elbow method)*. El punto donde se observa un cambio abrupto en el valor de la medida de rendimiento se considera el número óptimo de clústers.

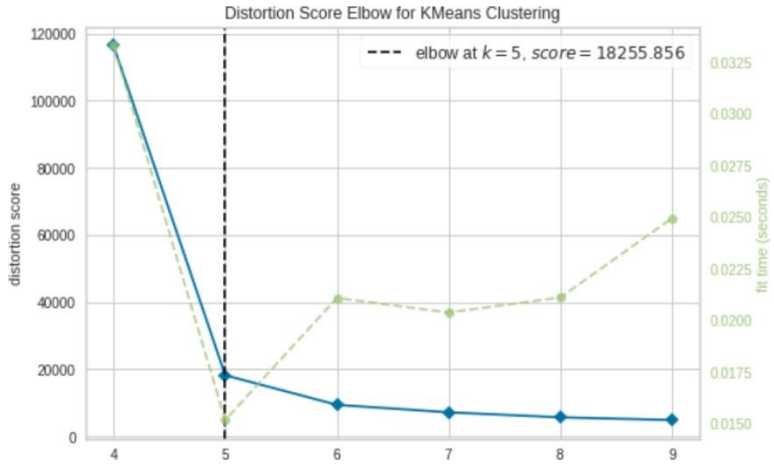



**Ilustración 30. Distribución *Score Elbow***

Inicializamos el *KMeans* con k=5, a continuación, visualizamos los *Clústers* en el espacio *t-sne* 2d.

En la ilustración 34 mostramos el resultado de aplicar el algoritmo de *Kmeans* sobre un conjunto de datos previamente reducidos mediante *t-sne* con k=5.

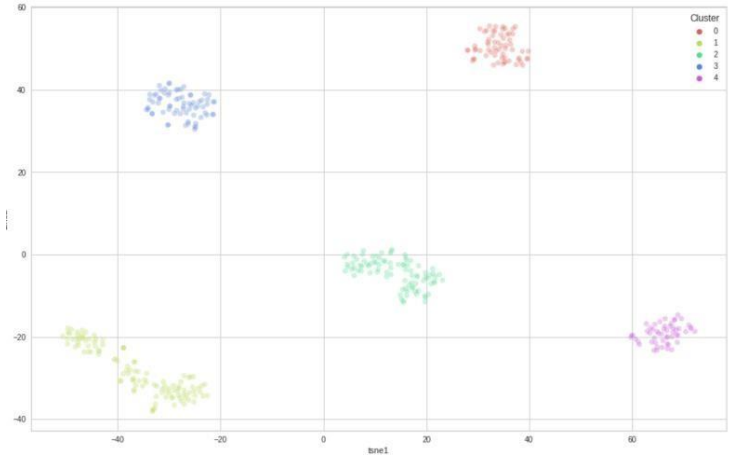



**Ilustración 31. *KMeans sobre t-sne k=5***

Asignaremos a cada una de las pinturas su correspondiente Clúster. Con toda esta última metodología tenemos un Data (en formato *pickle*) con las siguientes variables:

- Nombre de la Pintura y *URL* en *WikiArt*.
- Las dos dimensiones del *t-sne* (*t-sne1* y *t-sne2*).
- Número de Clúster Asignado (C0-C4).

### 3.2.5. Calculo de las distancias entre las pinturas y Grafo.

En este paso nuestro objetivo es transformar el *DF* anterior a un grafo donde en formato *Json* donde:

- Cada Nodo será una de las pinturas.
- Generaremos una Matriz Simétrica de 388x388 donde asignaremos un peso a las aristas calculando la inversa de la distancia euclídea de los valores del *t-sne*. (*t-sne1* y *t-sne2*).
- Calculando la Inversa de la Distancia Euclídea de los valores de *t-sne1* y *t-sne*2 obtenemos un índice que nos permita asignar un peso relativo a cada arista del Grafo y así visualizar de forma sencilla las pinturas. Cuanto menor sea el valor de la inversa de las Distancia Euclídea sobre los valores del TSNE más relacionados entre sí estarán las pinturas.

La pintura 0 y la pintura 2 con valor inverso de la Distancia Euclídea entre *t-sne1* y *t-sne2* es 0,014847 están mucho más relacionadas entre sí que el para de pintura 1 y la pintura 0 que es 1,566697.

En la tabla 2 mostramos una matriz simétrica utilizada para calcular la inversa de la distancia Euclídea entre los valores de *t-sne* para dos conjuntos de datos (*t-sne1* y *t-sne2*) y asignar un peso relativo a cada arista de un grafo.

| Nodos | Nodo1 | Nodo2 | Nodo3 | Nodo4 | Nodo5 |
|---|---|---|---|---|---|
| **Nodo1** | inf | 1,56 | 0,01 | 0,33 | 0,01 |
| **Nodo2** | 1,56 | inf | 0,01 | 0,36 | 0,01 |
| **Nodo3** | 0,01 | 0,01 | inf | 0,01 | 0,19 |
| **Nodo4** | 0,33 | 0,36 | 0,01 | inf | 0,01 |
| **Nodo5** | 0,1 | 0,01 | 0,19 | 0,01 | inf |

**Tabla 2. Matriz Simétrica Inversa Distancia Euclídea**

Mediante la librería networkx [27] de Python:

- Crearemos el Grafo.
- Todos los nodos están conectados entre sí.
- Asignaremos como nodo cada una de las Pinturas.
- Asignaremos a cada una de las aristas el peso de la matriz calculada anteriormente.
- Añadiremos información a cada nodo.
- Por último, crearemos un fichero en formato *Json* con el que trabajaremos en la visualización en *3DJS*.

En la ilustración 36 mostramos el resultado de una visualización interactiva de un clúster en un grafo mostrando los nodos de las pinturas de Goya agrupados en función de la inversa de distancia Euclídea entre los valores de *t-sne* de dos conjuntos de datos (*t-sne* 1 y *t-sne2*).

**Ilustración 32.Visualización Grafo**

### 3.2.6. Interpretación de los Clústeres.

A continuación, mostraremos un resumen de las características principales de las pinturas agrupadas en clústeres para darle un sentido y nombre a cada uno de ellos.

**Clúster 0. Retratos Oscuros:**

- 63 pinturas. 16,23%.
- Visualización *t-sne* Color rojo.
- Estilo Predominante Romanticismo.
- Tags Principales:
  - *Male-portraits*, *double-portraits*.
  - *Face, Head, Chin, Human, Standing, Christianity, famous-people.*
- Compuesto fundamentalmente por retratos de fondo oscuro. Tanto de visión cristiana (Saint Gregory) como de Personajes relevantes de la época (Moratín).
- Ilustración37 ejemplos de izquierda a derecha:
  - Portrait of Victor Guye (1810).
  - Martín Zapater (1797).
  - The Duke of Wellington (1812).

Ilustración 33. Clúster Retratos Oscuros

**Clúster 1. Caricaturas:**

- 112 pinturas. 28,86%.
- Visualización *t-sne* Color Verde Claro.
- Estilo Predominante Romanticismo Caricaturas.
- Tags Principales:
  - *allegories-and-symbols, Figure drawing, battles-and-wars, People, spirits-and-ghosts.*
- Compuesto fundamentalmente por caricaturas con alto contenido alegórico, Pintura.
- Ilustración38 ejemplos de izquierda a derecha:
  - Lunatic Behind Bars (1828).
  - Disparate furioso (1823).
  - The bagged (1823).

Ilustración 34. Clúster Caricaturas

**Clúster 2. Costumbrismo Español:**

- 89 pinturas. 22,93%.
- Visualización *t-sne* Color Verde Oscuro.
- Estilo Predominante Costumbrista.
- Tags Principales:
  - *arts-and-crafts, celebrations-and-festivals, Madrid, Holy places*.
- Compuesto fundamentalmente por costumbrismo donde busca recoger, el comportamiento social y la estética de la población de Madrid.
- Ilustración39 ejemplos de izquierda a derecha:
  - Otoño, o La cosecha de uva (1878).
  - Paseo por Andalucía o La Maja y los embozados (1777).
  - Peregrinación a la Iglesia de San Isidro (1788).

Ilustración 35. Clúster Costumbrismo Español

**Clúster 3. Retratos Aristocráticos:**

- 71 pinturas. 18,29 %.
- Visualización *t-sne* Color Azul.
- Estilo Predominante Retratos Romanticismo.
- Tags Principales: *famous-people, children portraits, Victorian fashion*.
- Retratos de mujeres, hombres y niños de las aristocracia y corte española. Destaca el estilo de la moda, y el trabajo de la luz.
- Ilustración40 ejemplos de izquierda a derecha:
  - La marquesa de Pontejos (1786).
  - Retrato de María Teresa de Borbón y Vallabriga (1783).
  - Fernando VII (1814).

Ilustración 36. Clúster Retratos Aristocráticos

**Clúster 4. Grabados y dibujos costumbristas:**

- 53 pinturas. 13,65 %.
- Visualización *t-sne* Color Violeta.
- Estilo Predominante Grabados, aguatinta, aguafuerte.
- Tags Principales:
  - *allegories-and-symbols, Stock photography, celebrations-and-festivals*.
- Compuesto fundamentalmente por grabados costumbristas.
- Ilustración41 ejemplos de izquierda a derecha:
  - La velocidad y el atrevimiento de Juanito Apiñani en el anillo de Madrid (1816).
  - Pareja con Sombrilla en el Paseo (1797).
  - El Gazul Morisco es el primero en torear con una lanza (1816).

Ilustración 37. Clúster Grabados y Dibujos

# 4. Conclusiones y Trabajo futuro.

Este Capítulo presenta las diferentes conclusiones extraídas al largo del trabajo realizado. En el primer punto, se exponen estas conclusiones después de desarrollar y realizar el experimento con datos que hemos trabajado. En un segundo punto, se presentan algunas futuras líneas de acción para guiar al lector sobre algunos puntos de interés con los que continuar este trabajo.

## 4.1. Conclusiones.

Repasaremos paso a paso los Objetivos

- *Scrapeo* de la web Wikiart para seleccionar y descargar las pinturas disponibles de Goya en formato jpg. Resuelto gracias a las funcionalidades de las librerías *Request* y *BeautifulSoup*.
- Especificar características del dataset (ruta donde se guardarán las pinturas, dataset donde podremos vincular estas pinturas al nombre, url...) Resuelto gracias a las funcionalidades de *Pandas Python.*
- Normalizar las pinturas Resuelto gracias a las funcionalidades del esquema *RGB* en *Opencv.*
- Acceder a la Red Convolucional *Resnet18* para extraer los Características Resuelto gracias a las funcionalidades de *Pytorch* para acceder a *ResNet18*
- Seleccionar algoritmo para reducir la dimensionalidad de los Feature de las pinturas a 2d. Resuelto gracias a la reducción de la Dimensionalidad con el algoritmo *t-sne*
- Dentro de este nuevo espacio crear un Clustering de las pinturas. Resuelto gracias al algoritmo *KMeans*
- Crear un grafo con la información de las pinturas. Resuelto gracias la librería *networkx* de *Python* asignando a cada nodo una pintura y al peso de la arista calculada a través de las distancias de los ejes del *t-sne*.
- Visualización del Grafo en formato *3DJS* Gracias a *3DJS* que nos permite la visualización de los grafos.
- Estudiar y analizar los Clústeres generados. Analizado y comprendido cada uno de los Clúster.

Las redes convolucionales son instrumentos fundamentales dentro del emergente campo de la visión artificial dentro del área del AI. Dentro de esta área se están realizando

un número importante de avances y desarrollos dentro de las Universidades y Empresas Privadas.

En este estudio hemos querido plasmar una visión alternativa, al margen de las visiones teóricas de los historiadores de arte, dirigida hacia las Redes neuronales y la información que estas nos brindan a través de las distintas capas de su arquitectura.

Nuestro punto de partida era no tanto realizar un Análisis Clúster a posteriori que nos hubiera obligado a etiquetar las distintas obras a través de distintas fuentes para ayudar a aprender a la red a qué grupo pertenece cada pintura, enfoque supervisado, sino lo queríamos dirigir hacia una investigación de los rasgos principales de cada una de las pinturas nos proporcionara las características o Características que la red Neuronal considerara más relevantes, un enfoque No Supervisado. La parte más destacada del estudio fue la extracción, manipulación y explotación de las Características que nos proporciona *ResNet18*. Es relevante destacar que a pesar de haber trabajado con otras redes públicas como *DenseNet*, que proporciona un número de capas mayor, tanto el rendimiento como la calidad de los resultados era inferior.

Otras de las partes fundamentales del TFM ha sido la visualización de las pinturas agrupadas en su Clúster vía web con información añadida para acceder a la web Wikiart, la cual nos ha permitido comprender la relación entre las pinturas dentro de cada grupo y como se diferencian entre el resto de los grupos.

Por último, nos gustaría destacar la sensibilidad del trabajo a la hora de la clasificación de los Clústeres, si bien la obra de Goya es compleja y podríamos haber forzado las métricas de los algoritmos *t-sne* para forzar a tener un mayor número de Clústeres, creemos que los resultados son más que satisfactorios y nos permite distinguir de forma sencilla y didáctica diferencias entre grupos.

## 4.2. Trabajo futuro.

Para futuros trabajos, se podría expandir este TFM aplicando técnicas de aprendizaje automático y teoría de grafos a una base de datos más amplia de obras de arte, incluyendo una variedad de estilos y épocas. De esta manera, podríamos obtener una visión más completa y detallada de la estructura y distribución de diferentes conjuntos de datos y hacer comparaciones entre ellos.

Por ejemplo, podríamos analizar cómo cambian las relaciones entre las pinturas a lo largo del tiempo y cómo estas relaciones reflejan los cambios en la sociedad y la cultura. También podríamos explorar cómo los artistas influyen en otros artistas y cómo las tendencias artísticas se transmiten a través de los siglos.

Además, podríamos profundizar en el análisis de las relaciones entre las pinturas utilizando técnicas de análisis de grafos más avanzadas y obtener una mejor comprensión de cómo están conectadas y cómo influyen unas en otras. Por ejemplo, podríamos utilizar medidas de centralidad y medidas de similaridad para identificar las pinturas más importantes o similares en nuestra base de datos.

También podríamos explorar la aplicación de técnicas de aprendizaje automático y teoría de grafos a otros campos, como la literatura, la música o el cine, y ver cómo estos métodos pueden aportar una visión más profunda y detallada de estos conjuntos de datos.

En conclusión, este TFM ha demostrado la utilidad de la teoría de grafos y las técnicas de aprendizaje automático para analizar y agrupar la obra de Francisco de Goya de una manera más profunda y detallada. Utilizando la teoría de grafos y la visualización, hemos podido acercar la obra de Goya a un público más amplio y ofrecer una experiencia pedagógica y atractiva para todos los interesados en la pintura.

# 5. Bibliografía.

[1] Wikiart (2022), https://www.wikiart.org/es/francisco-de-goya

[2] Google. (2020). *https://artsexperiments.withgoogle.com/tsnemap/*

[3] Alpaydin, E. Introduction to Aprendizaje Automático. MIT press, 2020.

Hinton, G. E., Sejnowski, T. J., Poggio, T. A., et al. Unsupervised learning: foundations of neural computation. MIT press, 1999.

[4] Chandra, R. V., & Varanasi, B. S. (2015). Python requests essentials. Packt Publishing Ltd.

[5] Baeza-Serrato, R., & Vázquez-López, J. A. (2014). Transición de un modelo de regresión lineal múltiple predictivo, a un modelo de regresión no lineal simple explicativo con mejor nivel de predicción: Un enfoque de dinámica de sistemas. Revista Facultad de Ingeniería Universidad de Antioquia, (71), 59-71.

[6] Diamond, S., Takapoui, R., & Boyd, S. (2018). A general system for heuristic minimization of convex functions over non-convex sets. Optimization Methods and Software, 33(1), 165-193.

[7] Haji, S. H., & Abdulazeez, A. M. (2021). Comparison of optimization techniques based on gradient descent algorithm: A review. PalArch's Journal of Archaeology of Egypt/Egyptology, 18(4), 2715-2743.

[8] Yu, Y., Adu, K., Tashi, N., Anokye, P., Wang, X., & Ayidzoe, M. A. (2020). Rmaf: Relu-memristor-like activation function for Aprendizaje Profundo. IEEE Access, 8, 72727-72741.

[9] Cilimkovic, M. (2015). Neural networks and back propagation algorithm. Institute of Technology Blanchardstown, Blanchardstown Road North Dublin, 15(1).

[10] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

[11] Wang, W., & Neumann, U. (2018). Depth-aware cnn for rgb-d segmentation. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 135-150).

[12] Yu, S., Park, B., & Jeong, J. (2019). Deep iterative down-up cnn for image denoising. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (pp. 0-0).

[13] Gao, Z., Wang, L., & Wu, G. (2019). Lip: Local importance-based pooling. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 3355-3364).

[14] Liu, L., Shen, C., & Van den Hengel, A. (2015). The treasure beneath convolutional layers: Cross-convolutional-layer pooling for image classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4749-4757).

[15] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6(02), 107-116.

[16] Guo, D., Niu, Y., & Xie, P. (2019). Speedy and accurate image super-resolution via deeply recursive CNN with skip connection and network in network. IET Image Processing, 13(7), 1201-1209.

[17] Hasan, B. M. S., & Abdulazeez, A. M. (2021). A review of principal component analysis algorithm for dimensionality reduction. Journal of Soft Computing and Data Mining, 2(1), 20-30.

[18] Arora, S., Hu, W., & Kothari, P. K. (2018, July). An analysis of the t-sne algorithm for data visualization. In Conference On Learning Theory (pp. 1455-1462). PMLR.

[19] Van Der Maaten, L. (2014). Accelerating t-SNE using tree-based algorithms. The Journal of Aprendizaje Automático Research, 15(1), 3221-3245.

[20] Hartigan, J. A., & Wong, M. A. (1979). Algorithm AS 136: A k-means clustering algorithm. Journal of the royal statistical society. series c (applied statistics), 28(1), 100-108.

[21[ Sinaga, K. P., & Yang, M. S. (2020). Unsupervised K-means clustering algorithm. IEEE access, 8, 80716-80727.

[22] Menéndez Velázquez, A. (1998). Una breve introducción a la teoría de grafos. Suma.

[23] Goldenberg, D. (2021). Social network analysis: From graph theory to applications with python. arXiv preprint arXiv:2102.10014.

[24] OpenCv (2022) *https://opencv.org/*

[25] Pytorch (2022) *https://pytorch.org/*

[26] Swapnarekha, H., Behera, H. S., Nayak, J., & Naik, B. (2022). Deep DenseNet and ResNet Approach for COVID-19 Prognosis: Experiments on Real CT Images. In Computational Intelligence in Pattern Recognition (pp. 731-747). Springer, Singapore.

[27] Networkx (2022) https://networkx.org/

[28] Bobadilla, J. (2021). Aprendizaje Automático y Aprendizaje Profundo: Usando Python, Scikit y Keras. Ediciones de la U.

[29] Borrero, I. P., & Arias, M. E. G. (2021). Aprendizaje Profundo (Vol. 19). Servicio de Publicaciones de la Universidad de Huelva.

## 6. Referencias Imágenes.

[Ilustración2] [Regresión Lineal] (n.d.). https://medium.com/iwannabedatadriven/machine-learning-modelos-de-regresi%C3%B3n-ii-18abc01a9848

[Ilustración4] [Función Coste Convexa] (n.d.). https://es.m.wikipedia.org/wiki/Archivo:Grafico_3d_x2%2Bxy%2By2.png

[Ilustración5] [Función Coste No Convexa] (n.d.). https://notebook.community/relopezbriega/mi-python-blog/content/notebooks/ecuaciones-diferenciales-en

[Ilustración6] [Función Coste No Convexa 3D] (n.d.). https://fr.wikipedia.org/wiki/Selle_de_sing

[Ilustración7] [Gradient Descent Algorithm] (n.d.). https://vitalflux.com/gradient-descent-explained-simply-with-examples/

[Ilustración8] [Redes Neuronales Artificiales] (n.d.). https://rua.ua.es/dspace/bitstream/10045/69432/1/Prediccion_de_la_probabilidad_de_exito_en_la_adqui_PAMIES_CARTAGENA_BENJAMIN.pdf

[Ilustración9] [Función ReLU] (Eddy Decena). https://medium.com/@eddydecena/entendiendo-las-redes-neuronales-part-1-fca3adf78c5b

[Ilustración10] [BackPropagation] (Eun Young Kim). https://www.researchgate.net/figure/A-Feed-forward-network-with-backpropagation-learning-input-data-flows-the-direction_fig4_221983860

[Ilustración11] [Planos RGB] (n.d.). https://cloud.tencent.com/developer/article/1809827

[Ilustración12] [Proceso de Convolución] Cortes Zarta, Juan F., Giraldo Tique, Yesica A., & Vergara Ramírez, Carlos F. (2021). http://scielo.senescyt.gob.ec/scielo.php?script=sci_arttext&pid=S1390-65422021000400088

[Ilustración13] [Convolutional Layer] (Adrià Ciurana). https://medium.com/dreamlearning/aprende-deep-learning-en-10-minutos-e4e9e8950cd8

[Ilustración14] [MaxPool & AveragePool] (n.d.). https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451

[Ilustración15] [Capas RESNET] (n.d.). https://www.redalyc.org/journal/1702/170262877013/html/

[Ilustración16] [SkipConnections] (n.d.). https://theaisummer.com/skip-connections/

[Ilustración17] [Rendimientos con menor número de capas] (n.d.). https://www.researchgate.net/journal/SN-Applied-Sciences-2523-3971

[Ilustración18] [TSNE de MNIST] (n.d.). https://www.oreilly.com/content/an-illustrated-introduction-to-the-t-sne-algorithm/

[Ilustración20] [Número óptimo de Clúster KElbow] (n.d.). https://www.codecademy.com/learn/machine-learning/modules/dspath-clustering/cheatsheet

[Ilustración21] [Grafo Etiquetado] (César A. Collazos). https://www.researchgate.net/figure/Figura-1-Ejemplo-de-una-representacion-de-red-como-grafo-dirigido-valorado_fig1_43601384

[Ilustración22] [Visualización Grafo 3DJS] (n.d.). https://www.dataviscourse.net/tutorials/lectures/lecture-d3-layouts/