Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Emparejamiento de imágenes tomadas desde diferentes perspectivas utilizando técnicas de Aprendizaje Máquina

Autor: Aitor Laiseca Valencia

Tutor: Francisco José Simois Tirado

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Emparejamiento de imágenes tomadas desde diferentes perspectivas utilizando técnicas de Aprendizaje Máquina

Autor:
Aitor Laiseca Valencia

Tutor:
Francisco José Simois Tirado
Profesor titular

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Proyecto Fin de Carrera: Emparejamiento de imágenes tomadas desde diferentes perspectivas utilizando técnicas de Aprendizaje Máquina

Autor: Aitor Laiseca Valencia

Tutor: Francisco José Simois Tirado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Estos años de carrera han sido unos años de aprendizaje, no solo a nivel académico, sino también a nivel personal. He pasado por muchas etapas malas, pero gracias a mi familia y amigos he sido capaz de sacar adelante todo el trabajo de estos años. Estuve a punto de dejar la carrera, pero mi familia me hizo no rendirme y seguir con ella. He conocido gente maravillosa en la carrera que también han sido pilares fundamentales para conseguir este logro.

Resumen

En este trabajo, se hará un resumen teórico de los distintos temas y ramas que engloban al campo de la Inteligencia Artificial, un campo que hoy en día está presente en todos los lugares por los contínuos avances y discusiones sociales de la actualidad. Se hace más enfásis en la explicación de las Redes Neuronales y, en concreto, de las técnicas de "Feature Detection and Matching".

Tras la explicación teórica, se hará uso de varios modelos de redes neuronales pre-entrenadas para resolver el problema del emparejamiento de imágenes. Una vez obtenidos los resultados de las distintas arquitecturas, se analizarán los resultados y se hará una comparativa de las mismas. Los modelos utilizados son implementaciones de las bibliotecas de estos proyectos disponibles en Python. La comparativa de resultados tiene como objetivo determinar el módelo más óptimo para resolver este problema y bajo las condiciones que se nos proponen en el desafio de Kaggle.

Abstract

In this project, a theoretical summary will be made of the different topics and branches that include the field of Artificial Intelligence, a field that nowadays is present everywhere due to the continuous advances and current social discussions. More emphasis is placed on the explanation of Neural Networks and, specifically, on the techniques of "Feature Detection and Matching".

After the theoretical explanation, several pre-trained neural network models will be used to solve the image matching problem. Once the results of the different architectures have been obtained, the results will be analyzed and compared. The models used are implementations of the libraries of these projects available in Python. The comparison of results aims to determine the most optimal model to solve this problem and under the conditions proposed in the Kaggle challenge.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

x	Producto vectorial
·	Producto escalar
f(.)	Función escalón
{. , .}	Set de valores
[. , .]	Intervalo cerrado
(. , .)	Intervalo abierto
Tanh	Parte real
Max(. , .)	Parte imaginaria
$\sum$	Sumatorio
$\leq$	Menor o igual
$\geq$	Mayor o igual
$\in$	Perteneciente a
$\forall$	Para todo
∇f	Gradiente de la función
$\frac{\delta .}{\delta .}$	Derivada parcial
Pc	Matriz de confianza

Abreviaturas

IA	Inteligencia Artificial
ANI	Inteligencia Artificial Reducida
AGI	Inteligencia Artificial General
ASI	Superinteligencia Artificial
IBM	International Business Machines Corporation
SE	Sistemas Expertos
RNA	Redes Neuronales Artificiales
ML	Machine Learning
AA	Aprendizaje Automático
RGB	Rojo, Verde y Azul
BGD	Descenso del gradiente en lotes
SCD	Descenso del gradiente estocástico
RNN	Red Neuronal Recurrente
RBF	Red de Base Radial
CNN	Red Neuronal Convolucional
LMS	Sistema de gestión de aprendizaje
GNN	Red Neuronal Gráfica
GPU	Unidad de Procesamiento Gráfico
CPU	Unidad Central de Procesamiento
RAM	Memoria de Acceso Aleatorio
CVPR	Conferencia sobre visión por computador y reconocimiento de patrones
SfM	Estructuras a partir del movimiento
SLAM	Localización y mapeo simultáneos
GUI	Interfaz gráfica de usuario
DCN	Red Convolucional Deformable
mAA	Precisión media
TP	Verdadero Positivo
TN	Verdadero Negativo
FP	Falso Positivo
FN	Falso Negativo

1 INTRODUCCIÓN

1.1 Motivación del Proyecto

En la era actual, la tecnología de la Inteligencia Artificial ha avanzado de manera exponencial y se ha convertido en un elemento omnipresente en diversos aspectos de la vida. Desde asistentes virtuales en nuestros móviles hasta aplicaciones más complejas en la medicina o la industria, la IA ha demostrado su utilidad y capacidad para transformar el mundo que nos rodea. Por ello, las empresas están invirtiendo muchos recursos en diversas aplicaciones que mejoren su productividad o lleguen a ser de uso cotidiano en la sociedad.

El propósito de este Proyecto es explorar y comprender más profundamente el potencial de la Inteligencia Artificial y su impacto en la sociedad. En un mundo donde la automatización, el análisis de datos y la toma de decisiones basada en algoritmos está cada vez más presente, es esencial acercarse a estas tecnologías para ver cómo pueden mejorar la eficiencia, precisión e innovación en muchos campos.

Para arrojar luz en el tema, se ha decidido elaborar este proyecto y así comprender las ventajas, desafíos y complicaciones de la Inteligencia Artificial.

1.2 Objetivos del Proyecto

Con el presente trabajo se pretende adquirir conocimientos básicos sobre el campo del Aprendizaje Automático y las Redes Neuronales para utilizarlos posteriormente en resolver un problema real, el emparejamiento de imágenes, y comparar resultados obtenidos por medio de modelos creados en el lenguaje de programación Python.

1.3 Organización de la Memoria

La memoria se divide en dos grandes partes, para llegar finalmente a las conclusiones del proyecto. La primera parte se ha dedicado a la explicación de los conceptos teóricos básicos para la comprensión y resolución del problema. La segunda parte se centra en la explicación de los diferentes modelos utilizados para realizar la comparativa de los resultados obtenidos con los mismos empleando los conceptos teóricos de la primera parte. En otras palabras, poner en práctica los conocimientos adquiridos.

Para la parte teórica, se busca empezar desde una perspectiva general del tema hasta llegar a una parte más detallada de la base en la que se sustenta nuestro problema a resolver. Se partirá de una introducción a la **Inteligencia Artificial** en la que se mostrarán todas sus posibles clasificaciones y utilidades, que servirá para poner en contexto el proyecto. A continuación, se explican más en detalle dos de sus ramas: el **Aprendizaje Automático** y la **Visión Artificial** pues las técnicas que utilizaremos se fundamentan en estos dos apartados que cooperan entre sí. Para concluir esta parte, se hará hincapié en los fundamentos de las **Redes Neuronales Artificiales** y, más en concreto, en las técnicas de **"Feature Detection and Matching"** pues en esto se basan los modelos utilizados.

Para la parte práctica, se exponen las **Soluciones Propuestas al Problema**. En este apartado se explicarán en detalle cada uno de las arquitecturas utilizadas para terminar con una comparativa de las mismas y las **Conclusiones** obtenidas.

2 INTELIGENCIA ARTIFICIAL

En este capítulo se busca dar una visión global de la Inteligencia Artificial. Es un buen punto de partida pues es la disciplina de la que surgen los métodos que se utilizaran posteriormente. Se empezará dando una definición y sus utilidades actuales y se terminará haciendo una clasificación en las que deriva: según su nivel de evolución y según su funcionalidad.

2.1 ¿Qué es la inteligencia artificial?

La inteligencia artificial (IA) se refiere a sistemas o máquinas que imitan la inteligencia humana para realizar tareas y pueden mejorar iterativamente a partir de la información que recopilan. Podemos ver algunos ejemplos en los que la IA se manifiesta para aportar un valor diferente:

- **Chatbots:** utilizan la IA para comprender más rápido los problemas de los clientes y proporcionar respuestas más eficientes.
- **Asistentes inteligentes:** utilizan la IA para analizar información crítica proveniente de grandes conjuntos de datos de texto libre para mejorar la programación.
- **Motores de recomendación:** utilizan la IA para automatizar las recomendaciones de productos según los hábitos de visualización de los usuarios.

2.2 Introducción a la inteligencia artificial

El término "Inteligencia Artificial" fue adoptado en 1956 pero este se ha vuelto más popular hoy en día debido al incremento en los volúmenes de datos, algoritmos y mejoras en el poder de cómputo y almacenaje.

Inicialmente, las investigaciones de la inteligencia artificial exploraban temas como la solución de problemas y métodos simbólicos, pero a partir de 1960, el Departamento de Defensa de los Estados Unidos, mostró un alto interés en este tipo de trabajos y comenzó a entrenar computadoras para que imitaran el razonamiento humano básico.

Este trabajo inicial abrió el camino para la automatización y el razonamiento que vemos hoy en las computadoras, incluyendo sistemas de soporte a decisiones y sistemas de búsqueda inteligentes que complementan y aumentan las capacidades humanas [1].

Aunque la IA se muestra al mundo como robots de aspecto humano de alto funcionamiento que se apoderan del mundo, la IA no pretende reemplazar a los humanos. Su objetivo es mejorar significativamente las capacidades y contribuciones humanas. Esto lo convierte en un activo empresarial muy valioso.

2.3 IA en la empresa

Actualmente, la IA mejora el rendimiento y productividad de la empresa mediante la automatización de procesos o tareas que antes requerían esfuerzo humano. La IA también puede dar sentido a los datos a una escala que ningún ser humano jamás podría. Por ejemplo, Netflix utiliza el "machine learning" para proporcionar un nivel de personalización que ayudó a la empresa a aumentar su base de clientes en más de un 25% en 2017 [3].

La mayoría de empresas ha hecho de la ciencia de los datos una prioridad y está realizando grandes inversiones en ella.

Según una encuesta de Gartner realizada en 2019 a más de 3.000 directores de tecnología y sistemas, clasificaron la analítica y la inteligencia empresarial como las tecnologías más importantes para sus organizaciones. También consideraban que estas tecnologías son las más estratégicas y por ello están atrayendo nuevas inversiones [2].

De acuerdo con la Harvard Bussines Review, las empresas utilizan la IA principalmente para:

- Detectar y corregir intrusiones de seguridad (44%).
- Resolver problemas tecnológicos de los usuarios (41%).
- Reducir el trabajo de la gestión de producción (34%).
- Medir el cumplimiento interno en el uso de los proveedores aprobados (34%).

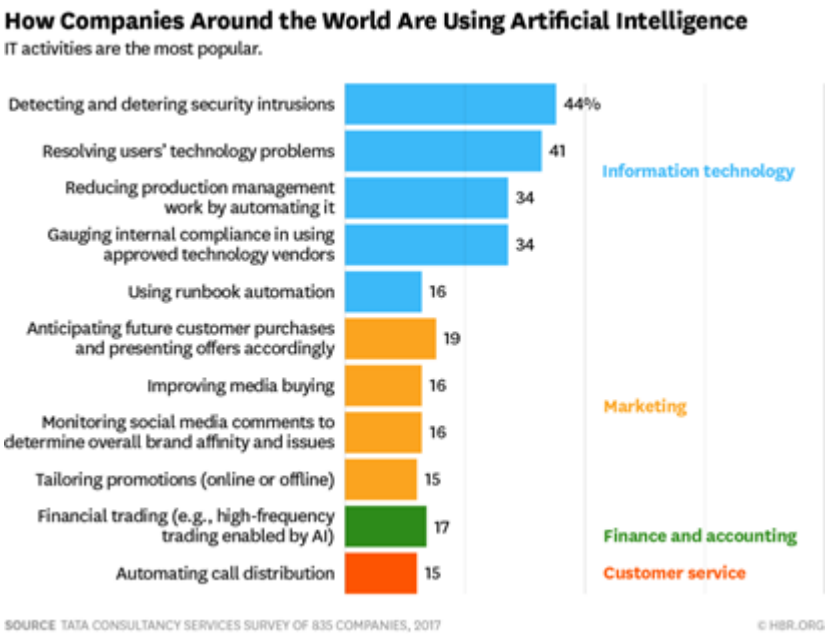


Figura 2-1. Resultado encuesta de Gartner sobre cómo usan las empresas las IA

2.4 Niveles de la Inteligencia Artificial

La inteligencia artificial tiene tres etapas o evoluciones determinadas por el grado de inteligencia del sistema.

2.4.1 Inteligencia Artificial Reducida

La inteligencia artificial reducida (también conocida como IA débil) o Artificial Narrow Intelligence (ANI) es un tipo de inteligencia artificial más inflexible, que no se amolda ni se adapta a los requisitos de un sistema o máquina en particular. Su función es centrarse en un trabajo único y dedicarle toda su capacidad.

El funcionamiento del modelo ANI se proyecta a través de la programación de sus acciones. En esa etapa, debe estar preparada para actuar en un solo rol, reduciendo su desempeño tanto como sea posible. Esto también garantizará que pueda desempeñar plenamente su papel. Puede que esto parezca una limitación, pero también puede verse como una dedicación amplia e integral [4].

Entre sus características, ANI es una inteligencia artificial con carácter reactivo y memoria limitada. Las clasificaciones técnicas ubican al ANI como una inteligencia incapaz de reproducir el comportamiento humano, solo es capaz de simularlo.

En la última década la IA débil ha experimentado numerosos descubrimientos los cuales vienen de la mano de los logros conseguidos por las técnicas de "Machine Learning" y "Deep Learning". Algunos ejemplos de estos resultados pueden ser [4]:

- Asistentes virtuales (Siri, Alexa, Cortana, etc.).
- Reconocimiento facial.
- Filtros de spam en correos electrónicos.

- Sistema de vehículos autónomos.

2.4.2 Inteligencia Artificial General

La inteligencia artificial general (AGI), también conocida como fuerte o de Nivel Humano, se refiere a aquellas máquinas que pueden realizar cualquier actividad intelectual que los seres humanos hagan.

A diferencia del tipo de Inteligencia Artificial nombrado anteriormente, crear estos procesadores es mucho más difícil y todavía no se ha logrado. Según Linda Gottfredson, la inteligencia humana es capaz de «razonar, planear, resolver problemas, pensar de manera abstracta, comprender ideas complejas, aprender rápidamente y aprender de la experiencia», por lo que, para estar en este nivel de Inteligencia Artificial, la máquina debería poder realizar todas esas acciones [5].

Es muy difícil definir el alcance que debería tener un procesador para lograr el nivel de la inteligencia humana debido a que estos tipos de Inteligencia Artificial deberían ser capaces de generar múltiples pensamientos al mismo tiempo, percibir el entorno y producir recuerdos que no necesariamente se relaciona de forma directa con la toma de decisiones de la máquina. Este desarrollo va muy ligado a la teoría de la mente donde no solo se busca una simulación o réplica de una acción sino crear la habilidad de discernir entre necesidades, emociones o sentimientos [5].

Hasta el momento, lo más cercano a la Inteligencia Artificial General son los robots humanoides que son capaces de imitar, analizar y registrar expresiones humanas [6].

Figura 2-2. Robot humanoide Ameca

De acuerdo a una encuesta realizada por KDNuggets a más de 1200 científicos repartidos por todo el mundo sobre cuándo se alcanzará la primera AGI se llegaron a los siguientes resultados [7]:

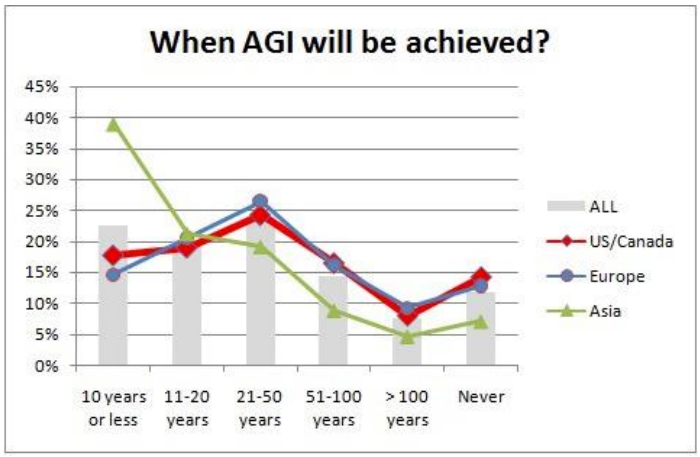


Figura 2-3. Resultado de encuesta sobre cuándo se logrará crear la primera AGI

Más del 50% de los encuestados creen que la primera AGI se logrará dentro del intervalo "entre 10 y 50 años". También podemos deducir que hay mayor esperanza (18%) de que se alcancé dentro de menos de 10 años que de que se logré en más de 100 años o nunca. Dentro de los países encuestados, Asia destaca como el continente más optimista a la hora de desarrollar estos tipos de Inteligencia Artificial opinando el 40% de ellos que se logrará en menos de 10 años frente al 20% del resto y con tan solo un 8% de los encuestados creyendo que nunca se logrará esto frente al 15% del resto de países.

2.4.3 Superinteligencia Artificial

La superinteligencia artificial (ASI) hace referencia a aquella inteligencia capaz de superar a la inteligencia humana en todos los ámbitos, desde el campo científico hasta las habilidades sociales y afectivas.

Algunos científicos como Stephen Hawking creen que cuando logremos tipos de Inteligencia Artificial que superen la inteligencia humana será el fin potencial de la humanidad. Otros como Demis Hassabis afirman que, contrario a la opinión de Hawking, cuando se alcance la súper-inteligencia, la inteligencia humana también aumentará y se podrá mejorar la relación con el medio ambiente, curar enfermedades, explorar el universo y tener una mayor comprensión de nosotros mismos [5].

Los sistemas superinteligentes siempre han sido asociados al mundo de la ciencia ficción y, actualmente, su construcción parece un logro inalcanzable.

La empresa Emerj, especializada en Inteligencia Artificial, realizó una encuesta a 32 investigadores expertos en este campo sobre cuándo se llegará a la singularidad tecnológica: un evento futuro hipotético en el que la inteligencia computacional superará a la humana [8].

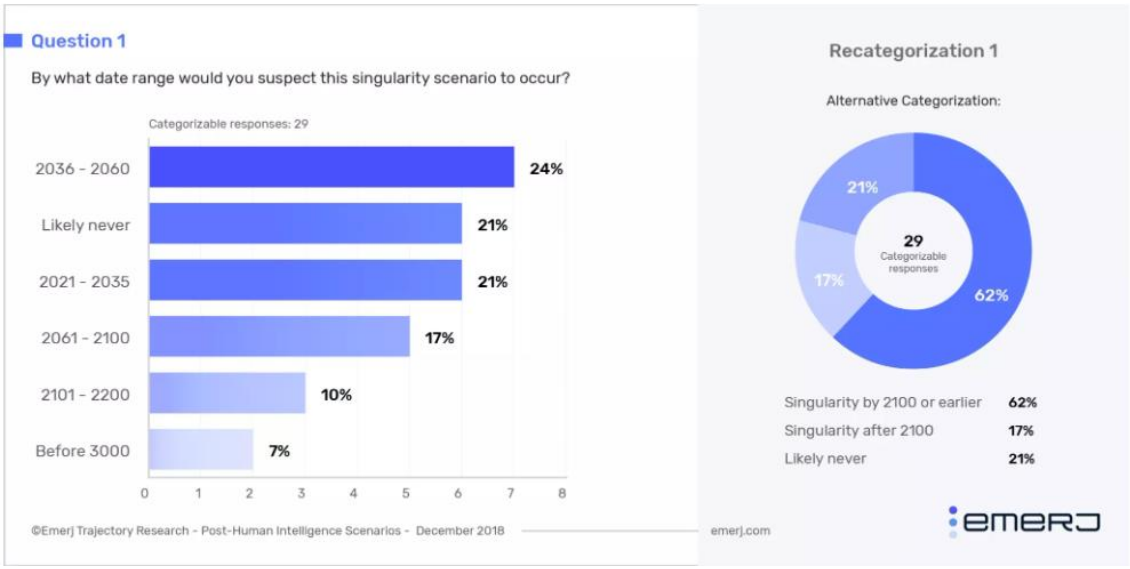


Figura 2-4. Resultado de encuesta de Emerj sobre cuándo se llegará a la singularidad tecnológica

En la encuesta, el 21% de los encuestados cree que la "Singularidad" no ocurrirá nunca frente al resto (79%) que cree que tarde o temprano se alcanzará este evento. El grueso de los encuestados, el 62%, opina que se producirá antes del año 2100 y el restante 17% cree que será después del año 2100. Viendo el resultado desde otra perspectiva, el intervalo con mayor porcentaje, 24%, es del año 2036 al 2060, seguido de cerca por el intervalo de 2021 a 2035 con un 21%. El menor porcentaje de los encuestados, un 7%, cree que se conseguirá este logro después del año 3000.

A partir de estos resultados podemos sacar en claro que hay una gran incertidumbre y disparidad de opiniones en la comunidad científica acerca de cuándo se alcanzará este hito e incluso si siquiera se logrará.

2.5 Sistemas inteligentes por funcionalidad

Una vez clasificados los sistemas según la capacidad que poseen, también se pueden clasificar a los sistemas inteligentes según la funcionalidad que son capaces de realizar.

2.5.1 Máquinas Reactivas

En este grupo se encuentran aquellas máquinas cuya función es reaccionar a eventos únicamente basándose en la percepción del mundo real cuando es estimulado.

Como no tienen memoria, esas máquinas no tienen la capacidad de aprender y administrar una base de datos interna para ejecutar lo que absorben. Así, solo tienen un rol de respuesta [9].

Uno de los ejemplos más famosos de este tipo de máquina es el proyecto de IBM, llamado Deep Blue, que fue capaz de vencer al campeón de ajedrez Garry Kasparov en 1997 [10].

Figura 2-5. Garry Kasparov enfrentándose a Deep Blue en una partida de ajedrez

2.5.2 Memoria Limitada

Los sistemas de memoria limitada son aquellos capaces de retener información durante un periodo de tiempo limitado. Estos sistemas también son completamente reactivos. Esto las hace más avanzadas ya que esta función les permite aprender de los datos. Es a partir de ahí que son capaces, cuando es necesario, de tomar pequeñas decisiones para responder a una solicitud o realizar cualquier acción [9].

Hoy en día, esa forma de inteligencia artificial se usa ampliamente. Un ejemplo actual de este tipo de sistemas lo encontramos en los coches de conducción autónoma. Estos vehículos son capaces de captar y almacenar información acerca de la velocidad de otros vehículos, la distancia entre ellos o los límites de velocidad. También los podemos ver hoy en día en los sistemas de reconocimiento facial, asistentes virtuales y chatbots.

2.5.3 Teoría de la Mente

Los sistemas englobados en el conjunto de Teoría de la Mente se definen como aquellos que serán capaces de imitar un modelo mental humano pudiendo comprender emociones, pensamientos e interactuar socialmente como humanos.

Actualmente, esa categoría todavía se ve como un futuro. Los expertos entienden que todavía se necesitan muchos avances en otras partes de los estudios de la inteligencia artificial. Por eso, la Teoría de la Mente es una idealización en ejecución, pero tiene el potencial de ser uno de los modelos más destacados [4].

2.5.4 Autoconciencia

La autoconciencia es solo una idea que el campo siempre ha soñado crear. Por ahora, todavía no hay creaciones concretas, pero la idea es que, en el futuro, las máquinas sean conscientes de sí mismas. Ese es el nivel de desarrollo más alto que puede alcanzar la inteligencia artificial [4].

La idea es que esa inteligencia artificial pueda comprender todas las emociones, tener las suyas y comprender cada detalle que pasa con quienes interactúan con ella. Sin embargo, no es posible señalar en cuántos años la tecnología podrá alcanzar ese nivel.

2.6 Ramas de la IA

Previamente se ha expuesto el objetivo principal de las Inteligencia Artificial pero esta tecnología se usa en diversas ramas distintas que aprovechan uno o varios de los procesos anteriores. A continuación, veremos algunas de estas ramas:

2.6.1 Sistemas Expertos

Los sistemas expertos (SE) son programas informáticos que tienen el objetivo de solucionar un problema concreto y utilizan la Inteligencia Artificial (IA) para simular el razonamiento de un ser humano. Se denominan sistemas expertos porque estos programas imitan la toma de decisiones de un profesional en la materia. Se crearon durante la década de los 60 y fueron uno de los primeros sistemas de Inteligencia Artificial utilizados con éxito [11].

Un sistema experto consta de 3 partes fundamentales [12]:

- **Base de conocimientos:** son las reglas que permiten representar los conocimientos del dominio de experto. Cada regla tiene de forma aislada su propio concepto. Normalmente solo reflejan una descripción de los conocimientos del experto.

- Motor de interferencias: es el responsable de seguir determinadas reglas (seleccionar las que considere oportunas) y ejecutarlas, con el objetivo de resolver el problema.

- **Base de hechos:** es la que almacena la información de las distintas fases de un sistema experto: datos de partida, criterios de parada y la actualización de la información conforme se ejecuta el sistema.

Además, podemos incluir una primera parte adicional denominada **interfaz de usuario** que será el medio por el cual los usuarios realizarán las preguntas al sistema y por donde recibirán las respuestas correspondientes.

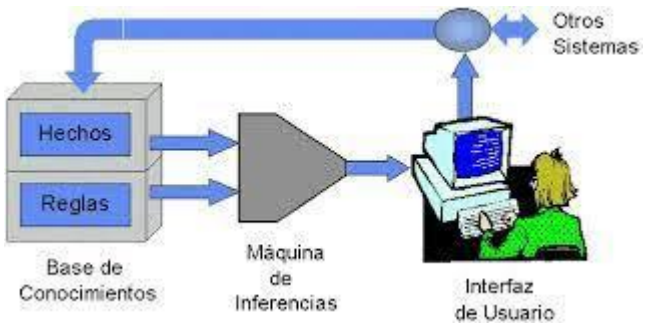


Figura 2-6. Esquema de un sistema experto

2.6.2 Robótica

Se puede hablar de robótica como la disciplina encargada de diseñar y fabricar el hardware de las máquinas (la parte física). Por el contrario, la Inteligencia Artificial sería tanto el producto como la disciplina encargada de diseñar y producir el software que hará funcionar las máquinas ensambladas por la robótica.

De este modo, se trata de elementos distintos pero que, en la mayoría de los casos actuales, tienden a darse de forma conjunta. Especialmente en el caso de robots o máquinas cuyas tareas son complejas y deben trabajar con gran cantidad de datos. Ambas disciplinas trabajan para diseñar y fabricar lo que se puede denominar en términos generales como máquinas o robots inteligentes que permitan realizar tareas con mayor fuerza o precisión que un ser humano [13].

2.6.3 Redes Neuronales

Las redes de neuronas artificiales (RNA) son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los humanos.

Consiste en simular las propiedades observadas en los sistemas neuronales biológicos a través de modelos matemáticos recreados mediante mecanismos artificiales (como un circuito integrado, un ordenador o un conjunto de válvulas). El objetivo es conseguir que las máquinas den respuestas similares a las que es capaz el cerebro que se caracterizan por su generalización y su robustez.

2.6.4 Aprendizaje automático: Machine Learning (ML)

El Machine Learning se trata de una rama de la informática de la IA que utiliza datos históricos para entrenar algoritmos de aprendizaje automático. Tras un cierto tiempo de entrenamiento, estos algoritmos mejoran su rendimiento y logran el objetivo de optimizar el proceso [14].

2.6.5 Aprendizaje profundo: Deep Learning

El Deep Learning se trata de una rama compleja del ML. El objetivo es construir y entrenar redes neuronales multicapas. El objetivo de estas redes es ordenar y clasificar los datos y encontrar anomalías en sus patrones [14].

2.6.6 Procesamiento del lenguaje natural: Natural Language Processing

El procesamiento del lenguaje natural es una rama de la IA que se refiere a la capacidad de una máquina para entender el habla humana o las palabras impresas, a diferencia de un ordenador que entiende un lenguaje de programación. Esta tecnología es utilizada por chatbots, asistentes digitales virtuales y motores de búsqueda para filtrar el spam [14].

3 APRENDIZAJE AUTOMÁTICO

Este capítulo trata de explicar todo el proceso de creación de un modelo basado en Aprendizaje Automático, indicando todos los pasos a seguir que podrá ayudar a entender la complejidad del problema.

3.1 Introducción

El Aprendizaje Automático (AA, o Machine Learning, por su nombre en inglés) es la rama de la Inteligencia Artificial que tiene como objetivo desarrollar técnicas que permitan obtener algoritmos capaces de generalizar comportamientos y reconocer patrones a partir de una información suministrada en forma de ejemplos. Es, por lo tanto, un proceso de inducción del conocimiento, es decir, un método que permite obtener por generalización un enunciado general a partir de enunciados particulares [15].

Cuando se plantean todos los casos particulares, la inducción se considera completa, por lo que la generación a la que da lugar se considera válida. Sin embargo, en la mayoría de los casos es imposible obtener una inducción completa por lo que el enunciado al que da lugar está sometido a un cierto grado de incertidumbre.

Una de las tareas del AA es intentar extraer conocimiento sobre algunas propiedades no observadas de un objeto basándose en las propiedades que sí han sido observadas de ese mismo objeto o, en otras palabras, "predecir" comportamiento futuro a partir de lo que ha ocurrido en el pasado. Un ejemplo sería el de predecir si un determinado producto le va a gustar a un cliente basándose en las valoraciones que ese mismo cliente ha hecho de otros productos que sí ha probado [15].

Hay un gran número de problemas que caen dentro de lo que llamamos aprendizaje inductivo. La principal diferencia entre ellos recae en el tipo de objetos que intentan predecir. Algunas clases son [15]:

- Regresión: intentan predecir un valor real. Por ejemplo, predecir el valor de la bolsa a partir del comportamiento de la bolsa que está almacenado.

- Clasificación (binario o multiclase): intentan predecir la clasificación de objetos sobre un conjunto de clases prefijadas. Por ejemplo. clasificar si una noticia es de deportes, entretenimiento, política, ...Si solo se permiten 2 posibles clases, entonces se llama clasificación binaria; si se permiten más, hablamos de clasificación multiclase.

- Ranking: intenta predecir el orden óptimo de un conjunto de objetos según un orden de relevancia predefinido. Por ejemplo, el orden en que un buscador devuelve recursos de internet como respuesta a una búsqueda.

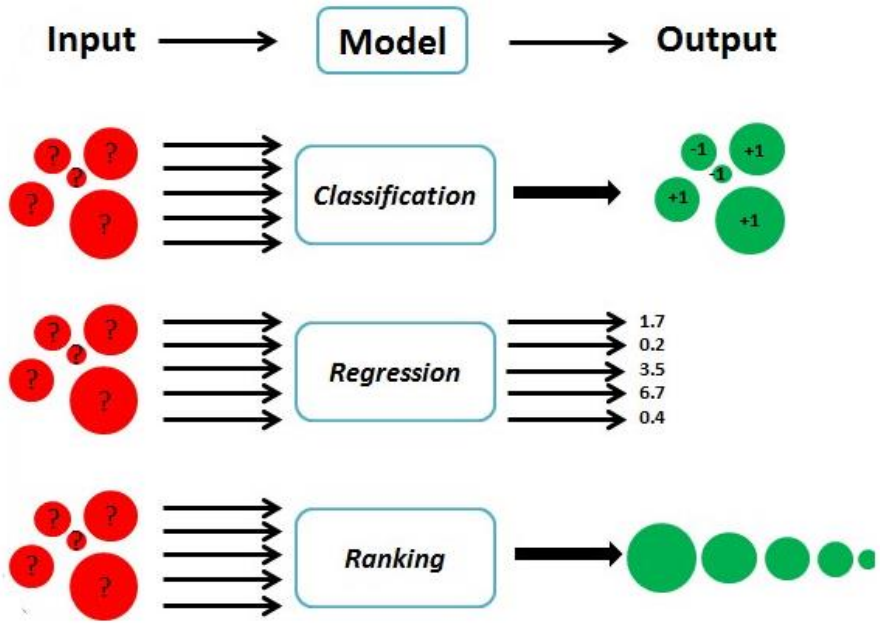


Figura 3-1. Clasificación de clases del Aprendizaje Automático

3.2 Etapas del proceso de Aprendizaje Automático

3.2.1 Fase 1: Entender el problema

Es muy importante entender el problema que tenemos que resolver. Entender el problema normalmente lleva bastante tiempo. Debemos ser realistas en cuanto a nuestro objetivo dado las características del problema, así como de los datos que tenemos a nuestra disposición [16].

En esta etapa es típico hacernos las siguientes preguntas:

A. ¿Qué deseamos hacer exactamente?

B. ¿Cómo podemos hacerlo?

C. ¿Es posible hacer lo que buscamos con los datos que tengo?

3.2.2 Fase 2: Recolección de los datos

Dependiendo del problema a resolver, deberemos investigar y obtener los datos que utilizaremos para "alimentar a la máquina". Es muy importante la calidad y cantidad de información ya que impactará directamente en lo bien o mal que luego funcione nuestro modelo [17].

3.2.3 Fase 3: Preparar los datos

Esta es una de las fases del Machine Learning que supone un mayor esfuerzo. Nos encontraremos las siguientes situaciones [16]:

- Datos incompletos: es normal no tener todos los datos que nos gustaría tener. Por ejemplo, en una encuesta puede haber clientes que no la rellenen entera. Para solucionar esto podremos eliminar aquellos datos defectuosos (esto está bien si son minoría) o bien completarlos manualmente con un valor razonable.

- Combinar datos de varias fuentes: tendremos que combinar datos de forma que los algoritmos de machine learning pueda considerar toda la información

- Darles el formato adecuado a los datos: si queremos usar librerías ya disponibles, debemos darles un formato a nuestros datos que reconozcan estas librerías tales como matrices o tensores.

- Calcular características relevantes (features): los algoritmos de Machine Learning funcionan

mucho mejor si les ofrecemos características relevantes en vez de los datos puros. Así, la tarea de aprendizaje es más fácil.

- **Normalización de datos:** en muchos casos, es útil normalizar los datos para hacer más fácil la técnica del aprendizaje a la máquina. Por normalizar nos referimos a poner todos los datos en una escala similar.

3.2.4 Fase 4: Elección del algoritmo

Durante esta fase tenemos que elegir qué tipo de técnica de machine learning queremos usar de acuerdo al objetivo que tengamos [17].

3.2.5 Fase 5: Entrenar el modelo

Un porcentaje de los datos, comúnmente el 70% del total, los utilizaremos como datos de entrenamiento. El algoritmo aprenderá automáticamente a obtener los resultados adecuados con los datos históricos que hemos preparado [17].

3.2.6 Fase 6: Validación del modelo

Dado que ya tenemos el modelo entrenado, usaremos el 30% restante de los datos. Simplemente ejecutaremos el algoritmo y evaluaremos los resultados obtenidos. Debemos ser conscientes que existe la posibilidad de que el modelo funcione bien para los datos de entrenamiento y no para los datos de validación. Volveremos a la etapa 6 hasta que nuestro modelo se ajuste bien a las dos particiones [17].

3.2.7 Fase 7: Predicción

Ya estamos listos para utilizar nuestro modelo de Aprendizaje Automático con nueva información y comenzar a predecir o inferir resultados en “la vida real”.

4 VISIÓN ARTIFICIAL

En este capítulo se busca dar una primera visión básica de la Visión Artificial centrándonos en las similitudes entre la percepción del ojo humano de las imágenes y la percepción de una computadora, así como los métodos que emplea.

4.1 Introducción a la visión artificial

La visión artificial es una parte de la inteligencia artificial (IA) que está en alza. Se centra en el desarrollo y el perfeccionamiento de técnicas que permiten a las máquinas ver, identificar y procesar imágenes de la misma manera que lo hace la visión humana.

Esta comprensión se consigue gracias a distintos campos como la geometría, la estadística, la física y otras disciplinas. La adquisición de los datos se consigue por varios medios como secuencias de imágenes, vistas desde varias cámaras de video o datos multidimensionales desde un escáner médico.

El objetivo último de la visión artificial es conseguir el desarrollo de estrategias automáticas para el reconocimiento de patrones complejos en imágenes de múltiples dominios. En la actualidad, muchos son los campos que se han visto beneficiados por este conjunto de técnicas. Uno de los más conocidos es el de la robótica, ya que los robots con cierta autonomía deben reconocer con precisión la localización de los objetos de su entorno para no colisionar contra ellos, por ejemplo. A menudo, esto lo consiguen por medio de sensores o de cámaras.

Sin embargo, la robótica no es el único ámbito que se ha visto beneficiado por este conjunto de técnicas. Podemos destacar el ámbito de la imagen médica, con sistemas capaces de reconocer, por ejemplo, patrones patológicos en una modalidad de imagen determinada y diagnosticar enfermedades de forma automatizada. También se emplean en otros ámbitos, como en sistemas de seguridad, seguimiento de objetos (por ejemplo, seguimiento de un futbolista en vídeo durante un partido de fútbol) o detección de anomalías en piezas fabricadas en una cadena de producción [18][19].

4.2 Visión Humana

La visión humana gira en torno a la luz y no implica repetición ni patrones, es decir, no necesitamos aprender a ver, está biológicamente arraigado en nosotros [20].

La visión humana consta de varios pasos. Primero, la luz rebota en la imagen y entra en los ojos a través de la córnea. Luego, la córnea dirige la luz a las pupilas y al iris, que trabajan juntos para controlar la cantidad de luz que ingresa al ojo. Una vez que la luz atraviesa la córnea, entra en la retina; la retina tiene sensores especiales llamados conos y bastones, que participan en la visualización del color.

Por lo tanto, la visión es, en primer lugar, una tarea de procesamiento de información ya que para poder entender lo que hay en una imagen, nuestro cerebro debe ser capaz de representar esta información como: color, forma, movimiento, detalle y belleza.

4.3 ¿Cómo funciona la Visión Artificial?

Para entender cómo percibe el mundo una computadora, comencemos por definir qué es una imagen digital y el procesamiento básico que realiza.

4.3.1 Imagen

Una imagen digital es una matriz. Una imagen digital está compuesta de un número finito de elementos, cada uno tiene una posición en un plano con coordenadas x, y, y un valor asociado al color de la imagen en ese punto. A estos elementos se les llama puntos elementales de la imagen o píxeles, siendo este el término usado para

denotar la unidad mínima de medida de una imagen digital.

4.3.2 Color

La representación del color de la imagen en un pixel, puede ser binaria, para el caso de blanco y negro, o un número que representa la intensisdad en la escala del gris o una gama de colores.

La primera imagen es una representación binaria de una imagen en blanco y negro. Se asigna 1 al color negro y 0 al color blanco.

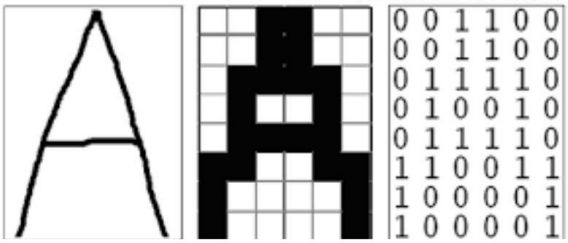


Figura 4-1. Paso de una imagen en blanco y negro a matriz binaria

La segunda imagen representa una imagen con 256 niveles de intensidad. Cada uno de los píxeles representa un número entero que es interpretado como el nivel de intensidad luminosa en la escala de grises.

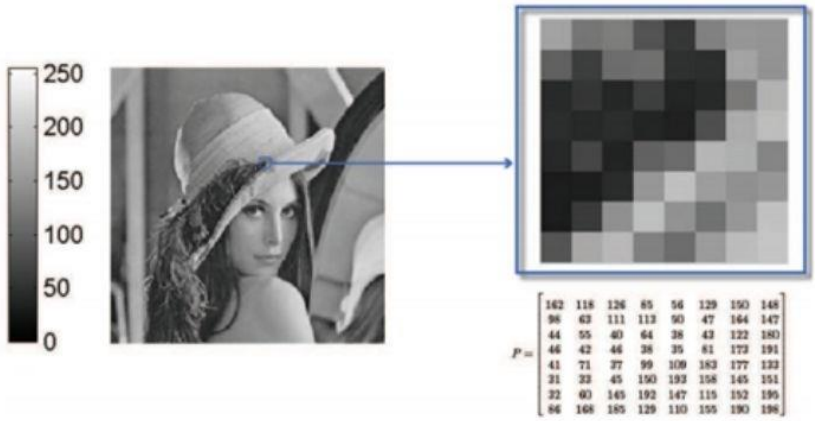


Figura 4-2. Representación de una imagen de forma matricial en escala de grises

En una imagen en color, cada píxel de la imagen se representa con tres valores que codifican su color como una combinación de la cantidad de rojo, verde y azul (conocido en inglés como RGB). El color resultante del píxel vendrá dado por tanto por la intensidad que tenga cada componente.

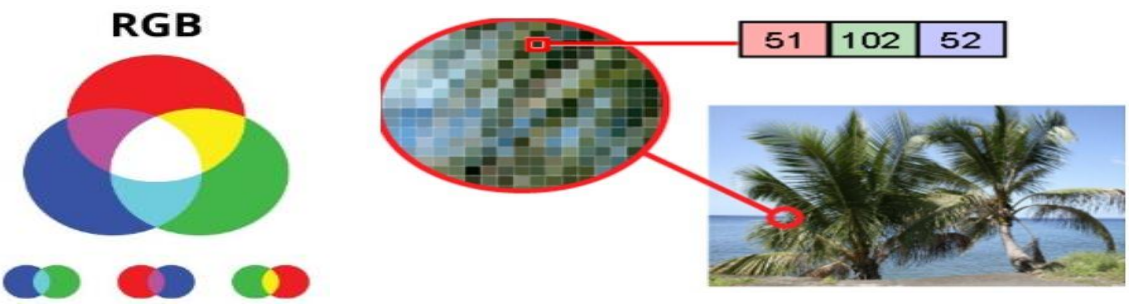


Figura 4-3. Representación de un píxel de una imagen a color siguiendo el patrón RGB

4.3.3 Resolución

La resolución es la cantidad de píxeles que contiene una imagen. La resolución de una imagen se representa en dos valores numéricos, donde el primero es la cantidad de columnas de píxeles (ancho) y el segundo es la cantidad de filas de píxeles (alto).

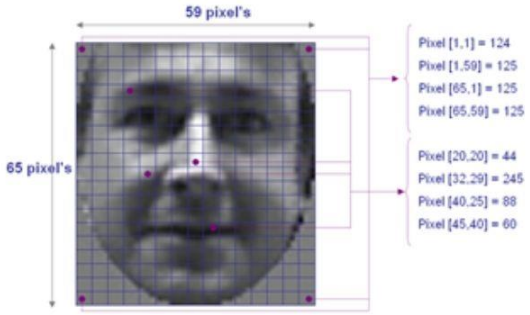


Figura 4-4. Representación de la resolución de una imagen

5 REDES NEURONALES ARTIFICIALES

Una vez explicadas las técnicas de creación de modelos y visión de las imágenes en las computadores, en este capítulo se aborda el tema de las Redes Neuronales Artificiales pues son las responsables de arrojarnos los resultados que buscamos aplicando lo aprendido en los capítulos anteriores. Así, a lo largo del capítulo, se hará una introducción a las Redes Neuronales Artificiales, como se clasifican, las formas de entrenarlas y los fundamentos matemáticos en los que se basan para obtener los resultados.

5.1 Introducción a las redes neuronales

Las Redes Neuronales fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, constituidos por un conjunto de unidades llamadas neuronas o nodos conectados unos con otros. El primer modelo de red neuronal fue propuesto en 1943 por McCulloch y Pitts. Este modelo era un modelo binario, donde cada neurona tenía un escalón o umbral prefijado, y sirvió de base para los modelos posteriores.

5.2 Redes Neuronales de tipo biológico

Se estima que el cerebro humano contiene más de cien mil millones (10^{11}) de neuronas y 10^{14} interconexiones (sinapsis) en el sistema nervioso. Estudios sobre la anatomía del cerebro humano concluyen que, en general, hay más de 1000 conexiones a la entrada y salida de cada neurona.

El tiempo de conmutación de las neuronas biológicas es casi un millón de veces mayor que el tiempo de los actuales componentes de los ordenadores y las neuronas naturales tienen una conectividad miles de veces superior a las artificiales [18].

Una neurona tiene tres partes:

1º El cuerpo de la neurona.

2º Las dendritas, que reciben las entradas.

3º El axón, que llevará la salida de la neurona a las dendritas de otras neuronas.

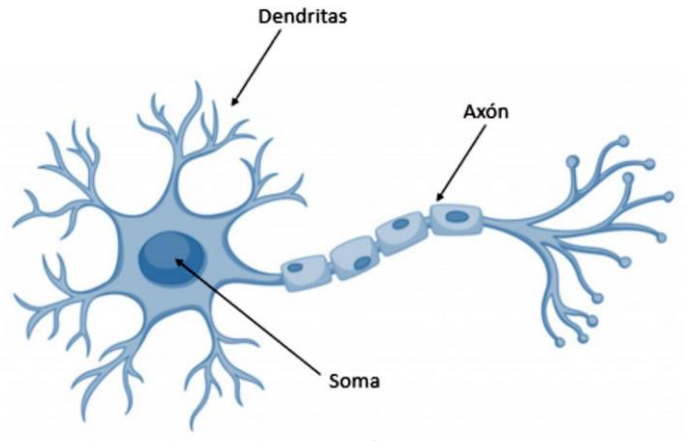


Figura 5-1. Imagen ilustrativa de las partes de una neurona biológica

Las neuronas y las conexiones entre ellas -sinapsis- constituyen la clave para el procesado de la información. La mayor parte de las neuronas poseen una estructura de árbol, llamada dendritas, que reciben las señales de entrada

procedentes de otras neuronas a través de las sinapsis.

Este proceso es a menudo modelizado como una regla de propagación representada por una función u (·). La neurona recoge las señales por su sinapsis sumando todas las influencias excitadoras e inhibidoras. Si las influencias excitadoras positivas dominan, entonces la neurona produce una señal positiva y manda este mensaje a otras neuronas por sus sinapsis de salida. En este sentido la neurona actúa como una simple función escalón f (·) [21].

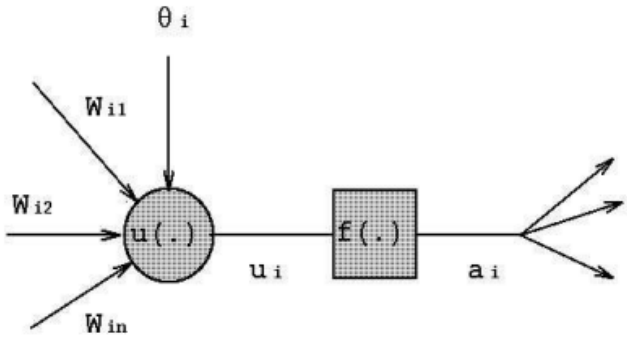


Figura 5-2. Modelo matemático de procesamiento de la información de una neurona

Las redes neuronales se enmarcan dentro del campo de la Inteligencia Artificial. La idea principal del Deep Learning es observar el cerebro humano e inspirarse en él para intentar reproducir de forma informática su comportamiento.

5.3 Redes Neuronales en Inteligencia Artificial

A nivel esquemático, una neurona artificial se representa así:

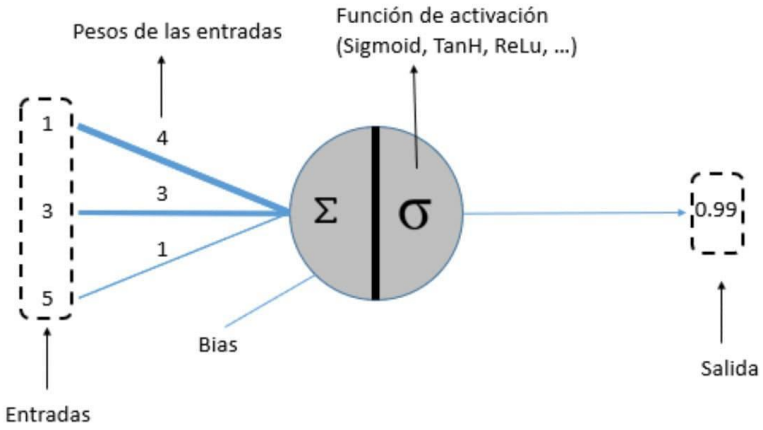


Figura 5-3. Esquema neurona artificial

En el caso de las neuronas artificiales, la suma de las entradas multiplicadas por sus pesos asociados determina el "impulso nervioso" que recibe la neurona. Este valor se procesa en el interior de la célula mediante una función de activación que devuelve un valor que se envía como salida de la neurona [22].

Una red neuronal artificial, al igual que nuestro cerebro, está formada por neuronas artificiales conectadas entre sí y agrupadas en diferentes niveles llamados capas.

Capa: Una capa es un conjunto de neuronas cuyas entradas provienen de una capa anterior (o de los datos de entrada en el caso de la primera capa) y cuyas salidas son la entrada de una capa posterior.

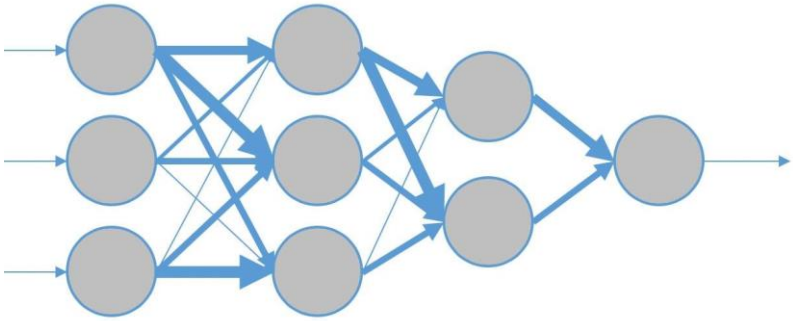


Figura 5-4. Modelo de red neuronal con cuatro capas

En esta imagen podemos ver una red con cuatro capas.

Las neuronas de la primera capa reciben como entrada los datos reales que alimenta a la red neuronal. Por eso la primera capa se denomina como *capa de entrada*. La salida de la última capa es el resultado visible de la red, por lo que la última capa se conoce como la *capa de salida*. Las capas que se sitúan entre la capa de entrada y la capa de salida se conocen como *capas ocultas* ya que desconocemos sus valores de entrada y de salida [22].

Así, podemos afirmar que una red neuronal siempre está compuesta por una capa de entrada, una capa de salida y puede contener 0 o más capas ocultas.

El concepto de Deep Learning nace a raíz de utilizar un gran número de capas ocultas en la red.

5.4 Funciones de activación

La función de activación se encarga de devolver una salida a partir de un valor de entrada. Las funciones de activación pueden ser binarias o continuas según el valor que envían a la siguiente capa. Las funciones binarias convertirán la combinación lineal de la entrada a solamente dos valores que suelen ser {0,1} o {-1,1}. Por otro lado, las funciones continuas devolverán un número cualquiera dentro de un rango predefinido, usualmente [-1,1].
Se buscan funciones cuyas derivadas sean simples, para minimizar con el coste computacional.

5.4.1 Tipos de funciones de activación

5.4.1.1 Sigmoide (Sigmoid)

Transforma los valores introduces a una escala (0,1) donde los valores altos tienden de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a 0 [23].

Características:

- Satura y mata el gradiente.
- Lenta convergencia.
- No está centrada en el cero.
- Está acotada entre 0 y 1.
- Buen rendimiento en la última capa.

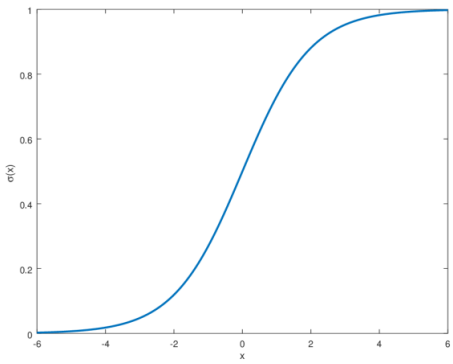


Figura 5-5. Representación función Sigmoide

5.4.1.2 Tangente hiperbólica (Tanh)

Transforma los valores introducidos a una escala (-1,1), donde los valores altos tienden de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a -1 [23].

Características:

- Satura y mata el gradiente.
- Lenta convergencia.
- Centrada en 0.
- Acotada entre -1 y 1.
- Se utiliza para decidir entre una opción y la contraria.
- Buen desempeño en redes recurrentes.

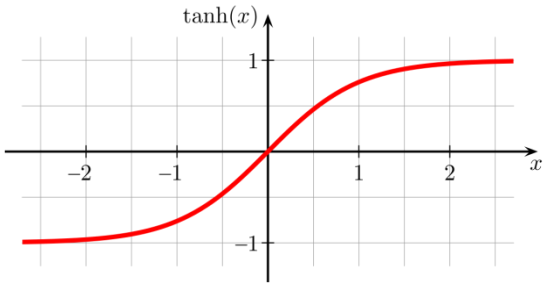


Figura 5-6. Representación función Tangente hiperbólica

5.4.1.3 Rectified Lineal Unit (ReLU)

Transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran [20].

Características:

- Activación Sparse – solo se activa si son positivos.
- No está acotada.
- Se pueden morir demasiadas neuronas.
- Se comporta bien con imágenes.
- Buen desempeño en redes convolucionales.

$$f(x) = \max(0, x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$$

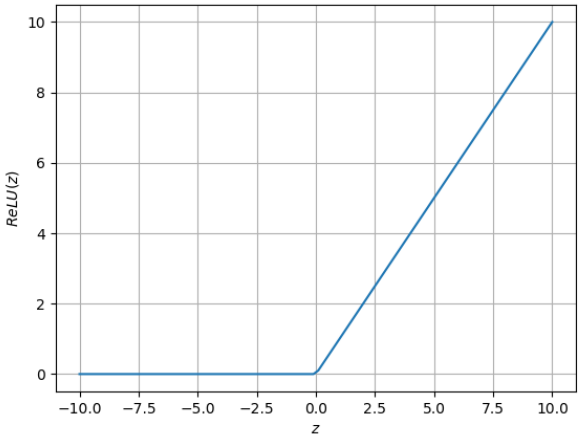


Figura 5-7. Representación función ReLU

5.4.1.4 Leaky Rectified Lineal Unit (Leaky ReLU)

Transforma los valores introducidos multiplicando los negativos por un coeficiente rectificativo y dejando los positivos según entran [23].

Características:

- Similar a la función ReLU.
- Penaliza los negativos mediante un coeficiente rectificador.
- No está acotada.
- Se comporta bien con imágenes.
- Buen desempeño en redes convolucionales.

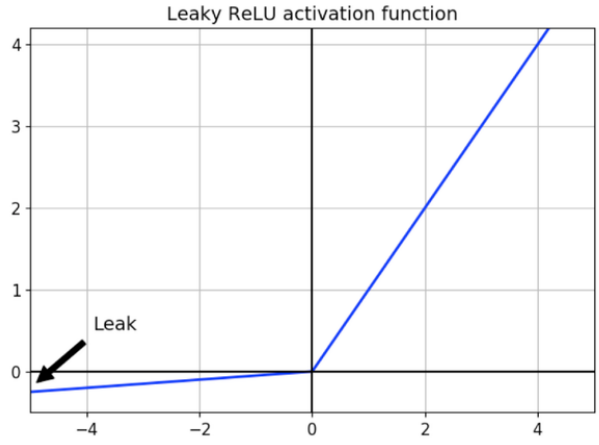


Figura 5-8. Representación función Leaky ReLU

5.4.1.5 Softmax

Transforma las salidas a una representación en forma de probabilidades, de tal manera que el sumatorio de todas las probabilidades de las salidas sea 1 [23].

Características:

- Se utiliza cuando queremos tener una representación en forma de probabilidades.
- Acotada entre 0 y 1.
- Muy diferenciable.
- Se utiliza para normalizar tipo multiclase.
- Buen rendimiento en las últimas capas.

$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

5.5 Algoritmo del descenso del gradiente

El algoritmo más utilizado para entrenar redes neuronales es el descenso del gradiente. De forma simple, el gradiente es un cálculo que nos permite saber cómo ajustar los parámetros de la red de tal forma que se minimice su desviación a la salida [24].

En RNA, los parámetros que buscamos ajustar son los pesos y sesgos. La función de coste estará en función de estos parámetros. Al comenzar el entrenamiento de un modelo, los valores de los parámetros, salvo que los usuarios indiquen lo contrario, son inicializados con unos valores aleatorios. Esto produce un valor inicial aleatorio de la función de coste, es decir, un punto de la representación multidimensional función de coste-parámetros [25].

Nuestro objetivo es minimizar el error pero no podemos conocer a priori todos los valores de la función de pérdida frente a los parámetros por lo que se opta por calcular la pendiente (derivada de la función de coste) en la posición actual. Al tratarse de una función multidimensional, calculamos las derivadas parciales para cada uno de los pesos, "w", y sesgos, "b", consiguiendo la pendiente en el eje de cada parámetro. Si agrupamos todas las derivadas parciales en un vector, se obtiene el gradiente de la función, ∇f [25].

$$\nabla f = [\frac{\delta coste}{\delta w1}, \frac{\delta coste}{\delta w2}, \dots, \frac{\delta coste}{\delta wN}, \frac{\delta coste}{\delta b1}, \frac{\delta coste}{\delta b2}, \dots, \frac{\delta coste}{\delta bL}] \quad (5.1)$$

Cuando una derivada parcial toma un valor grande, hace falta una modificación del parámetro en cuestión para reducir el error. Si toma un valor pequeño, significa que el valor actual no perjudica al desempeño de la RNA.

Para calcular cada una de las derivadas parciales, el algoritmo de descenso del gradiente utiliza otro algoritmo conocido como "backpropagation".

Este algoritmo comienza calculando las derivadas parciales en la "output layer" de la RNA para ir progresando hacia atrás hasta la primera "hidden layer", reutilizando las derivadas parciales calculadas y las salidas de las neuronas. El funcionamiento se basa en la composición de funciones y la regla de la cadena. Así, en la última capa, el coste es función de la salida, la salida es función de la combinación lineal y la combinación lineal es función de los pesos y sesgo de la capa anterior. Podemos resumir esto en sólo cuatro ecuaciones [25]:

$$\delta^L = \frac{\delta C}{\delta \alpha^L} \cdot \frac{\delta \alpha^L}{\delta Z^L} \quad (5.2)$$

$$\delta^{l-1} = \frac{\delta z^l}{\delta \alpha^{l-1}} \cdot \delta^l \cdot \frac{\delta \alpha^{l-1}}{\delta Z^{l-1}} \quad (5.3)$$

$$\frac{\delta coste}{\delta b^{l-1}} = \delta^{l-1} \quad (5.4)$$

$$\frac{\delta C}{\delta w^{l-1}} = \delta^{l-1} \cdot \alpha^{l-2} \quad (5.5)$$

Donde:

- Superíndice ***L*** representa la capa de salida de la RNA.
- Superíndice ***l*** representa la capa actual, ***l-1*** la anterior y ***l-2*** la anterior a esta.

- $\boldsymbol{\alpha}$ representa la salida de la neurona.

- **z** es la suma ponderada de pesos y sesgos.

- $\boldsymbol{\delta}$ es el error de la neurona.

La ecuación (5.2) es el cómputo del error de la última capa, calculado con la función de coste y la ecuación (5.3) es el error de la capa anterior a la última. Hay que tener en cuenta que las funciones de activación empleadas en las neuronas deben tener derivada distinta de cero o infinito ya que, si no, no podríamos utilizar el algoritmo de "backpropagation" en las ecuaciones (5.2) y (5.3) y no funcionaría el descenso del gradiente.

Una vez definido cómo se calcula nuestro vector $\boldsymbol{\nabla f}$, debemos tener en cuenta que la dirección de la función de coste hace que aumente su valor por lo que se produciría mayor error. Justo lo que no queremos que ocurra. Así, lo lógico es utilizar el "negativo del gradiente" para obtener una combinación de pesos y sesgos con la que obtener un menor valor de la función de coste inicial.

Si repetimos sucesivas veces este proceso, llegará un punto en el que iterar de nuevo no afectará prácticamente al resultado. Este punto suele tratarse de un mínimo local de la función de coste que puede llegar a ser global dependiendo del azar en la inicialización y del tipo de función de coste: convexa (un único mínimo) o no convexa (varios mínimos) [26].

Para terminar de completar el algoritmo debemos añadir un parámetro más, la tasa de aprendizaje o learning rate. Este valor define cuánto afecta el gradiente a la variación de los parámetros en cada iteración, es decir, la velocidad con la que se avanza en la dirección del negativo del gradiente. Es importante hallar un valor óptimo de este parámetro pues una tasa muy baja provoca un excesivo tiempo de aprendizaje pudiendo finalizar en un valor diferente al mínimo local. Por otra parte, un valor demasiado elevado, dificulta la convergencia en el mínimo, pudiendo hacer que se entre en un bucle infinito [26].

$$W_{i,j+1} = w_{i,j} - \alpha \boldsymbol{\nabla f_j}(i), \forall\, i \in [1, n] \tag{5.6}$$

En la ecuación (5.6) se muestra como para cada peso *i* se actualiza su valor de la iteración *j* a uno nuevo en la iteración *j+1* gracias a restarle un "learning rate" definido y multiplicado por el gradiente calculado en la iteración *j* y evaluado en la posición del peso *i*.

5.5.1 Versiones en función del número de muestras

Dependiendo del número de muestras que introduzcamos a la red en cada iteración podemos clasificar el algoritmo en las siguientes categorías.

5.5.1.1 Descenso del gradiente en lotes (o batch)

El descenso del gradiente por lotes calcula los errores de cada ejemplo del conjunto de datos de entrenamiento. Sin embargo, actualiza el valor de los pesos tras realizar una evaluación de todo el conjunto de datos. Habitualmente, a un periodo de entrenamiento con todos los datos del dataset de entrenamiento se le denomina "época" (epoch).

La ventaja de este método es su eficiencia computacional ya que desarrolla una convergencia en pocas iteraciones y muy robusta, pero requiere de mucha potencia computacional, sobre todo con datasets masivos [27].

5.5.1.2 Descenso del gradiente estocástico

Proporciona actualizaciones de los parámetros individuales para cada ejemplo de entrenamiento. Este procedimiento requiere que el orden del conjunto de datos de entrenamiento sea aleatorio para mezclar el orden en que se actualizan los coeficientes de los pesos. Debido a que los coeficientes se actualizan después de cada instancia de entrenamiento, el valor de los pesos y de la función de coste variará mucho de una iteración a otra. Esta fluctuación permite hallar mejores mínimos locales pero dificulta encontrar el mínimo exacto.

El aprendizaje es mucho más rápido para conjuntos de datos de entrenamiento grandes y, a menudo, solo necesita una pequeña cantidad de iteraciones para alcanzar un conjunto de coeficientes lo suficientemente bueno por lo

que es el algoritmo que requiere menor potencia computacional [27].

5.5.1.3 Descenso del gradiente (estocástico) en mini lotes (mini-batch)

Es el método más utilizado por los científicos de los datos por ser una mezcla de los conceptos de descenso de gradiente estocástico y de descenso de gradiente por lotes.

Divide el conjunto de datos en lotes y ejecuta una actualización para cada lote buscando así un equilibrio entre la eficiencia del BGD y la robustez del SCD.

Los valores más populares para los lotes oscilan entre 50 y 256. Este valor recibe el nombre de batch size (tamaño del lote). Además, al periodo de entrenamiento se le denomina step (paso). De esta forma, tras cada "step" se calcula el gradiente de la función de coste a "batch size" muestras y se actualizan los pesos.

La relación entre "epochs" y "steps" es la siguiente. El número de "epochs" es elegido por el usuario.

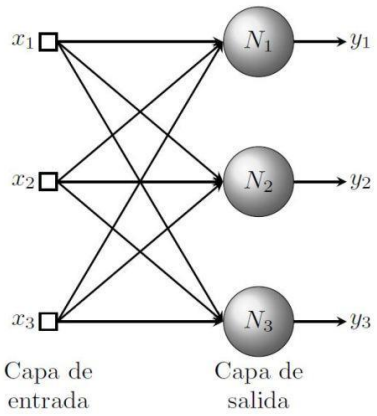
$$steps\ per\ epoch = \frac{nº\ datos\ de\ entrenamiento}{batch\ size} \tag{5.7}$$

Además de estas versiones del descenso del gradiente, existen otros algoritmos más complejos que tratan de optimizar el descenso del gradiente estocástico. Algunos de ellos son: "Momentum", "RMSProp" o "Adam" [28].

5.6 Tipos de Redes Neuronales Artificiales

5.6.1 Red Neuronal Monocapa (Perceptrón simple)

Se corresponde con la red neuronal más simple, está compuesta por una capa de neuronas que proyectan las entradas a una capa de neuronas de salida donde se realizan los diferentes cálculos [26].



$$f(x) = \begin{cases} 0 & for\ x < 0 \\ a \cdot x\ for\ x \geq 0 \end{cases}$$

Figura 5-9. Esquema Red Neuronal Monocapa

5.6.2 Red Neuronal Multicapa (Perceptrón multicapa)

Es una generalización de la red neuronal monocapa. La diferencia reside en que esta está compuesta por una capa de neuronas de entrada y una capa de neuronas de salida junto con un conjunto de capas intermedias (capas ocultas) entre estas.

Dependiendo del número de conexiones que presente la red esta puede estar total o parcialmente conectada [29].

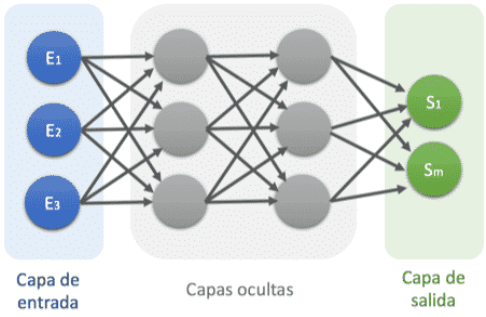


Figura 5-10. Esquema Red Neuronal Multicapa

5.6.3 Red Neuronal Recurrente (RNN)

No tienen una estructura de capas, sino que permiten conexiones arbitrarias entre las neuronas, incluso pudiendo crear ciclos. Con esto se consigue crear la temporalidad, permitiendo que la red tenga memoria. Los datos introducidos en el momento "t" en la entrada son transformadas y van circulando por la red incluso en los instantes de tiempo siguientes "t+1", "t+2", … [29].

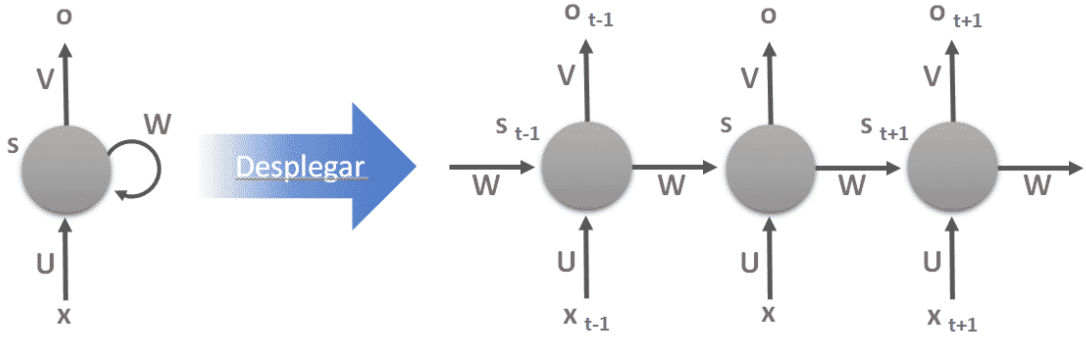


Figura 5-11. Esquema Red Neuronal Recurrente

5.6.4 Redes de base radial (RBF)

Calculan la salida de la función de la distancia a un punto denominado centro.

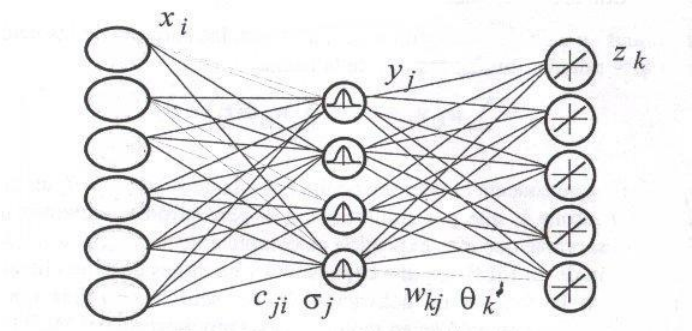


Figura 5-12. Esquema Red de Base Radial

5.6.5 Red Neuronal Convolucional (CNN)

La principal diferencia de la red neuronal convolucional con el perceptrón multicapa viene en que cada neurona no se une con todas y cada una de las capas siguientes, sino que solo con un subgrupo de ellas (se especializa). Con esto se consigue reducir el número de neuronas necesarias y la complejidad computacional necesaria para su ejecución [29].

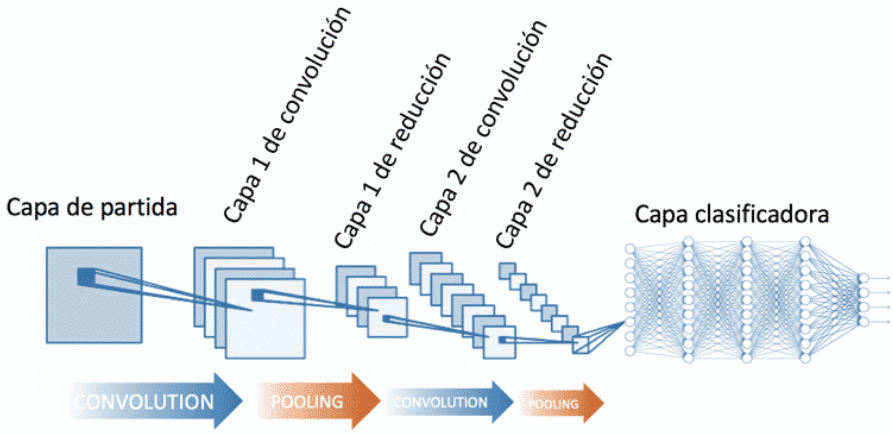


Figura 5-13. Esquema Red Neuronal Convolucional

5.7 Etapas redes convoluciones

5.7.1 Capa Entrada

Para comenzar, la red toma como entrada los píxeles de una imagen. Si tenemos una imagen de 28x28 pixeles de alto y ancho, esto equivale a utilizar 784 neuronas. Esto es sólo si tenemos 1 color (escala de grises). Si tuviéramos una imagen a color, se utilizarían 3 canales RGB y entonces usaremos 28x28x3 = 2352 neuronas. Esto constituye nuestra capa de entrada [30].

5.7.2 Pre-Procesamiento

Antes de alimentar la red, conviene convertir los valores entre 0 y 1 y por tanto tendremos que dividirlos todos entre 255 (esto es debido a que los colores de los píxeles tienen valores entre 0 y 255) [30].

5.7.3 Proceso convolución

Las "convoluciones" consisten en tomar grupos de píxeles cercanos de la imagen de entrada e ir operando matemáticamente (producto escalar) contra una pequeña matriz llamada kernel. Ese kernel irá "visualizando" todas las neuronas de entrada (de izquierda-derecha, de arriba-abajo) y generará una nueva matriz de salida, que en definitiva será nuestra capa de neuronas ocultas. El kernel tomará inicialmente valores aleatorios y se irán ajustando mediante backpropagation.

No usaremos 1 sólo kernel, si no que tendremos muchos kernel. Al conjunto de Kernerls se le denomina como filtros. Por ejemplo, podríamos tener 32 filtros, con lo que obtendremos 32 matrices de salida (este conjunto se conoce como "Feature Mapping" o mapa de características), cada una de 28x28x1 que dará lugar a 25.088 neuronas en la primera capa oculta de neuronas. A medida que vamos desplazando el kernel vamos obteniendo una "nueva imagen" filtrada por el kernel.

Estas "imágenes nuevas" lo que están haciendo es "dibujar" ciertas características de la imagen original [30].

5.8 Clasificación según método de aprendizaje

5.8.1 Aprendizaje supervisado

Se caracteriza porque el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un supervisor que determina la respuesta que se debe generar para cada entrada.

El supervisor controla la salida y si esta no es correcta, modifica los pesos de las conexiones, con el fin de que la salida obtenida se aproxime a la deseada [29].

- **Aprendizaje por corrección de error:** ajusta los pesos de las conexiones de la red en función del error cometido, es decir la diferencia entre los valores esperados y los obtenidos**.**

Ejemplos de algoritmos -> Perceptrón, LMS Error, Backpropagation

- **Aprendizaje estocástico:** realiza cambios aleatorios sobre los pesos. La predicción va mejorando o empeorando con cada uno de los cambios, quedándose evidentemente con los cambios que mejoren los resultados.

5.8.2 Aprendizaje no supervisado

Se caracteriza porque no requieren influencia externa para ajustar los pesos. Se busca encontrar las características, regularidades, correlaciones o categorías que se puedan establecer entre los datos que se presenten como entrada. La interpretación de sus datos depende de su estructura y del algoritmo de aprendizaje empleado.
La salida podía representar el grado de similitud entre los datos, un clustering o establecimiento de categorías [29].

- **Aprendizaje hebbiano:** permite medir la familiaridad o extraer las características de los datos de entrada.

- **Aprendizaje competitivo y comparativo:** permite realizar clasificaciones de los datos de entrada. La forma de actuación consiste en ir añadiendo elementos a una clase, si este nuevo elemento se determina que es de esta clase matiza los pesos, en caso contrario se puede crear una nueva clase con el elemento asociado a una serie de pesos propios.

5.8.3 Aprendizaje por refuerzo

Se considera un aprendizaje más lento que el aprendizaje por corrección de errores, en este caso no se dispone de un conjunto completo de los datos exactos de salida, sino que se le indica solamente si el dato es aceptable o no con esto el algoritmo ajusta los pesos basándose en un mecanismo de probabilidades.

6 FEATURE DETECTION AND MATCHING

En este capítulo, se hará una introducción a las técnicas de "Feature Detection and Matching" y los elementos en los que se basa. Este es un capítulo importante pues será la técnica principal en la que se basan los modelos utilizados para resolver el problema del emparejamiento de imágenes desde distintas perspectivas.

6.1 Introducción

En el campo de la visión artificial y el procesamiento de imágenes, la detección y coincidencia de imágenes (Feature Detection and Matching) es una técnica fundamental que juega un papel clave en numerosas aplicaciones, desde la recreación de estructuras hasta la navegación robótica o la fotografía computacional. Esta área de estudio se enfoca en identificar y comparar puntos de interés o características distintivas en imágenes para permitir el reconocimiento y seguimiento de objetos a través de diferentes vistas o escenas.

6.2 Aplicaciones

Algunas de las posibles utilidades en las que podemos aplicar algoritmos de "Feature Detection and Matching" son: la automatización del seguimiento de objetos, la coincidencia de puntos para calcular la disparidad, la segmentación basada en movimiento, la reconstrucción de objetos en 3D, utilidades para la navegación y pilotaje de robots o la recuperación e indexación de imágenes [34].

6.3 Características

En tareas de procesamiento de imágenes y visión artificial, es necesario representar las imágenes mediante características extraídas de las mismas. La imagen en bruto es ideal para que el ojo humano extraiga toda la información necesaria de ella. Sin embargo, esto no es así para los algoritmos informáticos. Las características pueden ser estructuras específicas en la imagen tales como puntos, bordes u objetos. También pueden ser el resultado de una operación de vecindad o detección de características aplicada a la imagen. Las imágenes se pueden representar con dos categorías [31]:

- **Características globales:** en esta representación, la imagen se representa por un vector de características multidimensional que describe la información de toda la imagen. Es decir, este método produce un sólo vector con valores que miden varios aspectos de la imagen, y posteriormente se pueden comparar dos imágenes mediante sus vectores de características. Estos vectores pueden representar propiedades tales como texturas, bordes o colores.

- **Características locales:** el objetivo de esta representación es representar de manera distintiva la imagen en función de algunas regiones destacadas, mientras que permanece invariante a los cambios de punto de vista e iluminación. Por lo tanto, la imagen se representa en base a sus estructuras locales dadas por un conjunto de descriptores de características. La mayoría de las características locales representan la textura del interior de las áreas de la imagen.

Dependiendo de la aplicación que se le vaya a dar se hará uso de uno u otro tipo de características. Por ejemplo, para grandes conjuntos de datos es más apropiado utilizar las características globales. También son útiles para aplicaciones donde se conoce una segmentación aproximada del objeto de interés. Por otro lado, para la coincidencia de imágenes y el reconocimiento de objetos es más útil fijarse en las características locales pues aportan información más detallada [32].

6.4 Características de los detectores de características

La finalidad de las características locales invariantes es aportar una representación que permita encontrar correspondencias entre estructuras locales de varias imágenes. En otras palabras, se busca obtener un conjunto

disperso de medidas locales que aporten la esencia subyacente de las imágenes de entrada y codifique sus estructuras de interés.

Para poder lograr esto, los detectores -y extractores- de características deben tener ciertas propiedades importantes para utilizar estas características [33]:

- **Robustez:** los algoritmos deberían ser capaces de detectar las mismas características independientemente del escalado, rotación, traslación, deformaciones fotométricas, artefactos de compresión y ruido.

- **Repetibilidad:** los algoritmos deberían ser capaces de detectar las mismas características de la misma escena u objeto repetidamente bajo una variedad de condiciones de visualización.

- **Precisión:** los algoritmos deberían ser capaces de localizar las características de la imagen, especialmente para tareas de correspondencia de imágenes, donde correspondencias precisas se requieren para estimar la geometría epipolar – la geometría que estudias las relaciones entre dos puntos de vista distintos.

- **Generalidad:** los algoritmos deberían ser capaces de detectar características que puedan ser usadas en diferentes aplicaciones.

- **Eficiencia:** los algoritmos deberían ser capaces de detectar características en nuevas imágenes para poder soportar aplicaciones en tiempo real.

- **Cantidad:** los algoritmos deberían ser capaces de detectar todos o la mayoría de las características de la imagen.

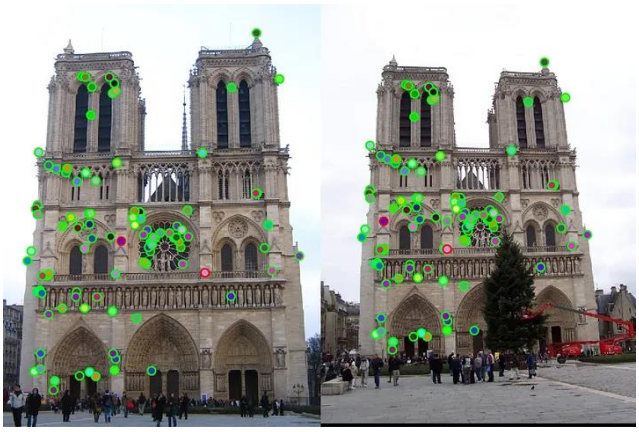


Figura 6-1. Representación de puntos de interés en una imagen

6.4.1 Algunos algoritmos utilizados

Estos son algunos de los algoritmos utilizados para la obtención y localización de características locales en una imagen:

- **SIFT**: Scale Invariant Feature Transform.
- **SURF**: Speeded Up Robust Feature.
- **BRISK:** Binary Robust Invariant Scalable Keypoints.
- **BRIEF:** Binary Robust Independent Elementary Features.
- **ORB**: Oriented FAST and Rotated BRIEF [35].

6.4 Coincidencia de características

La coincidencia de características, presente en muchas aplicaciones de la visión artificial, es la tarea de establecer correspondencias entre dos imágenes de la misma escena u objeto. Un enfoque para la coincidencia de imágenes

consiste en detectar un conjunto de puntos de interés, cada uno de los cuales estará asociado con descriptores de imágenes.
Una vez extraídas las características y sus descriptores de dos o más imágenes, el siguiente paso es establecer algunas coincidencias preliminares de características entre estas imágenes.

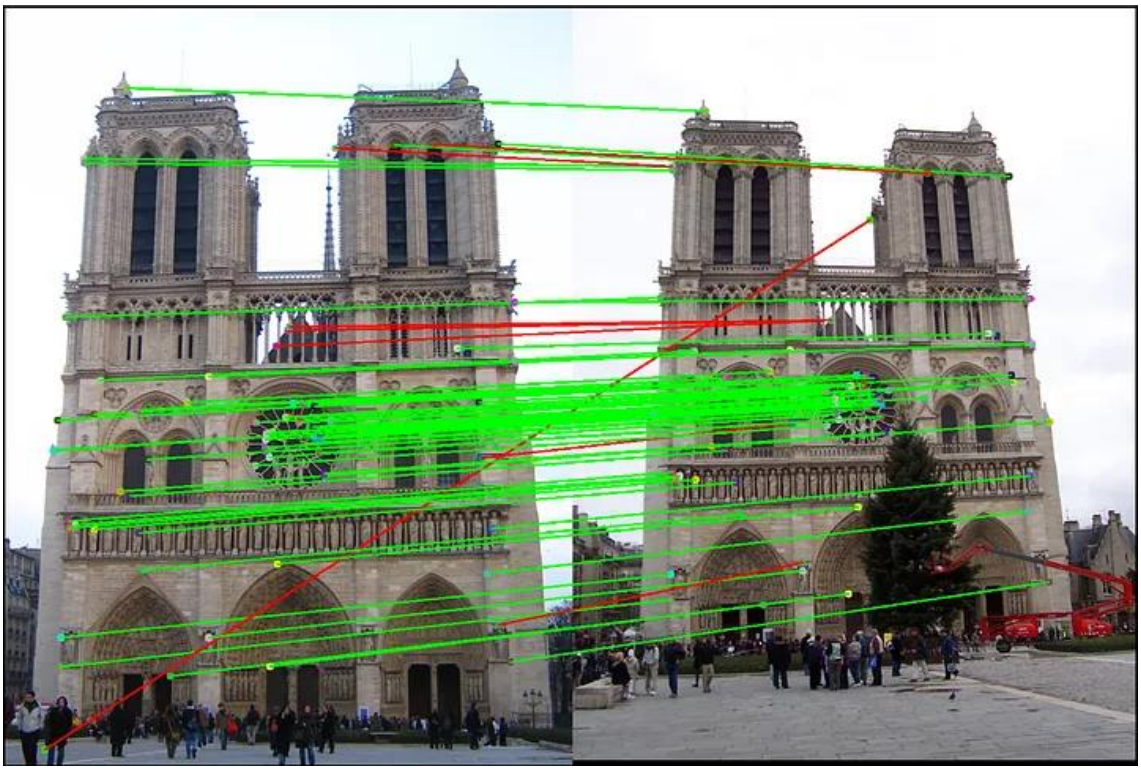


Figura 6-2. Representación de coincidencia de características de dos imágenes similares

El rendimiento de los métodos de coincidencia basados en puntos de interés depende tanto de las propiedades de los puntos como de la elección de los descriptores de imagen asociados. Por tanto, se deben utilizar detectores y descriptores apropiados para la aplicación.

Por ejemplo, si una imagen contiene células bacterianas, se debe usar el detector de manchas en lugar del detector de esquinas. Pero, si la imagen es una vista aérea de una ciudad, el adecuado será el detector de esquinas [34].

7 REDES NEURONALES GRÁFICAS (GRAPH NEURAL NETWORKS)

Para finalizar la parte teórica, se abordarán las Redes Neuronales Artificiales basadas en Grafos pues las arquitecturas que vamos a utilizar, basan parte de esta en una Red Neuronales Gráfica. Se hará una breve introducción al tema para luego explicar las distintas etapas por la que pasan los datos hasta obtener un resultado que se podrá dividir según cual sea nuestra finalidad.

7.1 Introducción

Las Redes Neuronales Gráficas son una clase de modelos de "deep learning" diseñados para trabajar con datos estructurados en forma de grafos.

Un grafo es una representación matemática que consiste en nodos (también llamados vértices) conectados por enlaces (o aristas). A diferencia de los datos tabulares o imágenes que son tratados por las redes neuronales más tradicionales, los grafos son inherentemente irregulares y no tienen una estructura fija haciendo que desarrollar modelos efectivos sea complicado [34][35].

La idea principal de las GNNs (Graph Neural Network) es aprovechar la información estructural del grafo y permitir que los nodos aprendan de sus vecinos. Cada nodo del grafo tiene un vector de características que representa ciertas propiedades del mismo. Las GNNs tienes una serie de capas neuronales que propagan y actualizan la información de los nodos vecinos para mejorar cada uno de los nodos de los grafos. Este proceso se llama "neighborhood aggregation". Esto hace que puedan aprender patrones y características complejas basadas en la topología del grafo y las relaciones entre los nodos [39].

7.2 Etapas de una GNN

Para que la GNN pueda obtener un resultado deberá pasar por una serie de procesos antes de realizar una predicción. Estos son [36][39]:

7.2.1 Representación del grafo

El primer paso es representar el grafo en una forma que pueda ser procesada por una red neuronal. Para ello, cada nodo del grafo se asigna a un vector de características inicial. Estas características pueden ser atributos asociados a cada nodo o simplemente identificadores únicos. Además, es posible representar los enlaces entre nodos mediante matrices de adyacencia o listas de adyacencia.

Dependiendo del dato que queramos representar seguiremos un proceso u otro. Nos centraremos en la representación de las imágenes como grafos pues es el dato que utilizaremos en nuestro problema, pero podríamos representar datos tales como textos, moléculas, redes sociales, redes de citas, etc.

7.2.1.1 Imágenes como grafos

La representación de imágenes mediante grafos es una técnica que se ha utilizado en diversas aplicaciones de procesamiento de imágenes y visión artificial.

Para lograr esto, se construye un grafo en el que los nodos representan las partes significativas de la imagen tales como regiones o características específicas extraídas de la imagen. Los enlaces entre los nodos se establecen para capturar las relaciones entre estas partes [36] [40].

Para representar imágenes con grafos tenemos tres representaciones:

- **Grafo de píxeles:** cada píxel de la imagen es un nodo en el grafo y los enlaces entre los píxeles se

establecen de acuerdo a alguna medida de proximidad o similitud entre ellos, como por ejemplo la distancia euclidiana o la intensidad de color. Esta representación se utiliza en tareas de segmentación de imágenes.

- **Grafo de superpíxeles:** a diferencia de la anterior representación, aquí las imágenes se procesan utilizando algoritmos de segmentación que agrupan píxeles similares en regiones más grandes llamadas superpíxeles. Estos superpíxeles se convierten en nodos del grafo y los enlaces entre ellos se basan en la proximidad espacial o características compartidas.

- **Grafo de características:** en lugar de utilizar píxeles como nodos, se extraen características relevantes de la imagen mediante técnicas de extracción de características tales como descriptores de texturas o descriptores de características visuales. Cada característica obtenida se convierte en un nodo del grafo y los enlaces se definen por similitud de características [36][40].

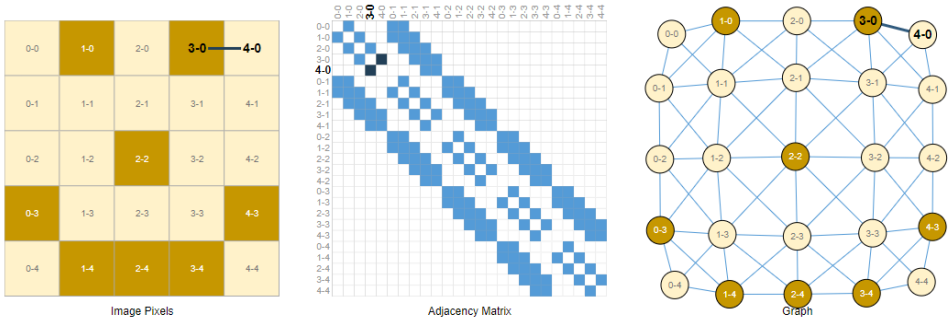


Figura 7-1. Formas de representar una imagen como grafo

7.2.2 Agregación de vecindario (neighborhood aggregation)

El proceso principal de una GNN es la propagación de información a través del grafo, permitiendo que los nodos aprendan de sus vecinos y actualicen sus características. En cada capa de propagación, cada nodo del grafo recopila información de sus vecinos y la combina para actualizar su propia representación. El proceso de agregación generalmente se realiza en dos pasos:

a. Paso de mensaje (message passing): Cada nodo envía un mensaje a sus vecinos con información sobre su estado actual (características). El mensaje se obtiene al aplicar una función de transformación a las características del nodo emisor.

b. Agregación (aggregation): Una vez que todos los nodos han enviado sus mensajes, cada nodo receptor recopila los mensajes entrantes de sus vecinos y los combina mediante una función de agregación, como la suma, el promedio o alguna función más compleja. Esta agregación captura la información global de los vecinos y la integra en la actualización del nodo.

7.2.3 Actualización de nodos

Después de que todos los nodos hayan agregado la información de sus vecinos, cada nodo actualiza sus características utilizando la información agregada y sus propias características actuales. Este paso se realiza mediante una función de actualización que toma en cuenta tanto el mensaje agregado como las características del nodo, lo que le permite capturar tanto la información local como la información de la estructura global del grafo.

7.2.4 Repetición en varias capas de propagación

El proceso de propagación (paso de mensajes y actualización de nodos) se repite a través de varias capas de la GNN. Cada capa refina gradualmente las representaciones de los nodos, permitiendo que la información se propague a través del grafo a diferentes niveles de profundidad. Esto permite a la GNN capturar patrones y características complejas basadas en la estructura del grafo.

7.2.5 Obtención de un resultado

Finalmente, una vez que la GNN ha propagado la información a través de varias capas, se puede usar una

estructura específica para realizar una tarea en particular. Por ejemplo, para clasificación de nodos, se puede agregar una capa de clasificación para predecir las etiquetas de los nodos; para predicción de enlaces, se puede usar una capa para predecir la existencia o el valor de enlaces entre nodos; para tareas de regresión, se puede utilizar una capa de regresión para predecir valores numéricos.

7.3 Tipos de tareas de predicción en gráficos

Hay tres tipos generales de tareas de predicción en gráficos: nivel de gráfico, nivel de nodo y nivel de borde [36].

7.3.1 Tareas de nivel de gráfico

En este tipo de tarea, nuestro objetivo es predecir la propiedad de un gráfico completo. Por ejemplo, para una molécula representada como un gráfico, podríamos querer predecir si esta se unirá a un receptor implicado en una enfermedad.

Esto es análogo a los problemas de clasificación de imágenes, donde queremos asociar una etiqueta a una imagen completa.

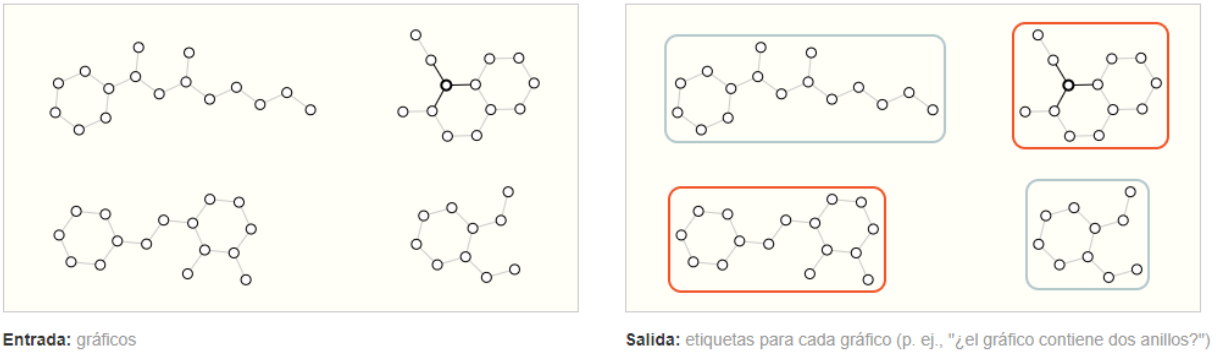


Figura 7-2. Entrada y salida de un gráfico intentando detectar qué patrón tiene dos anillos

7.3.2 Tarea a nivel de nodo

Las tareas a nivel de nodo se ocupan de predecir la identidad o el rol de cada nodo dentro de un gráfico. Un ejemplo clásico de un problema de predicción a nivel de nodo es el club de kárate de Zach.

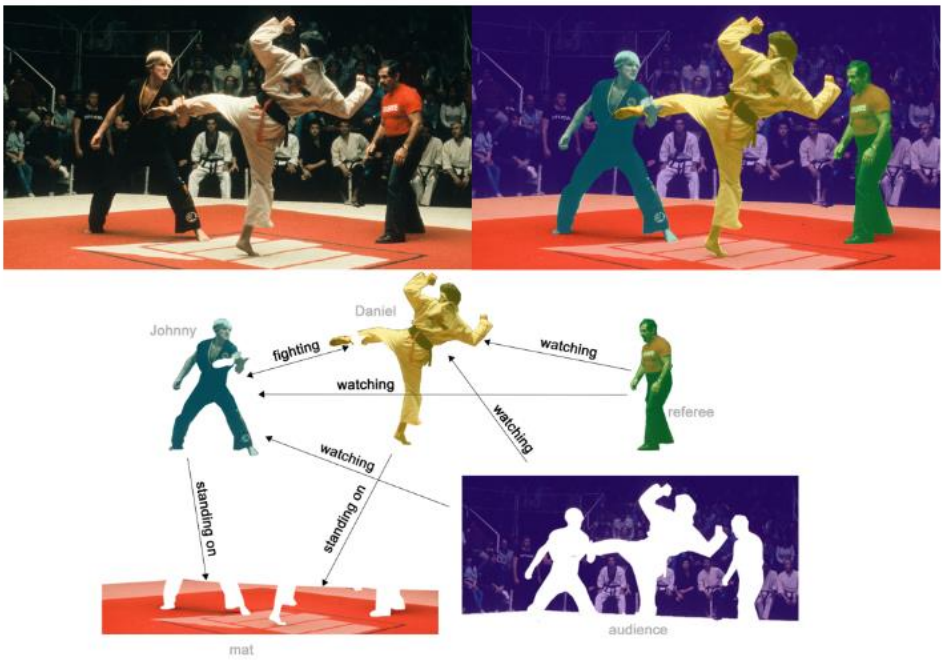
El conjunto de datos es un único gráfico de red social compuesto por personas que han jurado lealtad a uno de los dos clubes de kárate después de una ruptura política. Según cuenta la historia, una disputa entre el Sr. Hi (Instructor) y John H (Administrador) produce un revuelo en el club. Los nodos representan practicantes de karate individuales y los bordes representan interacciones entre estos miembros fuera del karate. El problema de predicción consiste en clasificar si un miembro determinado se vuelve leal al Sr. Hi o a John H, después de la disputa. En este caso, la distancia entre un nodo y el Instructor o el Administrador está altamente correlacionada con esta etiqueta.

Los problemas de predicción a nivel de nodo son análogos a la segmentación de imágenes, en las que intentamos etiquetar la función de cada píxel en una imagen.

7.3.3 Tarea de nivel de borde

Un ejemplo de uso es la comprensión de la escena de la imagen. Más allá de identificar objetos en una imagen, los modelos de aprendizaje se pueden utilizar para predecir la relación entre ellos.

Dados los nodos que representan los objetos en una imagen, deseamos predecir cuáles de estos nodos comparten un borde o cuál es el valor de ese borde. Si buscamos descubrir conexiones entre entidades, podríamos considerar el gráfico completamente conectado y, en función de su valor, llegar a un gráfico disperso.



En (b), arriba, la imagen original (a) ha sido segmentada en cinco entidades: cada uno de los peleadores, el árbitro, la audiencia y el tapiz. (C) muestra las relaciones entre estas entidades.

Figura 7-3. Representación de la extracción de información de una imagen a nivel de borde

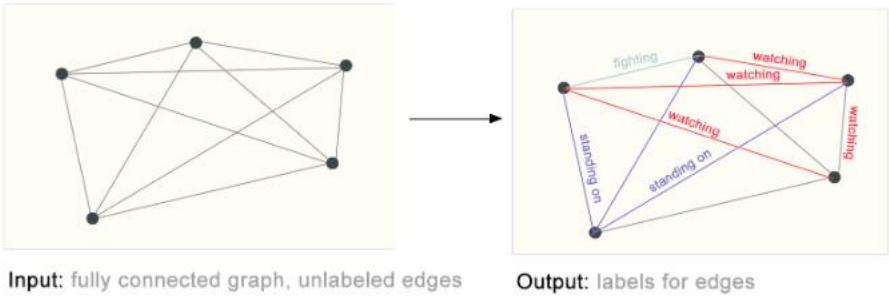


Figura 7-4. Representación de las conexiones de la figura 7-3 a nivel de nodos

8 PROBLEMA A RESOLVER

A la hora de solucionar el problema propuesto (emparejamiento de imágenes) lo más común es utilizar modelos de Redes Neuronales pre-entrenados sobre los que se realizan ajustes menores o se tratan los datos de modo que sean mejor interpretados por este debido a que se necesitan una cantidad inmensa de datos para que una red neuronal que trabaje en este campo obtenga resultados positivos. Así, en el capítulo se hará una introducción a nuestro problema y cómo trabajaremos.

8.1 Introducción al problema

El problema se ha sacado de la famosa página web Kaggle. Es una plataforma web que reúne a la comunidad del "Data Science" recibiendo más de 150 mil publicaciones al mes, que brindan todas las herramientas y recursos para progresar en data science.

Kaggle permite acceder de manera gratuita a un banco de trabajo para construir, entrenar y evaluar modelos además de acceso a GPUs. También ofrece una gran cantidad de datos y códigos publicados por la comunidad, e incluso hay tutoriales para iniciarse y avanzar en el mundo de las IA.

Kaggle ofrece una serie de competiciones o "challenges" con premios que son publicadas por empresas y van desde competencias para principiantes hasta tareas de investigación. En este trabajo nos vamos a centrar, precisamente, en uno de estos desafíos.

Concretamente, el problema es propuesto por Google Research y se corresponde a una competición con el nombre "Image Matching Challenge 2022" en la que participaron 642 equipos. El objetivo principal es crear un modelo capaz de identificar los puntos físicos coincidentes entre dos imágenes similares tomadas desde distinto ángulo. Se proporcionan dos sets de datos, uno para el entrenamiento del modelo y otro para el testeo del mismo. Los resultados se evalúan con una cierta métrica y el tiempo de ejecución del archivo debe ser inferior a 9 horas. Se pueden usar modelos pre-entrenados y datos externos públicos.

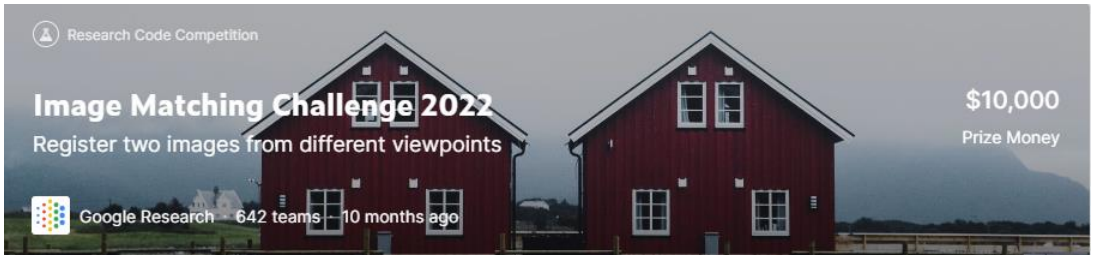


Figura 8-1. Competición "Image Matching Challenge 2022"

8.2 Entorno de trabajo

Para desarrollar se tienen dos opciones. O bien utilizar un ordenador personal o utilizar entornos de programación en línea como Kaggle o Google Colab. Estas son las características de cada uno de ellos:

	Ordenador sobremesa	Kaggle	Google Colab
RAM	16 GB	13 GB	12 GB
CPU	AMD Ryzen 5 5600X 6-Core Processor 3.70 GHz	2.30Ghz x4	2.30GHz x2
GPU	NVIDIA GeForce RTX 3060	NVIDIA Tesla P100	NVIDIA Tesla K80
Horas máximas por ejecución	Sin límite	12	12
Horas de uso por semana	Sin límite	De 30 a 40	84 (máximo 12 cada 24 horas)

Tabla 8-1. Comparación entornos de trabajo disponibles

Además de estas características, debemos tener en cuenta otros factores. Los entornos en línea pueden ejecutar el código tanto con los recursos del ordenador desde el que se accede como con los recursos propios que la plataforma reserva al usuario.

Aunque los entornos en línea tengan un límite de uso en cada sesión y a la semana, pueden ejecutarse aún con el navegador cerrado por lo que no es necesario que el ordenador esté encendido durante la ejecución del código, lo que es una gran ventaja en este problema pues el entrenamiento puede prolongarse durante varias horas. Adicionalmente, si se usan los entornos en línea, no hay que configurar nada en el ordenador ni descargar las librerías pues bastaría con importarlas sumado a que todas las versiones de un código quedan guardadas en la nube y se pueden consultar en cualquier momento sin riesgo de pérdida.

Dados los argumentos anteriores, nos decantaremos por los entornos en línea. Entre Google Colab y Kaggle se ha decidido utilizar la plataforma de Kaggle para el desarrollo del código debido a que son entornos muy parecidos, pero Kaggle cuenta con la ventaja de que todo el "dataset" que se va a utilizar se encuentra en esta plataforma y es muy cómodo utilizarlo ya que para utilizarlo en Google Colab habría que descargarlo y añadir código para poder incorporarlo [48].

Figura 8-2. Logo de la plataforma Kaggle

Durante la ejecución del código se utilizará la GPU que ofrece Kaggle en lugar de la CPU. Las CPUs consumen menos energía y son mejores para cálculos secuenciales pero, sin embargo, las GPU funcionan mejor en operaciones que necesitan paralelismo y están pensadas para problemas de computación gráfica tales como el mapeo de texturas, iluminación, proyecciones, ... Así, la GPU es la mejor opción para trabajar con el manejo de imágenes y CNN pues reducirá el tiempo de procesado y mejorará la eficiencia del modelo.

8.3 Lenguaje de programación

A la hora de programas problemas de esta índole, los lenguajes más utilizados son R y Python. El código se elaborará en lenguaje Python debido a los conocimientos previos que se tienen y la gran variedad de bibliotecas que incorpora junto con las bibliotecas de cada uno de los modelos de redes pre-entrenadas que vamos a utilizar.

8.4 Imágenes de testeo

Al utilizar redes pre-entrenadas, no necesitamos entrenar a nuestros modelos por lo que utilizaremos únicamente las imágenes de testeo para, posteriormente, enviar los resultados a Kaggle. Estas son las parejas de imágenes que queremos emparejar.

Figura 8-3. Primera pareja de imágenes de testeo

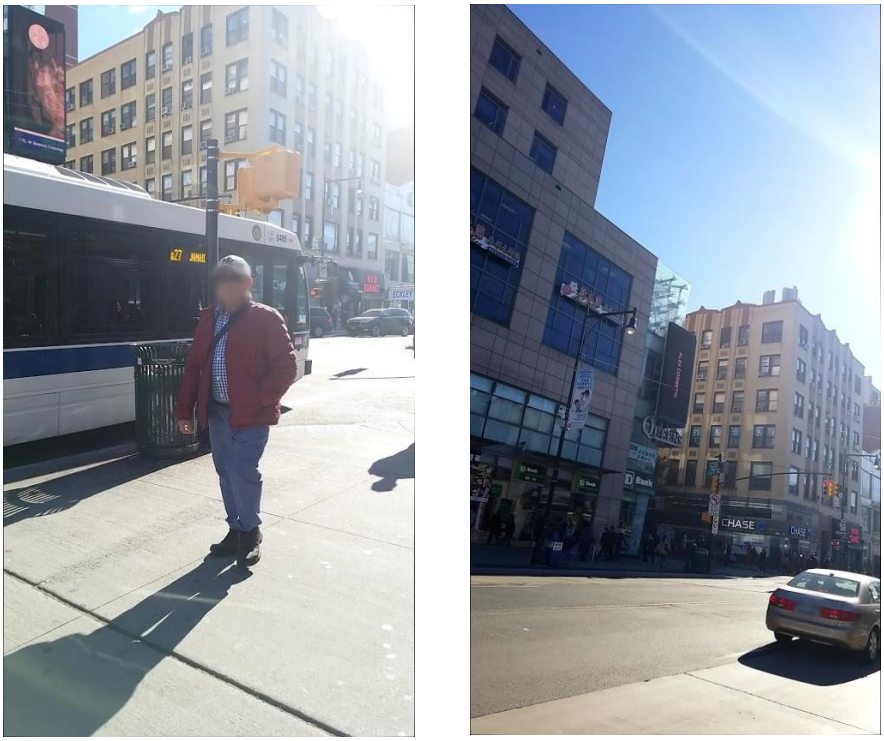


Figura 8-4. Segunda pareja de imágenes de testeo

Figura 8-5. Tercera pareja de imágenes de testeo

En cuanto a las imágenes de testeo, las dos primeras parejas parecen tener un grado de disparidad bajo con respecto a la tercera pareja.

Con la primera pareja se produce un movimiento de traslación pequeño y un cambio de luz lo que hace que los colores de las imágenes sean distintos pero las fachadas de los edificios están en posiciones similares.

La segunda pareja produce un movimiento de rotación, se aleja un poco el foco y se ha limpiado de obstáculos frente a los edificios, aunque las posiciones de los edificios son similares.

La tercera pareja de imágenes es la más complicada de emparejar pues cambia mucho el punto de vista del edificio rojo central. Tampoco tiene la referencia de los coches, el cartel o el otro edificio que sí aparece en la primera imagen.

8.5 Dataset de entrenamiento

Si quisiéramos entrenar un modelo que hemos creado, Kaggle también ofrece un amplio dataset de imágenes de entrenamiento junto con una serie de archivos .csv con información necesaria para que nuestro modelo sea entrenado correctamente. La estructura de este conjunto de datos e imágenes es el siguiente:

- train/*/calibration.csv
    - image_id: El nombre del archivo de imagen
    - camera_intrinsics: Una matriz 3x3 de calibración para una imagen, ajustado a un vector.
    - rotation_matrix: Una matriz 3x3 de rotación para una imagen, ajustado a un vector.
    - translation_vector: El vector de traslación.
- train/*pair_covisibility.csv
    - pair: Una cadena que identifica un par de imágenes separadas por un guion.
    - covisibility: La estimación del solapamiento entre las dos imágenes. Los números más altos indican un mayor solapamiento.
    - fundamental_matrix: la columna objetivo derivada de los archivos de calibración.
- train/scaling_factors.csv: Las poses de cada escena se reconstruyeron mediante "Structure-from-Motion", y sólo son precisas hasta un factor de escala. Este archivo contiene un escalar para cada escena que puede utilizarse para convertirlas a metros.
- train/*/images/: Un lote de imágenes tomadas cerca del mismo lugar.

```
Datos del dataset de entrenamiento

Número de monumentos diferentes:  16
Número de imágenes totales del dataset:  5678
Número de imágenes del monumento brandenburg_gate : 350
Número de imágenes del monumento british_museum : 176
Número de imágenes del monumento buckingham_palace : 446
Número de imágenes del monumento colosseum_exterior : 499
Número de imágenes del monumento grand_place_brussels : 236
Número de imágenes del monumento lincoln_memorial_statue : 214
Número de imágenes del monumento notre_dame_front_facade : 911
Número de imágenes del monumento pantheon_exterior : 321
Número de imágenes del monumento piazza_san_marco : 68
Número de imágenes del monumento sacre_coeur : 281
Número de imágenes del monumento sagrada_familia : 90
Número de imágenes del monumento st_pauls_cathedral : 142
Número de imágenes del monumento st_peters_square : 622
Número de imágenes del monumento taj_mahal : 399
Número de imágenes del monumento temple_nara_japan : 217
Número de imágenes del monumento trevi_fountain : 706
----------------------------------------
Número de combinaciones de imágenes totales del dataset:  1416814
Número de combinaciones de imágenes del monumento brandenburg_gate : 61075
Número de combinaciones de imágenes del monumento british_museum : 15400
Número de combinaciones de imágenes del monumento buckingham_palace : 99235
Número de combinaciones de imágenes del monumento colosseum_exterior : 124251
Número de combinaciones de imágenes del monumento grand_place_brussels : 27730
Número de combinaciones de imágenes del monumento lincoln_memorial_statue : 22791
Número de combinaciones de imágenes del monumento notre_dame_front_facade : 414505
Número de combinaciones de imágenes del monumento pantheon_exterior : 51360
Número de combinaciones de imágenes del monumento piazza_san_marco : 2278
Número de combinaciones de imágenes del monumento sacre_coeur : 39340
Número de combinaciones de imágenes del monumento sagrada_familia : 4005
Número de combinaciones de imágenes del monumento st_pauls_cathedral : 10011
Número de combinaciones de imágenes del monumento st_peters_square : 193131
Número de combinaciones de imágenes del monumento taj_mahal : 79401
Número de combinaciones de imágenes del monumento temple_nara_japan : 23436
Número de combinaciones de imágenes del monumento trevi_fountain : 248865
```

Figura 8-6. Datos relativos al número de imágenes del dataset de entrenamiento

9 REDES NEURONALES GRÁFICAS PRE-ENTRENADAS

En el problema de correspondencia de imágenes, se podrían aprovechar algunas regularidades del mundo: el mundo 3D es en gran medida liso y a veces plano, todas las correspondencias para un par de imágenes determinado derivan de una única transformación epipolar si la escena es estática, y algunas poses son más probables que otras. Además, los puntos clave 2D suelen ser proyecciones de puntos 3D destacados, como esquinas o manchas físicas:

i) un punto clave puede tener, como máximo, una única correspondencia en la otra imagen; y ii) algunos puntos clave no coincidirán debido a la opacidad y al fallo del detector.

Un modelo eficaz de correspondencia de rasgos debería tener como objetivo encontrar todas las correspondencias entre las reproyecciones de los mismos puntos 3D e identificar los puntos clave que tienen la misma correspondencia e identificar los puntos clave que no coinciden.

El problema a resolver se basa en la geometría proyectiva y la geometría epipolar.

La geometría proyectiva estudia las propiedades de incidencia de las figuras geométricas, abstrayéndose totalmente del concepto de medida. En otras palabras, la geometría proyectiva estudia las relaciones entre las figuras tridimensionales del espacio y sus proyecciones sobre el plano [47]. Apareció como solución al problema del artista para pintar el mundo tridimensional en sus lienzos bidimensionales [48].

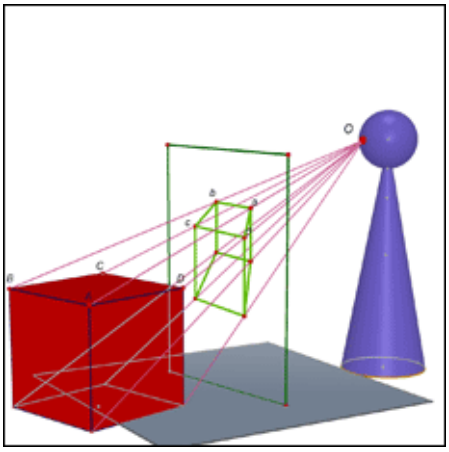


Figura 9-1. Imagen representativa de la geometría proyectiva

Por otro lado, la geometría epipolar se refiere a la geometría proyectiva entre dos puntos de vista distintos. No depende de la estructura de la escena y está ligada a los parámetros de la cámara y las posiciones de estas respecto al objeto [49].

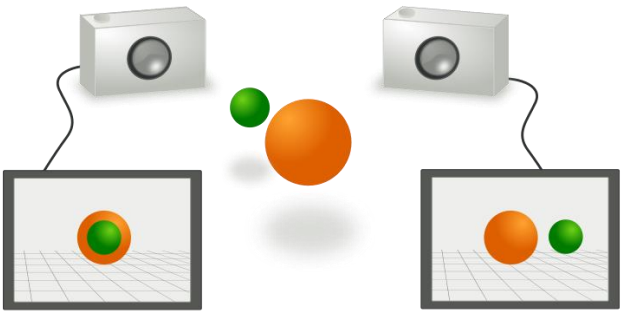


Figura 9-2. Imagen representativa de la geometría epipolar

9.1 Kornia LoFTR

9.1.1 Introducción

LoFTR es un proyecto de investigación enfocado en el campo del aprendizaje profundo para emparejar características e imágenes creado por Paul-Edouard Sarlin, Anurag Arnab, Mathieu Salzmann y Josef Sivic. Este proyecto fue presentado en el CVPR de 2021 y fue una de los mejores proyectos en el campo de la visión computacional de 2021 [41].

LoFTR trata de abordar las limitaciones de los métodos tradicionales de coincidencia de características, tales como la necesidad de un detector de características separado, que puede ser computacionalmente costoso y sensible a los cambios en la apariencia y el punto de vista de las imágenes. LoFTR aprende directamente a emparejar características locales utilizando arquitecturas basadas en transformadores sin la necesidad de un detector separado.

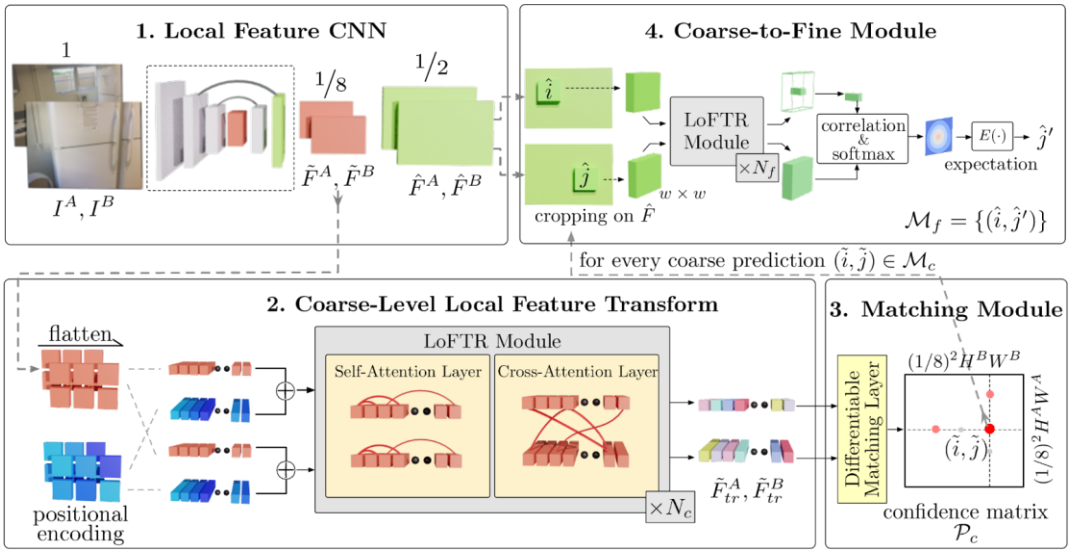


Figura 9.1-1: Arquitectura del modelo Kornia LoFTR

LoFTR tiene cuatro componentes principales: una red de detección de características locales CNN, emparejador basado en transformadores LoFTR, módulo emparejador y el módulo de refinamiento.

9.1.2 Arquitectura

9.1.2.1 Detector de características locales CNN

La tarea principal de esta capa es detectar y describir características locales (puntos clave) en las imágenes. Estos puntos clave son ubicaciones distintivas en la imagen que se consideran importantes para el emparejamiento. Para cada punto clave detectado, se genera un descriptor, que es una representación compacta de la información local de la imagen que rodea ese punto clave. Estos descriptores capturan la apariencia y las propiedades geométricas de los puntos clave y son fundamentales para el proceso de emparejamiento posterior.

9.1.2.2 Mecanismo de emparejamiento basado en transformadores

Los transformadores son conocidos por su efectividad en capturar dependencias de largo alcance e información contextual, lo que los hace adecuados para tareas como el emparejamiento de características entre imágenes. Una vez que la CNN extrae los puntos clave y descriptores de dos imágenes, entra en acción el mecanismo de emparejamiento basado en transformadores. Forma una representación de grafo donde los puntos clave son tratados como nodos, y los posibles emparejamientos entre puntos clave de las dos imágenes se representan como aristas en el grafo.

El mecanismo de emparejamiento basado en transformadores luego procesa este grafo, permitiendo que los puntos clave se comuniquen e intercambien información con sus puntos clave vecinos. Esta comunicación permite que el modelo aprenda la similitud entre puntos clave y sus posibles emparejamientos, considerando información contextual tanto local como global.

9.1.2.3 Módulo de emparejamiento

Después del emparejamiento basado en transformadores, el modelo aplica el sondeo lineal para aprender incrustaciones globales de imágenes a partir de los emparejamientos de características locales. El sondeo lineal es una transformación lineal que mapea los emparejamientos de características locales en una representación global de la imagen más compacta y significativa. Este proceso ayuda a capturar las características generales de la imagen, permitiendo que el modelo comprenda el contenido de la imagen a un nivel más alto.

Las incrustaciones globales obtenidas mediante el sondeo lineal permiten que LoFTR realice un emparejamiento de características robusto y preciso entre diferentes imágenes, incluso en escenarios desafiantes con cambios en el punto de vista, oclusiones y variaciones de escala.

Tras este proceso se obtiene un matriz de confianza *Pc.* Los puntos se seleccionan de acuerdo con el umbral de confianza y los criterios del vecino más cercano.

9.1.2.4 Módulo de refinamiento

Por último, este módulo se encarga de refinar la matriz de confianza obtenida a la resolución de la imagen original. Para cada predicción aproximada, se recorta una ventana local del mapa de características. Las coincidencias aproximadas se refinarán dentro de esta ventana local a un nivel de subpíxel como la predicción de coincidencia final.

9.2 SuperGlue

9.2.1 Introducción

SuperGlue es un proyecto de investigación del CVPR 2020 (*2020 Conference on Computer Vision and Pattern Recognition*) realizado en Magic Leap. La red SuperGlue es una red neuronal gráfica ("Graph Neural Network") combinada con una capa de coincidencia óptima que está entrenada para realizar la coincidencia en dos conjuntos de características de imagen dispersa [42].

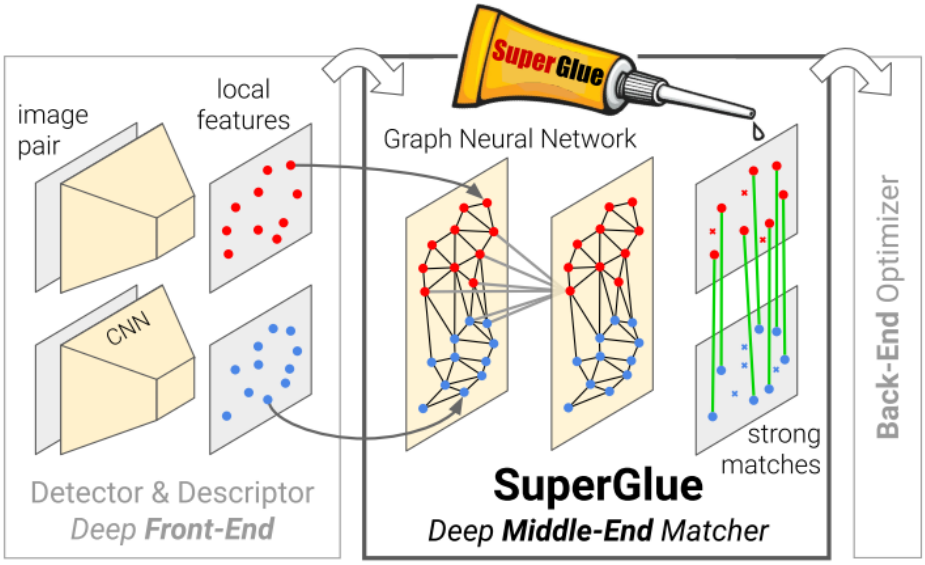


Figura 9.2-1: Imagen representativa fase de utilización SuperGlue

SuperGlue funciona como un "Middle-End Matcher", realizando agregación, coincidencia y filtrado de contexto en una única arquitectura de extremo a extremo. Esta red combina dos conjuntos de características locales al

encontrar correspondencias de forma conjunta y rechazar puntos que no coinciden. Las asignaciones se estiman resolviendo un problema de transporte óptimo diferenciable, cuyos costos son predichos por una red neuronal gráfica. El método propuesto realiza la comparación en tiempo real en una GPU moderna y puede integrarse fácilmente en los sistemas SfM (Structure-from-Motion) o SLAM (Simultaneous Localization and Mapping) modernos.

9.2.2 Arquitectura

Formulamos SuperGlue (véase la figura) como la resolución de un problema de optimización, cuyo coste predice una red neuronal profunda. Esto evita tener que recurrir a la experiencia y a la heurística -aprendemos las predicciones relevantes directamente de los datos.

Formulación: Consideremos dos imágenes A y B, cada una con un conjunto de posiciones de puntos clave *p* y descriptores visuales asociados *d*, que denominaremos conjuntamente *(p, d)* características locales. Las posiciones constan de las coordenadas *x* e *y* de la imagen, así como de un grado de confianza de detección *c*, *pi := (x, y, c)i*.

Descriptores visuales *di ∈ R^D* pueden ser los extraídos por una CNN como SuperPoint o descriptores tradicionales como SIFT.

Las imágenes A y B tienen *M* y *N* características locales, indexadas por *A := {1, ..., M}* y *B := {1, ..., N}*, respectivamente.

Asignación parcial: Las restricciones i) y ii) significan que las correspondencias se derivan de una asignación parcial entre los dos conjuntos de puntos clave. Para la integración en tareas posteriores y una mejor interpretabilidad, cada correspondencia posible debe tener un valor de confianza. En consecuencia, definimos una matriz de asignación parcial suave *P ∈ [0, 1]M×N* como: *P1N ≤ 1M y P>1M ≤ 1N*. Nuestro objetivo es diseñar una red neuronal que prediga la asignación P a partir de dos conjuntos de características locales.

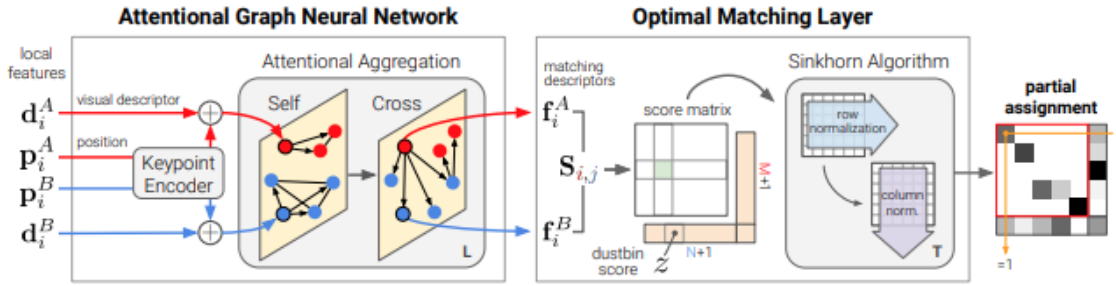


Figura 9.2-2: Arquitectura del modelo SuperGlue

SuperGlue consta de dos componentes principales: la red neuronal de grafos atencionales (Sección 9.2.3.1) y la capa de correspondencia óptima (Sección 9.2.3.2).

El primer componente utiliza un codificador de puntos clave para mapear posiciones *p* de los puntos clave y sus descriptores visuales *d* en un único vector y, a continuación, utiliza capas alternas de atención propia y cruzada (repetida L veces) para crear representaciones más potentes *f*. La capa de correspondencia óptima crea una matriz de puntuación M por N la aumenta con cubos de basura (dustbins) y, a continuación, encuentra la asignación parcial óptima mediante el algoritmo de Sinkhorn (para T iteraciones).

9.2.2.1 Attentional Graph Neural Network

La Red Neuronal Gráfica con Atención es el núcleo del mecanismo de coincidencia de SuperGlue. Es un tipo de red neuronal gráfica que opera en los puntos clave y sus descriptores extraídos de las imágenes de entrada. En el grafo, los puntos clave de una imagen se representan como nodos y las coincidencias potenciales entre los puntos clave de las dos imágenes se representan como aristas. La GNN está diseñada para realizar un proceso de propagación de mensajes entre los nodos (puntos clave) en el grafo. Durante este proceso, la GNN recopila información de los puntos clave vecinos, lo que permite que el modelo aprenda a establecer correspondencias entre puntos clave. El mecanismo de atención en la GNN le permite enfocarse en puntos clave relevantes y sus

coincidencias potenciales, lo cual es crucial para una coincidencia precisa y discriminativa de características.

9.2.2.2 Optimal matching layer

La Capa de Coincidencia Óptima es otro componente importante de la arquitectura de SuperGlue. Esta capa es responsable de calcular una puntuación de similitud para cada coincidencia potencial de puntos clave, basándose en la información agregada obtenida de la GNN. La puntuación de similitud representa qué tan bien se corresponden los puntos clave de las dos imágenes entre sí. La innovación clave de la Capa de Coincidencia Óptima es su capacidad para manejar coincidencias uno a uno y de muchos a uno. En los métodos tradicionales de coincidencia de características, los puntos clave suelen coincidir uno a uno, lo que significa que cada punto clave en una imagen se corresponde con exactamente un punto clave en la otra imagen. Sin embargo, en escenarios del mundo real, los puntos clave pueden no tener una correspondencia uno a uno, y un punto clave en una imagen puede coincidir con varios puntos clave en la otra imagen. La Capa de Coincidencia Óptima puede manejar estas coincidencias de muchos a uno, lo que hace que SuperGlue sea más robusto y efectivo en situaciones desafiantes.

En conjunto, la Red Neuronal Gráfica con Atención y la Capa de Coincidencia Óptima permiten que SuperGlue aprenda descriptores de características discriminativas y coincida de manera precisa los puntos clave entre imágenes. El entrenamiento de extremo a extremo de toda la arquitectura de SuperGlue garantiza que tanto la detección de características como la coincidencia de características se optimicen de manera conjunta, lo que da como resultado un sistema de coincidencia de características potente y eficiente para diversas aplicaciones de visión por computadora.

9.3 ASLFeat

9.3.1 Introducción

ASLFeat es un modelo de visión por computadora enfocado en el aprendizaje de características locales de imágenes que son robustas ante cambios de escala y oclusiones. Este proyecto fue presentado en el artículo de investigación "*ASLFeat: Learning Local Features of Accurate Shape and Localization*" por Zixin Luo, Lei Zhou, Xuyang Bai, Hongkai Chen, Jiahui Zhang, Yao Yao, Shiwei Li, Tian Fang and Long Quan en el CVPR de 2020 [43].

El objetivo principal de ASLFeat es abordar las limitaciones de los métodos tradicionales de aprendizaje de características locales, especialmente en escenarios donde los objetos en las imágenes experimentan cambios significativos de escala o están parcialmente ocultos. Estos desafíos pueden afectar negativamente el rendimiento de métodos de extracción de características locales tradicionales como SIFT o ORB.

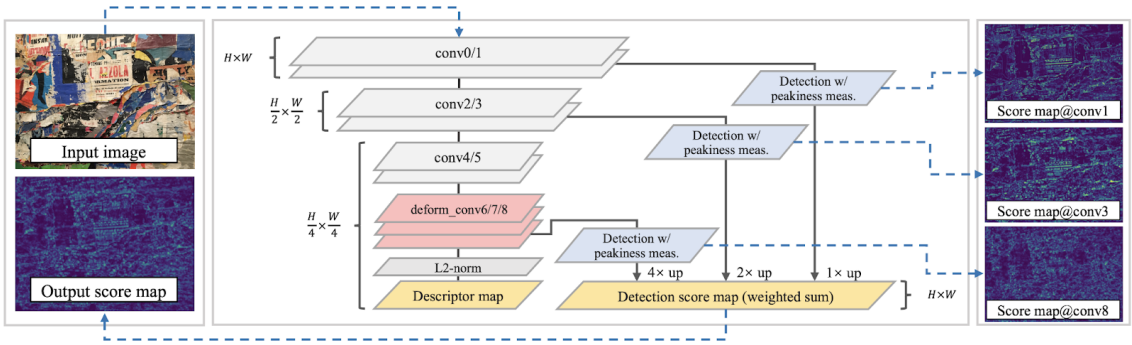


Figura 9.3-1: Arquitectura de ASLFeat

Arquitectura de la red, con la capa de red convolucional deformable (DCN), detección multinivel (MulDet) y medición de picos para la puntuación de puntos clave.

9.3.2 Arquitectura

9.3.2.1 Capa de Red Convolucional Deformable (DCN)

La capa de red convolucional deformable (DCN) es una parte fundamental de la arquitectura de ASLFeat. Esta capa permite que la red aprenda desplazamientos locales en las operaciones de convolución estándar. Esto es particularmente útil para capturar características locales que pueden estar deformadas o desplazadas en la imagen. La inclusión de la capa DCN es relevante para mejorar la robustez de ASLFeat ante cambios de escala y distorsiones en los objetos presentes en las imágenes.

9.3.2.2 Detección Multinivel (MulDet)

La detección multinivel (MulDet) es otro componente importante de ASLFeat. Esta detección se realiza en múltiples niveles de la pirámide de características de la red. La pirámide de características es una representación a diferentes escalas de la imagen, que permite detectar características locales en diversas dimensiones. Al realizar la detección en múltiples niveles de la pirámide, ASLFeat puede capturar características en diferentes tamaños y escalas, lo que mejora su capacidad para abordar la variabilidad de escala en los objetos presentes en las imágenes.

9.3.2.3 Medición de Picos para la Puntuación de Puntos Clave

La medición de picos es una parte crucial del proceso para obtener la puntuación de los puntos clave detectados por ASLFeat. Los puntos clave son ubicaciones distintivas en la imagen que son relevantes para el emparejamiento y otras tareas de visión por computadora. La medición de picos permite evaluar y asignar una puntuación a los puntos clave detectados, lo que ayuda a seleccionar los puntos clave más relevantes y discriminativos para la tarea en cuestión.

Estos componentes trabajan en conjunto para permitir que ASLFeat aprenda características locales precisas y robustas, especialmente en escenarios con cambios de escala y oclusiones. La combinación de la capa de red convolucional deformable, la detección multinivel y la medición de picos contribuye al éxito de ASLFeat en diversas aplicaciones de visión por computadora que requieren una extracción de características locales confiable y precisa.

9.3.3 Entrenamiento

ASLFeat se entrena de manera auto-supervisada, lo que significa que no requiere anotaciones explícitas de puntos clave o descriptores durante el entrenamiento. En su lugar, aprovecha la relación geométrica entre ejemplos positivos y negativos para aprender características locales discriminatorias.

9.4 DKM

9.4.1 Introducción

DKM es un modelo de visión por computadora enfocado en el establecimiento de correspondencias densas entre imágenes basado en la coincidencia de características kernelizadas. El proyecto de investigación fue presentado en el CVPR de 2020 en el artículo de investigación “DKM: Dense Kernelized Feature Matching for Geometry Estimation” por Johan Edstedt, Ioannis Athanasiadis, Marten Wadenback y ˚Michael Felsberg [44].

El objetivo principal de DKM es lograr una odometría visual precisa y en tiempo real, que es el proceso de estimar el movimiento (posición y orientación) de una cámara basado en imágenes consecutivas. Las correspondencias densas entre imágenes son esenciales para una estimación de movimiento precisa, especialmente en entornos desafiantes y dinámicos.

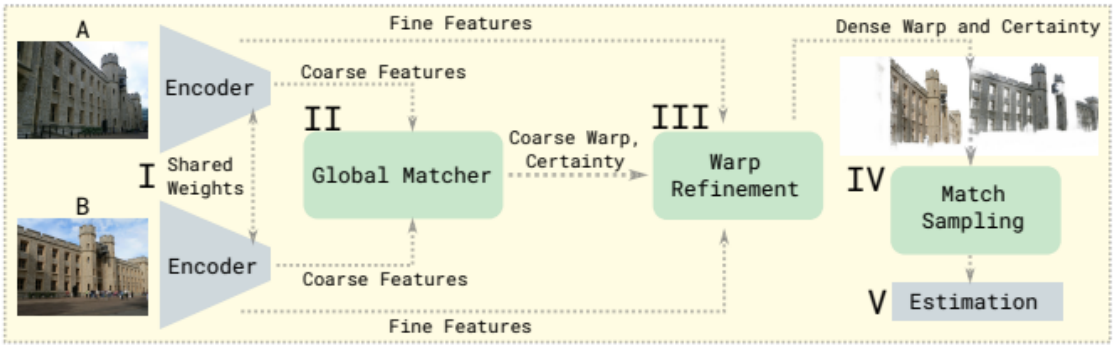


Figura 9.4-1: Arquitectura del modelo DKM

La arquitectura de DKM consta de un detector y un descriptor de características, seguidos de la kernelización de descriptores y la correspondencia de características. Todo esto termina con la estimación de la geometría para lograr una odometría visual precisa y en tiempo real.

9.4.2 Arquitectura

9.4.2.1 Detector de Características

El primer componente de la arquitectura es un detector de características que se encarga de identificar puntos clave o características distintivas en las imágenes. Estas características son ubicaciones importantes en las imágenes que pueden usarse para establecer correspondencias entre ellas.

9.4.2.2 Descriptor de Características

Una vez que se han detectado las características, el siguiente componente es el descriptor de características. Este componente se encarga de generar descripciones o representaciones compactas de las características detectadas. Los descriptores capturan la información local alrededor de cada característica y se utilizan para comparar y establecer correspondencias entre características en imágenes diferentes.

9.4.2.3 Kernelización de Descriptores

Uno de los aspectos clave de la arquitectura de DKM es la utilización de métodos de kernel para la coincidencia de descriptores. Los métodos de kernel permiten realizar mapeos no lineales de los descriptores a un espacio de características de mayor dimensión. Esto permite capturar relaciones complejas entre descriptores y mejorar el rendimiento de la coincidencia, especialmente en situaciones con cambios significativos en la apariencia y variaciones en el punto de vista.

9.4.2.4 Correspondencia de Características

El componente de correspondencia de características es responsable de emparejar los descriptores entre las imágenes. Utilizando los descriptores kernelizados, este componente busca establecer correspondencias precisas y densas entre características en las imágenes consecutivas. La correspondencia de características es fundamental para estimar el movimiento de la cámara en la odometría visual.

9.4.2.5 Estimación de la Geometría

Finalmente, la arquitectura de DKM utiliza las correspondencias de características para estimar la geometría del movimiento de la cámara entre las imágenes. Esto implica determinar la posición y orientación relativa de la cámara en cada imagen sucesiva. La estimación precisa de la geometría es crucial para lograr una odometría visual precisa y en tiempo real.

10 RESULTADOS

10.1 Mean Average Accuracy (mAA)

La media de la precisión media (o mean Average Accuracy) se utiliza comúnmente para analizar el rendimiento de los sistemas de segmentación y detección de objetos. La media de los valores de precisión promedio se calcula sobre los valores entre 0 y 1. La fórmula se basa en las siguientes submétricas [45] [46]:

- **Precisión:** la precisión mide el porcentaje de las predicciones que son correctas.

- **Recuperación:** la recuperación mide qué tan bien se encuentran los casos positivos en el modelo.

$$Precisión = \frac{TP}{TP+FP} \quad (10.1.1)$$

$$Recuperación = \frac{TP}{Tp+FN} \quad (10.1.2)$$

$$F1\ score = 2 \cdot \frac{precisión \cdot recuperación}{precisión+recuperación} \quad (10.1.3)$$

$$mAA = \frac{1}{N}\sum_{i=1}^{N} AP_i \quad (10.1.4)$$

TP = Verdadero positivo
TN = Verdadero negativo
FP = Falso positivo
FN = Falso negativo
AP = Precisión media

El mAA incorpora el equilibrio entre precisión y recuperación y considera tanto los falsos positivos como los falsos negativos. Esta propiedad hace que sea una métrica adecuada para la mayoría de las aplicaciones de detección.

10.2 Cálculo "a mano" del mAA para un conjunto reducido del dataset de entrenamiento

Debido al reducido número de imágenes de testeo que ofrece Kaggle para que la propia página nos ofrezca los resultados, se ha optado por calcular el mAA "a mano" para un conjunto reducido de parejas de imágenes del dataset de entrenamiento elegidas de forma aleatoria.

Dentro de todos los datos que se describieron en el apartado 8.5 del documento, vamos a tener en cuenta el valor medio de la covisibilidad de las parejas de imágenes que hemos utilizado para hacer esta nueva prueba. Indicar que se utilizaran las mismas 50 parejas de imágenes para todos los modelos.

	LoFTR mAA	**SuperGlue mAA**	**ASLFeat mAA**	**DKM mAA**	**Covisbilidad media**
Museo Británico	0.636	0.506	0.344	0.632	0.36268
Puerta de Brandeburgo	0.638	0.462	0.376	0.58	0.30838
Palacio de Buckingham	0.294	0.188	0.12	0.178	0.21836
Exterior del Coliseo Romano	0.302	0.152	0.228	0.256	0.3112
Gran plaza de Bruselas	0.354	0.304	0.238	0.380	0.2482

Estatua de Lincoln	0.790	0.712	0.732	0.784	0.33824
Fachada frontal de Notre-Dame	0.844	0.642	0.734	0.804	0.3621
Exterior del Panteón	0.742	0.336	0.392	0.448	0.37312
Plaza de San Marco	0.184	0.196	0.130	0.268	0.2889
Sagrado Corazón	0.724	0.532	0.518	0.628	0.28242
Sagrada Familia	0.696	0.556	0.614	0.756	0.4529
Catedral de San Pauls	0.588	0.388	0.422	0.480	0.40576
Plaza de San Pedro	0.406	0.176	0.198	0.250	0.27146
Taj Mahal	0.732	0.57	0.420	0.714	0.32914
Templo japonés de Nara	0.526	0.408	0.404	0.466	0.31606
Fontana di Trevi	0.734	0.55	0.566	0.682	0.49882
Media del dataset	0.57438	0.41738	0.40225	0.51913	

Tabla 10-1. Media de la precisión media de cada modelo para las imágenes de entrenamiento y su covisibilidad media

	GPU	**Tiempo de procesado**
LoFTR	P100	27min, 44s
SuperGlue	P100	3min, 12s
ASLFeat	P100	1h, 21min, 49s
DKM	P100	3min, 19s

Tabla 10-2. Tabla comparativa de los tiempos de procesado de los distintos modelos para los datos de entrenamiento

De la tabla 10-1, podemos ver como los modelos obtienen mejores y peores puntuaciones en los mismos monumentos. Obteniendo, por ejemplo, buenas puntuaciones en la Fachada frontal de Notre-Dame, entre 0.65 y 0.85, y bajas puntuaciones en monumento tales como la Plaza de San Marco, rondando valores entre 0.13 y

0.27. Por otro lado, se han elegido imágenes con una covisibilidad media baja para que esto suponga un desafío para nuestros modelos. A continuación, se muestran dos parejas de imágenes: una de la Plaza de San Marco y otra de la fachada frontal de Notre-Dame.

Figura 10-1: Pareja de imágenes de ejemplo de la Plaza de San Marco

Figura 10-2: Pareja de imágenes de ejemplo de la fachada frontal de Notre-Dame

Por último, comentar de esta tabla, que LoFTR se destaca como el modelo más preciso, con una puntuación de 0.57, seguido de DKM con un 0.52. SuperGlue y ASLFeat obtienen puntuaciones más bajas que los dos modelos anteriores y con valores muy similares.

En la tabla 10-2 se han recogido los tiempos de procesamiento de cada uno de los modelos. Tanto SuperGlue como DKM son modelos rápidos pues tardan en procesar los emparejamientos unos 3 minutos respecto a los 27 minutos que tarda LoFTR. ASLFeat se desmarca mucho de estos tiempos con más de 1 hora y 20 minutos.

10.3 Resultados envío Kaggle

Tras los resultados obtenidos con los datos de entrenamiento, se opta por resolver el propio problema que nos ofrece Kaggle. Esto es, a través de las imágenes de testeo. Para comprobar la precisión de las distintas arquitecturas de una manera sencilla se ha recurrido al envío de los resultados obtenidos con cada modelo a la propia competición de Kaggle. Esta te devolverá un resultado que consiste en calcular la mAA (mean Average Accuracy) de las matrices fundamentales (la matriz fundamental encapsula la geometría proyectiva entre dos vistas de la misma escena. No se ve afectada por el contenido de la escena y depende únicamente de los parámetros intrínsecos y extrínsecos de las dos cámaras) de las tres imágenes de testeo.

Dicho esto, se han obtenido los siguientes resultados:

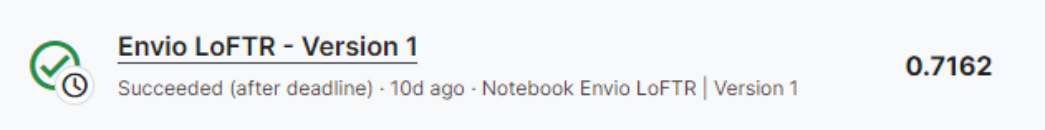


Figura 10-1: Resultado obtenido con LoFTR en Kaggle

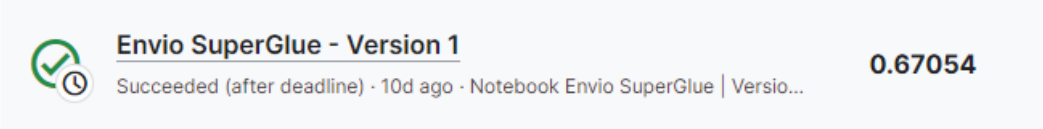


Figura 10-2: Resultado obtenido con SuperGlue en Kaggle

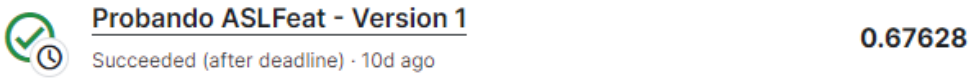


Figura 10-3: Resultado obtenido con ASLFeat en Kaggle

Figura 10-4: Resultado obtenido con DKM en Kaggle

Vemos que los resultados son muy similares y dan un grado de confianza muy similar. Esto puede ser así por el bajo tratamiento de las imágenes de testeo. Podemos ver que LoFTR es el modelo que mejores resultados da con un grado de confianza del 71.6% y esto se ha conseguido simplemente cambiando la resolución de la imagen a una más apropiada; a 840p.

10.4 Datos recuperación modelos

	Keypoints	Good Keypoints	%
LoFTR	883	511	57.87
	537	331	61.64
	238	54	22.69
SuperGlue	1024	196	19.14
	1024	209	20.41
	1024	54	5.27
ASLFeat	728	170	23.35
	339	45	13.27
	212	16	7.55
DKM	2000	1031	51.55
	2000	1357	67.85
	2000	600	30

Tabla 10-3. Tabla comparativa de "keypoints" utilizados y aceptados

Otra de las estadísticas que podemos obtener es el número de "keypoints" que ha utilizado cada uno de los modelos y, por consiguiente, el número de "keypoints" que ha conseguido emparejar correctamente entre las dos imágenes.

De la tabla anterior podemos comparar **DKM** y **SuperGlue** como modelos con un número fijo de características para las tres parejas de imágenes, utilizando 2000 y 1024 respectivamente. **SuperGlue** acepta como válidas el 19.14%, 20.41% y 5.27% de los puntos para las imágenes 1,2 y 3 y por otro lado, DKM acepta como válidas el 51.55%, 67.85% y 30% de los puntos.

Por otra parte, **LoFTR** y **ASLFeat** utilizan un número distinto para cada una de las imágenes, los cuales podemos ver en la tabla 9-3. También se recoge el porcentaje de puntos que toma como correctos.

Atendiendo a los porcentajes de los 4 modelos, podemos deducir que la segunda pareja de imágenes es la más fácil de emparejar, seguida de la primera pareja por un corto porcentaje para terminar con la tercera pareja que

se encuentra mucho más alejada en cuanto a porcentaje de puntos aceptados se refiere.

También podemos dividir los modelos en dos grupos según estos porcentajes pues **LoFTR** y **DKM** oscilan en torno al 50-60% para la primera pareja, 60-70% para la segunda y 20-30% para la tercera. **SuperGlue** y **ASLFeat** oscilan sobre el 19-25% para la primera, 13-21% para la segunda y 5-8% para la última.

ASLFeat se destaca como la única arquitectura que posiciona a la segunda pareja de imágenes como la más difícil de emparejar.

De los dos resultados que hemos obtenido, podemos destacar a **LoFTR** como el modelo más fiable al ser el segundo modelo que menos puntos utiliza y el segundo modelo que mayor porcentaje de puntos toma como válidos dando además la mayor fiabilidad de las 4 arquitecturas con un 72% aproximadamente.

Por otro lado, podemos ver los datos con otra perspectiva. Si queremos, posteriormente, utilizar nuestros resultados para realizar un modelo 3D de un elemento, buscaremos el modelo que nos ofrezca un mayor número de "Good Keypoints" puesto que, a mayor número de puntos, mayor fidelidad tendrá nuestra reconstrucción 3D. En ese caso DKM se destaca como el mejor modelo, seguido de LoFTR.

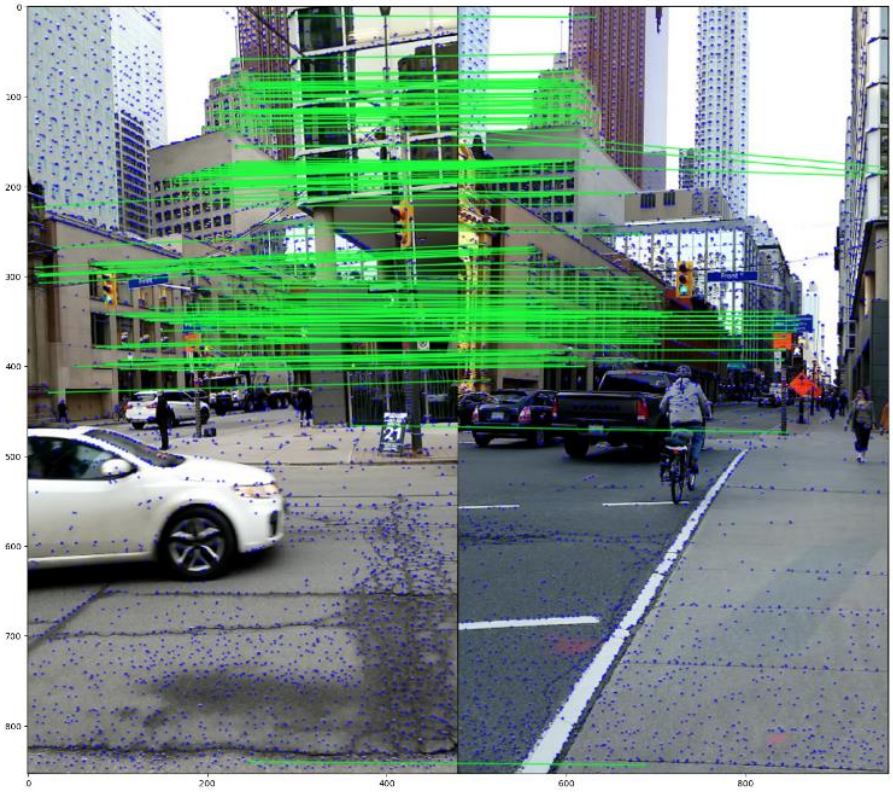


Figura 10-5. Ejemplo de emparejamiento pareja 1

Figura 10-6. Ejemplo de emparejamiento pareja 2

Figura 10-7. Ejemplo de emparejamiento pareja 3

Las imágenes 10-5, 10-6 y 10-7 son un ejemplo de cómo se representan los emparejamientos de los puntos que sacan los modelos. En este caso es de la arquitectura ASLFeat.

En las imágenes, podemos ver una serie de puntos representados en azul. Si entre dos de estos puntos hay una línea verde, significa que estos son dos puntos emparejados correctamente, según el modelo.

10.4 Tiempo de procesado de los modelos

	GPU	Tiempo de procesado
LoFTR	P100	112.1s
SuperGlue	P100	32.6s
ASLFeat	P100	48.7s
DKM	P100	87.8s

Tabla 10-4. Tabla comparativa de los tiempos de procesado de los distintos modelos

Otro de los datos que podemos extraer de los modelos es el tiempo que han tardado en procesar y obtener los resultados que buscamos. Indicar que en todos los casos hemos utilizado la misma GPU de entre todas las que nos ofrece Kaggle. Esta es GPU-P100.

El modelo más rápido es Superglue con 32.6s, seguido de ASLFeat, 48.7s. En tercer lugar, tenemos a DKM que es capaz de realizar todo el proceso en 87.8s para terminar con LoFTR con 112.1s.

11 CONCLUSIONES

Tras la implementación y comparación de varios modelos, se pueden realizar varias observaciones y sacar conclusiones:

- Los resultados en cuanto al cálculo de la precisión media del modelo con los datos de entrenamiento nos indican que hay ciertos modelos que encuentran dificultades a la hora de emparejar una serie de monumento, ya sea por la arquitectura del mismo, la luz o la rotación. Mientras que hay otros que es capaz de emparejar sus puntos con una alta precisión. LoFTR y DKM se diferencian como los mejores modelos para esta tarea. ASLFeat y SuperGlue tienen más dificultades para obtener resultados precisos.
- Los resultados en cuanto a la resolución del problema que nos ofrece Kaggle son muy similares en todos los casos, sólo se destaca a LoFTR como el modelo que mayor precisión ofrece. Los modelos que han sido entrenados con miles de imágenes no tienen grandes problemas para emparejar puntos en imágenes con pocas variaciones y sí encuentran más dificultades para aquellas parejas que difieren en mayor medida.
- El número de "Keypoints" que necesita cada modelo no es relevante en cuanto a la precisión o tiempo de procesado pues podemos ver cómo LoFTR es el modelo que más tiempo necesita para procesar las imágenes y sin embargo es el segundo con menos puntos utilizados.
- Por otro lado, el porcentaje de "Keypoints" tampoco es correlativo con la precisión del modelo puesto que, aunque podamos ver que LoFTR es el modelo que mayor porcentaje tiene y es el modelo más preciso, el resto de modelos – que tiene una precisión similar – tiene un porcentaje de aciertos mucho más dispar.
- Sin embargo, si podemos ver que hay una relación entre porcentaje de aciertos y el tiempo de procesado puesto que los modelos con un mayor porcentaje de aciertos, tardan más en procesar las imágenes.
- Si vemos el número de "Good Keypoints" con otra perspectiva, vemos que DKM es el modelo que nos ofrece un mayor número de puntos "aceptados", seguido de LoFTR. Si buscamos darle una aplicación futura para la reconstrucción de modelos 3D, esto es un dato muy relevante pues a mayor número de puntos, más información para montar nuestro modelo 3D.
- En cuanto al tiempo de procesado de los modelos, sí es un dato relevante a la hora de elegir nuestra arquitectura. Si necesitamos procesar una cantidad considerable de imágenes, DKM y SuperGlue ofrecen grandes resultados puesto que se diferencian en casi 25 minutos con respecto a LoFTR. Muy alejado se encuentra ASLFeat. Para pocas imágenes, los resultados son distintos. ASLFeat ahora necesita poco tiempo para procesar las tres parejas de imágenes. SuperGlue y DKM siguen siendo modelos rápidos y LoFTR se encontraría en última posición.
- Por último, indicar la comodidad y facilidades a la hora de utilizar cada uno de los modelos. LoFTR es el modelo más fácil de utilizar y que ofrece un mayor número de estadísticas y características útiles para sacar conclusiones de los resultados obtenidos. Seguido de LoFTR estaría SuperGlue muy cercano a este. En otra parte estarían DKM y ASLFeat como los modelos que tienen más dependencias y son menos intuitivos de utilizar, así como los modelos que menos estadísticas ofrecen para analizar los resultados.

De acuerdo con los puntos expuestos anteriormente y las pruebas realizadas, el modelo que utilizaría sería LoFTR si tuviera tanto un número de datos limitado como grandes cantidades de datos pues ofrece una mayor precisión y además es muy cómodo de utilizar junto con el gran número de características que se pueden usar. Por otro lado, DKM también se posiciona como un gran modelo de emparejamiento de imágenes ya que ofrece tiempos rápidos de procesamiento, una precisión similar a LoFTR y es capaz de generar un elevado número de "Good Keypoints" para posteriores aplicaciones.

Como evolución del proyecto elaborado y los conceptos aprendidos, el siguiente paso sería el crear un propio modelo de red neuronal artificial que fuera capaz de resolver el problema que se ha presentado. El primer modelo

que crearía no sería muy preciso, pero poco a poco habría que aprender más sobre redes neuronales hasta poder obtener resultados similares a los presentados por los distintos modelos.

Como utilidad para llevar del banco teórico a la práctica, se podría crear una aplicación que fuera capaz de recopilar miles y miles de resultados obtenidos al fotografiar un elemento desde distintas perspectivas para, posteriormente, ser capaz de crear un modelo 3D de dicho elemento. Esto me parece una aplicación muy interesante, sobre todo, de cara al futuro pues se podría llegar a obtener un banco digital con todos los elementos de nuestro mundo, ya sean pasados o presentes, para que posteriores generaciones tengan acceso a esta información y no se pierdan fragmentos de nuestra historia.

A lo largo de todo el documento, se ha realizado un viaje a través de los conceptos básicos de las RNA, del Aprendizaje automático, de las Inteligencias Artificiales, … Viendo sus distintos usos y utilidades provechosas que poco a poco la mayoría de los sectores incorporan en sus actividades del día a día. Hace unos años estas tecnologías sólo eran conocidas por las empresas, pero cada vez más personas de a pie van conociéndolas y comprendiendo su utilidad y funcionamiento. Este paso al "público general" solo hará que la tecnología se desarrolle más y más. Las Redes Neuronales Artificiales y, en general, la Inteligencia Artificial, es una tecnología tanto de presente cómo de futuro.

Anexo

En este apartado se incluirán los códigos utilizados para obtener los resultados mostrados en los apartados anteriores. Los códigos están basados e inspirados en soluciones propuestas por distintos usuarios de Kaggle.

LoFTR

[50], [51], [52]

Datasets necesarios

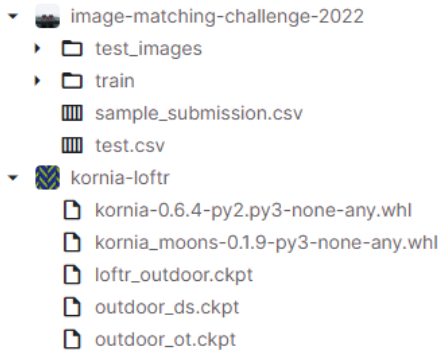


Figura A-1. Datasets necesarios para LoFTR

Librerias y dependencias

```
[ ]:
%%capture
#dry_run = False
!pip install ../input/kornia-loftr/kornia-0.6.4-py2.py3-none-any.whl
!pip install ../input/kornia-loftr/kornia_moons-0.1.9-py3-none-any.whl
```

```
[ ]:
import os
import numpy as np
import cv2
import csv
from glob import glob
import torch
import matplotlib.pyplot as plt
import kornia
from kornia_moons.feature import *
import kornia as K
import kornia.feature as KF
import gc
import pandas as pd
import time
```

Figura A-2. Librerias y dependencias utilizadas en LoFTR

Inicialización del “matcher”

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
matcher = KF.LoFTR(pretrained=None)
matcher.load_state_dict(torch.load("../input/kornia-loftr/loftr_outdoor.ckpt")['state_dict'])
matcher = matcher.to(device).eval()
```

Figura A-3. Inicialización “matcher” de LoFTR

Carga de las imágenes de testeo a utilizar

```
src = '/kaggle/input/image-matching-challenge-2022/'

test_samples = []
with open(f'{src}/test.csv') as f:
    reader = csv.reader(f, delimiter=',')
    for i, row in enumerate(reader):
        # Skip header.
        if i == 0:
            continue
        test_samples += [row]
```

Figura A-4. Carga imágenes de testeo LoFTR

Definición de funciones necesarias

```
def FlattenMatrix(M, num_digits=8):
    '''Convenience function to write CSV files.'''

    return ' '.join([f'{v:.{num_digits}e}' for v in M.flatten()])

def load_torch_image(fname, device):
    img = cv2.imread(fname)
    scale = 840 / max(img.shape[0], img.shape[1])
    w = int(img.shape[1] * scale)
    h = int(img.shape[0] * scale)
    img = cv2.resize(img, (w, h))
    img = K.image_to_tensor(img, False).float() /255.
    img = K.color.bgr_to_rgb(img)
    return img.to(device)
```

Figura A-5. Funciones necesarias en LoFTR

Extracción de Keypoints y “matches”

```
F_dict = {}
for i, row in enumerate(test_samples):
    sample_id, batch_id, image_1_id, image_2_id = row
    # Load the images.
    st = time.time()
    image_1 =load_torch_image(f'{src}/test_images/{batch_id}/{image_1_id}.png', device)
    image_2 = load_torch_image(f'{src}/test_images/{batch_id}/{image_2_id}.png', device)
    print(image_1.shape)
    input_dict = {"image0": K.color.rgb_to_grayscale(image_1),
              "image1": K.color.rgb_to_grayscale(image_2)}

    with torch.no_grad():
        correspondences = matcher(input_dict)

    mkpts0 = correspondences['keypoints0'].cpu().numpy()
    mkpts1 = correspondences['keypoints1'].cpu().numpy()
    print(f'mkpts0: {len(mkpts0)}')
    print(f'mkpts1: {len(mkpts1)}')

    if len(mkpts0) > 7:
        F, inliers = cv2.findFundamentalMat(mkpts0, mkpts1, cv2.USAC_MAGSAC, 0.5, 0.99999, 100000)
        inliers = inliers > 0
        assert F.shape == (3, 3), 'Malformed F?'
        F_dict[sample_id] = F
    else:
        F_dict[sample_id] = np.zeros((3, 3))
        continue
    gc.collect()
    nd = time.time()
    correctos=0
    for h in range(len(inliers)):
        if (inliers[h]==True):
            correctos = correctos + 1
    print(correctos)
```

Figura A-6. Código para obtener “matches” en LoFTR

Creación fichero de envío a Kaggle

```
with open('submission.csv', 'w') as f:
    f.write('sample_id,fundamental_matrix\n')
    for sample_id, F in F_dict.items():
        f.write(f'{sample_id},{FlattenMatrix(F)}\n')
```

Figura A-7. Creación fichero con matrices fundamentales para Kaggle

ASLFeat

[53], [54]

Datasets necesarios

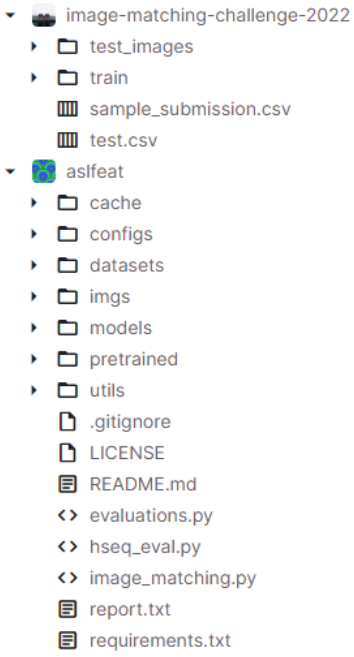


Figura A-8. Datasets necesarios para ASLFeat

Librerias y dependencias

```
import numpy as np
import pandas as pd
import csv
import cv2
import tensorflow as tf
import gc
import sys
import yaml
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm

#ASLFeat
sys.path.append('../input/aslfeat')
from models import get_model

tf.compat.v1.disable_eager_execution()
```

Figura A-9. Librerias y dependencias utilizadas en ASLFeat

Inicialización del modelo

```
# Importar Configuracion
with open('../input/aslfeat/configs/matching_eval2.yaml', 'r') as f:
        config = yaml.load(f, Loader=yaml.FullLoader)

# Eleccion modelo pre-entrenado
aslfeat_model_path = '../input/aslfeat/pretrained/aslfeatv2/model.ckpt-60000'
config['model_path'] = aslfeat_model_path
config['net']['config']['kpt_n'] = 8192 # Original config 8000

# Resumen config
print(config)
```

```
# Crear Modelo
model = get_model('feat_model')(aslfeat_model_path, **config['net'])
```

Figura A-10. Inicialización modelo de ASLFeat

Carga de las imágenes de testeo a utilizar

```
src = '/kaggle/input/image-matching-challenge-2022/'

test_samples = []
with open(f'{src}/test.csv') as f:
    reader = csv.reader(f, delimiter=',')
    for i, row in enumerate(reader):
        # Skip header.
        if i == 0:
            continue
        test_samples += [row]
```

Figura A-11. Carga imágenes de testeo ASLFeat

Definición de funciones necesarias

```python
# ASLFeat Functions
def load_imgs(img_paths):
    rgb_list = []
    gray_list = []

    for img_path in img_paths:
        img = cv2.imread(img_path)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)[..., np.newaxis]
        img = img[..., ::-1]
        rgb_list.append(img)
        gray_list.append(gray)

    return rgb_list, gray_list

def extract_local_features(gray_list):
    descs = []
    kpts = []

    for gray_img in gray_list:
        desc, kpt = [], []
        desc, kpt, _ = model.run_test_data(gray_img)
        descs.append(desc)
        kpts.append(kpt)

    return descs, kpts
```

```python
class MatcherWrapper(object):
    """OpenCV matcher wrapper."""

    def __init__(self):
        # Swapped BFMatcher to FlannBasedMatcher
        # FLANN parameters
        FLANN_INDEX_KDTREE = 0
        index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
        search_params = dict(checks = 100)   # or pass empty dictionary
        self.matcher = cv2.FlannBasedMatcher(index_params, search_params)

    def get_matches(self, feat1, feat2, cv_kpts1, cv_kpts2, ratio = 0.8, cross_check = True, err_thld = 0.5):
        """Compute putative and inlier matches.
        Args:
            feat: (n_kpts, 128) Local features.
            cv_kpts: A list of keypoints represented as cv2.KeyPoint.
            ratio: The threshold to apply ratio test.
            cross_check: (True by default) Whether to apply cross check.
            err_thld: Epipolar error threshold.
        Returns:
            good_matches: Putative matches.
            mask: The mask to distinguish inliers/outliers on putative matches.
        """

        init_matches1 = self.matcher.knnMatch(feat1, feat2, k = 2)
        init_matches2 = self.matcher.knnMatch(feat2, feat1, k = 2)

        good_matches = []
```

```python
    for i in range(len(init_matches1)):
        cond = True
        if cross_check:
            cond1 = cross_check and init_matches2[init_matches1[i][0].trainIdx][0].trainIdx == i
            cond *= cond1
        if ratio is not None and ratio < 1:
            cond2 = init_matches1[i][0].distance <= ratio * init_matches1[i][1].distance
            cond *= cond2
        if cond:
            good_matches.append(init_matches1[i][0])

    if type(cv_kpts1) is list and type(cv_kpts2) is list:
        good_kpts1 = np.array([cv_kpts1[m.queryIdx].pt for m in good_matches])
        good_kpts2 = np.array([cv_kpts2[m.trainIdx].pt for m in good_matches])
    elif type(cv_kpts1) is np.ndarray and type(cv_kpts2) is np.ndarray:
        good_kpts1 = np.array([cv_kpts1[m.queryIdx] for m in good_matches])
        good_kpts2 = np.array([cv_kpts2[m.trainIdx] for m in good_matches])
    else:
        good_kpts1 = np.empty(0)
        good_kpts2 = np.empty(0)

    # Calculate Fundamental Mask and inliers
    F, mask = get_fundamental_matrix(good_kpts1, good_kpts2, err_thld)
    return F, good_matches, mask

def draw_matches(self, img1, cv_kpts1, img2, cv_kpts2, good_matches, mask, match_color = (0, 255, 0), pt_color = (0, 0, 255)):
    """Draw matches."""
    if type(cv_kpts1) is np.ndarray and type(cv_kpts2) is np.ndarray:
        cv_kpts1 = [cv2.KeyPoint(cv_kpts1[i][0], cv_kpts1[i][1], 1) for i in range(cv_kpts1.shape[0])]
        cv_kpts2 = [cv2.KeyPoint(cv_kpts2[i][0], cv_kpts2[i][1], 1) for i in range(cv_kpts2.shape[0])]

    display = cv2.drawMatches(img1, cv_kpts1, img2, cv_kpts2, good_matches,
                              None,
                              matchColor = match_color,
                              singlePointColor = pt_color,
                              matchesMask = mask.ravel().tolist(), flags=4)
    return display
```

```python
def get_aslfeat_fmatrix(batch_id, img_id1, img_id2, plot = False):
    image_fpath_1 = f'{src}/test_images/{batch_id}/{img_id1}.png'
    image_fpath_2 = f'{src}/test_images/{batch_id}/{img_id2}.png'

    # Cargar Pareja de imagenes
    rgb_list, gray_list = load_imgs([image_fpath_1, image_fpath_2])

    # Extraer Local Features
    descs, kpts = extract_local_features(gray_list)

    # Emparejamiento.
    matcher = MatcherWrapper()
    fundamental_matrix, match, mask = matcher.get_matches(descs[0], descs[1], kpts[0], kpts[1],
                                                          ratio = None,
                                                          cross_check = True,
                                                          err_thld = 0.20)
    return fundamental_matrix
```

Figura A-12. Funciones necesarias en LoFTR

Obtener “matches” y matriz fundamental

```python
f_matrix_dict = {}
for i, row in tqdm(enumerate(test_samples)):
    sample_id, batch_id, img_id1, img_id2 = row

    # Set Plot
    plot = False
    if i < 3: plot = True

    # Obtener Matriz Fundamental con ASLFeat y FLANNBasedMatcher
    f_matrix_dict[sample_id] = get_aslfeat_fmatrix(batch_id, img_id1, img_id2, plot)
```

Figura A-13. Código para obtener “matches” en LoFTR

Creación fichero de envío a Kaggle

```python
with open('submission.csv', 'w') as f:
    f.write('sample_id,fundamental_matrix\n')
    for sample_id, F in F_dict.items():
        f.write(f'{sample_id},{FlattenMatrix(F)}\n')
```

Figura A-14. Creación fichero con matrices fundamentales para Kaggle

SuperGlue

[54], [55], [56]

Datasets necesarios

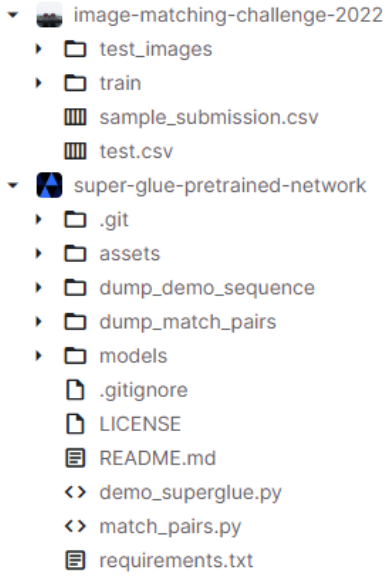


Figura A-15. Datasets necesarios para SuperGlue

Librerias y dependencias

```
import os
import csv
import random
from glob import glob
from tqdm import tqdm
from collections import namedtuple

import cv2
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import torch

import sys
sys.path.append("../input/super-glue-pretrained-network")
from models.matching import Matching
from models.utils import (compute_pose_error, compute_epipolar_error,
                          estimate_pose, make_matching_plot,
                          error_colormap, AverageTimer, pose_auc, read_image,
                          rotate_intrinsics, rotate_pose_inplane,
                          scale_intrinsics)
```

Figura A-16. Librerias y dependencias utilizadas en SuperGlue

Inicialización del “matcher”

```
device = "cuda" if torch.cuda.is_available() else "cpu"
resize = [-1, ]
resize_float = True

config = {
    "superpoint": {
        "nms_radius": 4,
        "keypoint_threshold": 0.005,
        "max_keypoints": 1024
    },
    "superglue": {
        "weights": "outdoor",
        "sinkhorn_iterations": 20,
        "match_threshold": 0.2,
    }
}
matching = Matching(config).eval().to(device)
```

Figura A-17. Inicialización “matcher” de SuperGlue

Carga de las imágenes de testeo a utilizar

```
src = '/kaggle/input/image-matching-challenge-2022/'

test_samples = []
with open(f'{src}/test.csv') as f:
    reader = csv.reader(f, delimiter=',')
    for i, row in enumerate(reader):
        # Skip header.
        if i == 0:
            continue
        test_samples += [row]
```

Figura A-18. Carga imágenes de testeo SuperGlue

Definición de funciones necesarias

```
def FlattenMatrix(M, num_digits=8):
    '''Convenience function to write CSV files.'''
    return ' '.join([f'{v:.{num_digits}e}' for v in M.flatten()])
```

Figura A-19. Funciones necesarias en SuperGlue

Extracción de Keypoints y “matches”

```
F_dict = {}
for i, row in tqdm(enumerate(test_samples)):
    sample_id, batch_id, image_1_id, image_2_id = row

    image_fpath_1 = f'{src}/test_images/{batch_id}/{image_1_id}.png'
    image_fpath_2 = f'{src}/test_images/{batch_id}/{image_2_id}.png'

    image_1, inp_1, scales_1 = read_image(image_fpath_1, device, resize, 0, resize_float)
    image_2, inp_2, scales_2 = read_image(image_fpath_2, device, resize, 0, resize_float)

    pred = matching({"image0": inp_1, "image1": inp_2})
    pred = {k: v[0].detach().cpu().numpy() for k, v in pred.items()}
    kpts1, kpts2 = pred["keypoints0"], pred["keypoints1"]
    matches, conf = pred["matches0"], pred["matching_scores0"]

    valid = matches > -1
    mkpts1 = kpts1[valid]
    mkpts2 = kpts2[matches[valid]]
    mconf = conf[valid]
    print(f"kpts1: {len(kpts1)}")
    print(f"kpts2: {len(kpts2)}")
    print(f"mkpts1: {len(mkpts1)}")
    print(f"mkpts2: {len(mkpts2)}")

    if len(mkpts1) > 8:
        F, inlier_mask = cv2.findFundamentalMat(mkpts1, mkpts2, cv2.USAC_MAGSAC, ransacReprojThreshold=0.25, confidence=0.99999, maxIters=10000)
        F_dict[sample_id] = F
    else:
        F_dict[sample_id] = np.zeros((3, 3))
```

Figura A-20. Código para obtener “matches” en SuperGlue

Creación fichero de envío a Kaggle

```
with open('submission.csv', 'w') as f:
    f.write('sample_id,fundamental_matrix\n')
    for sample_id, F in F_dict.items():
        f.write(f'{sample_id},{FlattenMatrix(F)}\n')
```

Figura A-21. Creación fichero con matrices fundamentales para Kaggle

DKM

[57], [58]

Datasets necesarios

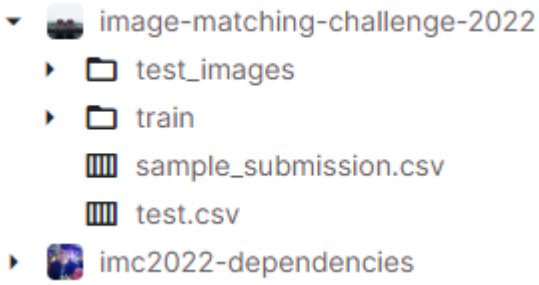


Figura A-22. Datasets necesarios para DKM

Librerias y dependencias

```
!mkdir -p pretrained/checkpoints
!cp /kaggle/input/imc2022-dependencies/pretrained/dkm.pth pretrained/checkpoints/dkm_base_v11.pth

!pip install -f /kaggle/input/imc2022-dependencies/wheels --no-index einops
!cp -r /kaggle/input/imc2022-dependencies/DKM/ /kaggle/working/DKM/
!cd /kaggle/working/DKM/; pip install -f /kaggle/input/imc2022-dependencies/wheels -e .
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import sys, os, csv
from PIL import Image
import cv2, gc
import matplotlib.pyplot as plt
import torch
sys.path.append('/kaggle/input/imc2022-dependencies/DKM/')

dry_run = False
import torch
if not torch.cuda.is_available():
    print('You may want to enable the GPU switch?')

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

torch.hub.set_dir('/kaggle/working/pretrained/')
from dkm import dkm_base
model = dkm_base(pretrained=True, version="v11").to(device).eval()
# model.load_state_dict(torch.load(WEIGHTS))
```

Figura A-23. Librerias y dependencias utilizadas en DKM

Carga de las imágenes de testeo a utilizar

```
src = '/kaggle/input/image-matching-challenge-2022/'

test_samples = []
with open(f'{src}/test.csv') as f:
    reader = csv.reader(f, delimiter=',')
    for i, row in enumerate(reader):
        # Skip header.
        if i == 0:
            continue
        test_samples += [row]
```

Figura A-24. Carga imágenes de testeo DKM

Definición de funciones necesarias

```
def FlattenMatrix(M, num_digits=8):
    '''Convenience function to write CSV files.'''
    return ' '.join([f'{v:.{num_digits}e}' for v in M.flatten()])
```

Figura A-25. Funciones necesarias en DKM

Extracción de Keypoints y “matches”

```
F_dict = {}

for i, row in enumerate(test_samples):
    sample_id, batch_id, image_1_id, image_2_id = row

    img1 = cv2.imread(f'{src}/test_images/{batch_id}/{image_1_id}.png')
    img2 = cv2.imread(f'{src}/test_images/{batch_id}/{image_2_id}.png')

    img1PIL = Image.fromarray(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
    img2PIL = Image.fromarray(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))

    dense_matches, dense_certainty = model.match(img1PIL, img2PIL)
    dense_certainty = dense_certainty.sqrt()
    sparse_matches, sparse_certainty = model.sample(dense_matches, dense_certainty, 2000)


    mkps1 = sparse_matches[:, :2]
    mkps2 = sparse_matches[:, 2:]

    h, w, c = img1.shape
    mkps1[:, 0] = ((mkps1[:, 0] + 1)/2) * w
    mkps1[:, 1] = ((mkps1[:, 1] + 1)/2) * h

    h, w, c = img2.shape
    mkps2[:, 0] = ((mkps2[:, 0] + 1)/2) * w
    mkps2[:, 1] = ((mkps2[:, 1] + 1)/2) * h

    F, mask = cv2.findFundamentalMat(mkps1, mkps2, cv2.USAC_MAGSAC, 0.3, 0.9999, 25_000)

    good = F is not None and F.shape == (3,3)
```

```
    correcto=0
    erroneos=0
    for k in range(len(mask)):
        if mask[k]==1:
            correcto=correcto+1
        else:
            erroneos=erroneos+1

    print(correcto)
    print(erroneos)


    if good:
        F_dict[sample_id] = F
        correctos = correctos + 1
    else:
        F_dict[sample_id] = np.zeros((3, 3))
        erroneos = erroneos + 1
        continue

    gc.collect()
```

Figura A-26. Código para obtener “matches” en DKM

Creación fichero de envío a Kaggle

```
with open('submission.csv', 'w') as f:
    f.write('sample_id,fundamental_matrix\n')
    for sample_id, F in F_dict.items():
        f.write(f'{sample_id},{FlattenMatrix(F)}\n')
```

Figura A-27. Creación fichero con matrices fundamentales para Kaggle

Añadidos generales para las pruebas con dataset de entrenamiento

[59], [60]

```
!cp -r ../input/imutils/imutils-0.5.3/ /
!pip install /imutils-0.5.3/
```

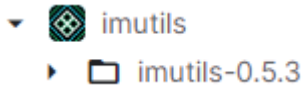


Figura A-28. Librerías y dependencias adicionales

```
eps = 1e-15

import imutils

def resize_keep_ratio(img, longest_size=LONGEST_EDGE):
    height, width = img.shape[:2]
    if np.maximum(height, width) <= longest_size: # no need to resize
        return img

    if height >= width:
        resized_img = imutils.resize(img, height=longest_size)
    else:
        resized_img = imutils.resize(img, width=longest_size)
    return resized_img

def load_torch_image(fname):
    img = cv2.imread(fname)
    img = resize_keep_ratio(img)
    img = cv2.resize(img, (img.shape[1]//8*8, img.shape[0]//8*8))  # input size should
    img = K.image_to_tensor(img, False).float() /255.
    img = K.color.bgr_to_rgb(img)
    return img
```

```python
def QuaternionFromMatrix(matrix):
    '''Transform a rotation matrix into a quaternion.'''

    M = np.array(matrix, dtype=np.float64, copy=False)[:4, :4]
    m00 = M[0, 0]
    m01 = M[0, 1]
    m02 = M[0, 2]
    m10 = M[1, 0]
    m11 = M[1, 1]
    m12 = M[1, 2]
    m20 = M[2, 0]
    m21 = M[2, 1]
    m22 = M[2, 2]

    K = np.array([[m00 - m11 - m22, 0.0, 0.0, 0.0],
                  [m01 + m10, m11 - m00 - m22, 0.0, 0.0],
                  [m02 + m20, m12 + m21, m22 - m00 - m11, 0.0],
                  [m21 - m12, m02 - m20, m10 - m01, m00 + m11 + m22]])
    K /= 3.0

    # The quaternion is the eigenvector of K that corresponds to the largest eigenvalu
    w, V = np.linalg.eigh(K)
    q = V[[3, 0, 1, 2], np.argmax(w)]

    if q[0] < 0:
        np.negative(q, q)

    return q

def ComputeErrorForOneExample(q_gt, T_gt, q, T, scale):
    '''Compute the error metric for a single example. The function returns two errors, over rotation an

    q_gt_norm = q_gt / (np.linalg.norm(q_gt) + eps)
    q_norm = q / (np.linalg.norm(q) + eps)

    loss_q = np.maximum(eps, (1.0 - np.sum(q_norm * q_gt_norm)**2))
    err_q = np.arccos(1 - 2 * loss_q)

    # Apply the scaling factor for this scene.
    T_gt_scaled = T_gt * scale
    T_scaled = T * np.linalg.norm(T_gt) * scale / (np.linalg.norm(T) + eps)

    err_t = min(np.linalg.norm(T_gt_scaled - T_scaled), np.linalg.norm(T_gt_scaled + T_scaled))

    return err_q * 180 / np.pi, err_t


def ComputeMaa(err_q, err_t, thresholds_q, thresholds_t):
    '''Compute the mean Average Accuracy at different tresholds, for one scene.'''

    assert len(err_q) == len(err_t)

    acc, acc_q, acc_t = [], [], []
    for th_q, th_t in zip(thresholds_q, thresholds_t):
        acc += [(np.bitwise_and(np.array(err_q) < th_q, np.array(err_t) < th_t)).sum() / len(err_q)]
        acc_q += [(np.array(err_q) < th_q).sum() / len(err_q)]
        acc_t += [(np.array(err_t) < th_t).sum() / len(err_t)]
    return np.mean(acc), np.array(acc), np.array(acc_q), np.array(acc_t)
```

```python
def ComputeFundamentalMatrix(K1, K2, R1, R2, T1, T2):
    '''Compute the fundamental matrix, given intrinsics and extrinsics for tw
    dR = np.dot(R2, R1.T)
    dT = (T2 - np.dot(dR, T1)).flatten()
    A = np.dot(K1, np.dot(dR.T, dT))
    C = np.array([[0, -A[2], A[1]], [A[2], 0, -A[0]], [-A[1], A[0], 0]])
    return np.matmul(np.linalg.inv(K2).T, np.matmul(dR, np.matmul(K1.T, C)))


def DecomposeFundamentalMatrixWithIntrinsics(F, K1, K2):
    '''Decompose the fundamental matrix into R and T, given the intrinsics.''

    # This fundamentally reimplements this function: https://github.com/openc
    # This is a pre-requisite of OpenCV's recoverPose: https://github.com/ope
    # Instead of the cheirality check with correspondences, we keep and evalu
    # Note that our metric does not care about the sign of the translation ve
    E = np.matmul(K2.T, np.matmul(F, K1))

    U, S, Vh = np.linalg.svd(E)
    if np.linalg.det(U) < 0:
        U *= -1
    if np.linalg.det(Vh) < 0:
        Vh *= -1

    W = np.array([[0, 1, 0], [-1, 0, 0], [0, 0, 1]])
    R_a = np.matmul(U, np.matmul(W, Vh))
    R_b = np.matmul(U, np.matmul(W.T, Vh))
    T = U[:, -1]

    return R_a, R_b, T
```

```python
def EvaluateSubmission(df):
    thresholds_q = np.linspace(1, 10, 10)
    thresholds_t = np.geomspace(0.2, 5, 10)

    scenes = df['scene'].unique()
    errors_dict_q = {scene: {} for scene in scenes}
    errors_dict_t = {scene: {} for scene in scenes}

    for i, row in df.iterrows():
        F_predicted    = row.fundamental_matrix
        scene          = row.scene
        scaling_factor = row.scaling_factor
        pair = row.image_id_1 + '-' + row.image_id_2

        K1, R1_gt, T1_gt = row.K1, row.R1, row.T1
        K2, R2_gt, T2_gt = row.K2, row.R2, row.T2

        R_pred_a, R_pred_b, T_pred = DecomposeFundamentalMatrixWithIntrinsics(F_predicted, K1, K2)
        q_pred_a = QuaternionFromMatrix(R_pred_a)
        q_pred_b = QuaternionFromMatrix(R_pred_b)

        dR_gt = np.dot(R2_gt, R1_gt.T)
        dT_gt = (T2_gt - np.dot(dR_gt, T1_gt)).flatten()
        q_gt = QuaternionFromMatrix(dR_gt)
        q_gt = q_gt / (np.linalg.norm(q_gt) + eps)

        err_q_a, err_t_a = ComputeErrorForOneExample(q_gt, dT_gt, q_pred_a, T_pred, scaling_factor)
        err_q_b, err_t_b = ComputeErrorForOneExample(q_gt, dT_gt, q_pred_b, T_pred, scaling_factor)
        assert err_t_a == err_t_b
        errors_dict_q[scene][pair] = min(err_q_a, err_q_b)
        errors_dict_t[scene][pair] = err_t_a

    maa_per_scene = {}

    for scene in scenes:
        maa_per_scene[scene], _, _, _ = ComputeMaa(list(errors_dict_q[scene].values()), list(errors_dict_t[scene].values()), thresholds_q, thresholds_t)

    return np.mean(list(maa_per_scene.values())), maa_per_scene, errors_dict_q, errors_dict_t
```

Figura A-29. Funciones adicionales

```
scaling_df = pd.read_csv(f'{dataset_path}/train/scaling_factors.csv')
scaling_df['scaling_factor'] = scaling_df['scaling_factor'].astype(float)
scaling_df
```

Figura A-30. Lectura de los factores de escalado

```
data_df = pd.DataFrame()
max_num_pairs = 50

for scene in scaling_df['scene'].values:
    pair_df = pd.read_csv(f'{dataset_path}/train/{scene}/pair_covisibility.csv')
    pair_df['scene'] = scene
    pair_df = pair_df[pair_df['covisibility']>0.1].copy()
    pair_df.reset_index(drop=True, inplace=True)

    if len(pair_df) > max_num_pairs:
        pair_df = pair_df.sample(n=max_num_pairs, random_state=42)
        pair_df.reset_index(drop=True, inplace=True)

    pair_df['fundamental_matrix'] = pair_df['fundamental_matrix'].apply( lambda x: np.array([float(v) for v in x.split(' ')]).reshape([3, 3]) )
    pair_df['image_id_1'] = pair_df['pair'].apply( lambda x: x.split('-')[0] )
    pair_df['image_id_2'] = pair_df['pair'].apply( lambda x: x.split('-')[1] )
    pair_df.drop( 'pair', axis=1, inplace=True )

    #calibration
    calib_df = pd.read_csv(f'{dataset_path}/train/{scene}/calibration.csv')
    calib_df['camera_intrinsics'] = calib_df['camera_intrinsics'].apply( lambda x: np.array([float(v) for v in x.split(' ')]).reshape([3, 3]) )
    calib_df['rotation_matrix'] = calib_df['rotation_matrix'].apply( lambda x: np.array([float(v) for v in x.split(' ')]).reshape([3, 3]) )
    calib_df['translation_vector'] = calib_df['translation_vector'].apply( lambda x: np.array([float(v) for v in x.split(' ')]).reshape([3, 1]) )

    pair_df = pd.merge( pair_df, calib_df.rename( columns={'image_id':'image_id_1', 'camera_intrinsics':'K1', 'rotation_matrix':'R1', 'translation_vector':'T1'} ), on=['image_id_1'], how='left')
    pair_df = pd.merge( pair_df, calib_df.rename( columns={'image_id':'image_id_2', 'camera_intrinsics':'K2', 'rotation_matrix':'R2', 'translation_vector':'T2'} ), on=['image_id_2'], how='left')

    data_df = pd.concat( [data_df, pair_df], axis=0 )
    data_df.reset_index(drop=True, inplace=True)

    del pair_df, calib_df

data_df = pd.merge( data_df, scaling_df, on=['scene'], how='left')
data_df.head()
```

Figura A-31. Creación del dataframe necesario para calcular el mAA

```
maa, maa_per_scene, errors_dict_q, errors_dict_t = EvaluateSubmission(data_df)

for scene, cur_maa in maa_per_scene.items():
    print(f'Scene:{scene:25s} ({len(errors_dict_q[scene])} pairs), mAA={cur_maa:.05f}')

print()
print(f'Full dataset: mAA={maa:.05f}')
```

Figura A-32. Presentación de los resultados obtenidas para todos los monumentos

REFERENCIAS

[1] SaS, Software de Analítica y Soluciones, <<*Inteligencia Artificial. Qué es la IA y por qué importa*>>, [En línea]. Available: https://www.sas.com/es_cl/insights/analytics/what-is-artificial-intelligence.html [Último Acceso: 26 agosto 2023].

[2] Satya Ramaswamy, <<*How Companies Are Already Using AI*>>, 14 abril 2017. [En línea]. Available: https://hbr.org/2017/04/how-companies-are-already-using-ai.

[3] Mario Gavira, *<<How Netflix uses AI and Data to conquer the world>>*, 2 julio 2018. [En línea]. Available: https://www.linkedin.com/pulse/how-netflix-uses-ai-data-conquer-world-mario-gavira/.

[4] Kellison Ferreira, <<*Tipos de inteligencia Artificial: conoce cuáles existen y cómo usarlos*>>, 20 julio 2021. [En línea]. Available: https://rockcontent.com/es/blog/tipos-de-inteligencia-artificial/#:~:text=Artificial%20Narrow%20Intelligence%20(ANI),M%C3%A1quinas%20reactivas.

[5] Redacción Futuro Eléctrico, <<*Tipos de Inteligencia Artificial | Débil, general y súper-inteligencia*>>, [En línea]. Available: https://futuroelectrico.com/tipos-de-inteligencia-artificial/#:~:text=Inteligencia%20Artificial%20D%C3%A9bil%20o%20Narrow,-Este%20tipo%20de&text=Acciones%20como%20la%20visi%C3%B3n%20computarizada,conducci%C3%B3n%20son%20Inteligencia%20Artificial%20Narrow. [Último Acceso 21 mayo 2023].

[6] Alexis Bahamondes, *<<Fabrican Robot Humanoide que imita a la perfección las expresiones humanas>>*, 27 agosto 2022.

[7] Gregory Piatetsky en KDnuggets, <<*Artificial General Intelligence (AGI) in less than 50 years*>>, 5 enero 2018. [En línea]. Available: https://www.kdnuggets.com/2018/01/poll-agi-50-years.html.

[8] Daniel Faggella, <<*When Will We Reach the Singularity? - A Timeline Consensus from AI Researchers*>>, [En línea]. Última actualización 18 marzo 2019. Available: https://emerj.com/ai-future-outlook/when-will-we-reach-the-singularity-a-timeline-consensus-from-ai-researchers/.

[9] Jaime Juan y Laura Cantero, <<*Los tipos de Inteligencia Artificial que deberías conocer*>>, 13 abril 2021, [En línea]. Available: https://iasolver.es/los-tipos-de-inteligencia-artificial-que-deberias-conocer/.

[10] Wikipedia, <<*Deep Blue versus Garri Kasparov*>>, 6 marzo 2023, [En línea]. Available: https://es.wikipedia.org/wiki/Deep_Blue_versus_Garri_Kasparov.

[11] UNIR, la universidad en internet, << *¿Qué es un sistema experto? Usos y aplicaciones en Inteligencia Artificial*>>, 29 marzo 2022 [En línea]. Available: https://www.unir.net/ingenieria/revista/sistema-experto/.

[12] Ceupe, *<< ¿Qué es un Sistema Experto? Definición, tipos y aplicaciones>>*, [En línea]. Available: https://www.ceupe.com/blog/sistema-experto.html?dt=1677163161350. [Último Acceso 26 julio 2023].

[13] DispatchTrack, <<*Robótica e Inteligencia Artificial, similitudes y diferencias*>>, [En línea]. Available: https://www.beetrack.com/es/blog/rob%C3%B3tica-e-inteligencia-artificial-similitudes-y-diferencias. [Último Acceso 27 julio 2023].

[14] EuroInnova, International Online Education, <<*Aprende las ramas de la inteligencia artificial*>>, [En línea]. Available: https://www.euroinnova.edu.es/blog/ramas-de-la-inteligencia-artificial. [Último Acceso 13 mayo 2023].

[15] Fernando Sancho Caparrini, <<*Introducción al Aprendizaje Automático*>>, [En línea]. Available: http://www.cs.us.es/~fsancho/?e=75. [Última modificación 23 septiembre 2017].

[16] Jose Martinez Heras, <<*Las 7 Fases del Proceso de Machine Learning*>>, 19 septiembre 2020, [En línea]. Available: https://www.iartificial.net/fases-del-proceso-de-machine-learning/.

[17] Velogig, <<*¿Qué es el Machine Learning y cómo es su proceso?*>>, 28 septiembre 2018, [En línea]. Available: https://velogig.com/blog/que-es-el-machine-learning-y-como-es-su-proceso/. [Último Acceso 24 abril 2023].

[18] CursoAula21, <<*Visión Artificial: tolo lo que necesitas saber*>>, [En línea]. Available: https://www.cursosaula21.com/que-es-la-vision-artificial/. [Último Acceso 10 julio 2023].

[19] Wikipedia, <<*Visión Artificial*>>, 22 julio 2023, [En línea]. Available: https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial. [Último Acceso 25 julio 2023].

[20] Leo Gamboa Uribe, <<*Computer Vision VS Human Vision: This is how computers see*>>, 21 junio 2021, [En línea]. Available: https://medium.com/be-tech-with-santander/computer-vision-vs-human-vision-this-is-how-computers-see-e2f4a5ed61dc. [Último Acceso 13 abril 2023].

[21] J. Miguel Marin, <<*Introducción a las redes neuronales aplicadas*>>, [En línea]. Available: https://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/DM/tema3dm.pdf. [Último acceso 15 abril 2023].

[22] Xeridia, <<*Redes Neuronales artificiales: Qué son y cómo se entrenan*>>, 16 septiembre 2019, [En línea]. Available: https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i. [Último Acceso 15 abril 2023].

[23] Diego Calvo, <<*Función de activación - Redes neuronales*>>, 7 diciembre 2018, [En línea]. Available: https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/. [Último Acceso 18 abril 2023].

[24] Jaime Durán, <<*Todo lo que Necesitas Saber sobre el Descenso del Gradiente Aplicado a Redes Neuronales*>>, 4 septiembre 2019, [En línea]. Available: https://medium.com/metadatos/todo-lo-que-necesitas-saber-sobre-el-descenso-del-gradiente-aplicado-a-redes-neuronales-19bdbb706a78.

[25] Dot CSV, <<*¿Qué es una Red Neuronal? Parte 3.5: Las Matemáticas de Backpropagation | DotCSV*>>, 14 octubre 2018, [Vídeo en línea]. Available: https://www.youtube.com/watch?v=M5QHwkkHgAA&t=780s.

[26] Dot CSV, << ¿Qué es el Descenso del Gradiente? Algoritmo de Inteligencia Artificial | DotCSV>>, 4 febrero2018, [Vídeo en línea]. Available: https://www.youtube.com/watch?v=A6FiCDoz8_4&t=539s.

[27] Data Science Team, <<*Comprensión de los fundamentos del descenso gradual*>>, 13 abril 2020, [En línea]. Available: https://datascience.eu/es/aprendizaje-automatico/descenso-gradual/. [Último Acceso 4 mayo 2023].

[28] Sebastian Ruder, *<<An overview of gradient descent optimization algorithms>>*, 19 de enero 2016, [En línea]. Available: https://www.ruder.io/optimizing-gradient-descent/. [Último Acceso 30 julio 2023].

[29] Diego Calvo, *<<Clasificación de redes neuronales artificiales>>*, 13 julio 2017, [En línea]. Available: https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/. [Último Acceso 21 abril 2023].

[30] Na8, *<< ¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador>>*, 29 noviembre 2018, [En línea]. Available: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/#:~:text=Si%20tenemos%20una%20imagen%20con,es%20nuestra%20capa%20de%20entrada.

[31] A. Oliva y A. Torralba, *<<Modeling the shape of the scene: a holistic representation of the spatial envelope>>*. Int. J. Comput. 145-175. Publicado en 2001.

[32] S. Bianco, D. Mazzini, D. Pau y R. Schettini, *<<Local detectors and compact descriptors for visual search: a quantitative comparison>>*. Digital Signal Process. 1-13. Publicado en 2015.

[33] K. Tuytelaars, T. y Mikolajczyk, *<<Local invariant feature detectors: a survey>>. Found. Trends Comput*. 177-280. Publicado en 2007.

[34] Deepanshu Tyagi, <<Introduction To Feature Detection and Matching>>, Data Breach, 3 enero 2019, [En línea]. Available: https://medium.com/data-breach/introduction-to-feature-detection-and-matching-65e27179885d. [Último Acceso 28 mayo 2023].

[35] Sanjaysdev0901, *<<Feature Detection and Matching with opencv python>>*, 3 enero 2023, [En línea]. Available: https://www.geeksforgeeks.org/feature-detection-and-matching-with-opencv-python/. [Último Acceso 12 julio 2023].

[36] Sanchez-Lengeling, et al., *<<"A Gentle Introduction to Graph Neural Networks">>*, Distill, 2021. [En línea]. Available: https://distill.pub/2021/gnn-intro/. [Último Acceso 2 junio 2023].

[37] Rick Merritt, *<< ¿Qué son las Graph Neural Networks?>>*, 31 octubre 2022, [En línea]. Available: https://la.blogs.nvidia.com/2022/10/31/que-son-las-graph-neural-networks/. [Último Acceso 4 junio 2023].

[38] Amrita Pathak, *<<Redes neurales gráficas explicadas en 5 minutos>>*, *GeekFlare*, 24 enero 2023, [En línea]. Available: https://geekflare.com/es/graph-neural-networks/. [Último Acceso 1 agosto 2023].

[39] Thomas Kipf y Max Welling, *<<Graph Neural Networks: A Review of Methods and Applications>>*, Revista arXiv año 2020, [En línea]. Available: https://arxiv.org/ftp/arxiv/papers/1812/1812.08434.pdf. [Último Acceso 3 agosto 2023].

[40] Saba Dadsetan, David Pichler, David Wilson, Naira Hovakimyan y Jennifer Hobbs, *<<Superpixels and Graph Convolutional Neural Networks for Efficient Detection of Nutrient Deficiency Stress from Aerial Imagery>>,* CVPR 2021, [En línea]. Available: https://openaccess.thecvf.com/content/CVPR2021W/AgriVision/papers/Dadsetan_Superpixels_and_Graph_Convolutional_Neural_Networks_for_Efficient_Detection_of_CVPRW_2021_paper.pdf. [Último Acceso 2 agosto 2023].

[41] @article{sun2021loftr,
 title={{LoFTR}: Detector-Free Local Feature Matching with Transformers},
 author={Sun, Jiaming and Shen, Zehong and Wang, Yuang and Bao, Hujun and Zhou, Xiaowei},
 journal={CVPR},
 year={2021}
} [En línea]. Available: https://zju3dv.github.io/loftr/ y https://arxiv.org/pdf/2104.00680.pdf.

[42] inproceedings{sarlin20superglue,
 author = {Paul-Edouard Sarlin and Daniel DeTone and Tomasz Malisiewicz and Andrew Rabinovich},
 title = {{SuperGlue}: Learning Feature Matching with Graph Neural Networks},
 booktitle = {CVPR},
 year = {2020},
 url = {https://arxiv.org/abs/1911.11763}
} [En línea]. Available: https://arxiv.org/pdf/1911.11763.pdf y
https://github.com/magicleap/SuperGluePretrainedNetwork.

[43] @article{luo2020aslfeat,
 title={ASLFeat: Learning Local Features of Accurate Shape and Localization},
 author={Luo, Zixin and Zhou, Lei and Bai, Xuyang and Chen, Hongkai and Zhang, Jiahui and Yao, Yao and Li, Shiwei and Fang, Tian and Quan, Long},
 journal={Computer Vision and Pattern Recognition (CVPR)},
 year={2020}
} [En línea]. Available: https://arxiv.org/pdf/2003.10071.pdf y https://github.com/lzx551402/ASLFeat.

[44] @inproceedings{edstedt2023dkm,
title={{DKM}: Dense Kernelized Feature Matching for Geometry Estimation},
author={Edstedt, Johan and Athanasiadis, Ioannis and Wadenbäck, Mårten and Felsberg, Michael},
booktitle={IEEE Conference on Computer Vision and Pattern Recognition},
year={2023}
} [En línea]. Available: https://arxiv.org/pdf/2202.00667.pdf y https://github.com/Parskatt/DKM.

[45] Deval Shah, *<<Mean Average Precision (mAP) Explained: Everything You Need to Know>>*, v7labs, 7 marzo 2022, [En línea]. Available: https://www.v7labs.com/blog/mean-average-precision

[46] Jonathan Hui, *<<mAP (mean Average Precision) for Object Detection>>*, 7 marzo 2018, [En línea]. Available: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

[47] Profesor de Dibujo, *<<Geometría Proyectiva>>*, [En línea]. Available: https://www.profesordedibujo.com/sistema-diedrico/fundamentos-del-sistema/geometria-proyectiva/

[48] Catetos de la geometría, *<<Origen de la geometría proyectiva: Renacimiento>>*, 14 marzo 2010, [En línea]. Available: http://catetos-catetos.blogspot.com/2010/03/origen-de-la-geometria-proyectiva.html.

[49] Addison Howard, Eduard Trulls, etru1927, Kwang Moo Yi, old-ufo, Sohier Dane, Yuhe Jin. (2022). *<<Image Matching Challenge 2022>>*. Kaggle. [En línea]. Available: https://kaggle.com/competitions/image-matching-challenge-2022.

[50] AMMARALI32, *<<IMC 2022-kornia LoFtr from 0.533 to 0.721>>*. Kaggle. [En línea]. Available: https://www.kaggle.com/code/ammarali32/imc-2022-kornia-loftr-from-0-533-to-0-721.

[51] RIGHT GOOSE, *<<IMC 2022-kornia; Score 0.725>>*. Kaggle. [En línea]. Available: https://www.kaggle.com/code/ilyaryabov/imc-2022-kornia-score-0-725.

[52] REMEK KINAS, *<<Whales feature matching LoFTR - Kornia (Open CV - SURF comparision)>>*. Kaggle. [En línea]. Available: https://www.kaggle.com/competitions/happy-whale-and-dolphin/discussion/310092.

[53] ROBIN SMITS, *<<Tensorflow ASLFeat Inference>>*. Kaggle. [En línea]. Available: https://www.kaggle.com/code/rsmits/tensorflow-aslfeat-inference.

[54] BOBFROMJAPAN, <<*SuperGlue and LoFTR and ASLFeat*>>. Kaggle. [En línea]. Available: https://www.kaggle.com/code/bobfromjapan/superglue-and-loftr-and-aslfeat.

[55] RICKY, <<*SuperGlue baseline*>>. Kaggle. [En línea]. Available: https://www.kaggle.com/code/losveria/superglue-baseline.

[56] YUXIANG HUANG, <<*50th Place's Solution: SuperGlue+LoFTR (lb0.810)*>>. Kaggle. [En línea]. Available: https://www.kaggle.com/code/anonymousyuxiang/50th-place-s-solution-superglue-loftr-lb0-810.

[57] CADAR, <<*Public Baseline DKM - 0.667*>>. Kaggle. [En línea]. Available: https://www.kaggle.com/code/radac98/public-baseline-dkm-0-667.

[58] CHAN KHA VU, <<*LoFTR + SuperGlue + DKM with Inspiration*>>. Kaggle. [En línea]. Available: https://www.kaggle.com/code/chankhavu/loftr-superglue-dkm-with-inspiration.

[59] EDUARD TRULLS, <<*imc2022-training-data*>>. Kaggle. [En línea]. Available: https://www.kaggle.com/code/eduardtrulls/imc2022-training-data?scriptVersionId=92062607.

[60] THE NAM, <<*LoFTR Validation Score* >>. Kaggle. [En línea]. Available: https://www.kaggle.com/code/namgalielei/loftr-validation-score.