Trabajo Fin de Grado

Segmentación multimodal de eventos en partidos de futbol mediante un sistema semisupervisado de redes neuronales.

Multimodal event segmentation in soccer matches using a semi-supervised neural network system.

Autor

David Tardós Pardos

Director

Antonio Miguel Artiaga

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2024

AGRADECIMIENTOS

En primer lugar, quiero transmitir mi más sincero agradecimiento a mi tutor, Antonio, por su ayuda y apoyo en el desarrollo de este TFG. En cada situación en la que algo no salía como debía, me ha aportado su conocimiento y experiencia para conseguir retomar el rumbo.

También quiero agradecer a todos los amigos que me han acompañado a lo largo de la carrera, compartiendo buenos momentos tanto dentro como fuera de la universidad. Mencionar de manera especial a Alejandro, que ha tenido el valor de aguantarme todos estos años.

En tercer lugar, agradecer también al resto de amigos fuera de la universidad que llevan toda la vida estando a mi lado.

Finalmente, me gustaría agradecer a mi familia por acompañarme y apoyarme en todo momento. Destacar de forma especial a mi madre, por estar siempre allí, incluso cuando yo no sabía que necesitaba que estuviera. Es alguien que se preocupa profundamente por mí, a veces hasta el punto de que acabamos chocando. Supongo que nunca cambiará el hecho de que me recuerde que debo ponerme el abrigo, incluso estando en ciudades diferentes o dentro de muchos años. Sin embargo, la quiero tal y como es.

Soy plenamente consciente de que no habría podido recorrer este camino sin vuestro apoyo, por lo que estoy eternamente agradecido.

Segmentación multimodal de eventos en partidos de futbol mediante un sistema semisupervisado de redes neuronales.

RESUMEN

El uso de estadísticas en eventos deportivos ha aumentado drásticamente en los últimos años ya que confieren una ventaja inherente y la capacidad de tomar decisiones basadas en datos objetivos. Uno de los campos emergentes es la obtención de las métricas de forma automatizada utilizando sistemas que implementen redes neuronales.

Este Trabajo de Fin de Grado, aborda el desafío de tratar de implementar un sistema que sea sencillo de mantener,modificar y proseguir con el desarrollo, así como la posibilidad de implementarlo en tiempo real. El objetivo es poder ofrecer un sistema que sea accesible para equipos que dispongan de menos recursos económicos con el propósito de que puedan beneficiarse también de las bondades de las estadísticas. Para ello se investigan métodos lo más optimizados posibles.

Se plantean las siguientes cuestiones: ¿Cómo conseguir datos necesarios para realizar los entrenamientos y que cantidad mínima puede servir? ¿Es posible realizar una implementación de la aplicación en tiempo real? ¿Cómo realizar el ajuste de los modelos? ¿Se puede evitar una perdida de precisión no tolerable para conseguir la velocidad necesaria? Al explorar diferentes técnicas se aspira a abrir diferentes enfoques.

Índice

1. Introducción **1**

1.1. Motivación y Objetivos 1
1.2. Contextualización del Trabajo 2
1.3. Fases del Proyecto y Cronograma 2
1.4. Estructura de la Memoria 3

2. Revisión Bibliográfica **5**

2.1. Redes Neuronales Artificiales 5
2.1.1. Perceptrón y MLP 5
2.1.2. Redes Convolucionales (CNN) 6
2.1.3. Redes Residuales 6
2.1.4. Redes Secuenciales 7
2.1.5. Transformer y Self-Attention 8
2.1.6. BERT 12
2.1.7. VIT 12
2.1.8. Modelos Multimodales 14
2.2. YOLO 17
2.2.1. Funcionamiento 18
2.2.2. Versiones de YOLO y sus Creadores 18
2.2.3. YOLOv8 19
2.2.4. Conclusión 19
2.3. Clustering y reducción de dimensiones 19
2.3.1. Clustering 19
2.3.2. Reducción de Dimensionalidad 20
2.4. Data Augmentation 22
2.4.1. Importancia del Data Augmentation en Imágenes 22
2.4.2. Técnicas Comunes de Data Augmentation en Imágenes 22
2.5. Tracking 24

3. Optimización en la ejecución **27**
3.1. Float32 27
3.1.1. Representación de números en Float32 27
3.1.2. Operaciones en Float32 28
3.2. Float16 29
3.2.1. Representación de números en Float16 29
3.2.2. Limitaciones y Operaciones en Float16 29
3.3. Int8 31
3.3.1. Operaciones en Int8 31
3.3.2. Ejecución de Operaciones y Alineamiento de Vectores 32
3.3.3. Optimización de Operaciones en Hardware 32
3.3.4. Tipos de cuantización 32
3.4. Quantization Aware Training 34

4. Experimentación y Resultados **37**
4.1. Resumen 37
4.1.1. Modificaciones respecto a Roboflow y motivaciones. 38
4.1.2. Dataset 39
4.2. Detección 40
4.2.1. Detección de objetos 40
4.2.2. Detección de keypoints 42
4.2.3. Visualización de Detecciones 43
4.3. Tracking 44
4.4. División equipos 45
4.5. Proyección 48
4.6. Aumento velocidad 49
4.6.1. Evaluación Prestaciones GPU 49
4.6.2. QTS 51
4.6.3. CPU 53

5. Discusión y Líneas Futuras **55**
5.1. Conclusión 55
5.2. Líneas Futuras 56

6. Bibliografía **59**

Lista de Figuras **65**

Lista de Tablas **67**

Apéndices **68**

A. Anexo **71**

A.1. Mecanismo atención Transformers 71
A.2. Técnicas de clustering 72
A.3. Homografía 74

Capítulo 1

Introducción

1.1. Motivación y Objetivos

En los últimos años, se ha visto la importancia que tienen las estadísticas en eventos deportivos, ya que a pesar de no capturar eventos inesperados, son capaces de revelar información que aparentemente puede estar escondida para nuestros ojos.

La importancia de las mismas es extremadamente relevante para los equipos, ya que permite tomar decisiones basadas en argumentos objetivos, lo que puede tanto mejorar la efectividad de las mismas, como justificarlas para evitar conflictos. Se pueden utilizar para el diseño de nuevas estrategias, personalizadas en función de cada rival, teniendo en cuenta las habilidades y el estado físico de cada jugador en ese momento.

Los equipos que son capaces de financiar proyectos para obtención de estadísticas pueden contar con una ventaja competitiva frente al resto. El objetivo es tratar de equiparar el tablero, proporcionando un método para la obtención de las mismas utilizando procedimientos de bajo coste y fácil implementación.

El uso de modelos para la extracción de información visual específica como es nuestro caso, resultan extremadamente costosos de entrenar, además de la dificultad añadida que resulta el encontrar bases de datos correctamente etiquetadas y de dimensión requerida. Para solventar este inconveniente, se puede aprovechar modelos ya entrenados para propósitos más generales cuya licencia lo permita, realizando un pequeño ajuste final de los mismos (*fine-tunning*).

En nuestro caso se utilizarán modelos con licencia *opensource* o de bajo coste para explorar la posibilidad de realizar una aplicación con un bajo retardo temporal que permita la obtención de datos con los que realizar un análisis estadístico.

El objetivo es implementar un sistema que permita ubicar en coordenadas reales sobre un campo de fútbol a los jugadores, árbitros y balón y realizar el desarrollo utilizando la librería proporcionada por los creadores del modelo (*Ultralytics*)[1], para facilitar la modificación, implementación,desarrollo y posterior mantenimiento, además de tratar de realizar una ejecución de la aplicación en tiempo real. Para ello se investigan métodos lo más optimizados posibles que permitan maximizar el uso del tiempo.

1.2. Contextualización del Trabajo

Este trabajo se sitúa en el marco de *Roboflow*[2], que ha desarrollado una implementación para la segmentación de objetos de interés y la utilización de una base de datos de tamaño reducido etiquetada por ellos. Además, se ha utilizado *pytorch* sobre *python*, con librerías de *cuda* de NVIDIA. Por ultimo, el *hardware* utilizado pertenece al grupo de investigación Vivolab de Unizar. En su clúster se han ejecutado y realizado las diferentes pruebas durante el desarrollo.

1.3. Fases del Proyecto y Cronograma

Nuestro proyecto ha sido dividido en seis fases, explicadas a continuación.

- **Fase 1**: Selección del *dataset* que se utilizará para entrenar los modelos. Se tratará de generarlo de manera automática en una primera instancia.
- **Fase 2**: Entrenamiento del modelo para detección de objetos. Se utilizará para la detección de los jugadores, árbitros y balón.
- **Fase 3**: Entrenamiento del modelo para la detección de *keypoints*. Se utilizarán para poder realizar la transformación de coordenadas.
- **Fase 4**: División de equipos. Se aplicarán algoritmos de reducción de dimensiones y *clustering* para poder realizar una separación de equipos.
- **Fase 5**: Transformación de las coordenadas de la imagen original a un mapa del campo en dos dimensiones, para poder situar en coordenadas reales la ubicación de los objetos.

[1] https://ultralytics.com
[2] https://roboflow.com/

– **Fase 6**: Realización del cambio de formato y cuantificación. Se tratará de conseguir mayores prestaciones en términos de velocidad, así como minimizar la pérdida de precisión.

En la figura 1.1 se puede ver el diagrama de Gantt del proyecto.

Figura 1.1: Diagrama de Gantt del Trabajo

1.4. Estructura de la Memoria

El contenido de la memoria esta estructurado en cinco capítulos y un anexo, su organización es la siguiente:

– **Capitulo 1**: ***Introducción.*** Se ha explicado las motivaciones para realizar el proyecto, los objetivos y las fases en las que está dividido.

– **Capitulo 2**: ***Revisión Bibliográfica.*** Se hace una revisión bibliográfica de la tecnología que se va a utilizar así como una explicación de como se ha llegado a la situación actual.

– **Capitulo 3**: ***Optimización en la ejecución.*** Se explica en detalle los diferentes formatos numéricos, así como los motivos para realizar los diferentes tipos de cuantización

– **Capítulo 4**: ***Caso de uso.*** Se explica el ejemplo realizado y su posterior evaluación

- **Capítulo 5**: ***Discusión y líneas futuras.*** Se finaliza con las conclusiones y líneas futuras.

Capítulo 2

Revisión Bibliográfica

2.1. Redes Neuronales Artificiales

2.1.1. Perceptrón y MLP

El perceptrón, desarrollado por Frank Rosenblatt en 1957[1], fue la primera red neuronal artificial. Estaba compuesta por una única capa como se puede ver en la figura 2.1, y su interpretación es la de una función matemática cuya salida es el resultado de aplicar una función no lineal al producto escalar entre los datos de entrada y los pesos de la capa. La frontera de separación que es capaz de crear es de tipo lineal, por lo tanto, solo es capaz de separar clases que puedan ser divididas utilizando una separación lineal. A pesar de su simplicidad, el perceptrón sentó las bases para el desarrollo de modelos de redes neuronales más complejas.

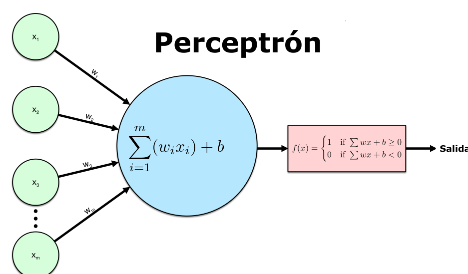


Figura 2.1: Diagrama explicativo del Perceptrón.[1]

Posteriormente se comenzó a trabajar en las redes multicapa, las cuales solventaban la incapacidad del perceptrón de capturar relaciones no lineales. Las redes multicapa, conocidas como perceptrones multicapa (MLP), introducen capas ocultas adicionales entre las capas de entrada y salida. En la década de 1980, el avance del algoritmo de retropropagación (backpropagation)[2] permitió un entrenamiento más eficiente de estas redes.

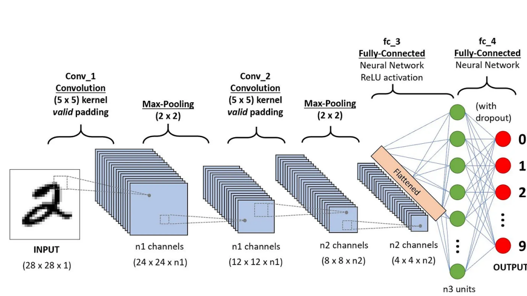
[1]Fuente: https://blog.josemarianoalvarez.com/2018/06/10/el-perceptron-como-neurona-artificial/

2.1.2. Redes Convolucionales (CNN)

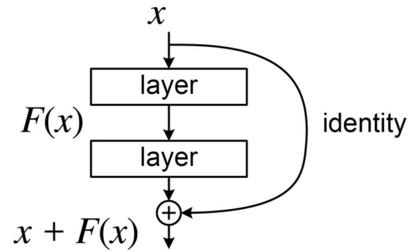
Las redes neuronales convolucionales (CNN)[3] representan una arquitectura específica de redes neuronales diseñada para el procesamiento eficiente de datos con estructura tipo rejilla, como las imágenes. Una capa de convolución realiza una operación lineal sobre la entrada, pero a diferencia del perceptrón, solo tiene en cuenta valores de muestras (píxeles) adyacentes hasta cierto tamaño definido por la máscara de convolución (*kernel*). Cuando se componen varias de estas capas con funciones no lineales intercaladas decimos que se trata de una red convolucional.
Inspiradas en la organización del córtex visual humano, estas redes han demostrado ser altamente efectivas en tareas de reconocimiento y clasificación de imágenes. Además, su aplicación se ha extendido a otros campos como el procesamiento de señales de audio y video [4]. En la figura 2.2a se puede ver el esquema que sigue su arquitectura.

2.1.3. Redes Residuales

El problema de la atenuación de la magnitud del gradiente en redes profundas, usualmente llamado *vanishing gradient problem* ocurre durante el proceso de entrenamiento de redes neuronales profundas mediante el algoritmo de retropropagación. En este proceso, los gradientes (derivadas parciales de la función de pérdida respecto a los pesos) se calculan y se utilizan para actualizar los pesos de la red. En redes muy profundas, los gradientes tienden a disminuir gradualmente a medida que se propagan desde las capas de salida hacia las capas de entrada. Esto significa que las primeras capas (más lejanas a la salida) reciben gradientes muy pequeños, con una estimación muy poco robusta y ruidosa, lo que ralentiza o incluso detiene su aprendizaje. Como resultado, la red puede tener dificultades para aprender representaciones útiles en sus primeras capas, limitando su capacidad para modelar funciones complejas. Esto dificultó el desarrollo de arquitecturas con gran profundidad en la década de 2010. La introducción de las redes residuales (ResNet)[5] mediante conexiones de salto ha sido una solución innovadora que ha permitido superar este desafío, facilitando el entrenamiento de redes mucho más profundas. En la actualidad se usan en la mayoría de arquitecturas. El bloque básico de una red residual se puede ver en la imagen 2.2b.



(a) Arquitectura Red Convolucional.[1]



(b) Diagrama de un bloque básico en red residual.[2]

Figura 2.2: Arquitecturas Redes Residuales y Convolucionales.

2.1.4. Redes Secuenciales

2.1.4.1. Redes Neuronales Recurrentes (RNN)

Debido a la necesidad de procesar y modelar datos secuenciales o con relaciones temporales, surgieron las Redes Neuronales Recurrentes (RNN)[6]. Su sistema de funcionamiento se basa en celdas interconectadas de manera que la salida de una celda se ve afectada por el valor de las anteriores celdas, dotándole de una ”pseudomemoria”. Sin embargo, las RNN tradicionales también sufren del problema de *vanishing gradient*, lo que llevó al desarrollo de variantes como las redes LSTM.

2.1.4.2. LSTM (Long Short-Term Memory)

Las LSTM[7] fueron introducidas para superar las limitaciones de las RNN. En ellas se introdujo celdas de memoria y puertas de control, lo que permite mantener la información a lo largo de secuencias.

Las LSTM cuentan con tres componentes principales. La puerta de olvido controla la cantidad de información proveniente del estado anterior es descartada. La puerta de entrada determina qué nueva información se agrega al estado de la celda. La puerta de salida controla qué parte del estado de la celda se utiliza para generar la salida actual. Gracias a la puerta de olvido se mitigan los problemas de *vanishing gradient* en dependencias largas. En la figura 2.3 se puede ver el diagrama básico.

[1] https://arxiv.org/abs/1512.03385
[2] https://en.wikipedia.org/wiki/Residual-neural-network

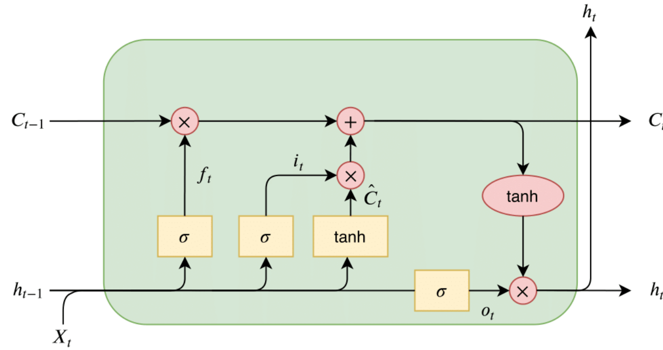


Figura 2.3: Diagrama de una celda LSTM. [2]

2.1.5. Transformer y Self-Attention

El modelo *Transformer*[8], fue presentado en 2017. Este revolucionó el procesamiento de datos secuenciales al eliminar la necesidad de recurrencia. El tranformer completo incluye dos grandes bloques de codificación (*encoder*), y decodificación (*decoder*), sin embargo; en la mayoría de aplicaciones solo se utiliza uno de los dos. Los modelos que se utilizan para representar la señal y no tienen restricciones de causalidad utilizan solo el encoder, como puede ser BERT [9]. Los que necesitan modelar de forma causal los datos, como pueden ser modelos de lenguaje basados en *transformer*, como GPT2, utilizan solo el decoder. Una de las mejoras más notables que aportó fue el mecanismo de *self-attention*, que permite capturar relaciones en contextos mucho mayores. En la figura 2.4 se puede ver el diagrama de bloques general de un Transformer.

Gracias a este mecanismo, se solucionó el problema existente en las redes neuronales recurrentes, ya que había un cuello de botella a la hora de transmitir la información de dependencias entre la información a través de los estados ocultos.

El *encoder* se encarga de generar un vector de *embeddings* $Z = (z_1, \ldots, z_n)$ a partir de un vector de entrada $(x_1, \ldots, x_n)$. El *embedding* permite obtener una representación multidimensional que concentra toda la información de la entrada, proporcionando al modelo una capacidad de abstración semántica avanzada para la comprensión del contexto. Dichos *embedding* se pasan al *decoder* que genera la secuencia de salida.

Un ejemplo interesante de esta capacidad es el siguiente: si tomamos el *embedding* que representa la palabra *rey*, le restamos el *embedding* correspondiente a *hombre* y le

[2] https://thorirmar.com/post/insight-into-lstm/

sumamos el *embedding* de *mujer*, el resultado será un *embedding* cuya representación más cercana es la palabra *reina*. Este ejemplo ilustra cómo los *embeddings* capturan relaciones semánticas entre palabras. Esto se observó por primera vez en los modelos *word2vec* [10].

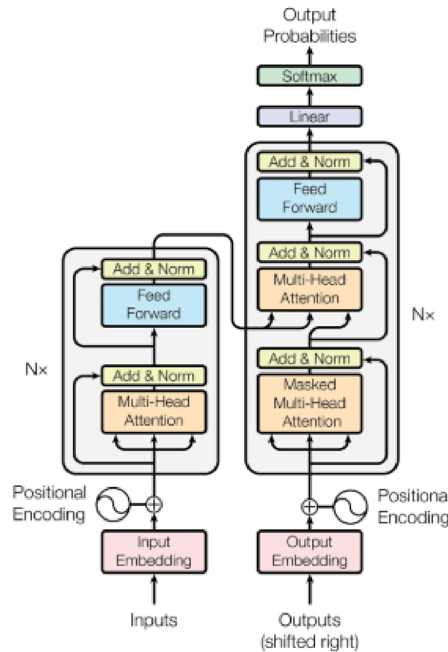


Figura 2.4: Arquitectura Transformer.[3]

2.1.5.1. Encoder

En primer lugar se transforma la secuencia de entrada en la unidad mínima de información, denominada *tokens*. Un *token* puede ser una parte de una palabra, o de una imagen o audio. Cada *token* es asociado con un vector multidimensional. Estos *tokens* son fijos en cada modelo y se pueden interpretar como un diccionario, llamado matriz de *embeddings*. El valor de dicha matriz se ajusta en el entrenamiento y si se realiza de forma exitosa, es capaz de almacenar información semántica de la entrada. Gracias a esta matriz asignamos a cada *token* su *embedding* correspondiente. Se puede ver como en el ejemplo citado anteriormente que dicho espacio multidimensional es capaz de guardar información semántica, así pues *tokens* con significados parecidos se encontraran representados en lugares cercanos dentro de este espacio, (torre, puente, edificio estarán ubicados cerca ya que guardan una relación entre ellos). Una forma de ver dicha distancia es realizar el producto escalar de los vectores, ya que es positivo en caso de que estén alineados y nulo en caso de que sean ortogonales.

[3] `https://arxiv.org/abs/1706.03762`

La capacidad de trabajar en paralelo del *Transformer* nos aporta muchas ventajas, sin embargo; esto implica que, por defecto, el *Transformer* no tiene una noción inherente del orden de los *tokens* en la secuencia. Sin embargo, el orden es crucial para entender el contexto y las relaciones. Por ello, se introducen los *embeddings* posicionales para incorporar información sobre la posición de cada *token* dentro de la secuencia.

Debido a la perdida de información posicional resultante de la capacidad de trabajar en paralelo, se incorporó un bloque adicional de codificación posicional que añade información sobre la posición de cada elemento.
El enfoque más común, introducido en el artículo original de "Attention is All You Need" [8], emplea funciones seno y coseno de diferentes frecuencias. A continuación, se detalla cómo funciona:

Cálculo de los Embeddings Para cada posición *pos* y cada dimensión *i* del *embedding*, se define:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{2.1}$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{2.2}$$

Donde:

- *pos* es la posición del *token* en la secuencia.
- i es el índice de la dimensión del *embedding*.
- d_{model} es la dimensión total del *embedding*.

Aplicación en el Modelo Los *embeddings* posicionales calculados se suman a los *embeddings* de los *tokens* antes de ser ingresados al *Transformer*. Es decir:

$$E_{input} = E_{tokens} + E_{pos} \tag{2.3}$$

Donde E_{tokens} es el *embedding* del *token* y E_{pos} es el *embedding* posicional correspondiente.

2.1.5.2. Mecanismo Multihead-Attention

El mecanismo de **Multihead Attention** es una de las innovaciones claves presentadas en el artículo. Esta permite la captura de multitud de tipos de relaciones y dependencias de los datos de entrada de manera simultanea.

1. Introducción a la Atención Multihead La atención *multihead* extiende el concepto de atención de producto escalar al ejecutar múltiples "cabezas" de atención en paralelo. Cada cabeza se centra en un tipo de relación de los datos, la cual se determina durante el entrenamiento. En la imagen 2.5 se puede ver el diagrama de bloques de las múltiples cabezas de atención.

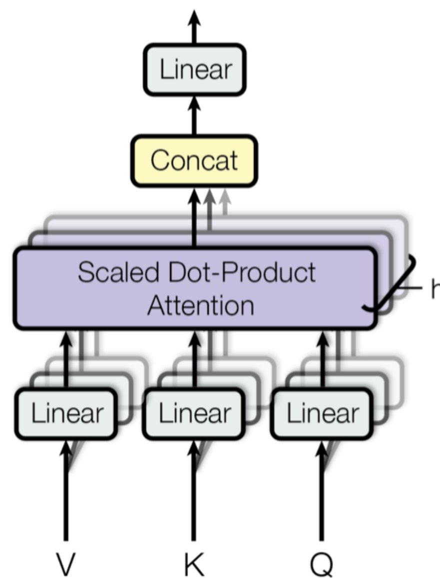


Figura 2.5: Diagrama de Bloques de atención Multicabeza.[4]

2. Componentes de la Atención Multihead El mecanismo de atención *multihead* se compone de los siguientes pasos:

1. **Proyección Lineal**: La entrada se proyecta linealmente en múltiples subespacios de menor dimensión utilizando matrices de pesos diferentes para las consultas (*Queries*), claves (*Keys*) y valores (*Values*).

2. **Atención Escalada de Producto Escalar**: Cada cabeza de atención aplica el mecanismo de atención de producto escalar de manera independiente:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.4}$$

[4] https://arxiv.org/abs/1706.03762

donde Q, K y V son las consultas, claves y valores proyectados, y d_k es la dimensión de las claves y la operación *softmax* se aplica en la dimensión de las columnas.

3. Ventajas de la Atención Multihead Permite al modelo enfocarse en diferentes partes de la secuencia, capturando diversas relaciones contextuales, lo que mejora la representación al aportar diferentes perspectivas y enriquecer el resultado final. Además, facilita el cálculo paralelo, mejorando la eficiencia computacional.

2.1.6. BERT

BERT (Bidirectional Encoder Representations from Transformers) [11], está construido sobre la arquitectura de *Transformers*, enfocándose en la parte del *encoder*. Su arquitectura consiste en múltiples capas apiladas de bloques de *self-attention* y capas *feedforward*. El mecanismo de autoatención permite que *BERT* evalúe la relación de cada palabra con todas las demás en una secuencia, capturando dependencias contextuales en ambas direcciones. Esta estructura bidireccional y profunda permite a BERT generar representaciones contextuales ricas para cada palabra en un texto, lo que mejora su capacidad para comprender y procesar el lenguaje natural en diversas tareas. A través del preentrenamiento con tareas como el modelado de lenguaje enmascarado y la predicción de la siguiente oración, *BERT* genera representaciones contextuales profundas y ricas.

2.1.7. VIT

Introducción

El **Vision Transformer (ViT)** [12] es una arquitectura de redes neuronales basada en el modelo *Transformer*, originalmente diseñado para tareas de procesamiento de lenguaje natural. Introducido por primera vez por investigadores de Google en 2020, el *ViT* supuso una revolución en el campo de la visión por computador, demostrando su capacidad superior a las Redes Neuronales Convolucionales en diversas tareas siempre y cuando se disponga de un volumen suficiente de datos de entrenamiento.

Arquitectura del Vision Transformer

La arquitectura del ViT es bastante similar a la del *Tranformer* para texto, considerando que una vez tienes los datos representados en sus respectivos *tokens* el proceso es bastante parecido. Toma prestada la arquitectura de BERT [11] pero como

se ha mencionado, en lugar de texto se utilizan los *patches* como entrada. Se puede ver su arquitectura en la figura 2.6

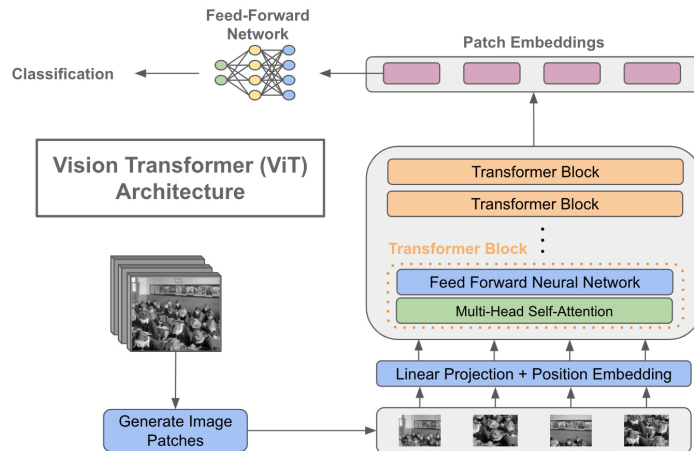


Figura 2.6: Diagrama de Bloques del Vision Transformer.[5]

Una imagen de entrada se divide en pequeños parches cuadrados de tamaño fijo (por ejemplo, 16x16 píxeles). Cada parche se *aplana* (flatten) y se proyecta linealmente a un espacio de características de dimensión D. Los parches proyectados se tratan como *tokens* de entrada, similares a las palabras en procesamiento de lenguaje natural. Además, se añade un *token* de clasificación ([CLS]) al inicio de la secuencia de *tokens*, y se utilizan *embeddings posicionales* para mantener la posición del parche dentro de la imagen.

La secuencia de *tokens* se procesa a través de múltiples capas de *Transformer*, que incluyen mecanismos de *auto-atención* y *redes neuronales feed-forward*. Este mecanismo de atención permite que el modelo capture las relaciones entre los datos de entrada. Finalmente, después de pasar por las capas del *Transformer*, el *embedding* correspondiente al *token* [CLS] se utiliza para realizar la clasificación final mediante una capa completamente conectada.

Variantes y Extensiones

Desde su introducción, se han propuesto múltiples variantes y mejoras sobre el *ViT* original para abordar sus limitaciones y mejorar su rendimiento:

- **DeiT (Data-efficient Image Transformers)**[13]: Introduce técnicas de entrenamiento más eficientes que permiten que los ViT se desempeñen bien incluso con conjuntos de datos más pequeños.

[5] `https://cameronrwolfe.substack.com/p/vision-transformers`

- **Swin Transformer**[14]: Utiliza ventanas deslizantes (shifted windows) para capturar información local y global de manera más eficiente, reduciendo la complejidad computacional.

- **Hybrid ViT** [15][16]: Combina convoluciones con Transformers para aprovechar las fortalezas de ambos enfoques, capturando tanto características locales como globales.

Conclusión

El *Vision Transformer* supuso un avance muy significativo en el campo de la visión por computador. Sus nuevas cualidades le dotan de capacidad para ser utilizado como una herramienta muy potente, sin embargo; el gran nivel de datos que son requeridos para realizar su entrenamiento y los recursos necesarios representan un desafío.

2.1.8. Modelos Multimodales

2.1.8.1. Introducción

En el ámbito de la inteligencia artificial, los **modelos multimodales** han ganado una relevancia significativa debido a su capacidad para integrar y procesar información de múltiples fuentes o modalidades, como texto e imágenes. Este enfoque permite una comprensión más rica y contextualizada de los datos, mejorando el desempeño en tareas complejas que requieren la correlación de diferentes tipos de información. En esta sección, nos centraremos en modelos avanzados como **Grounding DINO**, que combinan encoders de texto e imagen, y en el uso de **embeddings multimodales** para facilitar esta integración.

2.1.8.2. Definición y Características de los Modelos Multimodales

Un **modelo multimodal** es aquel que es capaz de procesar y relacionar datos cuya naturaleza es proveniente de diversas modalidades sensoriales. Las principales características de los modelos multimodales son la capacidad para combinar y relacionar los datos provenientes de diferentes modalidades, mejorando la comprensión de los mismos (integración de información), y la habilidad para representar datos de diferentes modalidades de manera conjunta (representaciones compartidas). Las modalidades más comunes son:

- **Visuales**: Imágenes y videos.

- **Textuales**: Texto escrito o hablado.

– **Auditivas**: Audio, como música o voz.

– **Sensoriales**: Datos provenientes de sensores, como temperatura, movimiento, etc.

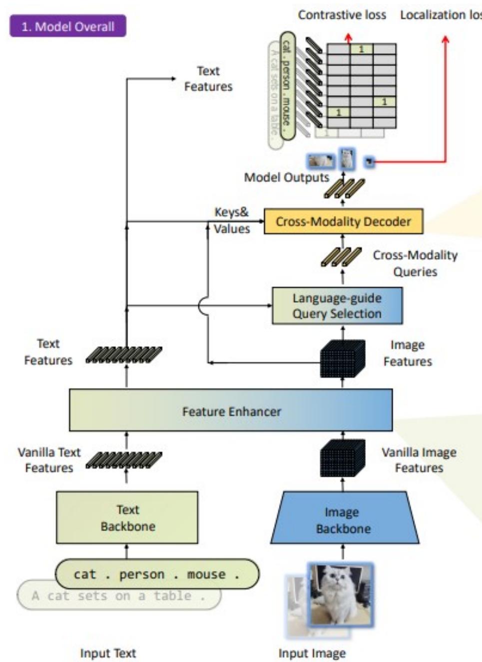


Figura 2.7: Ejemplo Arquitectura Multimodal Imagen y Texto.[6]

2.1.8.3. Arquitecturas Comunes en Modelos Multimodales

Existen diversas arquitecturas dedicadas a la implementación de modelos multimodales, dependiendo de la aplicación deseada. En nuestro caso nos vamos a centrar en las que utilizan una combinación de *encoders* de texto e imagen, para la generación de embeddings multimodales. En la figura 2.7 se puede ver un ejemplo de dicha arquitectura.

Modelos Basados en Encoders Independientes

Estos modelos utilizan *encoders* separados para cada modalidad (un encoder de texto y otro de imagen) y luego combinan sus representaciones en etapas posteriores [17]. El encoder de texto generalmente está basado en modelos de lenguaje como BERT [11] o GPT [18], que procesan y generan representaciones vectoriales del texto. Por otro lado, el encoder de imagen comúnmente utiliza redes neuronales convolucionales (CNN) o Vision Transformers (ViT) para extraer características visuales de las imágenes.

[6]https://arxiv.org/abs/2304.01497

Entre sus ventajas se encuentra la especialización de cada encoder en su modalidad respectiva y la flexibilidad para manejar diferentes tipos de datos. Sin embargo, presentan ciertas desventajas, como la fusión de representaciones que puede ser menos eficiente y la dificultad para capturar interacciones complejas entre modalidades.

Modelos con Fusión Intermedia

En esta arquitectura, las representaciones de cada modalidad se combinan en múltiples niveles del modelo, permitiendo una interacción más profunda y continua entre las modalidades [9].
Entre sus ventajas se encuentra la mejor captura de interacciones entre modalidades y la generación de representaciones más ricas y contextuales. Sin embargo, también presenta ciertas desventajas, como una mayor complejidad en la arquitectura y el entrenamiento, además de requerir una sincronización cuidadosa entre las ramas de las modalidades.

2.1.8.4. Grounding DINO

Los modelos multimodales, como *Grounding DINO* [19][20], nos proporcionan una herramienta avanzada, con la que somos capaces de abordar tareas que antes nos resultaban imposibles gracias a su capacidad de compresión más profunda al integrar datos provenientes de diferentes modalidades.

Nos encontramos ante una elección de arquitectura basada en el uso de dos *encoders* diferentes, a continuación se realiza una explicación sobre los mismos.

Arquitectura de Grounding DINO

La arquitectura de Grounding DINO está compuesta por las siguientes etapas principales:

En el *encoder* de imagen utiliza un *Vision Transformer* (ViT) para procesar la imagen de entrada y extraer características visuales, de esta manera generea los *embeddings* visuales.
En el encoder de texto emplea un modelo de lenguaje como BERT [11] para procesar la descripción textual. Este encoder genera embeddings textuales que capturan el contexto y las características semánticas del texto.

Posteriormente realiza una fusión de modalidades. Las representaciones visuales y textuales se combinan mediante mecanismos de atención cruzada. Esto permite que el modelo enfoque dinámicamente en las partes relevantes de la imagen en función de

la descripción textual proporcionada. Por último realiza una decodificación. Utiliza las representaciones fusionadas para realizar la detección de objetos. Genera *bounding boxes* y clasificaciones que están condicionadas por el texto, permitiendo una detección más precisa y contextualizada.

Las *bouding boxes*, son cajas rectangulares delimitadoras que envuelven al objeto detectado. Para proporcionarlas el modelo da como salida la coordenada de la esquina superior izquierda y la inferior derecha de la caja delimitadora. Dichas coordenadas se expresan en pixeles y permiten definir el área ocupada por el objeto. En la figura 2.8 se pueden ver ejemplos de las mismas.

Figura 2.8: Bounding Boxes de Dino con varias clases introducidas mediante promts de texto

2.2. YOLO

[21][22][23] **YOLO** (*You Only Look Once*) es una familia de modelos de detección de objetos en tiempo real desarrollada en el campo de la visión por computador. A diferencia de otros métodos que dividen la tarea en múltiples etapas (como generar propuestas de regiones y luego clasificarlas), YOLO realiza la detección de objetos en una única pasada de la imagen a través de una red neuronal, lo que le permite ser extremadamente rápido y eficiente.

2.2.1. Funcionamiento

En primer lugar se realiza una división de la Imagen en una Cuadrícula: La imagen de entrada se divide en una cuadrícula de tamaño $S \times S$ (por ejemplo, 7×7 en YOLOv1). Posteriomente tiene lugar la predicción de Cuadros de Delimitación (*Bounding Boxes*). Para cada celda de la cuadrícula, la red predice un número fijo de cuadros de delimitación. Cada cuadro incluye:

- Coordenadas (x, y, w, h): La posición y el tamaño del cuadro.
- Confianza (Confidence Score): Indica la probabilidad de que el cuadro contenga un objeto y la precisión de la predicción.

Además de los cuadros de delimitación, la red predice las probabilidades de clase para cada celda, indicando qué tipo de objeto está presente. Por último se realiza un Posprocesamiento en el que se utiliza *Non-Maximum Suppression* (NMS). Su objetivo es eliminar cuadros redundantes que se superponen, conservando solo los más confiables.

Este enfoque de una sola pasada permite a YOLO ser muy rápido, adecuado para aplicaciones en tiempo real como vehículos autónomos y sistemas de vigilancia.

2.2.2. Versiones de YOLO y sus Creadores

Es importante destacar que no todas las versiones de YOLO han sido creadas por los mismos autores. A lo largo de su evolución, diferentes investigadores y equipos han contribuido al desarrollo de nuevas versiones:

- **YOLOv1 (2015):** Creado por Joseph Redmon, Santosh Divvala, Ross Girshick, y Ali Farhadi. Introdujo el concepto de detección de objetos en una sola pasada.
- **YOLOv2 (2016):** Desarrollado por Joseph Redmon y Ali Farhadi, incorporó *Anchor Boxes*, *Batch Normalization*, y *Dimension Clustering.*
- **YOLOv3 (2018):** Introdujo *Darknet-53* y predicciones multi-escala para mejorar la detección de objetos pequeños.
- **YOLOv4 (2020):** Liderado por Alexey Bochkovskiy, utilizó técnicas como *CSPNet* y *Spatial Pyramid Pooling (SPP).*
- **YOLOv5 (2020):** Creado por Ultralytics, ofreció múltiples tamaños de modelo y migración a PyTorch.

- **YOLOv6 y YOLOv7 (2022):** Incorporaron arquitecturas mejoradas con *RepVGG Blocks* y *ELAN Blocks*.

- **YOLOv8 (2023):** Desarrollado por Ultralytics, ofrece mejoras significativas con un enfoque *anchor-free* y soporte para tareas avanzadas.

2.2.3. YOLOv8

YOLOv8 representa una de las versiones más recientes y avanzadas, con características destacadas. La arquitectura presenta varias mejoras clave, incluyendo un *backbone* optimizado llamado *C2f Module*, una variante mejorada de *CSPDarknet53*, y un diseño *anchor-free* que elimina la necesidad de *anchor boxes*, facilitando la detección de objetos de diferentes tamaños. Además, soporta múltiples tareas como detección de objetos, segmentación semántica y estimación de poses. En términos de rendimiento, logra una *AP (Average Precision)* de hasta 53,9 % en el conjunto de datos MS COCO y ofrece velocidades de 280 FPS en hardware de alto rendimiento. Asimismo, es compatible con múltiples versiones adaptadas a diversas necesidades de hardware, desde dispositivos móviles hasta servidores, optimizando tanto su implementación como su despliegue.

2.2.4. Conclusión

YOLO ha evolucionado significativamente desde su primera versión, adaptándose a las crecientes demandas de aplicaciones modernas. **YOLOv8**, en particular, combina precisión, velocidad y versatilidad, manteniendo a YOLO como una herramienta líder en visión por computador.

2.3. Clustering y reducción de dimensiones

2.3.1. Clustering

Hay algunas aplicaciones en la que la naturaleza de los datos corresponde a una distribución multimodal, esto significa que son datos en los que existe más de un máximo de probabilidad o modo. Debido a la necesidad de modelar estos datos surgió el *clustering* [24], o agrupamiento automático. Se trata de una técnica cuyo objetivo tiene la agrupación de los datos en "*clusters*" o "grupos". Es fundamental en diversas áreas como el análisis de datos, la minería de datos, la visión por computador y el procesamiento del lenguaje natural.

Las distribuciones multimodales suelen indicar que los datos provienen de diferentes subpoblaciones o procesos subyacentes. Por ejemplo, la altura de una población que incluye tanto adultos como niños puede mostrar una distribución bimodal (dos máximos locales). EL *clustering* se utiliza para modelar estas estructuras de datos.
En la figura 2.9 se puede ver un ejemplo de distribución Bimodal

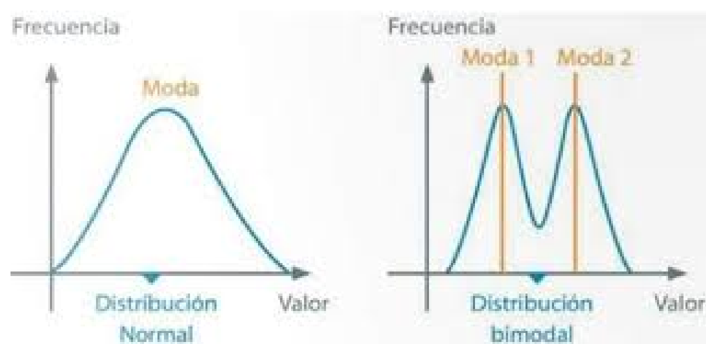


Figura 2.9: Comparativa Distribución Normal frente a Bimodal.[7]

A continuación, se detallan la técnica de *clustering* utilizada en el caso de uso.

2.3.1.1. Clustering Particional: K-Means

K-Means [25] es uno de los algoritmos de clustering más utilizados debido a su simplicidad y eficiencia. Este método divide los datos en k clusters, donde k es un número predefinido, y puede considerarse una versión simplificada del EM para GGM. El funcionamiento del algoritmo comienza seleccionando k centroides iniciales al azar. Luego, cada punto se asigna al cluster con el centroide más cercano, y se recalculan los centroides como la media de los puntos asignados. Este proceso se repite iterativamente hasta cumplir un criterio de parada, como un número máximo de iteraciones o un mínimo número de reasignaciones. Entre sus ventajas destacan su facilidad de implementación y rapidez en conjuntos de datos grandes, mientras que sus desventajas incluyen su sensibilidad a los valores iniciales y a los *outliers*, además de la necesidad de predefinir k.

2.3.2. Reducción de Dimensionalidad

La reducción de dimensionalidad busca simplificar los datos disminuyendo el número de variables, conservando la mayor cantidad posible de información relevante. En nuestro caso, utilizamos esta técnica para poder visualizar los *embeddings* y realizar el *clustering*. A continuación se detallan varias técnicas.

[7] `https://www.cca.org.mx/CCA-cursos/modulo8/modulo8-1.html`

1. **Análisis de Componentes Principales (PCA)** El *PCA* [26] es un método lineal que transforma los datos a un nuevo conjunto de variables no correlacionadas llamadas componentes principales, las cuales capturan la mayor varianza posible en los datos. Su funcionamiento comienza con el centrado de los datos, restando la media de cada variable. A continuación, se calcula la matriz de covarianza para medir la relación entre variables, seguido de la descomposición en valores propios, obteniendo vectores propios y valores propios. Finalmente, los datos se proyectan en los primeros componentes principales para reducir su dimensionalidad.

   Entre las ventajas del *PCA* se encuentran su capacidad para reducir la dimensionalidad de manera eficiente y mejorar la interpretabilidad al eliminar variables redundantes. Sin embargo, presenta limitaciones como capturar únicamente relaciones lineales y la dificultad para interpretar los componentes en términos de las variables originales.

2. **t-SNE (t-Distributed Stochastic Neighbor Embedding)** *El t-SNE* [27] es un método no lineal diseñado para visualizar datos de alta dimensión en espacios de 2 o 3 dimensiones, preservando principalmente las relaciones locales entre los puntos. Su funcionamiento se basa en tres etapas principales: primero, calcula las probabilidades de vecindad que miden la similitud entre puntos en el espacio original. Luego, define un mapa de baja dimensión y asigna posiciones a los puntos en este espacio. Finalmente, utiliza una optimización para minimizar la divergencia de Kullback-Leibler entre las distribuciones de similitud en los espacios original y reducido.

   Entre sus ventajas, el *t-SNE* es excelente para visualizar estructuras complejas y agrupamientos, y es muy efectivo en la preservación de relaciones locales. Sin embargo, tiene varias limitaciones: es computacionalmente intensivo y sus resultados pueden variar significativamente según las inicializaciones y los parámetros elegidos.

3. **UMAP (Uniform Manifold Approximation and Projection)** *UMAP* [28] es un método no lineal para reducción de dimensionalidad que preserva tanto las relaciones locales como las globales en los datos. Es más rápido y escalable que t-SNE, lo que lo convierte en una herramienta poderosa para grandes conjuntos de datos. Su funcionamiento se basa en dos etapas principales: primero, construye un grafo de vecinos cercanos que representa la estructura local de los datos. Luego, optimiza un espacio de baja dimensión utilizando teoría de variedades

para preservar la estructura topológica del grafo original.

Entre sus ventajas, UMAP es rápido y eficiente incluso en grandes conjuntos de datos, mantiene mejor la estructura global que t-SNE y es adecuado tanto para tareas de visualización como para preprocesamiento en modelos predictivos. Sin embargo, presenta limitaciones, como ser sensible a la elección de parámetros (número de vecinos, distancia mínima) y que la interpretación de los resultados puede ser menos intuitiva.

2.4. Data Augmentation

El *data augmentation* [4] o aumento de datos es una técnica utilizada en el campo del aprendizaje automático y la visión por computador para ampliar artificialmente el tamaño y la diversidad de un conjunto de datos existente. Esto se logra aplicando transformaciones y modificaciones a los datos originales, generando nuevas muestras que mantienen las características esenciales para el aprendizaje.

En el contexto de imágenes, el *data augmentation* es especialmente útil cuando se dispone de un conjunto de datos limitado. Al generar variaciones de las imágenes originales, se mejora la capacidad del modelo para generalizar y se reduce el riesgo de sobreajuste (*overfitting*).

2.4.1. Importancia del Data Augmentation en Imágenes

El *Data Augmentation* es crucial en el entrenamiento de modelos de visión por computadora por diversas razones. Mejora la generalización del modelo al exponerlo a una mayor variedad de ejemplos, permitiéndole aprender características que lo preparan mejor para datos no vistos. Además, previene el sobreajuste, reduciendo el riesgo de que el modelo memorice los datos de entrenamiento en lugar de identificar patrones útiles. También, permite la simulación de variaciones del mundo real, como rotaciones, cambios en la iluminación y ruido, lo que aumenta la robustez del modelo. Finalmente, maximiza el uso de los datos disponibles, siendo especialmente valioso en situaciones donde la adquisición de nuevos datos es costosa o impráctica.

2.4.2. Técnicas Comunes de Data Augmentation en Imágenes

A continuación, se detallan las técnicas más utilizadas para el aumento de datos en imágenes [29]. En la figura 2.10 se pueden ver varios ejemplos de manera visual.

1. **Transformaciones Geométricas**

– **Rotación:** Girar la imagen en torno a su centro en un ángulo determinado.

– **Volteo (Flip):** Horizontal (izquierda-derecha) y vertical (arriba-abajo).

– **Traslación:** Mover la imagen en el eje X, Y o ambos.

– **Escalado (Zoom):** Acercar o alejar el contenido de la imagen.

– **Recorte y Relleno (Cropping y Padding):** Eliminar bordes o añadir píxeles alrededor de la imagen.

– **Cizallamiento (Shearing):** Aplicar una transformación que desplaza una parte de la imagen, deformándola en forma de paralelogramo.

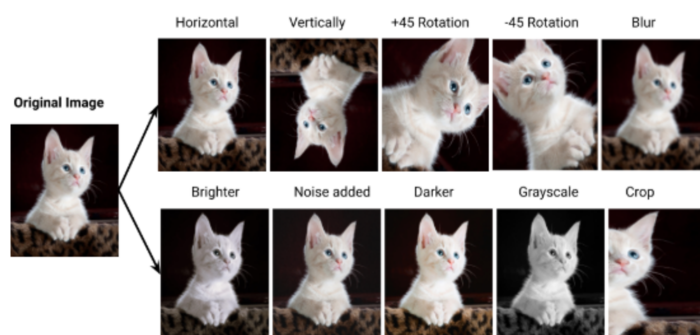


Figura 2.10: Diversas técnicas aplicadas de data augemtation.[8]

2. **Transformaciones de Color y Espacio de Color**

– **Ajuste de Brillo:** Aumentar o disminuir la luminosidad de la imagen.

– **Ajuste de Contraste:** Modificar la diferencia entre las áreas claras y oscuras.

– **Saturación y Tono:** Modificar la intensidad y el matiz de los colores.

– **Jittering de Color:** Aplicar perturbaciones aleatorias a los valores de color.

3. **Inyección de Ruido**

– **Ruido Gaussiano:** Añadir valores aleatorios de distribución normal a los píxeles.

– **Ruido Sal y Pimienta:** Píxeles aleatorios se establecen en valores mínimos o máximos.

– **Ruido Speckle:** Multiplica la imagen por un ruido gaussiano.

4. **Filtros y Convoluciones**

[8] https://ubiai.tools/what-are-the-advantages-anddisadvantages-of-data-augmentation

- **Desenfoque (Blurring):** Suaviza la imagen, reduciendo detalles finos.
- **Enfoque (Sharpening):** Aumenta la nitidez y el contraste de bordes.

5. **Ocultamiento y Eliminación Aleatoria**

   - **Borrado Aleatorio (Random Erasing):** Oculta regiones aleatorias de la imagen.
   - **Cutout:** Similar al borrado aleatorio, pero con regiones predefinidas.
   - **GridMask:** [30] Superpone una cuadrícula que oculta partes de la imagen en un patrón regular.

6. **Técnicas Avanzadas**

   - **Mixup:** [31] Combina dos imágenes y sus etiquetas mediante interpolación lineal.
   - **CutMix:** [32] Combina imágenes reemplazando una región de una imagen con la región correspondiente de otra imagen.
   - **Generative Adversarial Networks (GANs):** [33] Generan nuevas imágenes sintéticas que siguen la distribución de los datos originales.

2.5. Tracking

El *tracking* o seguimiento en aplicaciones para visión por computador, es una tarea que consiste en identificar un objeto como una entidad y seguirlo a través de un desplazamiento temporal (en una secuencia de imágenes).

Pasos del Tracking

El proceso de tracking en visión por computador comprende varias etapas fundamentales. Primero, la detección de objetos, donde se identifican los objetos a seguir mediante métodos de detección, que pueden ser tanto clásicos como basados en redes neuronales. A continuación, se realiza la asociación de objetos, relacionando las detecciones en diferentes *frames* con la identidad de un mismo objeto. Finalmente, en algunas aplicaciones, se incluye una predicción de movimiento para estimar la posición futura del objeto en caso de perder la detección.

Desafíos Comunes en el Tracking

El tracking en visión por computador enfrenta varios desafíos comunes que pueden afectar su precisión y robustez. Entre ellos se encuentran la oclusión, que ocurre cuando

un objeto es parcial o completamente ocultado por otro; la dificultad para diferenciar entre objetos con apariencia similar debido a características visuales compartidas; y los cambios de iluminación y escala, que incluyen variaciones en las condiciones de iluminación o el tamaño del objeto en el cuadro. Además, el seguimiento de objetos con movimiento rápido puede ser impreciso, y es necesario gestionar la entrada y salida de objetos de la escena, lo que implica manejar su aparición y desaparición de manera eficiente.

Algoritmo ByteTrack

ByteTrack [34] es un algoritmo de seguimiento de objetos que se destaca por su eficiencia y precisión, especialmente en escenarios de alta densidad de objetos y movimientos rápidos. Fue presentado como una mejora sobre métodos anteriores como DeepSORT [35], ofreciendo un mejor balance entre velocidad y exactitud.

Principios Fundamentales de ByteTrack

Los principios fundamentales de ByteTrack se centran en mejorar la asociación de detecciones y la capacidad de seguimiento en escenarios complejos. Este enfoque incluye la asociación de detecciones de alta y baja confianza, donde las detecciones de alta confianza se asignan a objetos ya rastreados para garantizar precisión, mientras que las detecciones de baja confianza, en lugar de descartarse, se utilizan para capturar objetos omitidos y mejorar el seguimiento en situaciones complicadas. Además, ByteTrack emplea un mecanismo de asociación mejorado que asigna eficientemente las detecciones a trayectorias existentes y aplica un filtrado robusto para aumentar la fiabilidad del sistema. Por último, utiliza modelos simples de movimiento para predecir la ubicación futura de los objetos, permitiendo un seguimiento más preciso y continuo.

Comparación con Otros Métodos

Gracias a las mejoras aplicadas en *ByteTrack*, es capaz de ofrecer diferencias positivas en términos de precisión y manejo de objetos con detecciones cuya confianza es baja. En comparación con *DeepSORT*, *ByteTrack* ha optimizado el proceso para reducir las pérdidas y aumentar la eficiencia.

Conclusión

El tracking en visión por computador es una tecnología esencial para numerosas aplicaciones modernas, y algoritmos como ByteTrack representan avances significativos en este campo. Al combinar detecciones de alta y baja confianza con técnicas de

asociación y predicción eficientes, ByteTrack ofrece una solución robusta y precisa para el seguimiento de múltiples objetos en tiempo real. Su capacidad para manejar escenarios complejos y su eficiencia computacional lo hacen una herramienta valiosa tanto en investigación como en aplicaciones industriales.

Capítulo 3

Optimización en la ejecución

3.1. Float32

El término **Float32** se refiere a números de punto flotante representados con 32 bits, también conocidos como precisión completa o *full-precision*. Hablando en términos de redes neuronales, los parámetros del modelo (principalmente pesos y *biases*) suelen representarse utilizando *Float32*. Esta representación permite una alta precisión en los cálculos y un amplio rango de valores, desde números muy pequeños hasta muy grandes.

3.1.1. Representación de números en Float32

El formato **Float32**, definido por el estándar IEEE 754, es una representación de números de punto flotante en 32 bits. Esta representación permite almacenar números reales de una amplia gama, incluyendo números muy pequeños y muy grandes, con una precisión significativa.

Un número en formato Float32 se divide en tres partes:

- **Bit de signo (1 bit)**: Indica si el número es positivo (0) o negativo (1).
- **Exponente (8 bits)**: Codifica el exponente, que ajusta la escala del número.
- **Mantisa (23 bits)**: Representa las cifras significativas del número.

La representación se calcula de la siguiente manera:

$$\text{Valor} = (-1)^{\text{signo}} \times 1.\text{mantisa} \times 2^{\text{exponente}-127} \tag{3.1}$$

Donde el exponente tiene un *offset* de 127, lo que permite la representación de exponentes positivos y negativos. Mostrado de manera visual en la figura 3.1

[1] https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

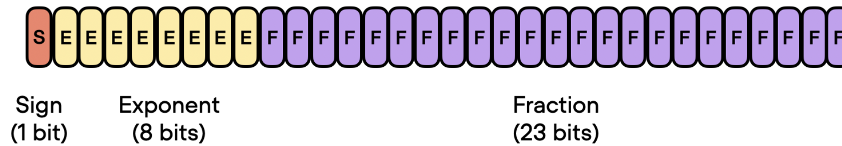


Figura 3.1: Formato Float32.[1]

3.1.2. Operaciones en Float32

Las operaciones aritméticas en punto flotante, como suma, resta, multiplicación y división, se realizan siguiendo algoritmos específicos para manejar la representación de signo, exponente y mantisa.

Por ejemplo, en la multiplicación de números en punto flotante:
Se multiplican las mantisas, se suman los exponentes y se resta el *offset* para corregirlo ya que en caso contrario se acumularía el doble del mismo. Se calcula el signo y posteriormente se normaliza la mantisa para que el bit más significativo sea 1. Por último se redondea la mantisa a 23 bits siguiendo las reglas de redondeo estándar IEEE 754.

En aplicaciones como cálculos gráficos o aprendizaje profundo, puede resultar interesante el realizar cambios de escala para realizar un ajuste del rango dinámico de los valores. En este caso, necesitamos seguir unos pasos para poder realizar la aritmética de suma en formato Float32 de dos números con escalas diferentes. En primer lugar, debemos de alinear sus exponentes, para ello se desplaza la mantisa hasta igualar el número del exponente menor con el del mayor. Una vez alineados realizamos la suma de mantisas y normalizamos si fuera necesario. Por último se realiza el redondeo en caso de ser necesario. Este proceso es más complejo que en caso de realizar las sumas en formato *Int* que será comentado posteriormente.

Las operaciones en punto flotante son complejas debido a la necesidad de manejar el exponente y la mantisa, así como las consideraciones de redondeo y normalización.

Debido al gran número de parámetros en las redes actuales (a menudo miles de millones), el uso de *Float32* implica un consumo significativo de memoria y recursos computacionales. Esto limita la capacidad de ejecutar estos modelos y hace necesario buscar métodos para reducir su tamaño sin comprometer demasiado la precisión.

3.2. Float16

Float16, o punto flotante de 16 bits, es una representación numérica que utiliza la mitad de bits que *Float32*, reduciendo así la memoria necesaria para almacenar los parámetros del modelo.

3.2.1. Representación de números en Float16

El formato **Float16** es una representación de punto flotante en 16 bits, también definida por el estándar IEEE 754. Es similar a *Float32* pero con menor precisión y rango.

La estructura de Float16 es:

- **Bit de signo (1 bit)**: Indica si el número es positivo o negativo.
- **Exponente (5 bits)**: Codifica el exponente con un sesgo de 15.
- **Mantisa (10 bits)**: Contiene las cifras significativas.

Tal y como se muestra en la Figura 3.2

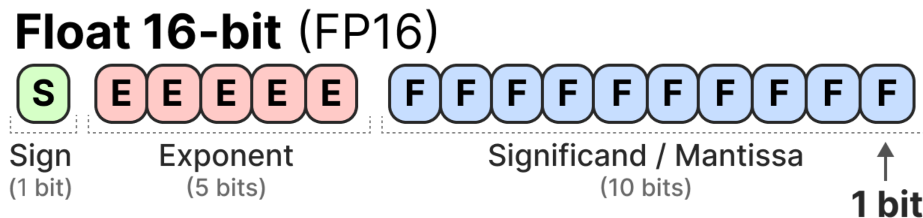


Figura 3.2: Formato Float16.[2]

El valor se calcula como:

$$\text{Valor} = (-1)^{\text{signo}} \times 1.\text{mantisa} \times 2^{\text{exponente}-15} \tag{3.2}$$

Al igual que en *Float32*, tenemos el *offset* en el exponente que nos permite representar los valores positivos y negativos del mismo.

3.2.2. Limitaciones y Operaciones en Float16

Debido al menor número de bits, *Float16* tiene menor precisión y un rango más limitado. Esto puede conducir a problemas de subdesbordamiento y sobreflujo, especialmente en cálculos que involucran números muy pequeños o muy grandes.

[2]https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Las operaciones aritméticas en *Float16* siguen principios similares a las de Float32, pero deben manejar con cuidado la precisión limitada y los casos especiales, como números denormalizados, infinitos y *NaN*. En la figura 3.3 se puede ver la diferencia de precisión.

Hay dos tipos principales de representaciones de 16 bits utilizadas en aprendizaje profundo:

- **FP16 (Half-Precision Floating Point)**: Reduce tanto el número de bits para la mantisa como para el exponente en comparación con Float32. Esto disminuye la precisión y el rango de valores que se pueden representar, lo que puede llevar a problemas de subdesbordamiento o sobreflujo en ciertos cálculos.

- **BF16 (Brain Floating Point Format)**: Mantiene el mismo número de bits para el exponente que Float32 pero reduce los bits de la mantisa. Esto preserva un rango amplio de valores mientras reduce la precisión. BF16 es especialmente útil en aplicaciones de aprendizaje profundo donde el rango de valores es amplio pero la precisión extrema no es crítica.

El uso de Float16 permite reducir a la mitad el tamaño del modelo y acelerar los cálculos, ya que las operaciones con números de menor precisión suelen ser más rápidas. Sin embargo, puede introducir errores de redondeo y afectar la estabilidad del entrenamiento, por lo que a veces se combina con técnicas como la normalización o la acumulación de gradientes en mayor precisión.

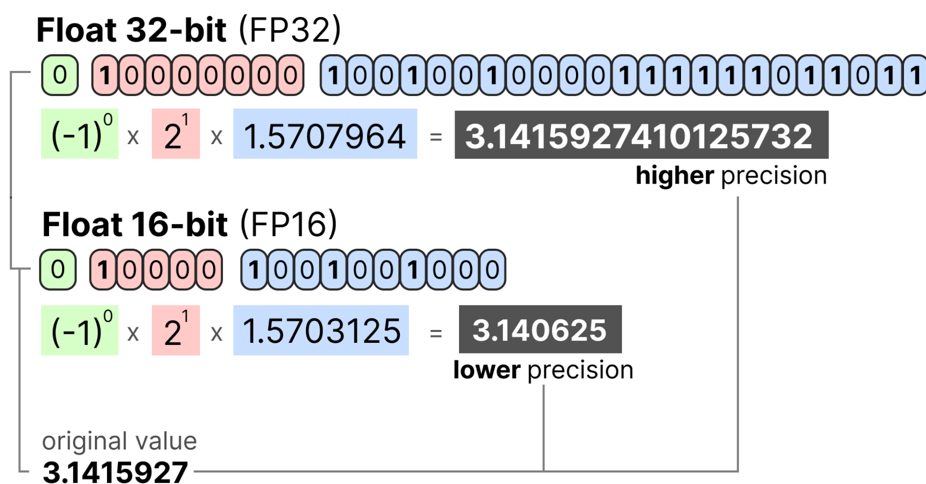


Figura 3.3: Comparativa Precisión Float32 y Float16.[3]

[3] https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

3.3. Int8

La representación **Int8** utiliza números enteros de 8 bits para almacenar los valores, lo que reduce aún más la memoria necesaria. La cuantificación a Int8 implica mapear los valores de punto flotante (pesos y activaciones) a un rango de enteros de 8 bits, lo cual puede introducir errores debido a la pérdida de precisión y rango. Además, el realizar las operaciones en *Int8* permite realizar tiempos de ejecución considerablemente menores. En la figura 3.4 se pueden ver las características de cada formato.

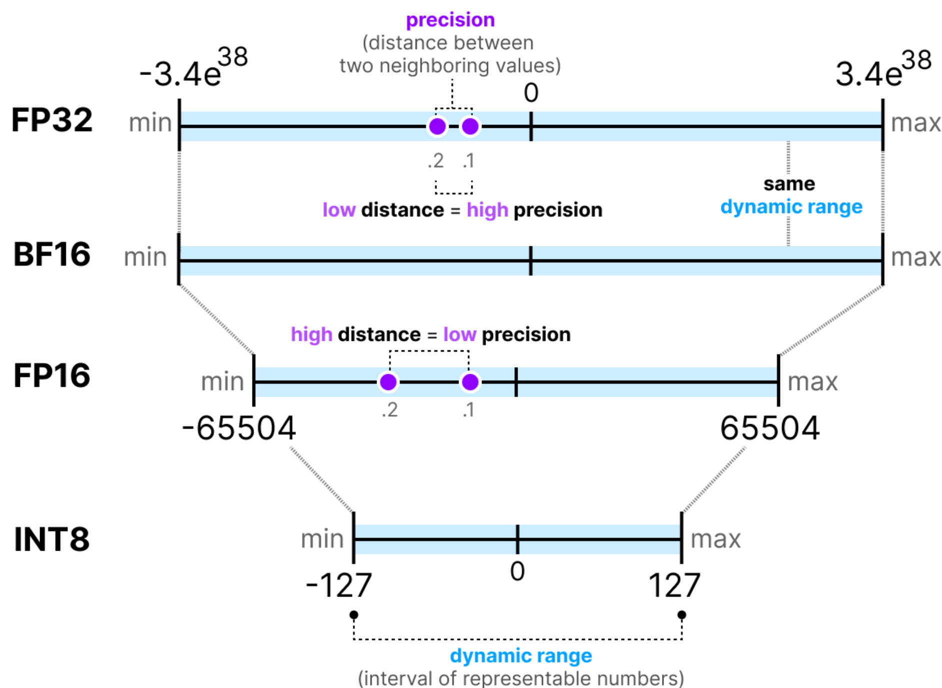


Figura 3.4: Precisión y rángo dinámico de los diferentes formatos. [4]

3.3.1. Operaciones en Int8

Las operaciones aritméticas con enteros son más simples y rápidas que con punto flotante, ya que no requieren manejar exponentes ni normalización. En el caso de que haya que realizar una operación de suma en formato entero y los números a sumar se encuentren en la misma escala, la operación suma es trivial, a diferencia de en formato *float*, que sería necesitar seguir el algoritmo anteriormente explicado para poder realizar la suma.

Sin embargo, al realizar operaciones como multiplicaciones y sumas con valores cuantizados, es crucial manejar adecuadamente el escalado y el desescalado para mantener la precisión del modelo, así como tener en cuenta el rango de valores en

[4] https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

el que nos vamos a mover, ya que al contar con un menor rango dinámico, puede darse el caso que perdamos toda la información.

3.3.2. Ejecución de Operaciones y Alineamiento de Vectores

La multiplicación en coma fija consiste en realizar una serie de sumas de los productos parciales. No ofrece demasiada ventaja respecto a hacer multiplicaciones en *Float*. La ventaja viene a la hora de realizar sumas, ya que lo único que hay que realizar es alinear los vectores en caso de estar en diferente escala y realizar la suma. La desventaja que ocurre es que si excedemos el rango dinámico que somos capaces de realizar nuestro sistema funcionará de forma errónea. Por lo tanto es esencial conocer el rango de valores en el que se va a mover nuestro sistema para evaluar la viabilidad de utilizar un formato u otro.

3.3.2.1. Comparación entre Coma Fija y Punto Flotante

Las operaciones en coma fija son más simples y pueden ser más eficientes en hardware que las operaciones en punto flotante, pero requieren una gestión cuidadosa del escalado y pueden ser más propensas a errores de redondeo y saturación.

Las operaciones en punto flotante permiten un rango dinámico más amplio y son más flexibles, pero son más complejas y requieren más recursos computacionales.

3.3.3. Optimización de Operaciones en Hardware

La optimización de operaciones matemáticas en redes neuronales y aprendizaje profundo se logra mediante el uso de hardware especializado. Las unidades de procesamiento en punto flotante (FPUs) manejan de manera eficiente las complejidades de los exponentes y mantisas, acelerando las operaciones en punto flotante. Por su parte, las unidades SIMD (*Single Instruction, Multiple Data*) y vectoriales permiten la ejecución paralela de operaciones en vectores de datos, mejorando el rendimiento en cálculos matriciales esenciales en redes neuronales. Además, los aceleradores de inteligencia artificial, como las TPUs de Google o los NPUs de otros fabricantes, están diseñados para realizar operaciones cuantizadas, especialmente en formatos como Int8, optimizando las convoluciones y multiplicaciones de matrices de forma altamente paralelizada y gestionando internamente el escalado y alineamiento para maximizar la eficiencia.

3.3.4. Tipos de cuantización

Existen dos métodos principales para realizar este mapeo:

El primero se basa en mapear el rango de valores de punto flotante a un rango simétrico alrededor de cero en el espacio cuantizado. Este método es sencillo y eficiente, pero puede no representar adecuadamente distribuciones de datos que no estén centradas en cero.

En el segundo se mapea los valores mínimos y máximos de los datos de punto flotante al rango completo de Int8, permitiendo una representación más precisa de distribuciones desplazadas. Requiere calcular un punto cero (*zero-point*) para ajustar el desplazamiento. La comparativa de ambas cuantizaciones se puede ver en la figura 3.5

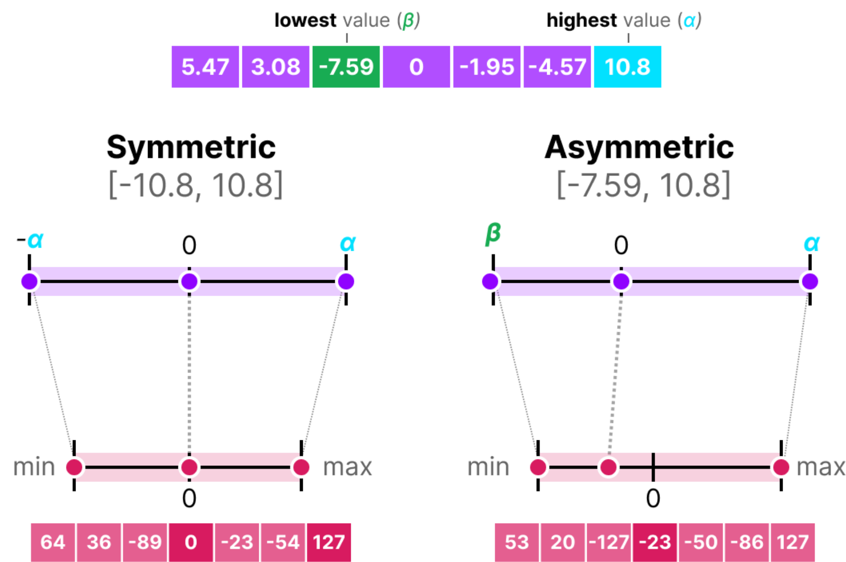


Figura 3.5: Comparación cuantización simétrica y asimétrica.[5]

La cuantización a Int8 es una técnica efectiva para reducir el tamaño del modelo y acelerar la inferencia, especialmente en hardware que está optimizado para operaciones con enteros. Sin embargo, es crucial manejar adecuadamente el rango y la distribución de los datos para minimizar el impacto en la precisión del modelo.

La cuantización se puede realizar de dos maneras:

- **PTQ**: la cuantización se realiza después de entrenar el modelo.

- **QAT**: el entrenamiento se realiza de manera que el modelo sea "consciente" de que va a ser cuantizado.

[5] `https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization`

3.4. Quantization Aware Training

El **Entrenamiento Consciente de Cuantización** (*Quantization Aware Training*, QAT) [36] es una técnica que incorpora los efectos de la cuantización durante el proceso de entrenamiento del modelo. En lugar de cuantizar el modelo después de entrenarlo (lo que puede llevar a una degradación en el rendimiento), el QAT simula la cuantización durante el entrenamiento para que el modelo aprenda a adaptarse a las limitaciones de precisión [6].

Durante el *QAT*, las redes neuronales se entrenan para operar con pesos y activaciones cuantizados, emulando las limitaciones de precisión del *hardware*. Sin embargo, las funciones de cuantización son no diferenciables, lo que impide la propagación estándar de gradientes necesaria para el entrenamiento mediante retropropagación. El Straight-Through Estimator (STE) [37] es un componente fundamental en el *QAT*. El *STE* aborda este desafío al permitir que los gradientes fluyan a través de las operaciones de cuantización reemplazando la función no diferenciable por una función diferenciable aproximada.

Específicamente, mientras que la cuantización se aplica en la propagación hacia adelante (*forward pass*), el *STE* utiliza una función identidad o una aproximación lineal en la retropropagación. Esto permite ajustar los pesos y sesgos del modelo a pesar de la presencia de operaciones no diferenciables, facilitando un entrenamiento efectivo de redes cuantizadas.La cuantización ficticia de pesos y activaciones consiste en cuantizar y luego de-cuantizar estos valores durante el entrenamiento, permitiendo que el modelo experimente los efectos de la cuantización sin perder la capacidad de calcular gradientes con precisión. Además, la optimización consciente de cuantización ajusta los pesos del modelo teniendo en cuenta el error introducido por la cuantización, buscando soluciones que sean robustas a la reducción de precisión.

```
1 import torch
2
3 def STE(x):
4     rounded = torch.round(x) # Redondea al entero mas cercano
5     return x + (rounded - x).detach() # 'detach()' bloquea la retropropagacin del gradiente
```

Listing 3.1: Implementación del Straight-Through Estimator (STE) en PyTorch

[6] https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

De manera visual queda reflejado en el Listing 3.1. La función STE realiza el proceso de cuantificación, en este caso mediante el redondeo del tensor ‘x‘ al entero más cercano utilizando ‘torch.round(x)‘, pero la función de redondeo no es diferenciable. Su derivada es cero en casi todas partes excepto en los puntos discontinuos que no está definida. Por lo tanto, no podemos calcular gradientes a través de ella usando métodos estándar de retropropagación. Para superar este problema, calculamos ‘(rounded - x).detach()‘, donde ‘rounded - x‘ es la diferencia entre el valor redondeado y el original, y ‘.detach()‘ crea un nuevo tensor sin conexión al grafo computacional, evitando que los gradientes fluyan a través de esta operación. Al sumar ‘(rounded - x).detach()‘ a ‘x‘, obtenemos efectivamente ‘rounded‘, pero durante la retropropagación, el gradiente fluye solo a través de ‘x‘ y no a través de ‘rounded‘, ya que ‘(rounded - x).detach()‘ no contribuye al gradiente, lo que significa que en el paso hacia adelante (*forward pass*) la función devuelve el valor redondeado de ‘x‘, útil para operaciones como cuantización o discretización, y en el *backward pass*, el gradiente se calcula como si la función fuera la identidad (‘y = x‘), permitiendo que el gradiente fluya sin interrupciones.

Básicamente consiste en calcular la diferencia entre la señal original y la cuantificada y bloquear el gradiente de este valor a la hora de calcular posteriormente el gradiente. Es decir, tratar este valor de corrección como una constante. La salida final se compone de la suma de la señal original a la que se le suma la diferencia que se ha calculado con el objetivo de que tenga el gradiente que habría tenido si no se hubiera cuantificado aunque a todos los efectos de los módulos posteriores su valor está cuantificado

La ventaja de utilizar QAT frente a PTQ, radica en que al realizar el entrenamiento de manera que los pesos sufran la consecuencia de que van a ser cuantizados, se llegará a un mínimo que puede ser diferente de si se hubiese realizado el entrenamiento de manera usual. Esto es debido a que puede darse el caso, en el que los pesos converjan a un mínimo absoluto, que en caso de tener los pesos en formato *float32* si que estaríamos en la situación de mínimas perdidas, pero, al realizar la cuantización, obtenemos una situación de mayores pérdidas que si los pesos hubiesen convergido a un valor con aparentes mayores perdidas, pero que al cuantizar acaba teniendo unas pérdidas menores. Esto se puede ver de manera gráfica en la figura 3.6

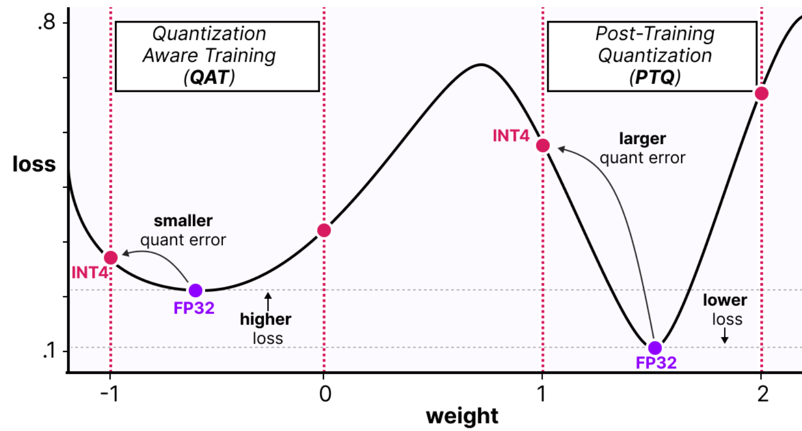


Figura 3.6: Comparativa de las pérdidas del modelo frente a realizar QAT o PTQ.[7]

El principal beneficio del QAT es que resulta en modelos que mantienen una alta precisión incluso después de ser cuantizados a representaciones de menor número de bits (como Int8). Esto es especialmente útil para implementaciones en dispositivos con recursos limitados, donde la eficiencia computacional y el uso de memoria son críticos.

Al integrar la cuantización en el proceso de entrenamiento, el modelo aprende a compensar la pérdida de precisión y puede ofrecer un rendimiento comparable al de modelos entrenados y ejecutados en mayor precisión.

[7]https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Capítulo 4

Experimentación y Resultados

El objetivo de este proyecto es desarrollar un sistema capaz de obtener datos en tiempo real durante eventos deportivos, centrándonos en los partidos de fútbol. La meta principal es registrar las coordenadas de los jugadores, árbitros y balón durante el juego.

El sistema ha de ser robusto frente a cambios en el ángulo y posición de la cámara, así como a cambios en el terreno de juego. Esto se puede abordar mediante la utilización de complejos y avanzados sistemas GPS. Gracias a ellos, se puede realizar el seguimiento de todos los objetos de interés anteriormente mencionados. El problema de este enfoque, es el coste y complejidad asociados. Nuestra solución busca poder ofrecer una alternativa viable con un menor coste y dificultad, permitiendo que equipos con menor poder adquisitivo puedan verse beneficiado de la obtención de dichas métricas.

4.1. Resumen

Para abordar el problema, hemos dividido el sistema en bloques, resolviendo cada uno de manera independiente. El proceso se enfoca en dos desafíos principales:

- **Detección de objetos de interés:** Identificar jugadores, árbitros y el balón.
- **Ubicación precisa en coordenadas reales:** Traducir la posición detectada a un sistema de coordenadas del campo.

Para resolver estos desafíos, utilizamos dos modelos:

- Un modelo especializado en detectar personas y el balón.
- Otro modelo diseñado para identificar puntos clave en el campo.

Además nuestro enfoque soluciona otro problema, y es la necesidad de tener mapeado cada campo donde se quiera implantar el sistema, ya que con el enfoque

tradicional de localización GPS, requiere de la realización de un mapeado del campo en el que se quiera aplicar.
Gracias a nuestra solución ofrecemos un despliegue que no requiere de personalización, solo se debe instalar la cámara en un lugar en la grada y conectarla a un sistema que permita procesar las imágenes. El objetivo es que sea un sistema de bajo coste y si es posible que no tenga la necesidad de conectarse a un servidor en internet con GPU. Esto influirá en las decisiones tomadas en cuanto al diseño y el análisis de prestaciones que se realizarán de forma posterior.

En la figura 4.1 del sistema vemos expresado los diferentes bloques por los que pasamos para proporcionar los resultados. En los siguientes capítulos vamos a ir desarrollándolos de manera secuencial.

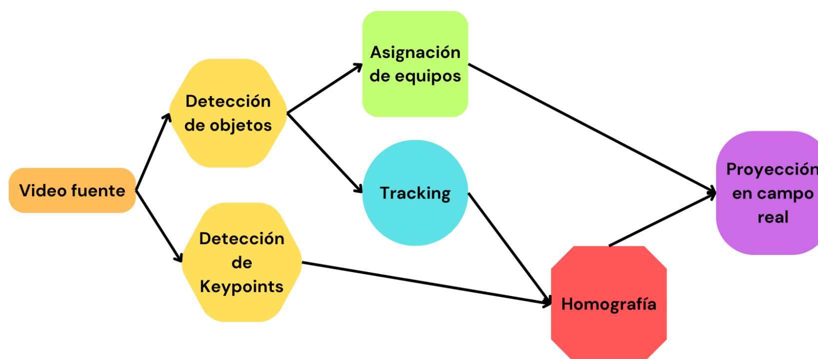


Figura 4.1: Diagrama de Bloques del sistema

4.1.1. Modificaciones respecto a Roboflow y motivaciones.

Durante este trabajo, se ha intentado implementar una aplicación desarrollada por *Roboflow*[1], pero utilizando directamente la librería de los creadores del modelo YOLOv8 [2], con el objetivo de no depender de librerías externas, simplificar el desarrollo las pruebas y futuro mantenimiento. El desarrollo se ha llevado a cabo empleando el código disponible en Ultralytics, a excepción del algoritmo de seguimiento (ByteTrack), que se ha integrado a través de *Roboflow*. Adicionalmente, se ha experimentado con diversas técnicas de extracción y representación de *embeddings* y *clustering*, además de varias técnicas de entrenamiento y de creación de *dataset* de manera automática.

[1]https://github.com/roboflow/sports
[2]https://ultralytics.com/es

Por último, se ha optimizado la ejecución del modelo en términos de velocidad mediante una variación en los formatos numéricos utilizados, además de una fusión de capas del modelo para incrementar aún más su rendimiento y evitar los altos gastos mensuales que supone alquilar servidores de sistemas *cloud*.

La motivación para desarrollar el sistema sin utilizar el código proporcionado radica en la complejidad y la falta de facilidad para el desarrollo en el código existente, así como en la necesidad de poder aplicarlo en situaciones que requieran una inferencia cercana al tiempo real. [3]

4.1.2. Dataset

Para entrenar nuestras redes neuronales, es fundamental contar con un conjunto de datos adecuado. En un primer intento, consideramos la opción de utilizar un modelo como *Grounding-DINO* para procesar nuestro *dataset* y generar automáticamente las etiquetas correspondientes. Sin embargo, tal y como se observa en la imagen 4.2, el desempeño obtenido no cumple con las expectativas.

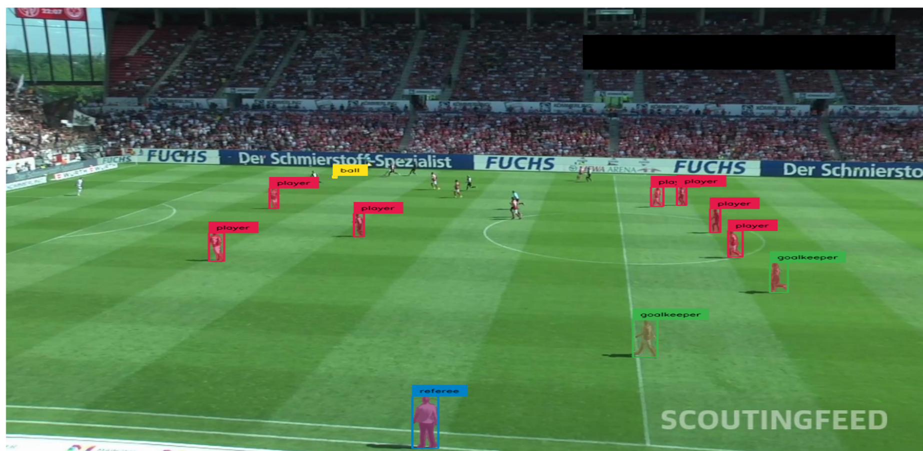


Figura 4.2: Etiquetado con Grouding Dino con promts para jugadores, arbitros y balón

Esto se debe a la complejidad inherente al problema. Aunque *Grounding-DINO* es un modelo *zero-shot* con una alta capacidad de abstracción, no logra segmentar correctamente las imágenes de nuestro *dataset*. Este problema surge porque las imágenes, al estar capturadas desde una distancia considerable para abarcar gran parte del campo, presentan los objetos de interés (jugadores, árbitros y balón) reducidos a unos pocos píxeles. Esta baja resolución de los objetos dificulta que el modelo alcance la precisión necesaria para considerarlo una opción viable. En vista de la siguiente tesitura,

[3]https://roboflow.com/

optamos por la utilización de un *dataset* ya existente perteneciente a la Bundesliga etiquetado por *Roboflow* [4]. El *dataset* elegido esta compuesto por un total de 380 imágenes etiquetadas, el cual ha sido dividido en un 80 % destinado al entrenamiento, y el 20 % para validación y test. [5]

4.2. Detección

En esta sección abordamos dos problemas fundamentales: la detección de objetos y la detección de puntos clave o *keypoints*. La detección de *keypoints* es esencial debido a que cada cámara está ubicada en una posición y ángulo diferentes, y además se encuentra en movimiento para seguir la acción del juego. Esto provoca que las coordenadas de los objetos detectados en cada imagen no correspondan directamente a las coordenadas reales en el campo de juego.

Para resolver este problema, necesitamos mapear las coordenadas de los objetos detectados en la imagen a sus posiciones reales en el campo. Aquí es donde entra en juego la matriz de homografía. La homografía es una transformación matemática que permite relacionar puntos entre dos planos, en este caso, el plano de la imagen capturada por la cámara y el plano del campo de juego.

La elección de YOLOv8 para la detección de objetos en tiempo real se fundamenta en su capacidad para ofrecer un rendimiento excepcionalmente rápido y preciso, junto con las ventajas inherentes de ser un modelo de código abierto. Para el desarrollo y ejecución hemos utilizado la librería proporcionada directamente por los creadores: Ultralytics. [6]

4.2.1. Detección de objetos

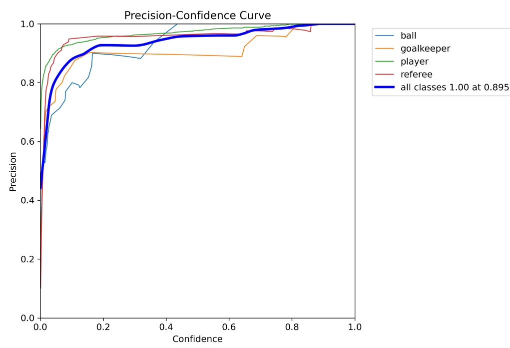
Realizamos el *fine-tunning* de un modelo YOLOv8, en su versión mas pequeña, ya que nuestro objetivo principal es la velocidad de procesado. El motivo de realizar un refinado de unos pesos ya entrenados en lugar de realizar un entrenamiento de la red desde cero, es debido a la cantidad de recursos y tiempos que supondría. Debido a esto, no nos sería posible obtener los resultados de precisión que obtenemos de otra forma. Una vez hemos entrenado el modelo con el *dataset* elegido, evaluamos unas métricas de precisión del mismo.

---

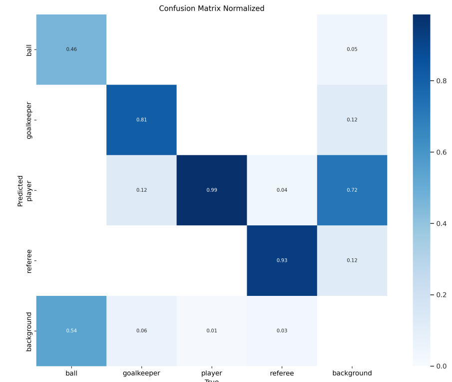
[4] https://universe.roboflow.com/roboflow-jvuqo/football-field-detection-f07vi/dataset/15

[5] https://roboflow.com/

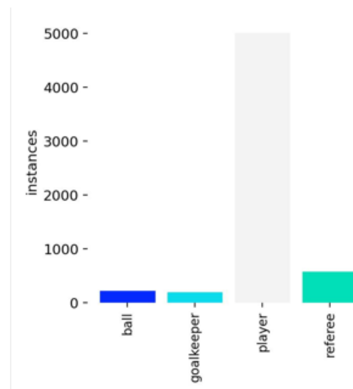
[6] https://docs.ultralytics.com/es



(a) Curva de precisión-confianza.



(b) Matriz de confusión normalizada.



(c) Distribución de etiquetas.

Figura 4.3: Rendimiento del modelo: análisis de precisión, matriz de confusión y distribución de etiquetas.

En la figura 4.3a, vemos la métrica precisión-confianza, que nos muestra la precisión del modelo para cada nivel de confianza. El umbral de confianza es el valor mínimo que debe tener una detección para ser considerada válida.

$$\text{Precisión} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Positivos}} \tag{4.1}$$

Podemos ver que el modelo muestra un buen equilibrio, con una precisión alta en promedio, cercana a 0.9 en el momento que establecemos una confianza necesaria superior a 0.3, sin embargo; la clase portero realiza un desempeño inferior a las demás. La elección del nivel de confianza tiene un impacto sobre la precisión y la sensibilidad del modelo. Las clases tienen un desempeño diferente, en la imagen 4.3c, se puede observar también el número de apariciones de cada clase en el *dataset*. Como es lógico, la clase más representada es la de jugadores, y las demás están menos representadas, siendo el portero la última. Esto puede ser una de las causas de que sea más difícil de detectar la clase portero para la red, ya que es la clase que menos ha visto.

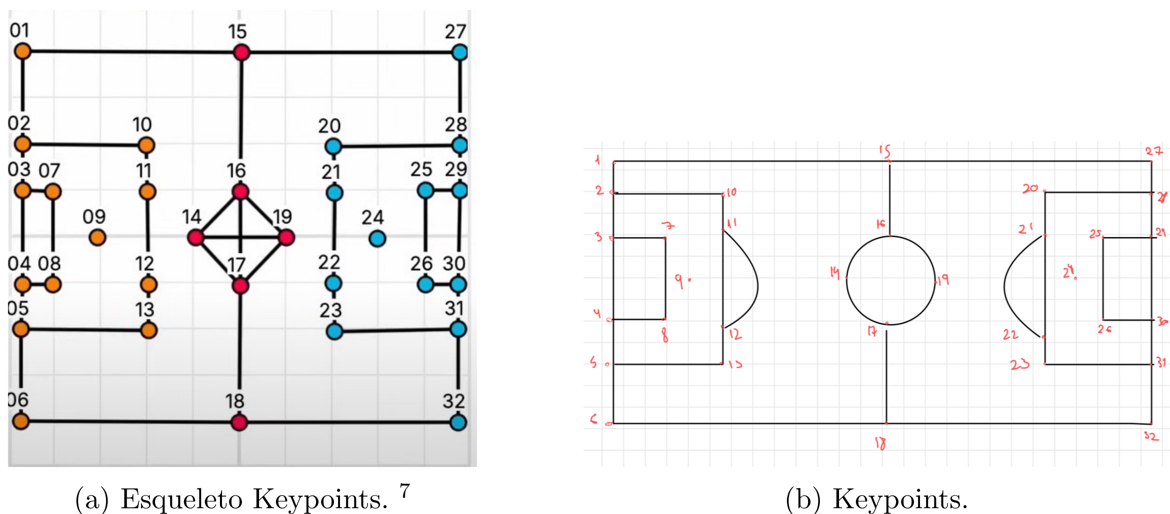
Por último, la matriz de confusión mostrada en la figura 4.3b permite identificar los errores cometidos por el modelo. En términos generales, este muestra un buen desempeño, ya que los valores en la diagonal principal son considerablemente altos. Sin embargo, es importante señalar que el modelo tiende a confundir la pelota con el campo de fútbol. Esto podría deberse a que interpreta ciertas líneas del campo como si fueran la pelota. Además en la tabla 4.1 podemos un resumen de métricas realizadas durante la validación del modelo.

Clase	Total Imágenes	Instancias	Precisión	Recall	mAP50	mAP50-95
all	43	1025	0.903	0.855	0.885	0.675
ball	39	39	0.833	0.513	0.64	0.336
goalkeeper	32	32	0.879	0.938	0.924	0.766
player	43	853	0.940	0.991	0.994	0.866
referee	43	101	0.958	0.980	0.983	0.730

Tabla 4.1: Métricas del modelo en dataset de Test

4.2.2. Detección de keypoints

Para calcular la matriz de homografía, es necesario conocer varios puntos de referencia cuyos valores sean conocidos en ambos planos. Estos puntos de referencia son los *keypoints*. Al identificar *keypoints* tanto en la imagen de la cámara como en el modelo del campo, podemos establecer correspondencias entre ellos. Esto nos permite calcular la matriz de homografía que transformará correctamente las coordenadas de los objetos detectados en la imagen a sus posiciones reales en el campo de juego [38]. Realizamos una selección de keypoints, en concreto 32, tal y como se puede ver en la imagen 4.4 .

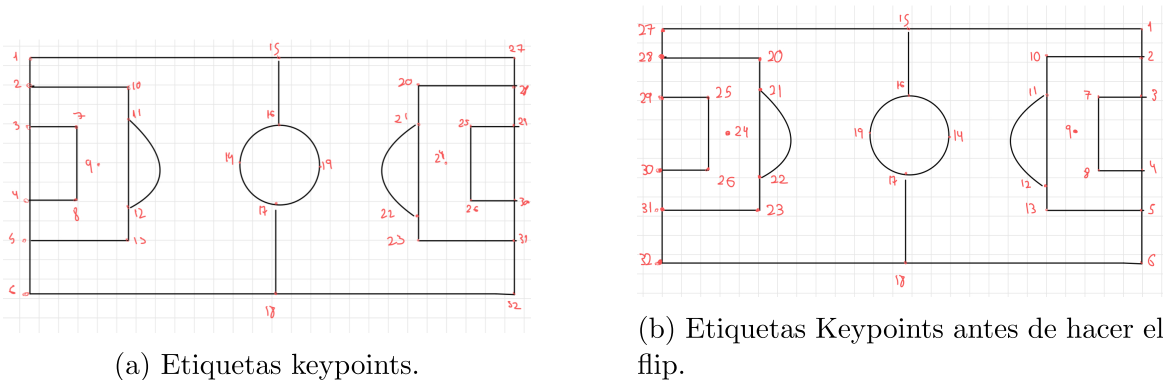


(a) Esqueleto Keypoints. [7]

(b) Keypoints.

Figura 4.4: Estructura Keypoints.

Además a la hora de entrenar el modelo aplicamos una técnica básica de *data augmentation*, lo cual nos permite contar con un *dataset* más amplio. Podemos realizar dicha operación gracias a que en nuestro caso, no nos vemos afectado si efectuamos una operación *Flip* en el eje horizontal. No obstante, hemos de tener en cuenta que al realizar dicha operación, tenemos que indicar el orden en que las etiquetas de los *keypoints* son reflejadas. Por ejemplo, si un punto esta etiquetado como el 1, al aplicar la reflexión debemos etiquetarlo como el 27. Esto se puede ver en la figura 4.5. La detección de los *keypoints* va a resultar un poco inestable, por lo que intentamos realizar un filtrado más exigente de los *keypoints* en el que solo los aceptamos con un nivel de confianza más exigente al resto.



(a) Etiquetas keypoints.

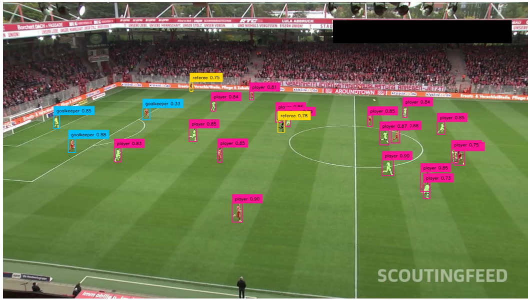
(b) Etiquetas Keypoints antes de hacer el flip.

Figura 4.5: Explicación del cambio de indices antes de hacer el flip para el data augmentation.

4.2.3. Visualización de Detecciones

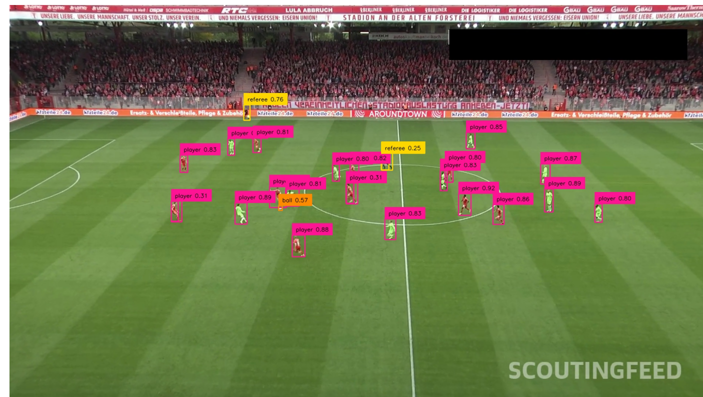
Podemos observar la detección de objetos en la figura 4.6. Aparecen etiquetadas tanto las clases como el nivel de confianza de la detección. Si prestamos atención podemos ver el problema comentado anteriormente, de que existe una dificultad en la detección del balón. Vemos como el modelo es capaz de diferenciar a los jugadores del resto de personas, como puede ser el entrenador o la grada, tal y como se reflejaban en las métricas de validación. Además podemos ver la confusión existente con la clase portero.

[7] `https://docs.roboflow.com`



(a) Visualización detección objetos.

(b) Visualización Keypoints.



(c) Visualización Balón.

Figura 4.6: Visualización combinada: detección de objetos y keypoints.

En la figura 4.6b podemos ver el otro problema de inestabilidad en la detección de keypoints comentado anteriormente. Además, cuando la pelota se mueve muy rápido tenemos dificultades en la detección, seguramente debido a la falta de representación de dicho objeto en dichas circunstancias, sumado a la dificultad de que al moverse no se aprecia bien en el *frame*.

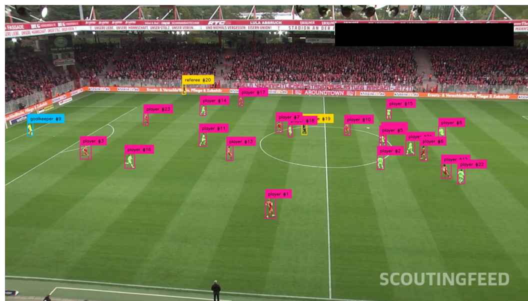
4.3. Tracking

Una vez que hemos detectado los objetos en cada fotograma, es necesario identificar cuándo dos objetos en fotogramas diferentes representan la misma entidad en distintos momentos temporales. Esto es fundamental para analizar el movimiento y el comportamiento de los objetos a lo largo del tiempo. Para lograrlo, utilizamos un algoritmo de seguimiento (tracking). En nuestra aplicación, hemos decidido emplear *ByteTrack*.

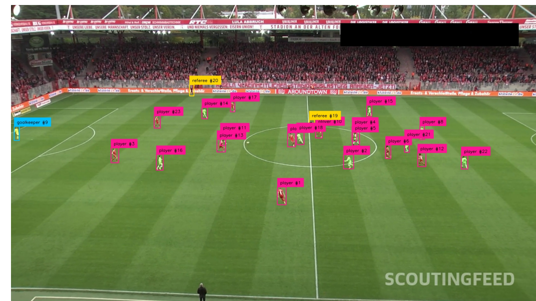
Aunque la librería *Ultralytics* incluye una implementación de *ByteTrack*, *Roboflow* ofrece una librería llamada *Sports*[8] que facilita considerablemente la representación de los resultados obtenidos con el modelo. Por esta razón, hemos optado por implementar

[8] https://github.com/roboflow/sports

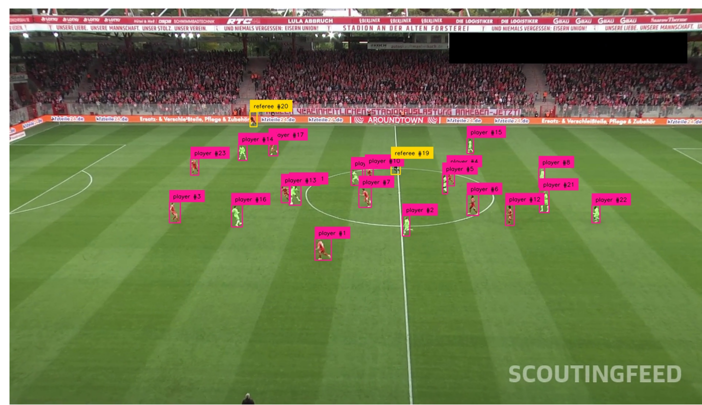
el seguimiento a través de *Roboflow* en lugar de hacerlo directamente con *Ultralytics*. Para ello, adaptamos la salida de la red al formato que acepta la librería.



(a) Frame 1.



(b) Frame 30.



(c) Frame 60.

Figura 4.7: Visualización del tracking aplicado en diferentes frames.

En la figura 4.7, podemos observar el seguimiento en acción, donde hemos asignado un ID a cada entidad detectada y cómo dicha entidad se propaga a través del tiempo.

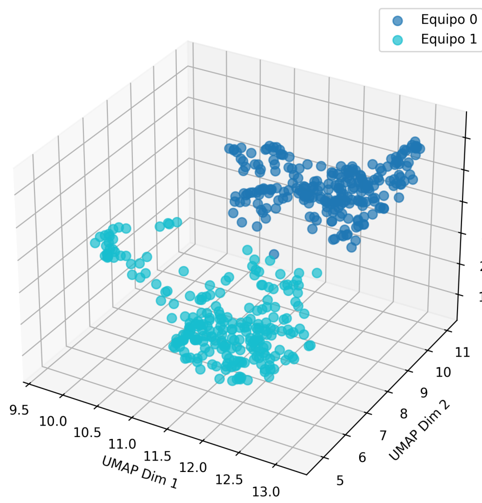
4.4. División equipos

En esta sección afrontamos el problema de separar los jugadores por equipos. Para poder realizar un sistema que sea robusto frente a cambios de luz y de lugar, recurrimos a utilizar un sistema de *clustering*. Gracias a esto aprovecharemos toda la información que nos pueda proporcionar la grabación para intentar realizar la clasificación en equipos.

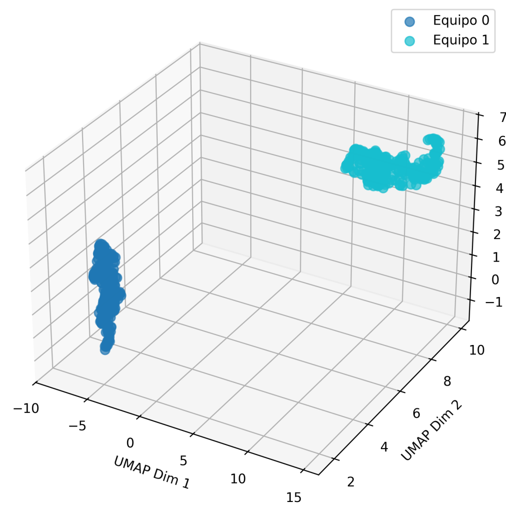
En primer lugar procedemos a obtener los pixeles contenidos en las *bounding boxes* delimitadoras proporcionadas por nuestro modelo de detección de objetos. Esto agregará un poco de retraso al principio ya que necesitamos esperar unos *frames* para recopilar las imágenes que necesitamos. Una muestra de cómo son los jugadores extraídos se puede ver en la figura 4.8.

Figura 4.8: Extracción de jugadores

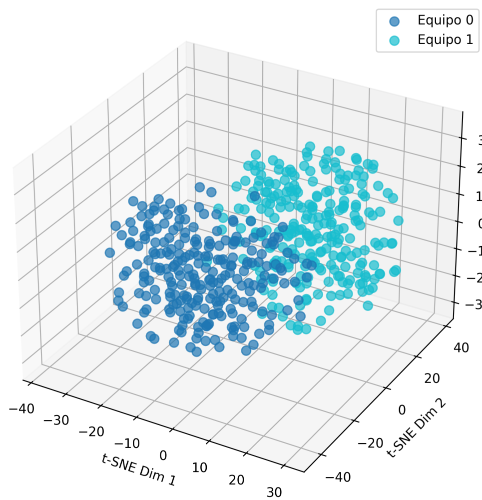
Con los jugadores recortados, procedemos a extraer la información contenida en sus imágenes mediante su representación en un espacio multidimensional (*embeddings*). Para ello, hemos experimentado con dos modelos previamente explicados.
Una vez que tenemos esta información en forma de vectores multidimensionales, aplicamos una herramienta de reducción de dimensionalidad, para reducir las dimensiones a tres. Finalmente, utilizamos un algoritmo de *clustering* para separar los datos en dos *clusters*. En la figura 4.9 y 4.10 se puede observar el resultado y la importancia de la capacidad de cada sistema para generar *embeddings* de calidad.



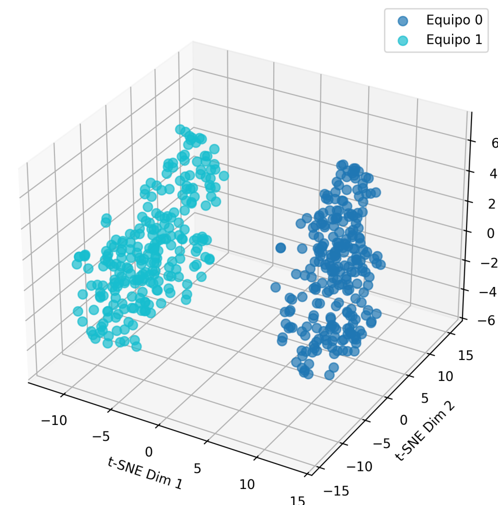
(a) Generador de embeddings VIT y reducción de dimensiones con UMAP.



(b) Generador de embeddings SIGLIP y reducción de dimensiones UMAP.



(c) Generador de embeddings VIT y reducción de dimensiones con t-SNE.



(d) Generador de embeddings SIGLIP y reducción de dimensiones con t-SNE.

Figura 4.9: Visualización de los equipos utilizando en dataset Test diferentes generadores de embeddings y métodos de reducción de dimensiones, utilizando k-means como método de clustering.

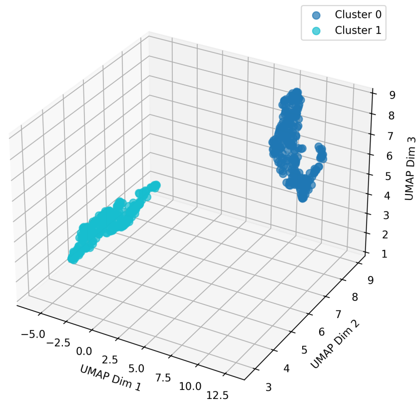
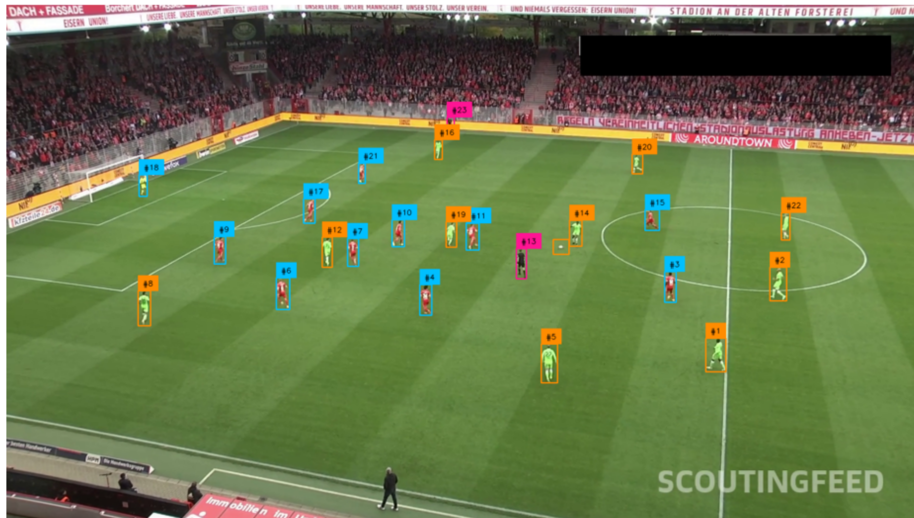


Figura 4.10: Visualización de los equipos en dataset Test utilizando como generador de embeddings SIGLIP, reducción de dimensiones UMAP y como método de clustering DBSCAN

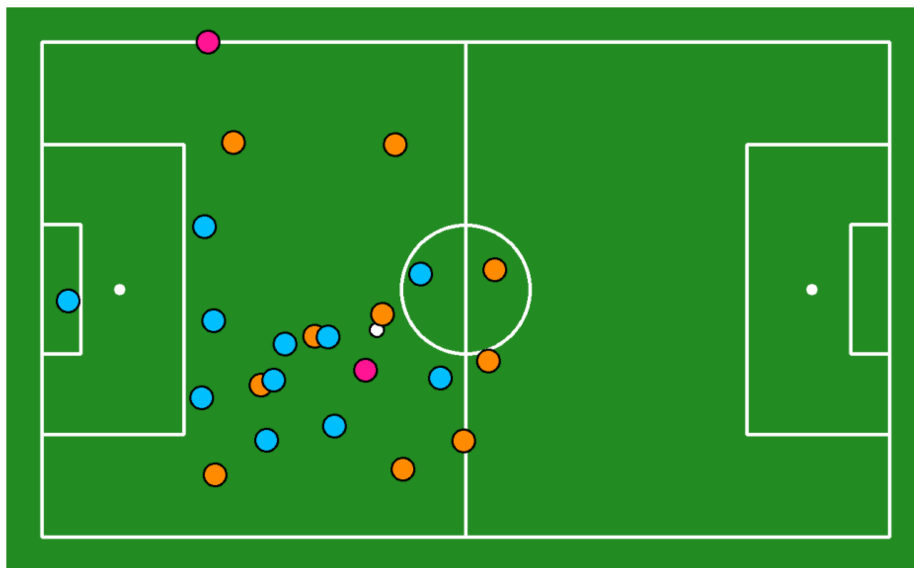
En base a los resultados obtenidos hemos decidido utilizar SIGLIP para realizar la representación en *embeddings*, y a UMAP para llevar a cabo la reducción de dimensiones. En cuanto al algoritmo de *clustering* nos hemos decantado por usar k-means por simplicidad debido a la ausencia de una razón basada en la precisión.

4.5. Proyección

Una vez que ya tenemos los jugadores identificados y los *keypoints* detectados, procedemos a realizar la proyección sobre el campo, para ello calculamos la matriz de homografía que nos permitirá realizar la traslación de coordenadas. En la figura 4.11 podemos ver el resultado de la proyección.



(a) Objetos etiquetados.



(b) Proyección de Objetos con coordenadas reales.

Figura 4.11: Proyecciones de jugadores, árbitros y balón.

4.6. Aumento velocidad

4.6.1. Evaluación Prestaciones GPU

Una vez hemos conseguido que nuestro sistema funcione, nos encontramos ante el problema de que necesitamos que sea una aplicación capaz de obtener resultados con un reducido margen de tiempo. La inmediatez no es imprescindible, se puede tolerar cierto retraso, pero no puede ser un proceso que se demore excesivamente.

La evalucación de todas las métricas ha sido realizada en una tarjeta NVIDIA

GeForce RTX 3090.

Hemos evaluado la velocidad de procesado del modelo en diferentes formatos de exportación para su posterior despliegue. Para ello, hemos realizado un conjunto de pruebas, en las cuales se mide tanto el tiempo de inferencia. En la imagen 4.12 se pueden ver los resultados.

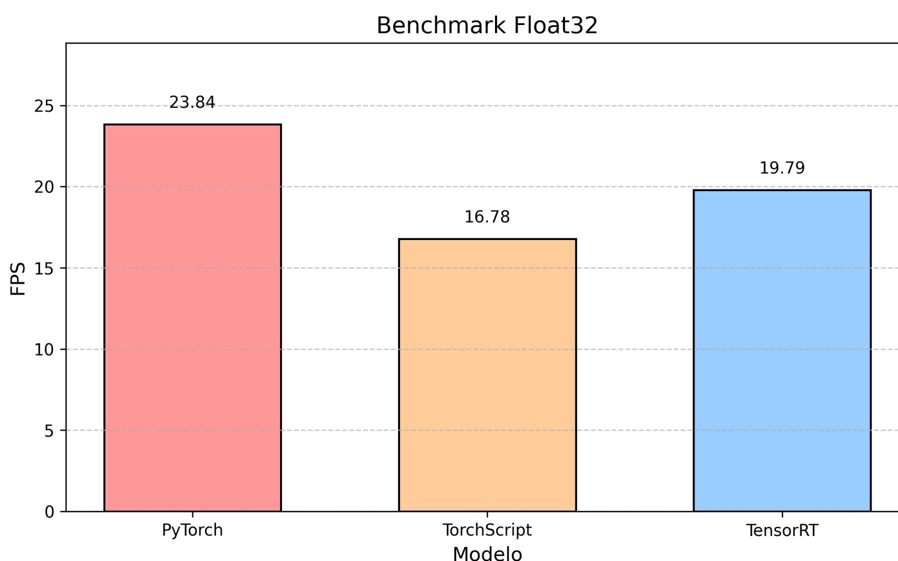


Figura 4.12: Frames por segundo calculados con varios toolkits de desarrollo de redes neuronales, realizado sobre el dataset de Test y con las redes en formato Float32. Calculos realizados sobre GPU

El problema como vemos, es que como máximo alcanzamos 23 fotogramas por segundo. A esto hay que tener en cuenta que es la velocidad que alcanzamos ejecutando un solo modelo, ahora habría que sumar que necesitamos otro modelo más, aparte de agregar los costos temporales de realizar los demás cálculos. Por esta razón, decidimos realizar una cuantización del modelo con el fin de disminuir tanto su tamaño como el tiempo requerido para realizar el procesamiento.

En primer lugar probamos a realizar un casting de los pesos del modelo a *float16* y procedemos a su siguiente evaluación.

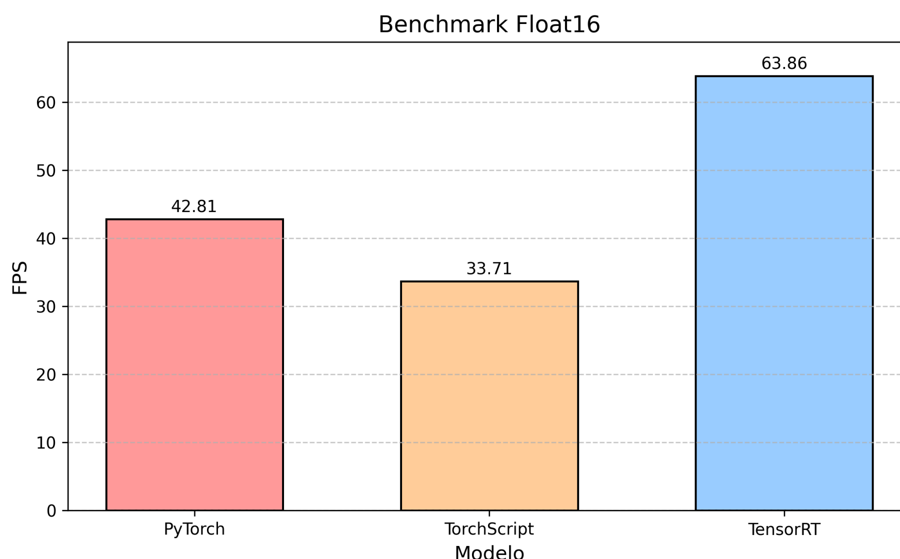


Figura 4.13: Frames por segundo calculados con varios toolkits de desarrollo de redes neuronales, realizado sobre el dataset de Test y con las redes en formato Float16. Cálculos realizados sobre GPU

Como podemos ver en la imagen 4.13, en este caso hemos obtenido una mejora del tiempo de inferencia. De media el modelo ha aumentado su velocidad de procesado y en un factor dos. La diferencia de factor presente entre diferentes formatos es debida a optimizaciones que utiliza cada formato para ejecutarse en los diferentes dispositivos.

Debido al hecho de que estamos buscando un rendimiento superior en cuanto a términos de velocidad, realizamos el salto de trabajar en coma flotante en 16 bits a trabajar en coma fija en 8 bits.

4.6.2. QTS

Para tratar de lidiar con el problema de la perdida de precisión debida a la cuantización y tratar de minimizarlo, entrenamos el modelo utilizando el procedimiento *QTS*. Para ello preparamos el modelo aplicando una configuración de la cuantización mediante la librería *torch*, de este modo insertamos unas operaciones ficticias que simulan el comportamiento cuantizado de la red, además de realizar un fusionado de las capas que lo permitan. Utilizan el estimador STE comentado en la revisión para poder realizar el entrenamiento

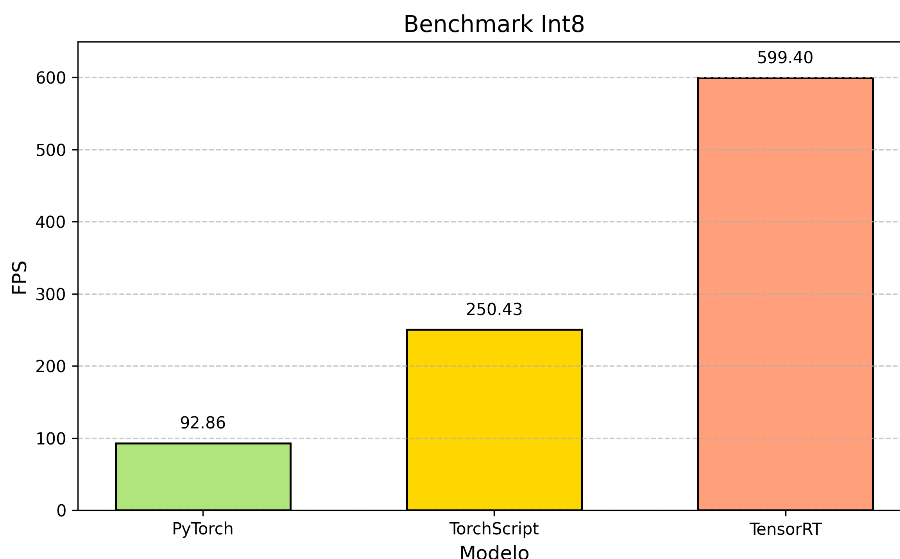


Figura 4.14: Frames por segundo calculados con varios toolkits de desarrollo de redes neuronales, realizado sobre el dataset de Test y con las redes en formato Int8. Cálculos realizados sobre GPU

En la figura 4.14 podemos ver los resultados que nos ofrece la cuantización. Hablando en términos de velocidad podemos constatar que damos por más que satisfecha la mejora obtenida permitiéndonos utilizar los modelos para nuestra aplicación sin necesidad de haciendo una inversión moderada.
El salto de velocidad es mucho mayor al cambiar a *Int8* desde *Float16*, que el conseguido anteriormente al pasar a *Float32*. Como se puede ver por ejemplo en el formato TorchScript, pasamos de alcanzar una velocidad de 33 FPS, a una de 250 FPS.

La posibilidad de realizar el procesado en un formato *Int8* abre las puertas a implementaciones futuras en dispositivos en los que además de la velocidad, pueda resultar un factor crítico el uso energético. Su capacidad de ejecución más rápida puede ser explotada tanto como para conseguir mayores velocidades, como para mantener las velocidades anteriores pero con dispositivos menos avanzados y que utilicen menos recursos.

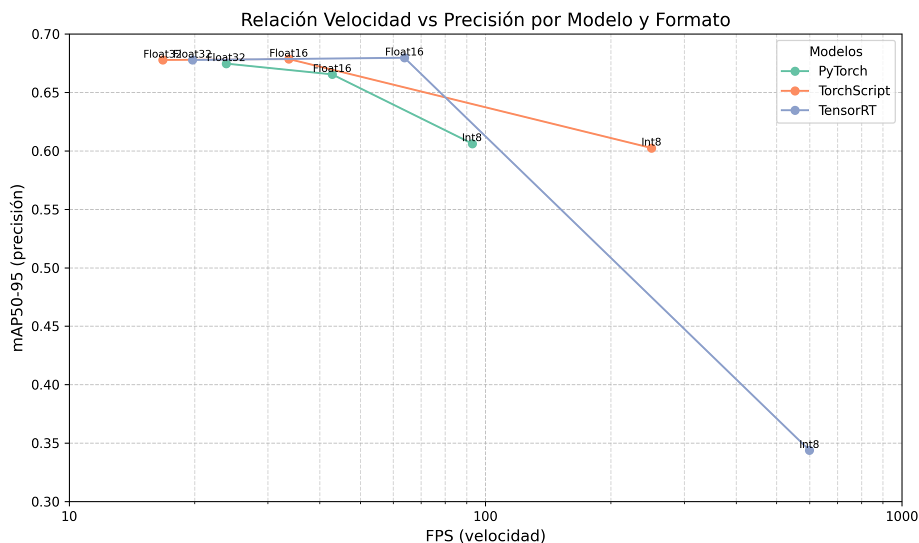


Figura 4.15: Representación de la velocidad alcanzada por los modelos en sus diferentes formatos frente a la precisión adquirida. La velocidad está en escala logarítmica. Los cálculos han sido realizados utilizando varios toolkits de desarrollo de redes neuronales, realizado sobre el dataset de Test y con las redes en formato Float32, Float16 e Int8. Calculos realizados sobre GPU

En la imagen 4.15 podemos ver una comparativa de los diferentes formatos. Hemos representado la velocidad del modelo frente a la precisión que es capaz de alcanzar. El punto óptimo sería la esquina superior derecha, ya que tendríamos el máximo de velocidad coincidiendo con el máximo de precisión. La perdida de precisión causada por la cuantización se puede observar de manera notable, sin embargo; es posible reducirla aun más mejorando la exportación del modelo.

La métrica de precisión utilizada para evaluar los modelos es la *mAP50-90*. Esta una superposición entre el *bouding box* obtenido por el modelo y la *bouding box* etiquetada real. Para poder considerar dicha predicción correcta, se utiliza un umbral, con el cual definimos que si la intersección entre ambas *bounding boxes* es superior al umbral, la consideramos correcta. En los *benchmark* que hemos realizado a continuación hemos utilizado un promedio entre 10 umbrales con valores extremos de 0.5 (menos exigente), a 0.95 (más exigente), con un salto de 0.05.

4.6.3. CPU

Por último, hemos realizado un análisis de prestaciones un CPU de los modelos para los diferentes formatos, para analizar su viabilidad en caso de que algún equipo quisiera ejecutarlo en un dispositivo sin GPU con el fin de ahorrar costes.

Las figura 4.16 muestra una comparativa del rendimiento de los modelos en cada correspondiente formato.

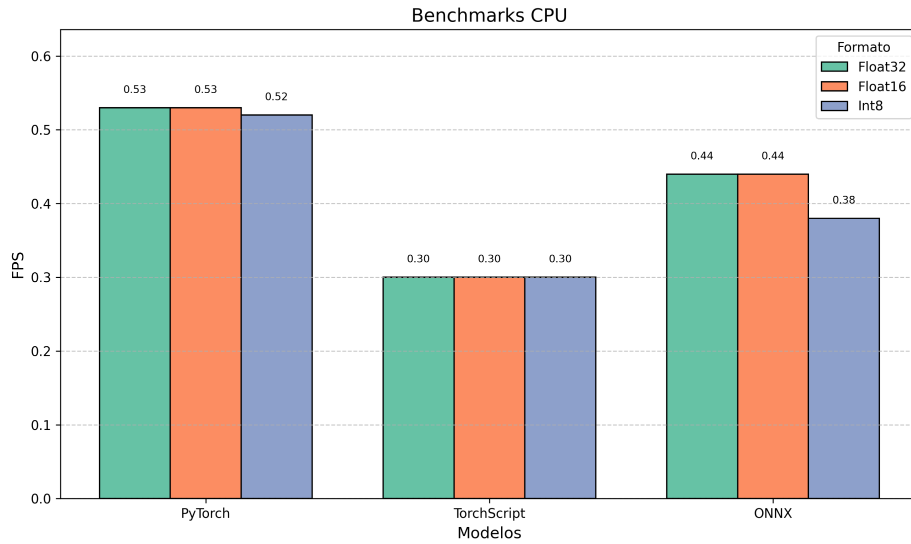


Figura 4.16: Frames por segundo calculados con varios toolkits de desarrollo de redes neuronales, realizado sobre el dataset de Test y con las redes en formato Float32, Float16 e Int8. Cálculos Realizados sobre CPU.

Debido a los resultamos obtenidos, descartamos la posibilidad de usar como dispositivo donde ejecutar nuestro sistema una CPU. No descartamos poder usarla en otras aplicaciones donde la velocidad no sea un factor crítico pero si lo sea la inversión disponible.

Capítulo 5

Discusión y Líneas Futuras

5.1. Conclusión

Hemos podido comprobar que hoy en día el uso de redes mediante *prompts* de texto es cada vez mayor y los modelos multimodales en el campo de la visión por computador son muy prometedores. Tareas muy complejas hace una década resultan prácticamente triviales en la actualidad con eficacias iguales o superiores a las que tendría un etiquetador humano. Cada vez hay más aplicaciones en las que se utiliza una combinación de redes para conseguir proporcionar un resultado que de otra manera habría resultado muy costoso.

En nuestro caso no se consigue una automatización completa del proceso, ya que en caso de que se generase un error o en un evento no contemplado como puede ser una situación de cambio de jugador, habría que indicar de manera manual que la identidad de ese objeto no corresponde con ninguna antes vista. Sin embargo, cabe destacar, que el trabajo que antes llevaría a una persona una cantidad razonable de tiempo y bastante esfuerzo, se realiza de forma cómoda y en un espaciado temporal considerablemente menor, lo que permite tener disponibles estadísticas prácticamente en tiempo real y la posibilidad de actuar en consecuencia de manera inmediata.

Se ha visto la posibilidad de conseguir buenos resultados a pesar de contar con una cantidad de datos bastante limitada. La posibilidad de realizar un pequeño entrenamiento (*fine-tunning*) de las redes en lugar de un entrenamiento completo, concede la oportunidad de personalizar redes para tareas bastante más específicas y con resultados que serían imposibles de alcanzar sin una cantidad ingente de datos.

Además también se ha observado la opción de realizar aumento de velocidad gracias a la cuantización. Esto abre las puertas a que se puedan realizar aplicaciones

que requieran de unas mayores prestaciones hablando en términos de velocidad, o en contraparte; conservar la velocidad pero utilizando menos recursos.

La posibilidad de realizar aplicaciones de bajo consumo puede permitir su aplicación en áreas donde el nivel de gasto energético sea un factor crítico, lo que habilitaría la opción de realizar pequeños dispositivos autónomos con la capacidad de ejecutar la aplicación.

Hay que mencionar también que una vez nos salimos del entorno de experimentación y entramos en el despliegue, hay que tener en cuenta el coste asociado a la ejecución de nuestro servicio. Se podría dar el caso de tener una aplicación muy buena pero incapaz de ser viable debido a los recursos necesarios para llevarla a cabo. Tener la capacidad de poder realizar la aplicación mediante recursos propios en lugar de necesitar de contratar servicios externos de cálculo en la nube puede abaratar mucho los costes. La cuantización abre la puerta a este mundo en el que más empresas podrán asumir los costes de realizar una implementación así.

Por último, cabe destacar el hecho de que se debería continuar el proceso realizado en este trabajo, para desarrollar e implementar el método de manera independiente de las librerías como *Ultralytics*, ya que gracias a ello ha posibilitado la implementación de cambios de forma más ágil. La dinámica de trabajo que se sigue últimamente en algunas librerías de código abierto que intentan integrar muchos modelos, trata de envolver mediante una infinidad de capas el modelo, para facilitar su uso final al usuario. No obstante, esta acción dificulta en gran medida si se precisa de utilizar la red para un fin que no ha sido diseñada y la interoperatividad entre modelos.

5.2. Líneas Futuras

En este trabajo se ha visto una primera aproximación a una posible solución de un problema que hasta hace unos años se consideraba extremadamente difícil. Sin embargo, hay bastantes áreas en los que se puede mejorar.

En primer lugar es evidente la necesidad de contar con un *dataset* cuyo contenido sea más extenso y variado. Únicamente se contaba con una cantidad cercana a las 350 unidades, y hablando en términos de entrenamiento para redes neuronales empleadas para resolver tareas complejas, como para la que la hemos utilizado, se trata de una cantidad bastante limitada. También cabe el caso de utilizar técnicas que mejoren el balance de representación entre clases.

Además se puede explorar el uso de nuevas arquitecturas, ya que gracias a los resultados que hemos podido obtener, vemos viable la utilización de versiones más grandes y complejas de la red utilizada, como también el uso de redes diferentes que han sido presentadas al mundo durante la duración de este trabajo. Se trata de un área en la que se están publicando cada mes nuevas arquitecturas y resultaría bastante interesante probar la nuevas arquitecturas que resuenan en la comunidad con fuerza como pueden ser *Mamba* [39].

Por ultimo otra posible mejora puede ser la implementación del algoritmo de *tracking* dentro de la propia red pero ese parece un camino un poco más complejo debido a la cantidad de objetos de los que se pretende realizar el *tracking* de manera simultánea.

Capítulo 6

Bibliografía

[1] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[2] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2012.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[6] John L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, pages 6000–6010, 2017.

[9] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[13] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Neil Houlsby. Training data-efficient image transformers & distillation through attention. In *International Conference on Learning Representations (ICLR)*, 2021.

[14] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[15] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Neil Houlsby. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[16] Zeyu Tang, Lin Xie, Bo Dai, and Chen Change Loy. Convit: Improving vision transformers with soft convolutional inductive biases. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[17] Yi-Hsuan Tsai, Shaojie Bai, Pei Liang, I-Hong Tseng, and Jonathan Hung. Multimodal transformer for unaligned multimodal language sequences. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. In *Proceedings of the Workshop on Deep Learning for NLP (DL4NLP) at NeurIPS*, 2018.

[19] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, Q. Jiang, C. Li, J. Yang, H. Su, and J. Zhu. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

[20] Peiyu Zhou, Wuwei Pang, Hong Yu, Lei Li, Ehsan Xie, Wenzhe Wang, Xiaohua Shi, and Xiaochun Liang. Grounding dino: A general and efficient object detector with open-vocabulary capability. *arXiv preprint arXiv:2304.02643*, 2023.

[21] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma. A review of yolo algorithm developments. *Procedia Computer Science*, 199:1066–1073, 2022.

[22] Juan Terven, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, 5(4):1680–1716, 2023.

[23] Muhammad Hussain. Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, 11(7):677, 2023.

[24] Anil K. Jain and Richard C. Dubes. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 20(11):1169–1178, 1999.

[25] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[26] Ian T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics, 2nd edition, 2002.

[27] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[28] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NeurIPS 2018)*, pages 8360–8368, 2018.

[29] Xinlei Chen and Yangqing Jia. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[30] Yu-Hao Zhang, Jianchao Yang, Hong-Kun Wang, Yuchen Zhang, and Zhihua Zhang. Gridmask data augmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11177–11186, 2019.

[31] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations (ICLR)*, 2017.

[32] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyun Woo, Jong Chul Ye, and Byoung-Tak Zhang. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6023–6032, 2019.

[33] A. Antoniou, H. Edwards, and A. Storkey. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.

[34] Yifu Wang, Anpei Chen, Xiaowei Hu, Xiaogang Wang, and Bo Li. Bytetrack: Multi-object tracking by associating every detection box. *arXiv preprint arXiv:2110.06864*, 2021.

[35] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking with a deep association metric. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649, 2016.

[36] Bo Chen Menglong Zhu Matthew Tang Andrew Howard Hartwig Adam Dmitry Kalenichenko Benoit Jacob, Skirmantas Kligys. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2704–2713, 2018.

[37] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[38] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK, second edition edition, 2003.

[39] A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint*, arXiv:2312.00752, 2023.

[40] Xingyi Yang, Yuke Wang, Bo Dai, Sanja Fidler, Jeff Schneider, Deva Ramanan, Erik Gavves, and Tom Tuytelaars. Grounding dino: A strong, transferable detector for foundation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[41] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[42] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc V. Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[43] Lloyd Kaufman and Peter J. Rousseeuw. *Clustering by Means of Medoids.* John Wiley & Sons, 1987.

[44] Joseph H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[45] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231, 1996.

[46] Martin Ankerst, M. M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 49–60, 1999.

[47] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–38, 1977.

[48] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, volume 14, pages 849–856, 2002.

[49] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma. A review of yolo algorithm developments. *Procedia Computer Science*, 199:1066–1073, 2022.

[50] T. Diwan, G. Anirudh, and J. V. Tembhurne. Object detection using yolo: Challenges, architectural successors, datasets, and applications. *Multimedia Tools and Applications*, 82(6):9243–9275, 2023.

[51] J. Terven, D. M. Córdova-Esparza, and J. A. Romero-González. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, 5(4):1680–1716, 2023.

[52] Y. H. Lee and Y. Kim. Comparison of cnn and yolo for object detection. *Journal of the Semiconductor Display Technology*, 19(1):85–92, 2020.

[53] Linjie y Yu Licheng y Kholy Mohamed y Ahmed Hao y Gan Zhe y Batra Dhruv Chen, Yen-Chun y Li. Uniter: Universal image-text representation learning. *European Conference on Computer Vision (ECCV)*, 2020.

[54] Richard Hartley and Peter Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2):146–157, 1997.

Lista de Figuras

1.1. Diagrama de Gantt del Trabajo 3

2.1. Diagrama explicativo del Perceptrón.[1] 5
2.2. Arquitecturas Redes Residuales y Convolucionales. 7
2.3. Diagrama de una celda LSTM. [2] 8
2.4. Arquitectura Transformer.[3] 9
2.5. Diagrama de Bloques de atención Multicabeza.[4] 11
2.6. Diagrama de Bloques del Vision Transformer.[5] 13
2.7. Ejemplo Arquitectura Multimodal Imagen y Texto.[6] 15
2.8. Bounding Boxes de Dino con varias clases introducidas mediante promts de texto 17
2.9. Comparativa Distribución Normal frente a Bimodal.[7] 20
2.10. Diversas técnicas aplicadas de data augemtation.[8] 23

3.1. Formato Float32.[9] 28
3.2. Formato Float16.[10] 29
3.3. Comparativa Precisión Float32 y Float16.[11] 30
3.4. Precisión y rángo dinámico de los diferentes formatos. [12] 31
3.5. Comparación cuantización simétrica y asimétrica.[13] 33
3.6. Comparativa de las pérdidas del modelo frente a realizar QAT o PTQ.[14] 36

4.1. Diagrama de Bloques del sistema 38
4.2. Etiquetado con Grouding Dino con promts para jugadores, arbitros y balón 39
4.3. Rendimiento del modelo: análisis de precisión, matriz de confusión y distribución de etiquetas. 41
4.4. Estructura Keypoints. 42
4.5. Explicación del cambio de indices antes de hacer el flip para el data augmentation. 43
4.6. Visualización combinada: detección de objetos y keypoints. 44

4.7. Visualización del tracking aplicado en diferentes frames. 45
4.8. Extracción de jugadores 46
4.9. Visualización de los equipos utilizando en dataset Test diferentes generadores de embeddings y métodos de reducción de dimensiones, utilizando k-means como método de clustering. 47
4.10. Visualización de los equipos en dataset Test utilizando como generador de embeddings SIGLIP, reducción de dimensiones UMAP y como método de clustering DBSCAN 48
4.11. Proyecciones de jugadores, árbitros y balón. 49
4.12. Frames por segundo calculados con varios toolkits de desarrollo de redes neuronales, realizado sobre el dataset de Test y con las redes en formato Float32. Calculos realizados sobre GPU 50
4.13. Frames por segundo calculados con varios toolkits de desarrollo de redes neuronales, realizado sobre el dataset de Test y con las redes en formato Float16. Cálculos realizados sobre GPU 51
4.14. Frames por segundo calculados con varios toolkits de desarrollo de redes neuronales, realizado sobre el dataset de Test y con las redes en formato Int8. Cálculos realizados sobre GPU 52
4.15. Representación de la velocidad alcanzada por los modelos en sus diferentes formatos frente a la precisión adquirida. La velocidad está en escala logarítmica. Los cálculos han sido realizados utilizando varios toolkits de desarrollo de redes neuronales, realizado sobre el dataset de Test y con las redes en formato Float32, Float16 e Int8. Calculos realizados sobre GPU 53
4.16. Frames por segundo calculados con varios toolkits de desarrollo de redes neuronales, realizado sobre el dataset de Test y con las redes en formato Float32, Float16 e Int8. Cálculos Realizados sobre CPU. 54

Lista de Tablas

4.1. Métricas del modelo en dataset de Test 42

Apéndices

Apéndices A

Anexo

A.1. Mecanismo atención Transformers

1. Detalle del Cálculo de Q, K y V En el mecanismo de atención de los *Transformers*, las **Consultas** (*Queries*), **Claves** (*Keys*) y **Valores** (*Values*) son componentes fundamentales que permiten al modelo determinar qué partes de la secuencia de entrada son más relevantes para cada elemento de la secuencia de salida.

Definición y Cálculo

- **Consultas (Q)**: Representan las características de los elementos de la secuencia de salida que buscan información relevante en la secuencia de entrada.
- **Claves (K)**: Representan las características de los elementos de la secuencia de entrada que serán comparadas con las consultas para determinar su relevancia.
- **Valores (V)**: Contienen la información que se extraerá y utilizará para generar la representación final de la salida.

Las consultas, claves y valores se obtienen mediante proyecciones lineales de los embeddings de los tokens de entrada. Matemáticamente, se definen como:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V \tag{A.1}$$

Donde:

- X es la matriz de embeddings de la secuencia de entrada.
- W^Q, W^K y W^V son matrices de pesos para las consultas, claves y valores, respectivamente. Estas se ajustan durante el entrenamiento.

2. Funcionamiento del Mecanismo de Atención El proceso de atención se lleva a cabo de la siguiente manera:

1. **Cálculo de Pesos de Atención**: Se calcula la similitud entre cada consulta Q y todas las claves K mediante el producto escalar, dando lugar a la matriz de atención:
$$\text{Pesos} = Q \cdot K^T \tag{A.2}$$

2. **Escalado de Pesos**: Para estabilizar los gradientes, los pesos se escalan dividiéndolos por la raíz cuadrada de la dimensión de las claves d_k:
$$\text{Pesos Escalados} = \frac{Q \cdot K^T}{\sqrt{d_k}} \tag{A.3}$$

3. **Aplicación de Softmax**: Se aplica la función *softmax* a los pesos escalados para obtener pesos de atención que sumen 1:
$$\text{Pesos de Atención} = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \tag{A.4}$$

4. **Ponderación de Valores**: Se realiza el producto entre los pesos de atencion y los valores V, produciendo una representación ponderada:
$$\text{Salida de Atención} = \text{Pesos de Atención} \cdot V \tag{A.5}$$

A.2. Técnicas de clustering

1. **Clustering Jerárquico**[44]

   - **Aglomerativo (Bottom-Up)**

     - **Descripción:** Comienza considerando cada punto como un cluster individual y, en cada paso, fusiona los clusters más cercanos hasta formar un único cluster que contiene todos los puntos.
     - **Métodos de enlace:**
       - Enlace simple: Distancia mínima entre puntos de clusters diferentes.
       - Enlace completo: Distancia máxima entre puntos de clusters diferentes.
       - Enlace promedio: Promedio de las distancias entre todos los pares de puntos.
     - **Ventajas:** No requiere predefinir el número de clusters.

- **Desventajas:** Altamente sensible a los outliers y puede ser computacionalmente costoso.

– **Divisivo (Top-Down)**

- **Descripción:** Comienza con todos los puntos en un solo cluster y divide recursivamente los clusters en subclusters.
- **Ventajas y Desventajas:** Similar al método aglomerativo, pero menos utilizado debido a su complejidad.

2. **Clustering Basado en Densidad**

– **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**[45]

- **Descripción:** Identifica clusters como áreas de alta densidad separadas por áreas de baja densidad.
- **Funcionamiento:**
  a) Define puntos como "centrales", "bordes" o "ruido" basándose en la densidad local.
  b) Forma clusters conectando puntos centrales adyacentes.
- **Ventajas:** Puede encontrar clusters de forma arbitraria y es robusto a outliers.
- **Desventajas:** Sensible a los parámetros de densidad y puede fallar en datasets con densidades variables.

– **OPTICS (Ordering Points To Identify the Clustering Structure)**[46]

- **Descripción:** Extensión de DBSCAN que maneja mejor variaciones en la densidad.
- **Ventajas:** Proporciona una ordenación de los puntos que refleja la estructura de densidad.
- **Desventajas:** Más complejo y requiere interpretación adicional.

3. **Clustering Basado en Modelos**

– **Gaussian Mixture Models (GMM)**[47]

- **Descripción:** Asume que los datos son una combinación de múltiples distribuciones gaussianas y utiliza el algoritmo EM (Expectation-Maximization) para estimar los parámetros.

- **Ventajas:** Flexibilidad en la forma de los clusters.
- **Desventajas:** Requiere predefinir el número de clusters y puede ser computacionalmente intensivo.

4. **Clustering Espectral**[48]

   - **Descripción:** Utiliza técnicas de álgebra lineal y grafos para realizar el clustering, basándose en la similitud entre puntos.
   - **Funcionamiento:**
     a) Construye una matriz de similitud entre los puntos.
     b) Calcula los vectores propios (eigenvectors) del Laplaciano del grafo.
     c) Utiliza estos vectores para reducir la dimensionalidad y aplicar clustering (como K-Means).
   - **Ventajas:** Eficaz en datos con clusters no convexos y estructura compleja.
   - **Desventajas:** Escalabilidad limitada en datasets muy grandes.

A.3. Homografía

La homografía [54] es una transformación matemática que relaciona puntos entre dos planos proyectivos. En el contexto de la visión por computador, una homografía permite mapear puntos de una imagen a otra cuando ambas imágenes capturan un plano desde diferentes perspectivas. Esta transformación se representa mediante una matriz y es capaz de describir rotaciones, traslaciones, escalados y proyecciones en perspectiva. Es esencial cuando se desea alinear imágenes tomadas desde distintos ángulos o corregir distorsiones debidas a la perspectiva.
Para poder realizar el cálculo de la matriz, es necesario conocer las coordenadas de una serie de puntos en ambos planos.

La homografía es fundamental en aplicaciones como la creación de panoramas, donde se combinan varias imágenes para formar una vista amplia, o en la rectificación de imágenes, que corrige la perspectiva para obtener una vista frontal de un plano inclinado. Una vez calculada, esta matriz permite transformar cualquier punto de una imagen a su posición correspondiente en la otra.