Universidad
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Curso Académico 2022/2023

DETECCIÓN Y POSPROCESADO DE INSCRIPCIONES Y ESCUDOS EN IMÁGENES DE FACHADAS USANDO REDES NEURONALES Y VISIÓN ARTIFICIAL

Autor: Alejandro Mateos Domínguez

Tutor: Ángel Sánchez Calle

Resumen

El objetivo de este Trabajo Fin de Grado es resolver el problema de la detección y clasificación de relieves grabados en piedra, como escudos e inscripciones, en imágenes de elementos históricos, como monumentos, iglesias o catedrales, utilizando una red neuronal profunda. Para lograrlo, se ha recopilado una gran cantidad de datos, que incluyen una colección de fotografías propias e imágenes obtenidas de Internet. Esto ha permitido una experimentación continua y una búsqueda constante de los mejores resultados.

La red neuronal convolucional YOLOv5 y la herramienta *Roboflow* se utilizaron para la gestión y etiquetado de imágenes. Además, se han utilizado librerías como *PyTorch* y *CV2* para el entrenamiento, la ejecución y la evaluación del modelo. Esto incluye mejorar las regiones de la imagen con texto y correlacionar las regiones con relieves.

Se ha utilizado una técnica de trabajo basada en la experimentación y el aumento constante del conjunto de datos. El tamaño del conjunto, el número de *epochs* y el uso de pesos previamente entrenados en la red han variado. Se han evaluado otras métricas como *F1-score, precision* y la matriz de confusión, y se ha establecido una precisión mínima aceptable del 70% para la detección.

Finalmente, se discutirán las formas en que se podrían mejorar los resultados en función de los hallazgos y se discutirá cómo integrar esta herramienta con otras similares para que pueda ser utilizada en un contexto interdisciplinario y facilitar el trabajo de estudio y conservación de elementos históricos.

Palabras clave: detección y clasificación de elementos**,** YOLOv5, Python, inscripción, escudo, red neuronal.

Summary

The aim of this Final Degree Project is to solve the problem of detecting and classifying reliefs engraved in stone, such as coats of arms and inscriptions, in images of historical elements, such as monuments, churches or cathedrals, using a deep neural network. To achieve this, a large amount of data has been collected, including a collection of own photographs and images obtained from the Internet. This has allowed for continuous experimentation and a constant search for the best results.

The YOLOv5 convolutional neural network and the *Roboflow* tool were used for image management and labelling. In addition, libraries such as *PyTorch* and *CV2* have been used for model training, execution and evaluation. This includes enhancing image regions with text and correlating regions with reliefs.

A working technique based on experimentation and constant augmentation of the dataset has been used. The size of the set, the number of epochs and the use of weights previously trained on the network have been varied. Other metrics such as accuracy, F1 curve and confusion matrix have been evaluated and a minimum acceptable accuracy of 70% for detection has been established.

Finally, we will reflect on possible improvements based on the results obtained and explore ways of integrating this tool with other similar tools, in order to use it in an interdisciplinary field and thus facilitate the task of study and conservation of historical elements.

Keywords: element detection and classification, YOLOv5, Python, inscription, shield, neural network.

Agradecimientos

La finalización de este trabajo ha sido un proceso complicado de casi un año lleno de dificultades. A pesar de ello, he tenido la suerte de contar con el apoyo de personas increíbles a lo largo de este proyecto. Estoy muy agradecido a mis compañeros y profesores de la carrera por su apoyo y motivación para lograr este objetivo.

De hecho, este proyecto habría sido imposible de completar sin la ayuda de este grupo. Por lo tanto, me siento muy afortunado de haber encontrado a personas tan valiosas en este camino que me han acompañado en momentos buenos y malos. Su ayuda y apoyo han sido esenciales para mi éxito en este proyecto.

Agradezco a Ángel Sánchez Calle, mi tutor de TFG, por su dedicación y apoyo en este proyecto. Ángel ha invertido mucho tiempo compartiendo su conocimiento y experiencia conmigo de manera comprensiva, amable e instructiva durante todo el proceso.

Me siento muy afortunado de haber tenido la oportunidad de trabajar con Ángel, cuyo compromiso con mi crecimiento y desarrollo profesional ha sido fundamental para el éxito de este trabajo. Estoy muy agradecido por su apoyo y orientación en todo momento, y estoy muy agradecido por su ayuda en el desarrollo de este proyecto.

Mi más sincero agradecimiento a mi familia, especialmente a mis padres, por su paciencia y comprensión durante todo este proceso. Su incondicional apoyo me ha permitido enfrentar los desafíos de este proyecto con confianza y determinación.

Agradezco profundamente su dedicación y recursos para ayudarme a lograr mis metas. Su apoyo y amor han sido cruciales para mí, y les estaré eternamente agradecido por su inquebrantable ayuda.

Mi familia es mi principal fuente de motivación y fortaleza en este camino, y una vez más, quiero expresar mi gratitud. No habría sido posible alcanzar mis objetivos sin su ayuda.

Tabla de contenido

Resumen iii
Summary v
Agradecimientos vii
Índice de figuras xi
Índice de tablas xiii
1.Introducción 1
1.1 Motivación del proyecto 1
1.2 Patrimonio histórico sobre piedra 2
1.2.1 Tipos de relieves detectados en imágenes sobre piedra 3
1.2.2. Elementos gráficos y textuales. 3
1.3. Objetivos del proyecto 5
2. Marco teórico 7
2.1. Aprendizaje profundo (*Deep Learning*) 7
2.1.1. Redes neuronales profundas (RNP) 7
2.1.2. Redes neuronales convolucionales (CNN) 8
2.1.3. YOLO y su evolución 11
2.1.4. Arquitectura de YOLOv5 14
2.2. Preprocesado de imágenes de bajo contraste y con ruido 18
2.3. Mejora del texto en imágenes de inscripciones sobre piedra basado en componentes conexas 18
2.4. Correlación de los escudos basado en Similaridad Estructural 19
2.4.1. Índice de Similaridad Estructural (SSIM) 19
2.4.2. Otros métodos utilizados 20
3. Estado del arte 23
4. Métodos y solución informática 27
4.1. Bases de datos de imágenes de relieves 27
4.2. Preproceso de los datos 27
4.3. Anotación de las imágenes (Herramienta Roboflow) 28
4.4. Solución propuesta 31
4.4.1. Equipo y software usado 31
4.4.2. Jerarquía de ficheros 32
4.4.3. Entrenamiento de la red 33
4.4.4. Detección de elementos 36
4.4.5. Mejora del texto en inscripciones 37
4.4.6. Emparejamiento de escudos 38

5. Pruebas 41
5.1. Conjunto de datos. 41
5.2. Métricas de evaluación 44
5.2.1. Verdadero Positivo 45
5.2.2. Verdadero Negativo 45
5.2.3. Falso Positivo 46
5.2.4. Falso Negativo 46
5.2.5 *Precision* 46
5.2.6. *Recall* 46
5.2.7. *F1-score* 47
5.2.8. Matriz de confusión 47
5.3. Experimentos 47
5.3.1. Experimento 1: Valor de *batch* = 16 48
5.3.2. Experimento 2: Valor de *batch* máximo 51
5.3.3. Experimento 3: Pesos preentrenados 54
5.3.4. Experimento 4. Métodos de correlación de escudos (SSIM) 56
5.3.5. Experimento 5. Otros métodos de correlación de escudos 58
5.3.6. Experimento 6. Mejora de las detecciones de la clase 'Inscripción' 62
5.3.6. Resumen de los experimentos 64
6. Conclusión 67
6.1. Conclusiones 67
6.2 Trabajo futuro 68
Referencias 70

Índice de figuras

Fig. 1: Ejemplo de elementos gráficos 3
Fig. 2: Ejemplo de escudo 4
Fig. 3: Ejemplo de inscripción 4
Fig. 4: Ejemplo de convolución de Sobel. 9
Fig. 5: Ejemplo de pooling. 10
Fig 6: Arquitectura de YOLOv5. Imagen extraída de [21] 15
Fig. 7: Ejemplo de división en celdas (s = 7) y detección de objetos. Imagen extraída de [22] .. 16
Fig. 8: Imagen antes del recorte 27
Fig. 9: Imagen después del recorte 28
Fig. 10: Etiquetado con la herramienta Roboflow 29
Fig. 11: Operación de brillo utilizando Data Augmentation en Roboflow 30
Fig. 12: Versionado del dataset con Roboflow 30
Fig. 13: Arquitectura del sistema propuesto. 31
Fig. 14: Jerarquía de ficheros para un proyecto YOLOv5. 32
Fig. 15: Mapa de color de la clase Inscripción. Imagen obtenida con la herramienta Roboflow 42
Fig. 16: Mapa de color de la clase Escudo. Imagen obtenida con la herramienta Roboflow 43
Fig. 17: Representación esquemática de la fórmula para calcular el IoU. Imagen extraída de [27] 44
Fig. 18: Diferencia entre el ground truth y una detección. Imagen extraída de [28] 44
Fig. 19: Gráficas de precision y recall del experimento 1. 48
Fig. 20: Matriz de confusión del experimento 1 49
Fig. 21: Ejemplo de detecciones del experimento 1 50
Fig. 22: Gráficas de precision y recall del experimento 2 51
Fig. 23: Matriz de confusión del experimento 2 52
Fig. 24: Ejemplo de detecciones del experimento 2 53
Fig. 25: Gráficas de precision y recall del experimento 3. 54
Fig. 26: Matriz de confusión del experimento 3 55
Fig. 27: Ejemplo de escudos con un Índice de Similaridad intermedio 56
Fig. 28: Ejemplo de escudos con un Índice de Similaridad alto 57
Fig. 29: Escudos con una gran cantidad en común de puntos de interés 59
Fig. 30: Escudos con una baja cantidad en común de puntos de interés 59
Fig. 31: Escudo resultante de restar dos escudos. 60
Fig. 32: Escudos con media de correlación baja. 61
Fig. 33: Escudos con media de correlación alta. 62
Fig. 34: Pasos de postproceso para la mejora de legibilidad de las inscripciones. 63

Índice de tablas

Tabla 1: Desglose del número de objetos por categoría 42
Tabla 2: Resultados cuantitativos del Experimento 1 49
Tabla 3: Resultados cuantitativos del Experimento 2 52
Tabla 4: Resultados cuantitativos del experimento 3 55
Tabla 5: Resumen de los resultados de los tres primeros experimentos de detección usando YOLOv5 64

1.Introducción

1.1 Motivación del proyecto

Las inscripciones grabadas en piedra constituyen un valioso documento de la historia humana. Las inscripciones han evolucionado desde las primeras manifestaciones en el Paleolítico [1], hasta las construcciones complejas del Neolítico, como los dólmenes de Carnac o Stonehenge, también desde formas rudimentarias como pinturas y grabados en paredes de cuevas hasta esculturas y grabados más elaborados.

Gracias a la tecnología actual, es posible analizar, clasificar y manipular automáticamente estos grabados e inscripciones. Para ayudar a la comunidad científica, este proyecto permitirá procesar grabados y relieves históricos de piedra de manera rápida y precisa. De esta manera, el conocimiento disponible se puede ampliar y hacer posible la realización de inferencias sobre la historia y la cultura de los pueblos antiguos.

Por lo tanto, este trabajo se centra en el problema de la detección y el postprocesado de relieves sobre piedra, con el objetivo de mejorar esos relieves, como escudos e inscripciones, en imágenes de elementos históricos, como monumentos, iglesias o catedrales. Este estudio no solo proporciona a la comunidad científica una herramienta útil, sino que también abre la posibilidad de integrar esta tecnología con otras similares para facilitar la conversión y el estudio de elementos históricos.

La clasificación y el análisis de los escudos y relieves de piedra no solo puede revelar la época o el propósito de las obras, sino que también puede ayudar en la conservación y restauración de los elementos históricos. Al tener una *base de datos (BD)* precisa y detallada sobre cada relieve y su estado de conservación, los especialistas pueden planificar y ejecutar mejor las tareas de mantenimiento y restauración. Además, esta información puede ser útil para la educación y la investigación en historia y arqueología. Por lo tanto, el uso de técnicas de procesamiento de imágenes y análisis de datos puede tener un impacto significativo en la preservación y difusión del patrimonio cultural y artístico.

1.2 Patrimonio histórico sobre piedra

La Ley 16/1985 del 25 de junio del Patrimonio Histórico Español [2] define el Patrimonio Arqueológico como el patrimonio histórico protegido para su preservación, en particular en sus artículos 46 y 50, de la siguiente manera:

"*Las Administraciones públicas adoptarán medidas especiales para la protección de los monumentos con decoraciones escultóricas y en relieve, y se prohibirá toda intervención que dañe o desvirtúe su carácter histórico o artístico"* [2].

"*La realización de intervenciones en los bienes integrantes del Patrimonio Histórico deberá ajustarse a los criterios de conservación, restauración, protección y seguridad que se establezcan reglamentariamente, y se someterá al procedimiento administrativo previsto para las intervenciones en los bienes integrantes del Patrimonio Histórico*" [2].

Además, es importante destacar la labor de los profesionales dedicados a la arqueología, que contribuyen al conocimiento y preservación de nuestro patrimonio histórico y cultural realizando estudios y trabajos de excavación y conservación en yacimientos arqueológicos. Para asegurar la preservación de estos bienes y su acceso a las futuras generaciones, también es esencial la colaboración y la concienciación de la sociedad.

Estos documentos y grabados son indudablemente valiosos para el patrimonio cultural porque nos ayudan a comprender mejor las relaciones y costumbres de los grupos y personas que los crearon, así como su contexto histórico y cultural. Además, muchos de ellos tienen una gran relevancia religiosa o simbólica para ciertas comunidades, lo que los convierte en un componente crucial para comprender la evolución de las creencias y prácticas culturales a lo largo del tiempo.

Por lo tanto, contar con herramientas efectivas para analizar y preservar este vasto conjunto de datos es fundamental para que nuestra sociedad pueda continuar aprendiendo de nuestro pasado y transmitiendo ese conocimiento a las futuras generaciones. Solo así podremos continuar ampliando nuestra comprensión de nosotros mismos y del mundo.

1.2.1 Tipos de relieves detectados en imágenes sobre piedra

El análisis de relieves en imágenes de piedra puede encontrar varios elementos, ya sea textuales o gráficos, que requieren diferentes tipos de procesamiento. El texto, varios elementos gráficos (escudos, figuras humanas, etc.) y el fondo de la imagen han sido considerados en este proyecto. La Figura 1 ilustra algunos de estos elementos.

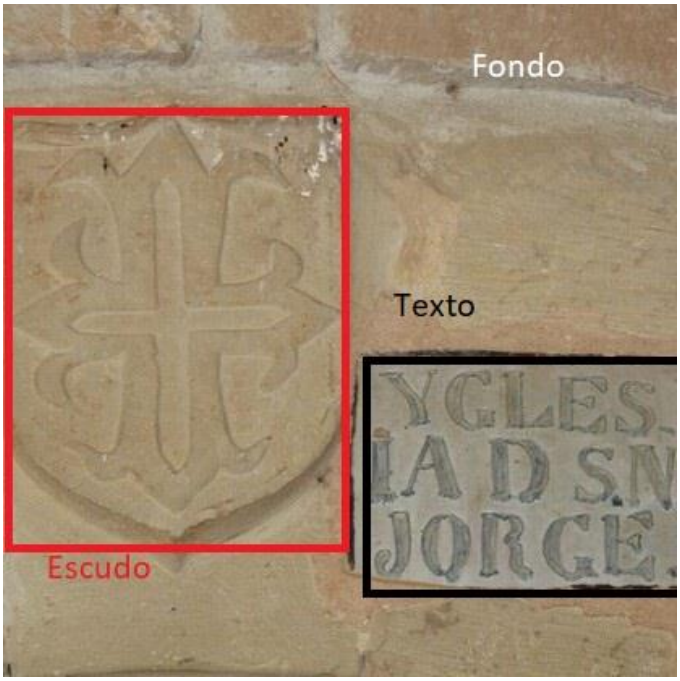

Fig. 1: Ejemplo de elementos gráficos

1.2.2. Elementos gráficos y textuales.

- **Escudos:** Los elementos grabados en piedra que identifican a una casa, familia o territorio que no contienen texto se denominan escudos. Estos escudos pueden estar compuestos por varios elementos gráficos, como cascos, ramos o coronas, y suelen tener un tamaño de entre 1 y 2 metros y proporciones rectangulares o incluso cuadradas. En la Figura 2 se ilustra un ejemplo de escudo.

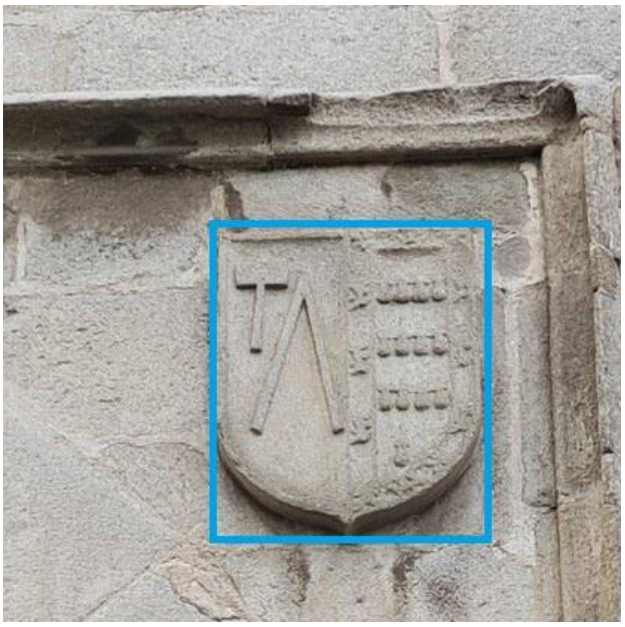

Fig. 2: Ejemplo de escudo

- **Inscripciones:** Se consideran inscripciones todos los elementos que contienen un conjunto de letras y/o números que forman palabras o frases. En la Figura 3 se ve un ejemplo de inscripción. Estas inscripciones pueden aparecer solas o acompañadas de otros elementos gráficos, como escudos. Si una inscripción aparece junto a un escudo, el elemento se clasifica como "escudo con inscripción".

Fig. 3: Ejemplo de inscripción

1.3. Objetivos del proyecto

El objetivo del proyecto actual es utilizar redes neuronales profundas para localizar de manera precisa y automática relieves en monumentos y patrimonio histórico, como escudos e inscripciones. Posteriormente, se buscará mejorar la calidad de las inscripciones en imágenes para que sean más fáciles de leer y analizar. Asimismo, se realizará una búsqueda por contenido para encontrar similitudes con otros escudos y agruparlos. Dado que hay pocos trabajos que aborden este problema, se cree que este proyecto puede ser de gran utilidad para la comunidad científica y para los historiadores porque puede facilitar su trabajo. Para lograr el objetivo general, es necesario alcanzar una serie de objetivos específicos para garantizar la calidad del proyecto. Algunos de estos subobjetivos pueden ser:

1. *Analizar los diversos tipos de elementos históricos y sus rasgos.* De manera que se pueda definir de manera precisa el significado de los objetos a detectar y desarrollar una metodología adecuada para su análisis, es necesario comprender las partes que conforman los relieves a analizar.
2. *Crear un conjunto de datos*, también conocido como *dataset*, sin sesgos para garantizar que el detector funcione correctamente. Tener un equilibrio en el número de objetos por clase y una cantidad suficiente de imágenes de alta calidad en la *BD* es esencial para lograr este punto.
3. *Investigar la arquitectura del modelo de detección seleccionado*, (YOLOv5), para lograr la identificación de objetos en los relieves. Es fundamental comprender cómo funciona esta herramienta para optimizar su rendimiento y obtener los mejores resultados posibles.
4. *Entrenar varios modelos basados en YOLOv5 utilizando el dataset construido*. Para mejorar el rendimiento de los modelos y lograr una detección precisa de los objetos de interés en los relieves, es necesario experimentar con diferentes hiperparámetros y técnicas de optimización.
5. *Calcular métricas de evaluación como precision, recall y F1-score*. Para evaluar la eficacia de los modelos propuestos en este proyecto, se realizará experimentación con el *dataset* y se calcularán los valores de las métricas de evaluación previamente establecidas.

6. *Correlacionar los escudos con similitudes y mejorar la legibilidad de las inscripciones mediante unos procedimientos específicos.*
    a. Se utilizarán técnicas de procesamiento de imágenes como aumento de contraste, ecualización, umbralización y eliminación de ruido para mejorar la legibilidad de las inscripciones. Además, se utilizarán técnicas de filtrado de texto para eliminar el fondo y hacer que el texto sea más fácil de leer.
    b. Se utilizarán técnicas de comparación de imágenes como correlación cruzada, detección de puntos de interés y coeficientes de similitud estructural para correlacionar los escudos que presentan similitudes. Estas operaciones tienen como objetivo agrupar los escudos en grupos donde los escudos de cada grupo sean similares para facilitar operaciones futuras sobre cada grupo en función de sus similitudes.
7. *Realizar un análisis detallado de los hallazgos obtenidos durante el proceso de recopilación de información sobre relieves de piedra.* Se pueden mostrar los resultados en gráficos o comparar los resultados con los datos originales para lograrlo. Además, es crucial considerar los objetivos del proyecto y determinar si los resultados obtenidos cumplen con ellos. Además, se deben analizar las posibles limitaciones del proceso y encontrar soluciones para mejorar la calidad de los resultados.

Se puede afirmar que el logro del objetivo principal del proyecto se verá significativamente beneficiado por la ejecución adecuada de estos subobjetivos.

2. Marco teórico

En este apartado se explicarán en detalle las bases teóricas que sustentan las tecnologías utilizadas en el proyecto, que incluyen la red neuronal YOLOv5 y conceptos más generales de *Inteligencia Artificial*, *Redes Neuronales* y *Visión Artificial*.

2.1. Aprendizaje profundo (*Deep Learning*)

El *Machine Learning* es una subcategoría del aprendizaje automático y se define como una rama de la Inteligencia Artificial que busca dar a las computadoras la capacidad de aprender a partir de un conjunto de datos de entrada y generalizar su conocimiento para resolver problemas específicos. Las redes neuronales profundas, que tienen capas ocultas entre la capa de entrada y la capa de salida, son esenciales para el aprendizaje profundo. En la actualidad, estas redes pueden ser muy grandes, con miles de capas ocultas, y han logrado resultados impresionantes en una variedad de tareas, como el reconocimiento de imágenes, la clasificación de texto y el procesamiento del lenguaje natural, entre otras. Luego se abordará los conceptos teóricos de las redes neuronales profundas.

2.1.1. Redes neuronales profundas (RNP)

Algunos tipos de redes neuronales artificiales están estructuradas en capas, y cada capa recibe la salida de la capa anterior en su entrada. Las redes más básicas están formadas únicamente por una capa de entrada, donde se recopila la información necesaria para tratar el problema, y una capa de salida, que calcula el resultado utilizando pesos, sesgos y funciones de activación específicas.

Las *RNP* son la base del aprendizaje profundo y se extienden a modelos de redes tradicionales como el *Perceptrón Multicapa (MLP)*. Las *RNP* incluyen una gran cantidad de capas intermedias con múltiples funcionalidades entre la entrada y la salida, conocidas como "capas ocultas". Las neuronas de una capa están completamente conectadas con las neuronas de la siguiente capa [3].

El algoritmo de *backpropagation* [4] y el descenso del gradiente [5] son dos ideas fundamentales que sustentan el funcionamiento de las *RNP*. La red puede aprender automáticamente los pesos y sesgos ideales para resolver un problema [3].

Una de las principales desventajas de este tipo de redes es el alto costo computacional asociado con las interconexiones entre todas las neuronas de la capa anterior y posterior. Además, esta característica puede aumentar la probabilidad de *overfitting*, un efecto que ocurre cuando el algoritmo de aprendizaje se entrena demasiado con ciertos datos para los cuales se sabe el resultado, lo que puede llevar a que la red memorice patrones específicos en lugar de generalizar a nuevos patrones [6].

2.1.2. Redes neuronales convolucionales (CNN)

Las *CNN* han revolucionado la *Visión Artificial*. Gracias a ellas se han logrado importantes avances en el campo, que actualmente es uno de los más utilizados y se considera un conjunto de sistemas clave para resolver una variedad de problemas relacionados con el tratamiento y clasificación de imágenes, incluyendo el problema de detectar ciertas clases de objetos en las imágenes [3].

La capacidad de las *CNN* para resolver problemas de clasificación y detección de objetos en imágenes ha sido un gran avance en el campo de la *Visión Artificial*. De hecho, en la actualidad se utilizan ampliamente y se consideran el estado del arte en este campo.

La construcción de redes neuronales artificiales comienza con modelos teóricos más básicos, como el *Perceptrón*. Estas redes tienen como objetivo emular la interacción entre topologías de configuraciones de neuronas dispuestas. Las *CNN*, una evolución del *MLP*, son uno de los modelos más conocidos y utilizados. Para comprender cómo funcionan estas redes, primero debemos entender qué es una convolución.

La convolución, una función matemática clave en el funcionamiento de las *CNN*, permite aplicar un *kernel* o matriz con unos pesos específicos a una porción de la imagen, normalmente una región local del mismo tamaño. Al ajustar estos pesos, se pueden extraer diferentes características de las imágenes. Estos resultados se pueden ver como mapas de características. Algunos *kernels* consideran la dimensión espacial, lo que permite considerar las celdas adyacentes en las operaciones y obtener un conocimiento de conjunto que facilita la clasificación de patrones.

Además, es posible combinar esta operación matemática en secuencia con una agrupación de *maxpooling* para disminuir la resolución de la imagen y utilizar convoluciones posteriores, manteniendo los mapas de características extraídos. Esta técnica de agrupación mejora la generalización del modelo al eliminar gradualmente información redundante y ruido de la imagen.

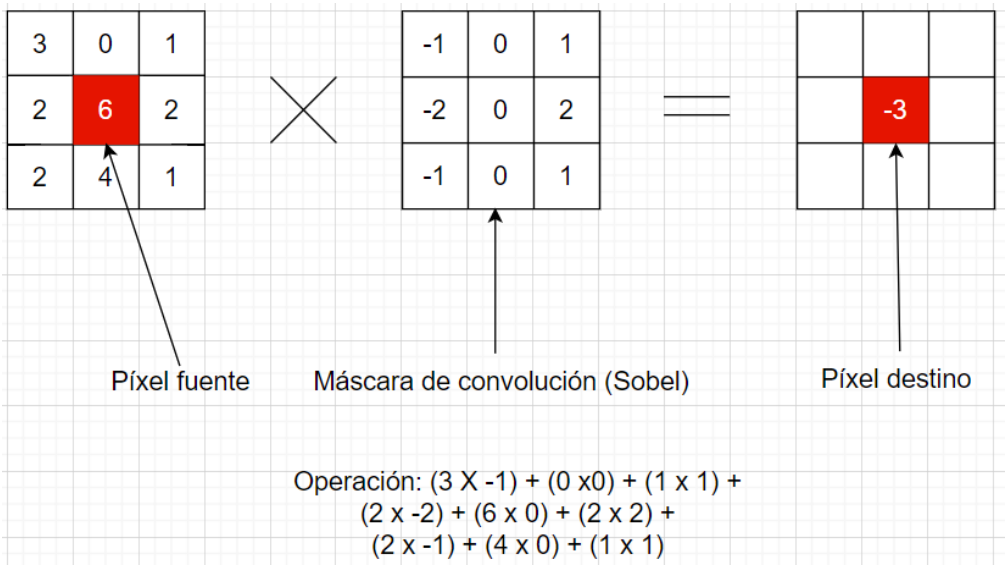

Fig. 4: Ejemplo de convolución de Sobel.

La convolución, en este caso con un filtrado de Sobel, se puede ver en la Figura 4. Este proceso implica multiplicar los valores de la máscara por cada valor de la capa de entrada y luego sumar los valores de la vecindad correspondiente. Esta operación produce la capa convolucionada. Es importante tener en cuenta que la operación no puede aplicarse a los bordes de la imagen, lo que resulta en capas convolucionadas cada vez más pequeñas. En ocasiones, es necesario reducir la resolución de las capas convolucionadas para mejorar el rendimiento computacional del modelo. En este caso, se emplea la técnica de *pooling*.

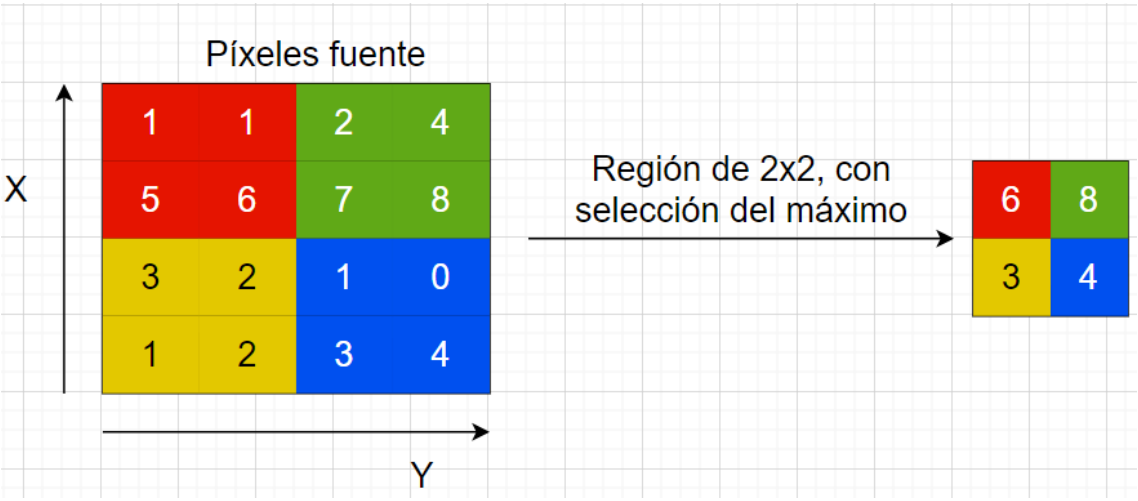

Fig. 5: Ejemplo de pooling.

Las *CNN* utilizan el *pooling* para reducir la resolución de las imágenes. El proceso es simple: se elige una región de píxeles y se realiza una operación de reducción, como tomar el valor máximo o promedio de la región. Esta operación produce un nuevo valor en la capa de reducción. La Figura 5 muestra cómo se utiliza esta estrategia para mejorar el desempeño computacional del modelo al reducir la cantidad de datos que deben procesarse en capas posteriores.

Las *CNN* son *RNP*, una evolución de las redes neuronales clásicas. La principal diferencia entre las *RNP* y otras redes neuronales es que las *RNP* introducen capas ocultas entre la capa de entrada y la capa de salida. Las operaciones de convolución que dan sentido y nombre a esta red se llevan a cabo en estas capas ocultas.

Las *CNN* tienen la intención de imitar el funcionamiento del córtex visual del cerebro humano, que detecta patrones y características espaciales en las imágenes, como cambios de contraste, superficies planas, formas geométricas y patrones circulares, entre otros [7]. La capacidad de las *CNN* para realizar varias convoluciones en cascada, que pueden extraer varias características importantes de la imagen, es lo que las hace tan poderosas. La red ajusta los pesos y tamaños del *kernel* durante el proceso de entrenamiento para aprender a extraer los mapas de características necesarios para resolver el problema en cuestión.

La capacidad de realizar convoluciones sucesivas es fundamental para que una red neuronal convolucional funcione correctamente. Cada convolución aplica un *kernel* o filtro a un segmento de la imagen de entrada, produciendo un mapa de características que destaca patrones o características específicas de la imagen. Una extracción de características jerárquica y progresiva se logra mediante el uso de múltiples convoluciones, donde cada capa aprende a reconocer patrones de mayor complejidad y abstracción.

Además, generalmente se aplica una capa de submuestreo o *pooling* después de cada capa de convolución, que implica reducir la resolución de la imagen. Esto se hace para capturar las características invariantes a escala y traslación de manera más efectiva, así como para reducir la cantidad de parámetros de la red y evitar el *overfitting*. Al aplicar *pooling* después de cada capa de convolución, la resolución espacial de la imagen disminuye a la par con el aumento de la profundidad de la red. Esto permite capturar información relevante en una variedad de escalas y niveles de abstracción, lo que permite extraer características locales y globales de la imagen.

2.1.3. YOLO y su evolución

YOLO, que significa "Solo miras una vez" en español, es una red neuronal convolucional diseñada para detectar objetos en imágenes. Se trata de una red de tipo *one-stage*, lo que significa que predice tanto la región donde se encuentra el objeto de interés como la clase a la que pertenece dentro de un conjunto de categorías predefinidas en una sola pasada.

Desde su creación en 2015, Joseph Redmon ha mejorado y actualizado la aplicación. Al ser YOLO una red *one-stage*, es más rápida y eficiente que otras redes como *DPM* y *R-CNN*. YOLO también usa un método innovador para dividir la imagen en celdas y realizar la detección en cada una de ellas, lo que le permite detectar objetos de diferentes tamaños y formas con mayor precisión [8]. YOLO ha demostrado ser superior en métricas como el *mean avergae precision* (mAP) en comparación con otras redes [8].

Posteriormente, se han desarrollado versiones mejoradas del modelo original, entre las que se incluyen:

- YOLOv2

Una versión mejorada del modelo original se lanzó en 2016, y Ali Farhadi se unió al proyecto. Esta versión, YOLOv2, mejoró la velocidad y la precisión del modelo. La introducción del concepto de *anchor box*, que son áreas predeterminadas dentro de la imagen que representan la posición ideal del objeto a detectar, fue la mejora principal de esta versión. Se utiliza un umbral de IoU (*Intersection over Union*) para determinar si estas áreas predefinidas son consideradas parte de la detección. Además, *Darknet19*, que está formado por 19 capas convolucionales y 5 capas de agrupamiento, sirve como *backbone* para la red neuronal [9].

- YOLOv3

YOLOv3 fue lanzado en 2018 y es una versión mejorada de YOLOv2 creada por los mismos creadores. Esta versión ofrece una mejora adicional al aumentar la velocidad de detección por cuatro mientras mantiene el mismo *mAP* a 0,5 de IoU (un valor de la métrica *mean average precisión* para un umbral de *intersection over union* de 0,5). Para mejorar el rendimiento, los investigadores cambiaron el modelo de *backbone* a *Darknet53*, redujeron el tamaño del modelo a 75 capas convolucionales e implementaron la técnica de optimización *Pyramid Network* [8].

- YOLOv4

La versión 4 de YOLO, que se lanzó en 2020, es una de las versiones más recientes de la serie. El equipo de desarrollo liderado por Alexey Bochoknovskiy, Chien-Yoa Wang y Hong-yuan Mark Liao mejoró la velocidad de detección y la precisión del modelo en esta versión. YOLOv4 utiliza un modelo de *backbone* más grande y complejo llamado *CSPDarknet53*, con más de 50 capas convolucionales, a diferencia de las versiones anteriores. El equipo de desarrollo también implementó técnicas avanzadas de *data augmentation* y una arquitectura de red neuronal basada en *CSPNet*, que mejora la eficiencia de la propagación hacia atrás. YOLOv4 superó la velocidad y la precisión de los modelos anteriores con estas mejoras [11].

En la investigación, los investigadores presentan la idea de *Bag of Freebies*, que incluye una variedad de métodos de optimización mediante el aumento de datos. Estas técnicas mejoran la precisión de la red, pero disminuyen la velocidad. Algunos de estos métodos incluyen:

1. *Mixup*: se basa en la combinación aleatoria de pares de imágenes y sus respectivas etiquetas de clasificación para crear una nueva imagen y una etiqueta asociada. Dado que obliga a la red neuronal a aprender rasgos más amplios y sólidos, esta técnica ayuda a mejorar la generalización de la red neuronal.
2. Mosaic Data Augmentation: combina cuatro imágenes en una sola imagen. Consiste en dividir la imagen principal en cuatro partes y luego combinarlas con otras imágenes seleccionadas al azar. De esta manera, se crea una imagen compuesta que contiene una variedad de objetos y fondos, lo que mejora la capacidad de la red neuronal para reconocer objetos en una variedad de posiciones y contextos.
3. CutMix: mejora la generalización y la robustez del modelo. Esta técnica crea una imagen combinada cortando y pegando partes de una imagen en otra. De esta manera, se obliga al modelo a aprender características más fuertes y a generalizar mejor para imágenes que contienen objetos parcialmente cubiertos o superpuestos.
4. Dropblock Regularization: elimina aleatoriamente ciertos píxeles de la imagen de entrada para ayudar a la red a aprender características más sólidas y generalizables. Esto reduce el *overfitting* y mejora la precisión del modelo en imágenes con difuminado o ruido. Al eliminar píxeles, la red se hace más fuerte ante perturbaciones o deformaciones al aprender a reconocer objetos a partir de fragmentos de la imagen.

Además, se incluyen mejoras adicionales que los autores denominaron *Bags of Specials* (BOS). Estas estrategias de optimización aumentan la precisión del modelo, pero aumentan el costo computacional de la inferencia. Algunos de estos métodos incluyen:

- *Una amplia gama de funciones de activación no lineales*: como *ReLu*, *SELu*, *Leaky* y *Mish*. El modelo puede aprender patrones más complejos y no lineales en los datos gracias a estas funciones, lo que mejora la capacidad de detección. El rendimiento del modelo puede depender en gran medida de la función de activación elegida, que dependerá en gran medida de las características específicas del conjunto de datos y del modelo utilizado.

- *Non-maximum suppression (NMS)*: es una técnica utilizada en la etapa posterior al procesamiento de los resultados de detección de objetos en imágenes. Esta estrategia se emplea para eliminar detecciones redundantes que se solapan mucho entre sí.

  Después de la fase de detección, generalmente se obtienen varios cuadros delimitadores que encierran el objeto detectado, cada uno de los cuales tiene una puntuación de confianza que indica la probabilidad de que contenga el objeto. El método de *NMS* selecciona el cuadro delimitador con la mayor puntuación de confianza y elimina todos los cuadros delimitadores que se solapan mucho con él.

  Como resultado, *NMS* permite disminuir la cantidad de detecciones incorrectas y aumentar la precisión general del modelo de detección.
- *Spacial attention modules (SAM):* Mejoran la precisión de la detección en imágenes con objetos pequeños o de alta densidad de objetos.

- YOLOv5

Glenn Jocher tomó el mando del proyecto YOLO y lanzó una nueva versión en 2020. Al incorporar PyTorch, esta actualización modernizó el marco y se ajustó a las tendencias del sector y las preferencias de la comunidad. Además, es importante mencionar que todo el código fuente de esta versión está disponible como software libre en GitHub [10]. La incorporación de técnicas *CSP* (*Cross-Stage Partial Network*) y *PAN* (*Path Aggregation Network*), así como el aprendizaje automático de los *anchor-box* son novedades en esta versión.

A continuación, se presentan algunas de las partes más importantes de la arquitectura y el funcionamiento del modelo YOLOv5.

2.1.4. Arquitectura de YOLOv5

Aunque ha recibido varias mejoras en cada iteración, la arquitectura de YolOv5 sigue siendo la misma que la de sus predecesoras. El cambio del *framework DarkNet* por PyTorch es la diferencia más notable entre YOLOv4 y YOLOv5. Esto convierte al proyecto en software libre y permite una implementación más adaptable y un entrenamiento de GPU más rápido.

Es una red neuronal convolucional con múltiples bloques de capas convolucionales intercalados con capas de normalización por lotes y *pooling*. Las redes convolucionales modernas utilizan esta arquitectura, que permite la extracción de características de la imagen a diferentes escalas.

Las capas convolucionales con la función de activación *ReLu* se utilizan en YOLOv5 para procesar imágenes y crear mapas de características que representan objetos y sus ubicaciones en la imagen.

YOLOv5 utiliza otras técnicas para mejorar la precisión y la velocidad de la detección de objetos, como la ya mencionada *SAM*, además de las capas convolucionales. Para mejorar la generalización del modelo y evitar el sobreajuste, también se utilizan técnicas como el mosaico y el *mixup*.

Por último, pero no menos importante, YOLOv5 es una red convolucional altamente optimizada que utiliza técnicas avanzadas de procesamiento de imágenes para la detección de objetos eficiente y precisa.

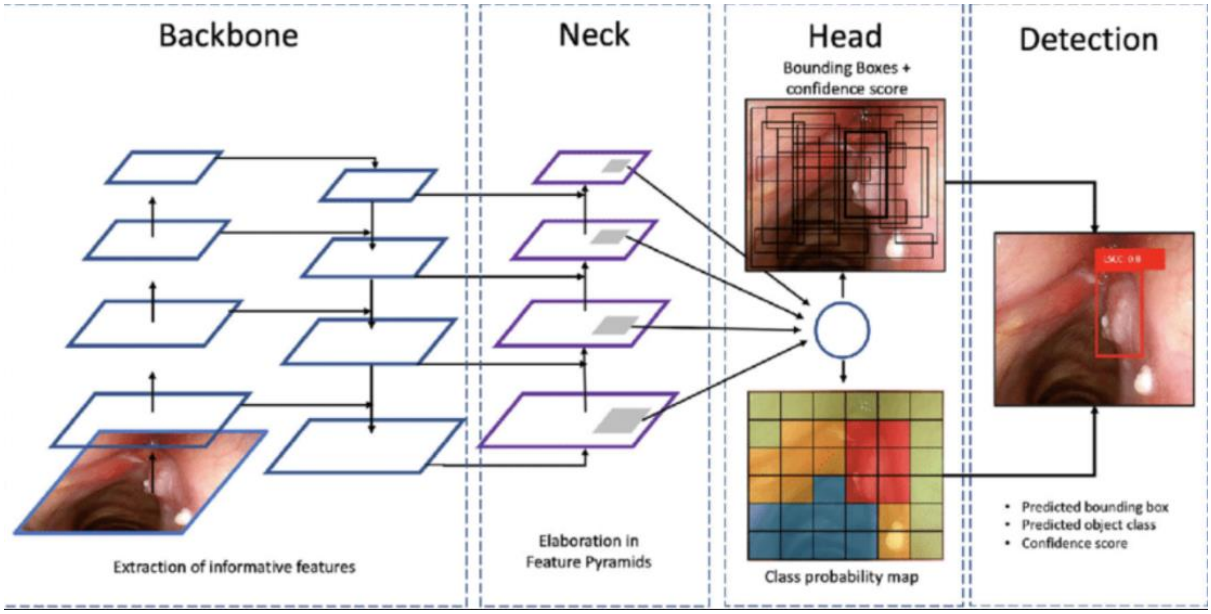

Fig 6: Arquitectura de YOLOv5. Imagen extraída de [21]

La Figura 6 muestra la estructura de YOLOv5, donde se puede ver que la red recibe una imagen de 640×640, aunque esto puede ser modificado, y utiliza varias capas de convolución y conglomeración para extraer mapas de características. Las dimensiones del tensor resultante son:

$$s \times s \times (b \times 5 + c)$$

Aquí, b es el número de cajas delimitadoras por celda, c es el número de clases y s es el tamaño de la rejilla utilizada para dividir la imagen de entrada. Es importante destacar que YOLOv5 permite trabajar con una gran variedad de tamaños de rejilla, lo que permite adaptarse a una variedad de tipos de objetos y resoluciones de imágenes. Por ejemplo, la Figura 7 muestra que s = 7.

Es fundamental comprender cómo YOLO gestiona el flujo de datos. Como se indicó anteriormente, YOLO divide la imagen en una cuadrícula de celdas, como se muestra en la Figura 7. Se requiere que el modelo proporcione tantas predicciones para cada celda como indica el parámetro b; estas predicciones se ajustarán a las *anchor box* definidas previamente. Luego, se utilizan los métodos de validación antes mencionados para determinar si la detección es válida.

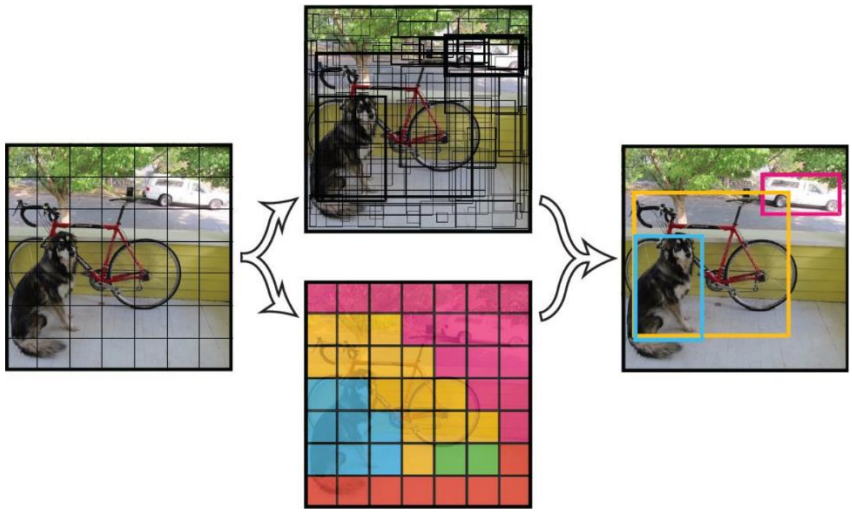

Fig. 7: Ejemplo de división en celdas (s = 7) y detección de objetos. Imagen extraída de [22]

Aunque no se utilizarán versiones posteriores a YOLOv5 en el proyecto, se describirán brevemente las mejoras que presentan en relación con YOLOv5.

- YOLOv7

YOLOv7 mejora significativamente la precisión de detección de objetos en tiempo real sin aumentar el costo de inferencia. YOLOv7 puede lograr una mayor velocidad de inferencia y una mayor precisión de detección, además de reducir de forma efectiva alrededor del 40% de los parámetros y el 50% del cálculo de las detecciones de objetos en tiempo real más avanzadas.

En general, YOLOv7 proporciona una arquitectura de red más rápida y sólida que ofrece un método de integración de características más eficiente, un rendimiento de detección de objetos más preciso y una mayor eficiencia en la asignación de etiquetas y el entrenamiento de modelos.

Como resultado, YOLOv7 se puede ejecutar en un hardware informático mucho más barato que otros modelos de aprendizaje profundo. Sin pesos previos de entrenamiento, se puede entrenar mucho más rápido en conjuntos de datos más pequeños.

- YOLOv8

YOLOv8 es la versión más reciente de la serie de detectores de objetos en tiempo real YOLO y ofrece una velocidad y precisión superiores. YOLOv8 incorpora nuevas funciones y optimizaciones basadas en los avances de las versiones anteriores de YOLO, lo que lo convierte en la mejor opción para una variedad de tareas de detección de objetos en una variedad de aplicaciones. Algunas de sus características clave son:

- *Arquitecturas de backbone y neck avanzadas*: YOLOv8 utiliza arquitecturas de *backbone* y *neck* de última generación, lo que mejora la extracción de características y el rendimiento de la detección de objetos.
- *Head Ultralytics anchor-free*: YOLOv8 adopta un *head Ultralytics anchor-free*, que contribuye a una mayor precisión y a un proceso de detección más eficaz en comparación con los enfoques basados en *anchor-box*.
- *Compromiso optimizado entre precisión y velocidad*: YOLOv8 es adecuado para tareas de detección de objetos en tiempo real en una variedad de áreas de aplicación porque se enfoca en mantener un equilibrio ideal entre precisión y velocidad.
- *Varios modelos preentrenados*: YOLOv8 proporciona una variedad de modelos preentrenados para satisfacer una variedad de tareas y requisitos de rendimiento, lo que facilita la selección del modelo adecuado para su caso de uso específico.

2.2. Preprocesado de imágenes de bajo contraste y con ruido

Se realizan operaciones adicionales para identificar los elementos deseados, en este caso escudos e inscripciones grabados en piedra:

- Mejorar la comprensión y el análisis de las detecciones que son inscripciones.
- Comparar escudos con otros similares.

Sin embargo, para facilitar estas operaciones, primero debemos hacer una mejora previa de las detecciones. Los postprocesos más importantes se enumeran a continuación.

2.3. Mejora del texto en imágenes de inscripciones sobre piedra basado en componentes conexas

Se ha utilizado un procedimiento específico para mejorar la legibilidad de las inscripciones encontradas:

1. Convertimos la imagen a escala de grises porque la legibilidad de la inscripción no depende del color y agilizamos el procesamiento al eliminar las capas de color.
2. Con el objetivo de segmentar la inscripción del fondo de la imagen de manera óptima, realizamos un umbralizado adaptativo utilizando una máscara gaussiana. Esta técnica ajusta el umbral localmente en cada región de la imagen en función de las variaciones de intensidad locales. Al aplicar una ecuación de *Gaussiana*, podemos suavizar los valores de píxeles y reducir el impacto de las irregularidades o el ruido en la imagen.
3. Usamos una técnica para eliminar las áreas oscuras de la imagen. Esto nos permite comparar cada píxel de la imagen binaria que se obtuvo después del umbralizado adaptativo con el umbral que se estableció por el valor medio de la imagen. Al llevar a cabo esta comparación, creamos una máscara binaria en la que los píxeles con valores que están por debajo del umbral se clasifican como verdaderos (valor 255) mientras que los píxeles que representan las regiones negras correspondientes a la inscripción son representados como falsos (valor 0).
4. Eliminamos las regiones pequeñas de las regiones oscuras de la imagen original, calculadas en el capítulo 3, cuyo tamaño es menor que el valor promedio. Este proceso elimina ruidos o elementos no deseados que puedan dificultar la lectura de la inscripción.

5. Finalmente, creamos una imagen binaria en la que los píxeles de valor máximo (255) resaltan las áreas de inscripción, mientras que el resto de la imagen se mantiene en cero. Esto facilita el análisis y la legibilidad posterior de la inscripción al proporcionar una representación visual clara.

2.4. Correlación de los escudos basado en Similaridad Estructural

Nuestro objetivo cuando se detecta un escudo es identificar características que son comunes a todas las detecciones de escudos. Esto nos permitirá realizar una clasificación posterior basada en la similitud entre las detecciones, lo que facilitará la diferenciación en un análisis futuro por épocas, tipos, familias u otros criterios. Se han seguido los siguientes pasos para lograrlo.

1. Para garantizar la consistencia en la representación de los escudos detectados, redimensionamos las detecciones después de pasarlas a escala de grises a un tamaño de 416×416 píxeles.
2. Creamos una matriz que utilizaremos como ruido aleatorio. Esta matriz sigue una distribución *Gaussiana* similar a la de la imagen.
3. Sumamos la matriz de ruido creada anteriormente con la imagen del escudo. Este paso se omitirá una vez que se haya comprobado la robustez del algoritmo al procesar una imagen "dañada" con ruido y comparar su rendimiento con otros algoritmos.
4. Usamos el método de similaridad estructural, que se explicará a continuación, para calcular el índice de similitud entre los escudos. Le pasamos las dos imágenes de los escudos para comparar. Dado que las imágenes son a color, el parámetro *multichannel* se establece a *True*, lo que nos permite evaluar la similitud basada en la estructura y contenido de los canales de color.

2.4.1. Índice de Similaridad Estructural (SSIM)

El *SSIM* [15] es una medida que compara cómo se parecen dos imágenes en términos de su estructura o distribución espacial de píxeles. Se utiliza para evaluar la similitud entre dos imágenes en su organización interna. La fórmula matemática completa de la *SSIM* es la siguiente:

$$SSIM\ (x, y) = \frac{(2\mu_x\mu_y\ + c_1)(2\sigma_{xy}\ +\ c_2)}{(\mu_x^2\ +\ \mu_y^2\ +\ c_1)(\sigma_x^2\ +\ \sigma_y^2\ +\ c_2)}$$

donde:

- x e y representan las dos imágenes que se comparan;
- μ_x y μ_y son las medias de intensidad de los píxeles en las imágenes x e y respectivamente;
- σ_x^2 y σ_y^2 son las varianzas de intensidad de los píxeles en las imágenes x e y, respectivamente;
- σ_{xy} es la covarianza de intensidad entre los píxeles de las imágenes x e y; y
- c_1 y c_2 son constantes pequeñas que se utilizan para evitar divisiones por cero y para estabilizar el cálculo;

La *SSIM* calcula una medida general de similitud utilizando las medias, varianzas y covarianzas de intensidad de los píxeles en las imágenes x e y. Esta medida tiene en cuenta tanto las intensidades promedio de los píxeles (representadas por μ_x y μ_y) como las variaciones de intensidad en las imágenes (representadas por σ_x^2, σ_y^2 y σ_{xy}). Un valor *SSIM* alto (el valor de *SSIM* varía entre -1 y 1) indica que las imágenes son similares en términos de su estructura espacial, mientras que un valor bajo indica que son diferentes.

2.4.2. Otros métodos utilizados

Sin embargo, podemos correlacionar las detecciones de escudos con otros métodos además del *SSIM*. Existen métodos alternativos que se han utilizado y se explicarán brevemente en la sección de Experimentos; estos métodos también pueden utilizarse:

- *Detector de puntos de interés*: Podemos usar un detector de puntos de interés, particularmente un detector *SIFT* [16], iterando sobre las detecciones. Una vez que se identifican los puntos de interés y los descriptores de las imágenes, se selecciona un descriptor del primer conjunto y se intenta emparejarlo mediante un algoritmo de fuerza bruta, también conocido como *Brute Force matcher*. Además, se utiliza un algoritmo de distancias, particularmente *KNN*, para emparejar todos los descriptores del segundo conjunto [17].

- *Cálculo de la diferencia de imágenes*: el método *absdiff* de OpenCV es otro método para calcular la resta de las dos detecciones, pero es menos preciso. Este método reduce los valores de intensidad de las dos imágenes píxel a píxel y obtiene el valor absoluto de la diferencia. Se crea una nueva imagen utilizando estos valores absolutos para destacar las diferencias entre las dos imágenes originales y mostrar las áreas donde las intensidades de los píxeles difieren en mayor medida.
- *Índice de correlación*: el método *matchTemplate* de OpenCV es una opción adicional. Este método realiza una búsqueda de plantilla en una imagen utilizando una técnica de correlación de coeficientes normalizados. Este método permite iterar sobre las detecciones y generar una imagen de coincidencias, en la que los valores de intensidad representan la similitud entre la plantilla y las regiones superpuestas en la imagen de entrada. Los valores altos en la imagen de coincidencias muestran áreas en la imagen de entrada que son similares a la plantilla, mientras que los valores bajos muestran áreas que son diferentes.

3. Estado del arte

En relación con los trabajos publicados sobre el problema abordado en este proyecto, se han identificado los siguientes estudios:

- **«Restoring ancient text using deep learning: a case study on Greek epigraphy»** [18]:

El artículo presenta Pythia, un modelo de restauración de textos antiguos que utiliza *RNP* para recuperar los caracteres que faltan en textos dañados. La arquitectura del modelo se ha diseñado cuidadosamente para manejar información contextual a largo plazo y tratar eficazmente las representaciones de caracteres y palabras dañadas. El modelo se entrena utilizando un proceso para convertir el mayor corpus digital de inscripciones griegas antiguas en texto procesable por la máquina, llamado *PHI-ML*. Las predicciones de Pythia en *PHI-ML* alcanzan una tasa de error de caracteres del 30,1%, en comparación con el 57,3% de los epigrafistas humanos. Además, en el 73,5% de los casos, la secuencia real se encontraba entre las 20 mejores hipótesis de Pythia, lo que demuestra el impacto de este método de asistencia en el campo de la epigrafía digital y establece el estado del arte en la restauración de textos antiguos.

- **«Estimating Gothic Facade Architecture from Imagery»** [19]:

El artículo propone un método para estimar la forma de los elementos de mampostería en la fachada de un edificio gótico a partir de una sola imagen. El enfoque consta de dos pasos: la estimación de arcos y aberturas rectangulares, y la estimación de la mampostería que rodea estas estructuras. Se utilizan algoritmos de detección y estimación basados en contornos activos con restricciones para preservar la forma. Los elementos se agrupan en función de su similitud de forma y posición, y se repiten varias veces para extraer disposiciones jerárquicas. Los píxeles no incluidos como parte de los elementos arquitectónicos se consideran mampostería y se segmentan en mortero y ladrillos. Este enfoque promete ser una forma eficaz de obtener un modelo detallado de los componentes reales de la fachada de un edificio gótico a partir de imágenes, lo que puede facilitar los esfuerzos de conservación y proporcionar mediciones cuantitativas para la investigación en la construcción de estos edificios.

- **«Inscriptions and Images in Secular Buildings: Examples from Renaissance Scania, Sweden, ca. 1450–1658»** [20]:

El artículo examina cómo los habitantes del Renacimiento en Escania, Suecia, inscribían su identidad en los edificios a través de tablillas de piedra y vigas de madera con inscripciones, imágenes y fechas. Se analiza la colocación de estos objetos sobre las puertas de residencias nobles en áreas rurales o edificios de la élite urbana, y se investiga quiénes podían ver y entender los mensajes transmitidos. El artículo también explora cuándo, cómo y por qué surgió esta tradición de inscribir objetos en los edificios en el contexto escandinavo, destacando cuestiones de identidad e individualidad.

- **«A Method for Extracting Text from Stone Inscriptions Using Character Spotting»** [23]:

Este artículo presenta una técnica interactiva para extraer caracteres de texto de imágenes de inscripciones en piedra. La técnica está diseñada para procesar imágenes en el lugar de la adquisición, en palacios, monumentos y templos históricos. Se basa en elementos robustos de análisis de caracteres, como características *HoG*, diacríticos vocálicos y líneas de escaneo delimitadas. El proceso permite la localización de caracteres y la extracción de información inscrita para convertirla en texto editable, lo cual podría ayudar a los arqueólogos en tareas de epigrafía, transliteración y traducción de inscripciones rupestres, especialmente aquellas que están degradadas, con ruido o con diferentes estilos debido a su origen. Los caracteres manchados también pueden utilizarse para crear una *BD* para el análisis de escrituras antiguas y trabajos arqueológicos relacionados. La técnica ha sido probada en inscripciones en piedra en sitios patrimoniales de Karnataka, India, con resultados prometedores. Además, se ha desarrollado una aplicación Android para ayudar a los epigrafistas en el estudio de las inscripciones utilizando tabletas o teléfonos móviles.

Como conclusión del Estado del Arte, se destaca lo siguiente:

Se ha observado que las redes neuronales se utilizan para detectar y clasificar elementos específicos, como inscripciones y elementos de fachadas. Sin embargo, los problemas de detección y procesamiento de escudos e inscripciones utilizando una misma red han demostrado una falta de aplicación de redes neuronales.

Según este análisis, existe la posibilidad de investigar y desarrollar una técnica que utilice una red neuronal capaz de abordar simultáneamente la detección y procesamiento de inscripciones y escudos. Al hacerlo, se podría encontrar una solución más completa y efectiva para este problema específico. Esta propuesta mejoraría la automatización y precisión de los sistemas existentes y abriría nuevas oportunidades en el campo de la detección de elementos en imágenes.

4. Métodos y solución informática.

Este capítulo detallará la técnica utilizada para recopilar el conjunto de datos del proyecto. Además, se proporcionarán detalles sobre la manipulación y procesamiento del *dataset*, así como la solución informática utilizada para resolver el problema abordado en este trabajo.

4.1. Bases de datos de imágenes de relieves

Dado que no existe *BD* de imágenes de relieves e inscripciones grabadas en piedra y anotadas, este proyecto requiere la creación de una. Se han empleado fotografías tomadas por el autor en diferentes lugares de España, como Madrid, Ávila, Toledo y Santiago de Compostela, entre otros, desde el siglo XIV hasta el siglo XIX, así como imágenes tomadas de Internet, todas ellas de acceso y uso gratuito.

4.2. Preproceso de los datos

En nuestro caso, todas las imágenes que vamos a utilizar deben redimensionarse a un tamaño de 416×416 píxeles antes de usar la herramienta de anotación. Además, se pueden eliminar elementos no deseados en la imagen, como el cielo, los edificios y la vegetación. Aunque estos componentes no dañarán la imagen, pueden causar problemas al entrenar la red neuronal al introducir ruido innecesario. Antes del preproceso, la Figura 8 muestra una imagen con escudo, mientras que la Figura 9 muestra la misma imagen después del preproceso.

Fig. 8: Imagen antes del recorte

Fig. 9: Imagen después del recorte

El programa *Roboflow* [24] se utilizará para preprocesar y etiquetar las imágenes. El brillo de las imágenes se modificó, así como la exposición y el ruido.

4.3. Anotación de las imágenes (Herramienta Roboflow)

Roboflow [24] es una herramienta para crear aplicaciones de *Visión Artificial*. En un principio, su utilidad se limitaba a facilitar el manejo de grandes conjuntos de imágenes y simplificar las tareas de etiquetado y empaquetado. Actualmente, *Roboflow* puede realizar todos los procesos auxiliares de un sistema de *Visión Artificial*, desde la recopilación de datos hasta el preprocesamiento, y mostrar los resultados en una interfaz amigable.

Las funciones de esta herramienta incluyen:

- Cargar anotaciones ya existentes en formatos reconocidos por la herramienta, como XML, JSON, COCO y YAML, entre otros.

La interfaz sencilla y fácil de entender de *Roboflow* se muestra en la Figura 10.

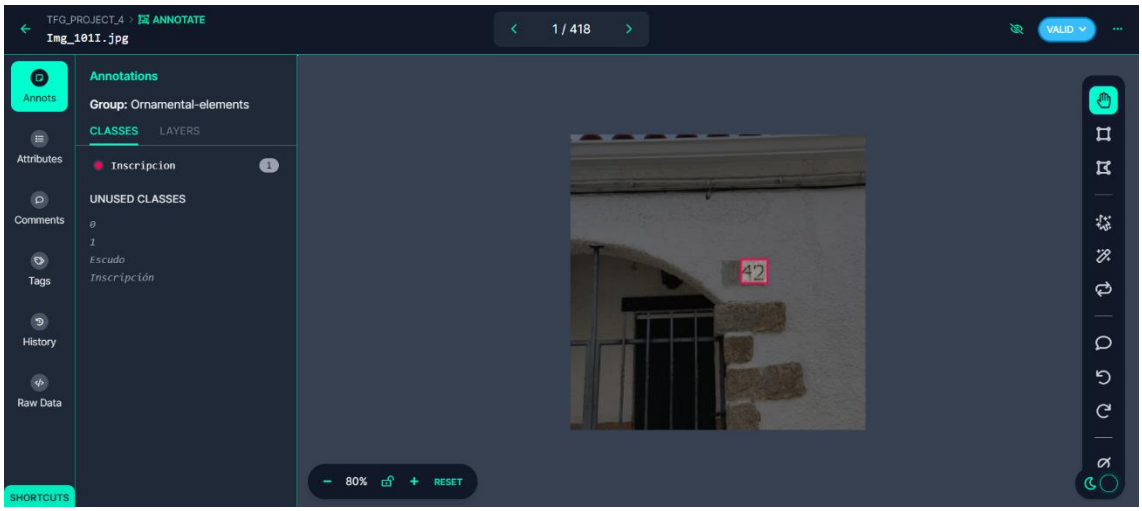

Fig. 10: Etiquetado con la herramienta Roboflow

Las siguientes son algunas de las operaciones importantes que se pueden realizar con Roboflow:

- *Conversión de anotaciones de formato*: Útil para cambiar de red neuronal sin tener que etiquetar manualmente las imágenes.
- *Preprocesamiento de imágenes*: permite ajustar el color y el contraste, transformar las imágenes a escala de grises y reescalarlas. Todas las imágenes de este proyecto se redimensionaron a una resolución de 416×416 píxeles, que es el tamaño de la capa de entrada de la red neuronal utilizada para la detección.
- *Capacidades de data augmentation o aumento de datos*: Esto permite crear nuevas imágenes a partir de las originales utilizando técnicas matemáticas como rotaciones, escalados, espejos de imágenes y adición de ruido aleatorio. La Figura 11 muestra el resultado de la operación de *data augmentation* para ajustar el brillo para aclarar y oscurecer las imágenes.

Fig. 11: Operación de brillo utilizando Data Augmentation en Roboflow

Una ilustración del procesamiento de una imagen de un conjunto de datos utilizando la operación de brillo (*brightness*) se muestra en la Figura 11.

- Funcionalidad de control de versiones de *Roboflow* para crear diferentes instantáneas del conjunto de datos en función de los experimentos realizados, como se muestra en la Figura 12. Esto permite mantener un registro de las modificaciones realizadas en el conjunto de datos durante el proyecto y facilita la comparación de los resultados entre diferentes versiones.

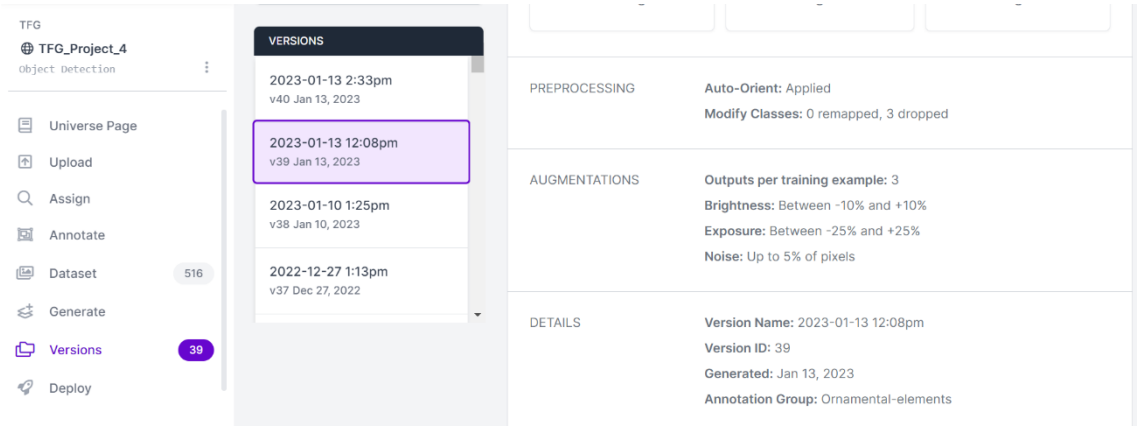

Fig. 12: Versionado del dataset con Roboflow

- El entrenamiento de modelos neuronales a partir de conjuntos de datos no se ha utilizado en este proyecto, debido a los límites de presupuesto. Pero para aquellos que deseen usar su plataforma para entrenar sus modelos y aprovechar sus habilidades de procesamiento y optimización de modelos de visión artificial, *Roboflow* ofrece esta funcionalidad.

4.4. Solución propuesta

En este proyecto, la *Visión Artificial* se utiliza para reconocer escudos e inscripciones grabadas en piedra. Debido a sus características de alta precisión y velocidad de cómputo, el modelo de red neuronal de YOLOv5 se eligió para crear un sistema de detección de objetos en imágenes. Esto lo sitúa en la vanguardia de los detectores *one-stage* de objetos. Un diagrama de la estructura de la solución se muestra en la Figura 13.

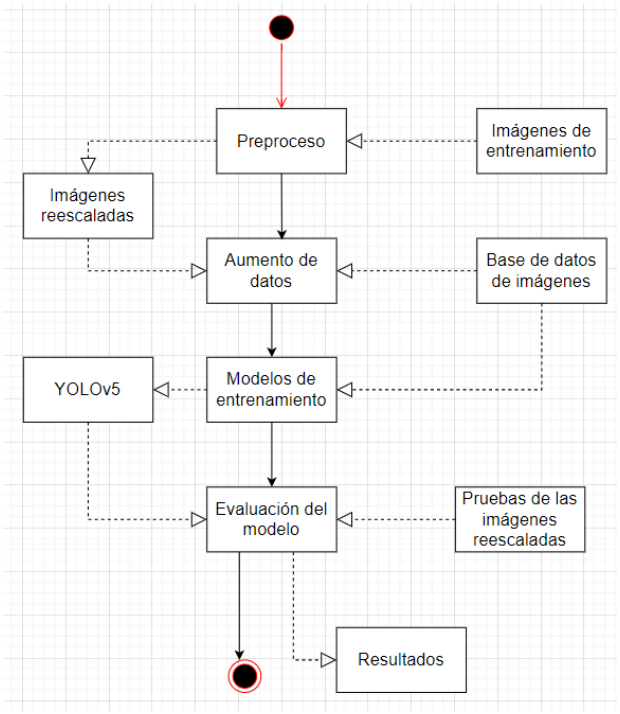

Fig. 13: Arquitectura del sistema propuesto.

4.4.1. Equipo y software usado

El proyecto se ha desarrollado usando un ordenador personal (PC) equipado con:

- Procesador: Intel ® Core ™ i7 – 10510U CPU @ 1.80 GHz
- Memoria RAM del sistema: 16 GB, 2.13 MHz
- GPU: Intel ® UHD Graphics
- Memoria de la GPU: 8 GB

La herramienta *Google Colab*, que proporciona un entorno en la nube con acceso a GPU dedicada, se ha utilizado. Se ha creado un cuaderno (*notebook*) de Jupyter utilizando esta herramienta para ejecutar el código de *Python* necesario para el proyecto. A pesar de las limitaciones del plan gratuito, como la cantidad de tiempo limitada para el entrenamiento (menos de una hora y media), ha sido suficiente y no ha requerido la instalación de software adicional en el PC.

4.4.2. Jerarquía de ficheros

Se debe establecer una jerarquía de directorios en la raíz del proyecto con una carpeta que contenga todas las imágenes que se utilizarán durante el proceso de entrenamiento, validación y prueba para que YOLOv5 encuentre las imágenes correctamente, como se muestra en la Figura 14. Si se utiliza en el entorno local, como *Google Colab*, esta jerarquía debe respetarse.

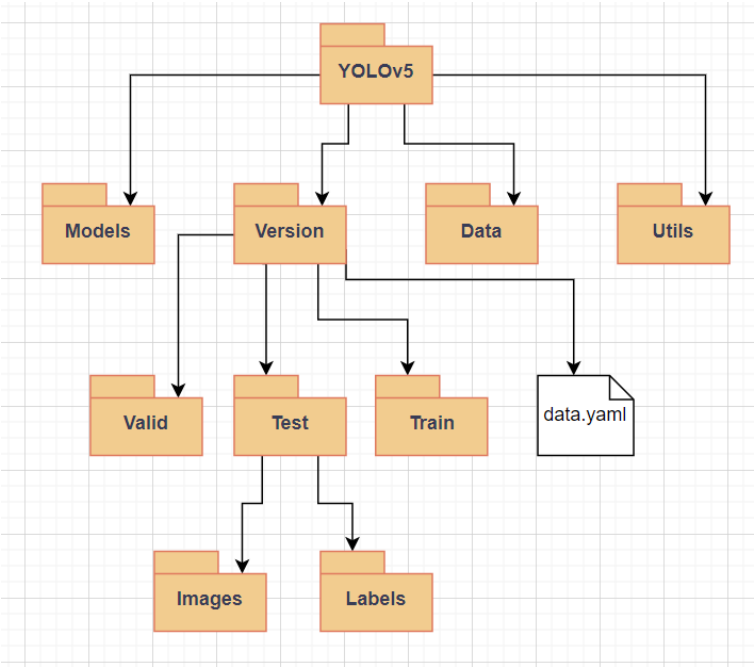

Fig. 14: Jerarquía de ficheros para un proyecto YOLOv5.

4.4.3. Entrenamiento de la red

Se requieren imágenes etiquetadas con las clases y los objetos correspondientes para entrenar una red neuronal. Para cada imagen, es necesario disponer de un archivo.txt con la información de las anotaciones, que incluye el identificador de la clase, las coordenadas del centro del rectángulo en el eje X, las coordenadas del centro del rectángulo en el eje Y, el ancho y el largo. Cada elemento en las imágenes debe tener una línea.

Para comenzar el entrenamiento del modelo, debe configurar los parámetros proporcionados por YOLOv5. Se han probado varios valores de estos parámetros durante los experimentos con el objetivo de mejorar la calidad del detector.

- Image size (tamaño de la imagen): Indica la resolución de las imágenes utilizadas para entrenar el modelo. Antes de usar las imágenes para el entrenamiento, es necesario reescalarlas a esta resolución (416×416), ya que este tamaño corresponde a la resolución de la capa de entrada del modelo.
- *Batch-size (lote)*: Se refiere al número de muestras que se procesan en una iteración del proceso de entrenamiento, es decir, antes de actualizar los pesos. Cuanto más alto sea este valor, más satisfactorio será el resultado en menos tiempos. Sin embargo, la limitación de este valor es la memoria de la GPU. Puede ocurrir que el modelo pierda su capacidad de generalización, lo que puede reducir la calidad de los resultados, a pesar de que se ha mencionado que un valor más alto produce mejores resultados.
- *Epochs (épocas):* Es el número de iteraciones que realiza el modelo con todo el conjunto de imágenes de entrenamiento. Todos los datos de entrenamiento se procesan una vez en cada *epoch*. Según la documentación de YOLOv5, un valor inicial adecuado es 300, y a medida que se aumenta este valor, se debe verificar si hay sobreajuste (*overfitting*). Se debe reducir el número de *epochs* si hay *overfitting*, si no hay, se puede aumentar el valor de este parámetro. En resumen, el valor de tiempo influye en la precisión y el tiempo de entrenamiento, por lo que se debe elegir cuidadosamente para obtener el mejor equilibrio entre ambos factores.
- *Hyperparameters:* Este parámetro indica la ruta del archivo de configuración de los hiperparámetros utilizado en el entrenamiento del modelo. En principio, no se aconseja alterar este parámetro para evitar el sobreajuste, pero en caso de obtener resultados deficientes en la evolución de los hiperparámetros, se puede intentar hacerlo. Los hiperparámetros que se pueden cambiar incluyen:

- *Fliplr (flip left-right*): Este parámetro se utiliza para determinar si las imágenes deben girar a la izquierda o a la derecha de manera aleatoria durante el proceso de *data augmentation*. Al utilizar esta estrategia, se pueden producir imágenes nuevas que permiten al modelo generalizar mejor y detectar objetos en diferentes orientaciones.
- *Flipud:* Es un parámetro que se puede usar para determinar si las imágenes deben voltearse hacia arriba o hacia abajo de manera aleatoria. Es una técnica de aumento de datos que mejora el rendimiento del modelo, como Fliplr.
- *Momentum*: El parámetro utilizado en el algoritmo de descenso del gradiente para ayudar a la optimización de los pesos de la red neuronal. La proporción en la que se utilizan el gradiente actual y el gradiente anterior para actualizar los pesos en cada iteración se conoce como *momentum*. Este parámetro ayuda a que el algoritmo converja más rápido y puede evitar que se quede atrapado en locales mínimos. El *momentum* tiene un valor promedio de 0,9.
- S*caling:* se utiliza para aumentar el tamaño de las imágenes que se utilizarán en el entrenamiento de la red neuronal. Este proceso de escalado puede ser necesario por una variedad de razones, como mejorar la precisión del modelo u optimizar el rendimiento durante el entrenamiento. Es posible que las imágenes de entrada tengan una resolución muy alta o baja en algunos casos, lo que puede dificultar la detección de objetos. Es posible obtener mejores resultados en el entrenamiento de la red neuronal al aumentar el tamaño de las imágenes.
- *Mosaic*: es una técnica de data augmentation que se ha explicado previamente en este trabajo. En resumen, consiste en combinar cuatro imágenes en una sola y alimentar a la red neuronal con ella, lo que puede mejorar la detección de objetos y la eficiencia del modelo.
- *Degree:* El grado de rotación de una imagen para crear una nueva imagen de entrenamiento se puede determinar utilizando esta técnica de *data augmentation*. Para evitar que el modelo se sobreajuste a un conjunto de datos específico, la rotación se realiza de forma aleatoria en el rango de grados especificado. Es un método comúnmente utilizado para aumentar la variabilidad de los datos de entrenamiento y mejorar la capacidad del modelo para generalizarse a nuevas imágenes.

- *Weight-decay*: es un método de regularización que ayuda a prevenir el *overfitting* en la *Red Neuronal*. Tiene como objetivo agregar un valor numérico a la función de activación de pesos durante el proceso de *backpropagation*. Esto tiene como resultado reducir el valor de los pesos, lo que reduce su magnitud y, por lo tanto, su impacto en la función de activación. De esta manera, el riesgo de que la *Red Neuronal* memorice los datos de entrenamiento en lugar de generalizar para datos nuevos se reduce.

El siguiente es un ejemplo del comando utilizado para entrenar el modelo:

```
!python train.py --img 416 --batch-size 130 --epochs 300 --
data {dataset.location}/data.yaml --cfg ./models/custom_yolov5s.yaml  --
name yolov5s_results [25]
```

Este comando utiliza los siguientes argumentos:

- --img 416: Indica que la imagen debe ser de 416×416 píxeles.
- --batch-size 130: Establece un lote de entrenamiento de 130 muestras por iteración.
- --epochs 300: Define el número de épocas de entrenamiento en 300.
- --data {dataset.location}/data.yaml: Indica la ubicación del archivo **data.yaml** con los datos de entrenamiento.
- --cfg ./models/custom_yolov5s.yaml: Establece la ruta del archivo de configuración del modelo personalizado para el entrenamiento.
- --name yolov5s_results: Indica el nombre del directorio donde se guardarán los resultados del entrenamiento.

Se generan una serie de archivos de rendimiento para evaluar el desempeño del modelo después de completar el proceso de entrenamiento de YOLOv5. La información sobre ellos se ampliará en el apartado de experimentos. Es importante destacar que el entrenamiento crea un archivo .pt con los mejores pesos y otro archivo .pt con los pesos más recientes.

4.4.4. Detección de elementos

Se puede utilizar el archivo best.pt como parámetro en el comando de detección una vez que se cuenta con el modelo entrenado. Es esencial que las imágenes utilizadas para la detección tengan la misma resolución que las del modelo, en este ejemplo de 416×416 píxeles. De lo contrario, no se alcanzarán los resultados óptimos.

Se deben agregar parámetros al comando detect.py para configurar algunas características de detección. Estos son los parámetros disponibles y sus funciones:

- Source*:* Indica dónde buscar la imagen o el vídeo. Una sola imagen, una carpeta con varias imágenes o un archivo de vídeo se pueden usar.
- Iou: Establece el valor límite de IoU (*Intersection over Union*) para fusionar regiones detectadas. El valor estándar es 0.45.
- Weights: Indica donde se encuentra el archivo junto con los pesos del modelo. El archivo *best.pt* creado durante el entrenamiento se utiliza por defecto.
- Img*:* Cambia la resolución de las imágenes que se reciben. El valor inicial es 640.
- Device: Establece el mecanismo de procesamiento para la red neuronal. Puede ser *cpu*, *cuda* o *xla*, con *cuda* como valor predeterminado.
- Conf: Establece el valor mínimo de confianza para mostrar una detección. El valor *default* es 0.25.

Para realizar la detección de objetos con YOLOv5, por ejemplo, se puede ejecutar el siguiente comando:

```
!python detect.py --weights runs/train/yolov5s_results/weights/best.pt --img 416 --conf 0.5 --source /content/yolov5/TFG_Project_4-39/test/images
```

El resultado de la detección se guarda en la carpeta:

```
yolo/runs/detect/[nombre del experimento]
```

Y allí, junto con las imágenes procesadas por el detector, se pueden encontrar los archivos de texto correspondientes que contienen la información sobre las detecciones realizadas en cada imagen.

4.4.5. Mejora del texto en inscripciones

Utilizamos el método descrito anteriormente en la sección 2.3 para realizar una serie de operaciones después de encontrar las inscripciones. Este método implica el uso de umbrales y componentes conexos. Esto se puede lograr ejecutando los siguientes comandos:

```
img_umbr1 = cv2.adaptiveThreshold(gray_image,
255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,21,10)
```

Se aplica un umbral adaptativo a la imagen ***gray_image*** en escala de grises en esta operación. El umbral adaptativo se calcula con el método de umbralización adaptativa de *Gaussiana* (cv2.ADAPTIVE_THRESH_GAUSSIAN_C). Este método utiliza un área local alrededor de cada píxel de la imagen para ajustar el umbral.

El parámetro 21, que determina el tamaño de la ventana utilizada para calcular el umbral, indica el tamaño del área local. La constante de umbral que se resta al resultado del cálculo para obtener el umbral adaptativo final se muestra en el parámetro 10.

La imagen umbralizada img_umbr1 se crea como resultado de esta operación. Los píxeles que superan el umbral se ponen en el valor máximo (255) y los píxeles que no lo superan se ponen en el valor mínimo (0).

Una vez tenemos la imagen umbralizada, realizamos la siguiente operación:

```
blobs = img_umbr1 < img_umbr1.mean()
```

El objetivo de esta operación es identificar las áreas de interés dentro de la imagen umbralizada. La comparación de cada píxel de la imagen con el valor promedio de la intensidad de todos los píxeles de la imagen (img_umbr1.mean()) es lo que hace. Si un píxel tiene un valor menor que el promedio, se considera parte de una región de interés y se establece como True en la matriz blobs; de lo contrario, se establece como False.

Posteriormente, se realizan operaciones adicionales para etiquetar y eliminar áreas pequeñas que pueden representar artefactos o ruido no deseados. Esto se logra mediante la operación siguiente:

```
pre_version = morphology.remove_small_objects(blobs_labels, average)
```

La matriz blobs_labels, que contiene las regiones etiquetadas en la imagen umbralizada, se utiliza como entrada para la operación remove_small_objects de la biblioteca morphology. La función también recibe el tamaño promedio (average) de las regiones calculadas anteriormente.

Por lo tanto, la función remove_small_objects elimina las áreas cuyo tamaño es menor que el promedio. Las áreas más grandes, que son el tamaño promedio, se mantienen intactas, mientras que estas áreas pequeñas, que pueden ser ruido o artefactos, se eliminan de la imagen.

Finalmente, debe aplicarse una umbralización a la imagen pre_version porque la operación remove_small_objects devuelve una imagen en niveles de gris. Para lograr esto, se realiza la operación siguiente:

```
im_bin = (pre_version > 0) * 255
```

El objetivo de esta operación es convertir la imagen pre_version en una imagen binaria, en la que los píxeles con valores superiores a 0 se asignan como 255, mientras que los píxeles con valores inferiores a 0 se mantienen como 0.

Para lograr esto, se utiliza una comparación booleana (pre_version > 0) que da como resultado una matriz de valores True o False. Los píxeles con valores superiores a 0 se consideran True y los píxeles con valor inferior a 0 se consideran False. Luego, esta matriz se multiplica por 255, lo que da al valor 255 en blanco a los píxeles True y al valor 0 en negro a los píxeles False, lo que da como resultado la imagen binaria resultante.

4.4.6. Emparejamiento de escudos

Después de completar la detección de los escudos, pasamos a correlacionarlos utilizando el *SSIM*, que se describe en la sección 2.4. Esto se puede lograr ejecutando el comando siguiente:

```
ssim = structural_similarity(escudo1, escudo2, multichannel=True)
```

La función structural_similarity, que se encuentra en la biblioteca de procesamiento de imágenes skimage.measure, recibe dos imágenes de escudo (escudo1 y escudo2) en formato de matriz. La opción Multichannel=True indica que las imágenes son de varios canales, lo que permite incluir información de color adicional en el cálculo de similitud.

El *SSIM* se calcula utilizando la operación de structural_similarity entre las dos imágenes de escudo. Este índice evalúa la similitud entre las estructuras locales de las imágenes y acepta valores que oscilan entre -1 y 1. Un valor alto indica una similitud perfecta y un valor bajo indica una diferencia significativa.

5. Pruebas

Este capítulo tiene como objetivo mostrar los resultados experimentales de este trabajo en relación con la detección de escudos e inscripciones usando YOLOv5, en los que se varían algunos hiperparámetros considerados relevantes, como el número de *batches* y *epochs*. Al variar el número de *batches*, se busca estudiar cómo afecta a la calidad del modelo. La teoría sugiere que es recomendable fijar este valor en el más alto posible que permita el hardware utilizado, lo que haría posible la realización de entrenamientos más largos sin que aparezca el fenómeno de *overfitting* o, en su defecto, obtener mejores resultados con el mismo número de épocas. Sin embargo, aumentar este valor puede hacer que el modelo no generalice adecuadamente y "aprenda" solo las imágenes de entrenamiento, lo que podría conducir a un peor desempeño del modelo, ya que el número de imágenes de entrenamiento es limitado. Por lo tanto, debido a la falta de generalización del modelo, aumentar el número de *batches* conducirá a un desempeño más bajo del modelo.

En esta sección, describiremos la composición del conjunto de datos, definiremos las métricas más importantes para el análisis posterior de los resultados y concluiremos con la ejecución y el análisis de los experimentos.

5.1. Conjunto de datos.

Se ha buscado crear un conjunto de datos sin sesgos, lo que implica tener el mismo número de objetos a detectar para cada clase ("*Escudo*" e "*Inscripción*", respectivamente). Sin embargo, el número total de imágenes se ha reducido debido a los problemas para encontrar las imágenes adecuadas para este trabajo. Para obtener los mejores resultados, según el creador y mantenedor de YOLOv5 en su wiki de *GitHub*, se necesitan al menos 2000 imágenes, y dado que el problema es de tipo multiclase, se necesitarían aún más. Aunque no se ha logrado la cantidad ideal de imágenes, se ha logrado un equilibrio entre las clases.

El conjunto de datos utilizado contiene un total de 516 imágenes con las siguientes entradas.

Clase	Número de objetos
Escudos	450
Inscripciones	156
Total	606

Tabla 1: Desglose del número de objetos por categoría

Para garantizar un entrenamiento efectivo y una evaluación confiable del modelo, se han dividido las imágenes en proporciones adecuadas para el entrenamiento, la validación y el test. El conjunto de entrenamiento, el conjunto de validación y el conjunto de prueba recibieron el 84% de las imágenes. Para evitar un sesgo en el modelo, es importante señalar que se ha mantenido la misma proporción de clases en cada uno de los conjuntos.

Para detectar correctamente los relieves, es esencial conocer dónde se encuentran en las piedras. Para lograrlo, *Roboflow* ofrece una herramienta de diagnóstico que incluye mapas de calor para cada clase específica. Un mapa de calor muestra un color más cálido (verde y amarillo) cuando hay más ocurrencias de un objeto en ese espacio, mientras que un color más frío (azul) cuando hay menos ocurrencias. Si no hay objetos de esa clase en el área, el mapa no mostrará ningún color. Un ejemplo de estos mapas de calor se muestra en la Figura 15, concretamente con la clase Inscripción.

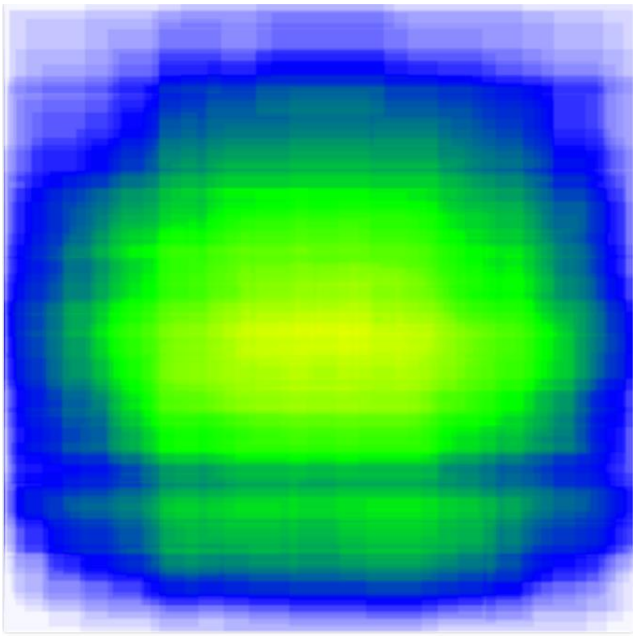

Fig. 15: Mapa de color de la clase Inscripción. Imagen obtenida con la herramienta Roboflow

La mayoría de las inscripciones se ubican en el centro de la imagen, extendiéndose por todo su ancho, tal como se muestra en la Figura 15. La red probablemente utilice esta información espacial para encontrar elementos de este tipo y distinguirlos de otras categorías. Sin embargo, surge la pregunta de si esto puede causar sesgos en el proceso de detección de inscripciones. ¿Es apropiado clasificar un elemento como inscripción si se encuentra en una posición que no le corresponde, como en una escritura oriental vertical? No hay una sola respuesta a esta pregunta. Si se cree que es crucial clasificar correctamente estas inscripciones, independientemente de su ubicación en la imagen, se puede abordar este problema de sesgo mediante el uso de técnicas de *data augmentation*, como la rotación de la imagen.

El mapa de calor para una imagen de la clase "*Escudo*" se muestra en la Figura 16. En este caso, se cree que un escudo debe estar en el centro de la imagen y su forma generalmente es circular. Dado que un escudo siempre debe estar en el centro de la imagen, se cree que esta información espacial será útil para el modelo y no producirá sesgo.

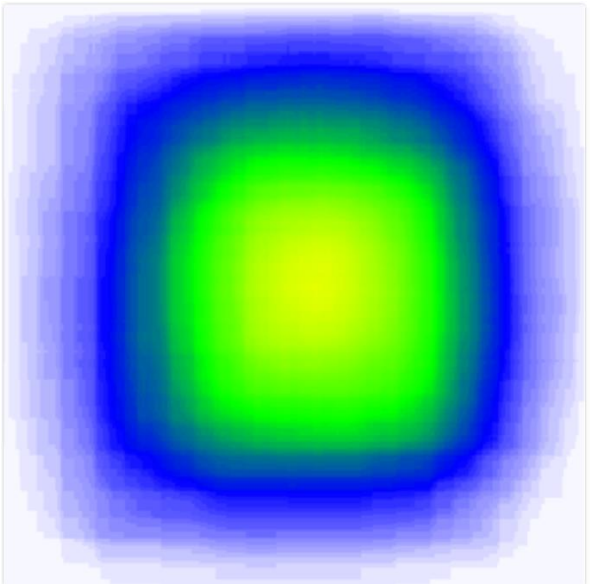

Fig. 16: Mapa de color de la clase Escudo. Imagen obtenida con la herramienta Roboflow

5.2. Métricas de evaluación

Las métricas de evaluación son instrumentos utilizados para evaluar la eficacia de un detector de objetos. En este ámbito, *la Intersection over Union* (IoU) es una de las medidas más utilizadas [26]. El *ground truth* se utiliza para comparar la detección producida por el modelo con la posición real del objeto. El cálculo de esta métrica se realiza comparando dos conjuntos finitos, en este caso conjuntos de píxeles. La similitud entre estos dos conjuntos se mide por el coeficiente de IoU, que se calcula dividiendo la intersección de los conjuntos por su unión, como se muestra en la Figura 17.

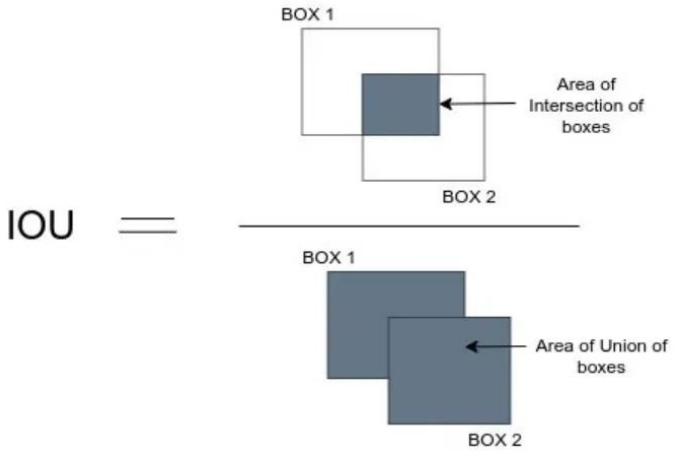

Fig. 17: Representación esquemática de la fórmula para calcular el IoU. Imagen extraída de [27]

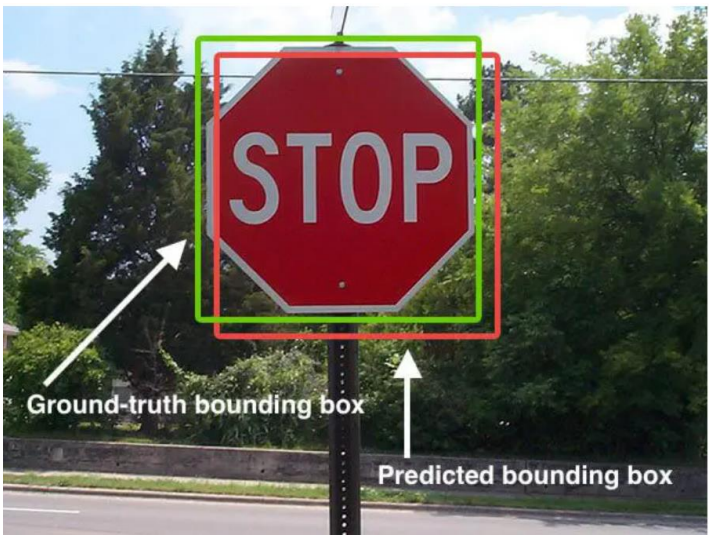

Fig. 18: Diferencia entre el ground truth y una detección. Imagen extraída de [28]

La Figura 18 muestra un ejemplo de las áreas en las que la operación de *IoU* debe implementarse. Es común que el área de detección no coincida con el área definida en el *ground truth*. No obstante, esto no implica que la detección sea incorrecta. Para abordar esta situación, se introduce el concepto de umbral, que se describe en el apartado siguiente.

5.2.1. Verdadero Positivo

Para establecer las métricas básicas de detección de objetos, es necesario tener en cuenta dos umbrales: la confianza de la red y el umbral *IoU*. El umbral de confianza de la red es un valor numérico que muestra cuán confiable es la predicción del modelo. El umbral de *IoU* es una medida de la similitud entre la detección y el *ground truth*, y se establece un valor mínimo para considerar que la detección es correcta.

Un *Verdadero Positivo* es una detección que ha superado tanto los requisitos de confianza como los requisitos de *IoU*. Esto es lo que se puede decir matemáticamente sobre esta definición:

$$TP(p) = (Conf(p) \geq Conf_u \cap IoU(p) \geq IoU_u)$$

En este caso, $Conf_u$ es el umbral de confianza, IoU_u es el umbral de intersección sobre la unión y $Conf_u$ y es el umbral de intersección sobre la unión [29] y $Conf(p)$ es la confianza de la detección

5.2.2. Verdadero Negativo

Las detecciones que no han superado el umbral de confianza de la red ni el umbral de intersección sobre la unión pertenecen a la clase de detección *Verdadero Negativo*. Esto indica que en esa área de detección no hay nada que se deba detectar, y el modelo ha reconocido esta situación correctamente. La siguiente es la definición matemática de esta clase [8]:

$$TN(p) = (Conf(p) < Conf_u \cap IoU(p) < IoU_u)$$

5.2.3. Falso Positivo

Un *Falso Positivo* ocurre cuando la confianza de la red en la detección supera el umbral de confianza establecido por el modelo, pero no alcanza el umbral de intersección sobre la unión, lo que lleva al modelo a realizar una detección errónea [30]. Se define matemáticamente como:

$$FP = (Conf(p) > Conf_u \cap IoU(p) < IoU_u)$$

5.2.4. Falso Negativo

Cuando el modelo no puede detectar un objeto en la imagen porque la confianza de la detección no supera el umbral establecido, se produce un *Falso Negativo*. En otras palabras, el modelo no puede encontrar una detección precisa. La diferencia entre el número total de objetos que se deben detectar en la imagen y el número de *TP* encontrados en la imagen se define matemáticamente como la siguiente:

$$FN = NO(i) - |TP(i)|$$

En este caso, $NO(i)$ indica la cantidad de objetos en la imagen y $TP(i)$ indica la cantidad de detecciones verdaderamente positivas encontradas en la misma imagen.

5.2.5 *Precision*

Se dice que la *precision* es la proporción de los *Verdaderos Positivos* en comparación con la suma de los *Falsos Positivos*. Por decirlo de otra manera, muestra la cantidad de detecciones correctamente clasificadas en comparación con el total de detecciones clasificadas como positivas [31].

$$Precision = \frac{TP}{TP + FP}$$

5.2.6. *Recall*

El *recall*, también conocido como exhaustividad o sensibilidad, es una medida de calidad que se utiliza para evaluar el desempeño de un modelo de detección. El *recall* se enfoca en la cantidad de detecciones correctas entre todos los objetos que el modelo debería haber detectado, en lugar de la *precision*. El *recall* es directamente proporcional a los *Verdaderos Positivos* e inversamente proporcional a la suma de los *Verdaderos Positivos* y *Falsos Negativos* matemáticamente [32].

$$Recall = \frac{TP}{TP + FN}$$

5.2.7. *F1-score*

El *F1-score* es una medida que combina la *precision* y el *recall* en una sola medida utilizando la media armónica de ambas. La fórmula matemática del *F1-score* indica que aumentar el valor de cualquier variable aumentará el *F1-score* porque la multiplicación crece más rápidamente que la suma. Esta medida es útil para crear un indicador que integre los datos de las dos métricas anteriores [33].

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

5.2.8. Matriz de confusión

Para evaluar modelos de clasificación, se utiliza la matriz de confusión. El número de predicciones correctas e incorrectas clasificadas en cada una de las clases que se están evaluando se muestra en la matriz. Los cuatro elementos ya mencionados en la matriz son *Verdadero Positivo, Falso Positivo, Verdadero Negativo* y *Falso Negativo*. Se pueden calcular métricas de evaluación como *precision*, *recall* y *F1-score* a partir de los valores de la matriz. Estas métricas brindan una evaluación más amplia del rendimiento del modelo [34].

5.3. Experimentos

Se realizarán cinco experimentos con diferentes parámetros utilizando las métricas descritas anteriormente para analizar la calidad del detector y planificar estrategias de mejora de la red, además de los postprocesos para las inscripciones y escudos Esto permitirá identificar errores de detección y mejorar la *precision* y el *recall* del modelo.

La técnica consistirá en entrenar un modelo con los parámetros elegidos, aplicando las métricas mencionadas a las imágenes de validación. Luego se aplicará el modelo a las imágenes de prueba para mostrar los resultados y analizar la calidad del detector.

El tamaño del *batch* y los *epochs* serán los dos parámetros principales que se examinarán. El objetivo es determinar cómo estos dos parámetros afectan el funcionamiento del modelo de detección de objetos y en qué medida pueden mejorar la *precision* y el *recall*. Para lograr esto, el modelo se entrenará utilizando una variedad de combinaciones de valores de *batch* y *epochs* y se evaluará su desempeño utilizando imágenes de prueba.

5.3.1. Experimento 1: Valor de *batch* = 16

El objetivo de este experimento es investigar cómo el tamaño del *batch* afecta el resultado final del modelo. Para lograr esto, se ha entrenado al modelo con los hiperparámetros fijos siguientes: *momentum* = 0.937, *weight-decay* = 0.0005 y *epoch* = 300. El tamaño del *batch* y la duración serán modificados en los diferentes experimentos. En el primer experimento, estos parámetros se fijaron en 16 y 300, respectivamente.

El primer experimento se ha realizado utilizando los parámetros mencionados anteriormente y ejecutando el siguiente comando:

```
!python train.py --img 416 --batch-size 16 --epochs 300 --
data {dataset.location}/data.yaml --cfg ./models/custom_yolov5s.yaml  --
name yolov5s_results
```

Se ha obtenido los siguientes resultados de entrenamiento (Ver Figura 19):

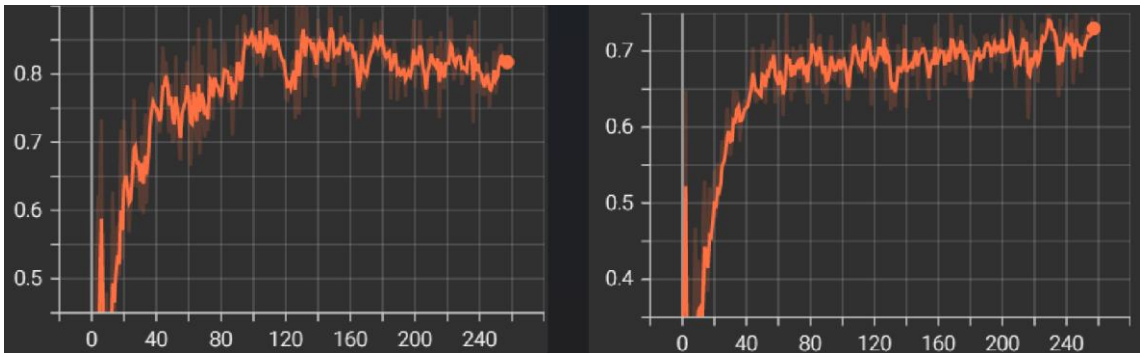

Fig. 19: Gráficas de precision y recall del experimento 1.

Como se puede ver, los resultados de entrenamiento son bastante aceptables, con las mejores iteraciones del proceso cercanas al 0,9 de *precision* y 0,8 de *recall*, sin el fenómeno de *overfitting*. Además, es notable cómo las métricas encuentran un punto donde la mejora es mínima en cada iteración, lo que indica que no es necesario entrenar el modelo con más épocas.

Después del entrenamiento, es crucial evaluar el desempeño del modelo utilizando imágenes de prueba que no haya visto previamente. YOLOv5 proporciona un *script* en *Python* para realizar esta tarea, lo que permite realizar el proceso de validación con las imágenes de prueba. El siguiente es el comando que se utilizó:

```
!python val.py --weights runs/train/yolov5s_results/weights/best.pt --
data {dataset.location}/data.yaml --img 416 --workers 0 --save-txt --conf-thres 0.6 --
task val
```

El programa realiza el proceso sobre las imágenes de prueba al especificar el parámetro *–task test* Esto producirá una variedad de métricas que podrán analizarse. La matriz de confusión creada como resultado de la ejecución de este comando se muestra en la Figura 20.

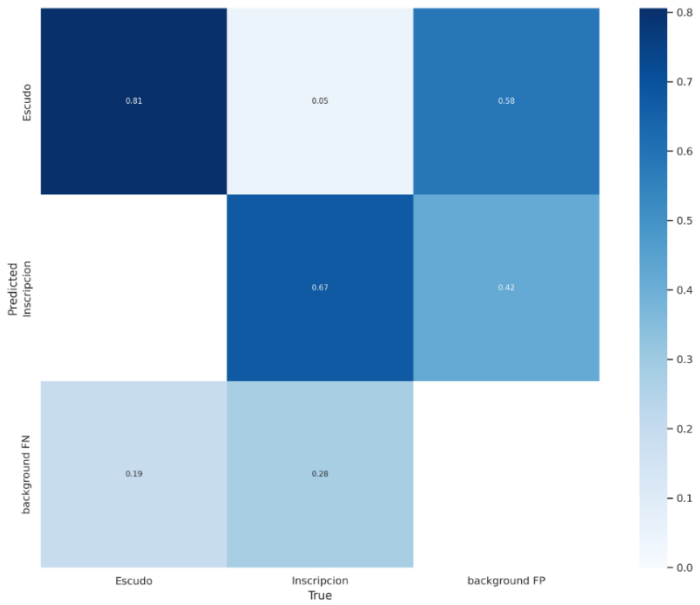

Fig. 20: Matriz de confusión del experimento 1

	Precision	**Recall**	***F1-score***	**Tiempo computacional por imagen**
Experimento 1	0.89	0.53	0.66	3,9 ms

Tabla 2: Resultados cuantitativos del Experimento 1

Los datos mostraron una *precision* de 0.89 y una *recall* de 0.53. Como se puede ver en la Figura 20, la mayoría de los falsos positivos pertenecen a la clase de escudos. A pesar de que la *precision* y el *recall* de esta prueba son relativamente altos, se entiende que estos falsos positivos no son muy grandes en volumen. En cuanto a los falsos negativos, se encontró que el 58 % pertenecía a la categoría de escudo, lo cual tiene sentido porque es la categoría más difícil de clasificar porque es fácil confundir un relieve en piedra con un escudo. Las inscripciones representan el 42% de los falsos positivos. Esto se cree que se debe a la generalidad de esta clase, ya que es una agrupación de varias subclases con diferentes tipos de escritura y orientaciones, lo que puede resultar en procesos de generalización no deseados. Sin embargo, la categoría de inscripciones tiene menos falsos positivos que la categoría de escudo.

En la Figura 21 se muestra un ejemplo de detección de inscripciones y escudos producidos por esta configuración de la red YOLOv5. Por cada detección producida, se muestra el valor de confianza de esta.

Fig. 21: Ejemplo de detecciones del experimento 1

5.3.2. Experimento 2: Valor de *batch* máximo

Para este experimento, se utilizó el parámetro '-1' para aumentar el tamaño del lote (*batch*) al máximo permitido por el equipo. Esto significa que se ajustará automáticamente al número máximo de imágenes en cada lote. La resolución ha sido de 416×416 y los hiperparámetros han sido los mismos que en el Experimento 1. Según la teoría, aumentar este parámetro aumenta la posibilidad de aumentar el número de *epochs* y reduce la probabilidad de *overfitting* y pérdida. Se llevará a cabo el proceso de entrenamiento y validación del modelo utilizando las imágenes de prueba, como en la muestra anterior.

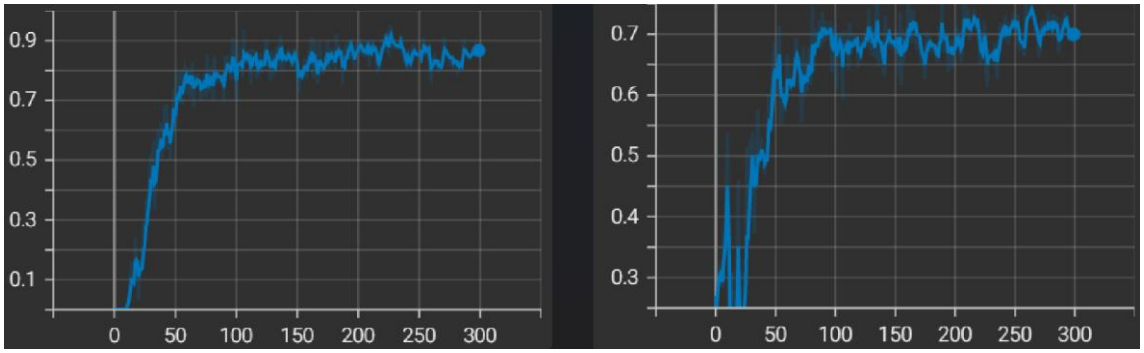

Fig. 22: Gráficas de precision y recall del experimento 2

Aunque las métricas obtenidas son muy similares a las del experimento anterior, se puede ver en la Figura 22 que, durante el entrenamiento, las medidas muestran mucha menos dispersión en las gráficas. Las métricas evolucionan a un ritmo más lento, como se esperaba con el aumento del número de *batch*. Como resultado de esto, será posible aumentar este valor para lograr mejores resultados en el futuro. Al final del curso, las métricas se estabilizan y se obtienen resultados algo mejorados.

Para evaluar el modelo entrenado en el Experimento 2, se usa el mismo comando que se utilizó en el Experimento 1, pero cambiando la dirección del nuevo modelo generado. La Figura 23 muestra la matriz de confusión del Experimento 2.

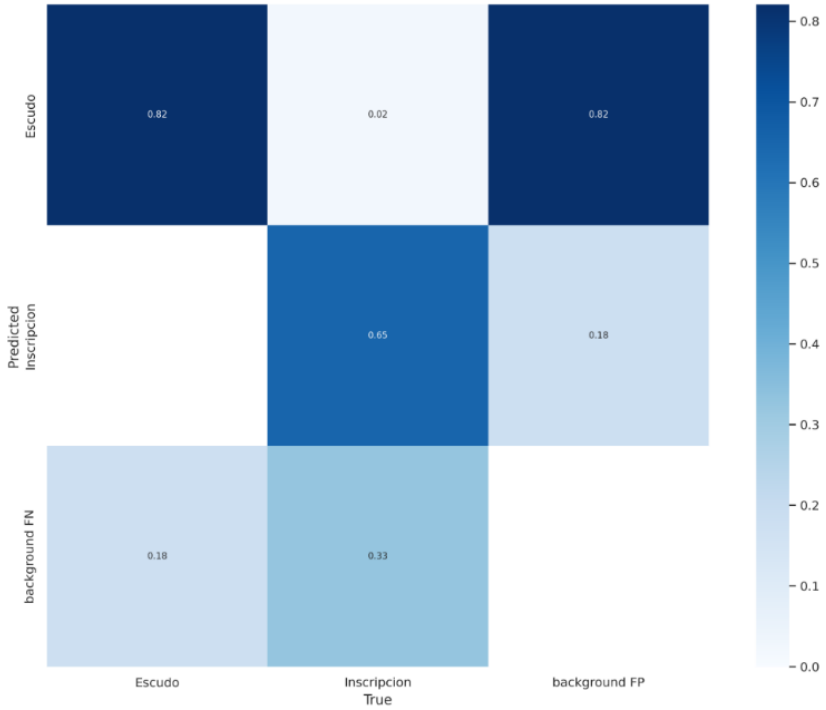

Fig. 23: Matriz de confusión del experimento 2

	Precision	**Recall**	***F1-score***	**Tiempo computacional por imagen**
Experimento 2	0.91	0.65	0.76	3,5 ms

Tabla 3: Resultados cuantitativos del Experimento 2

Se puede ver que el 33% de los falsos negativos pertenecen a la categoría de "*Inscripción*" y el 18% a la categoría de "*Escudo*", según la matriz de confusión de la Figura 23. De nuevo, estos errores ocurren en las clases más comunes y complicadas. En cuanto a los falsos positivos, la diferencia entre las clases "*Escudo*" e "*Inscripción*" ha aumentado, con más falsos positivos en la clase "*Escudo*" y menos falsos positivos en la clase "*Inscripción*". Esto podría deberse a que la clase "*Inscripción*" comparte características visuales con la clase "*Escudo*", lo que puede llevar al modelo a confusiones. Además, hay más instancias de la clase "*Escudo*" en el conjunto de datos que de la clase "*Inscripción*", lo que podría afectar los resultados. Para mejorar su desempeño en estas clases difíciles, es crucial continuar analizando los resultados y ajustando el modelo.

La Tabla 3 muestra valores de métricas superiores a los del experimento anterior. Sin embargo, se debe considerar la posibilidad de entrenar el modelo durante más épocas para poder aprovechar verdaderamente la potencia de entrenar con un tamaño de *batch* elevado. En el futuro, puede ser interesante realizar este experimento.

La Figura 24 muestra los resultados del procesamiento de imágenes de prueba con el modelo entrenado.

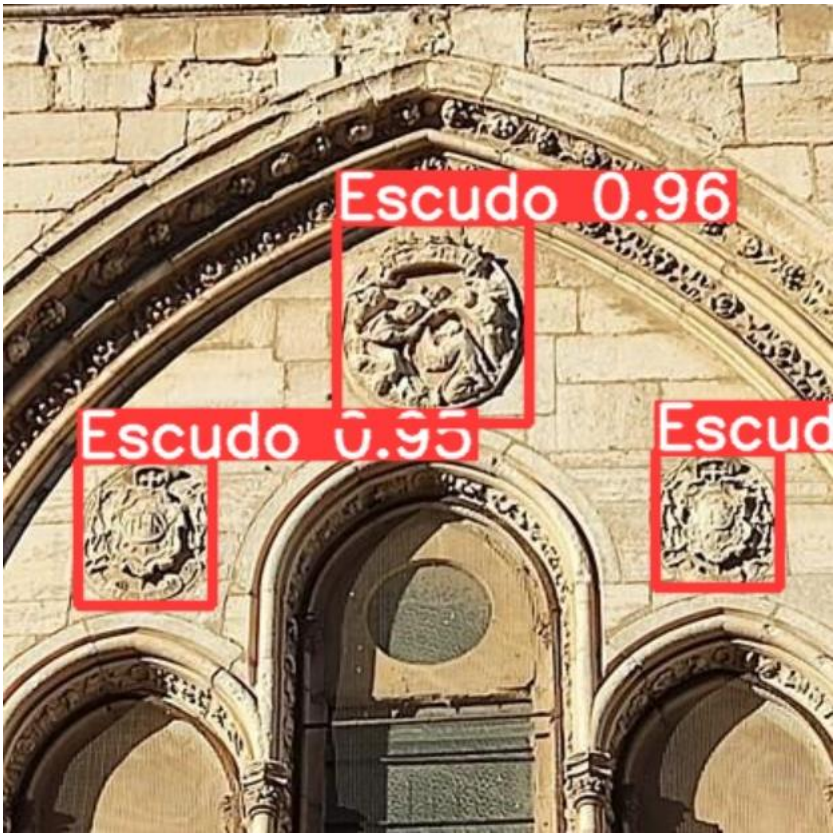

Fig. 24: Ejemplo de detecciones del experimento 2

5.3.3. Experimento 3: Pesos preentrenados

El objetivo de este experimento es investigar cómo los pesos preentrenados afectan la calidad de la detección de objetos. Para ello, los hiperparámetros de la red del Experimento 2 se mantienen, pero se utilizan pesos preentrenados para entrenar la red mediante el parámetro *weights*. Estos pesos se han entrenado utilizando conjuntos amplios como *COCO (Common Objects in Context)* junto con métodos de *PyTorch* Según la teoría, el uso de pesos preentrenados debería permitir que la red aprenda más rápido y obtenga mejores resultados porque estos pesos ya han sido ajustados a un gran conjunto de datos.

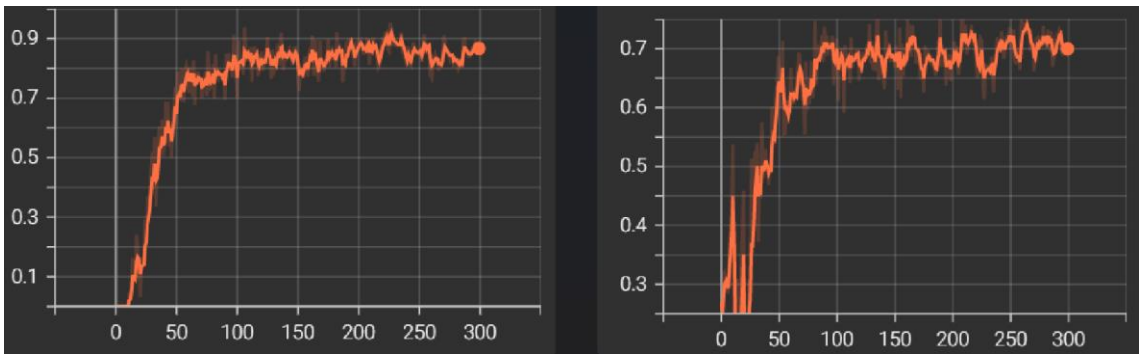

Fig. 25: Gráficas de precision y recall del experimento 3

Las métricas son buenas durante el entrenamiento y no hay pérdida ni *overfitting*, como se muestra en la Figura 25. Los resultados son bastante similares a las métricas del Experimento 2, aunque es cierto que se observa cierta dispersión en la *precision* y el *recall*. Además, al final del entrenamiento se observa un aplanamiento de la curva, lo que indica que no es necesario aumentar la cantidad de veces que se entrena el modelo porque no se lograría una mejora significativa. La Figura 26 muestra la matriz de confusión del Experimento 3.

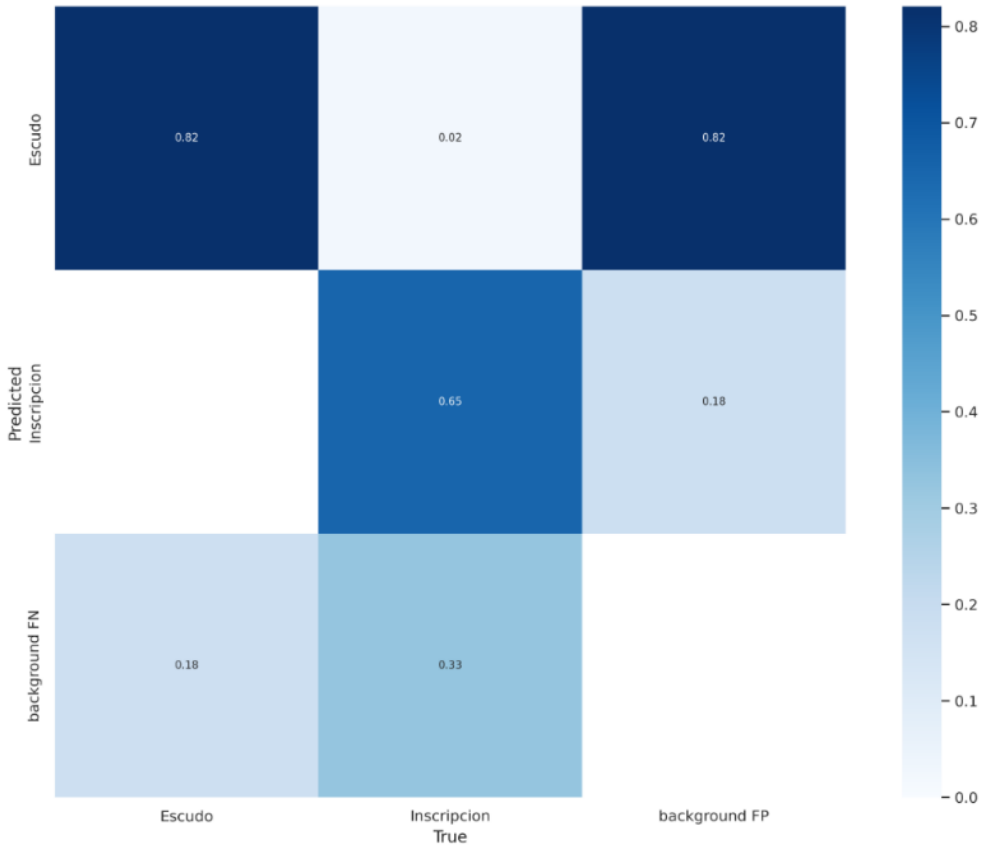

Fig. 26: Matriz de confusión del experimento 3

	Precision	**Recall**	***F1-score***	**Tiempo computacional por imagen**
Experimento 3	0.92	0.66	0.77	3,4 ms

Tabla 4: Resultados cuantitativos del experimento 3

La Figura 26 muestra cómo se obtiene una matriz de confusión idéntica a la del Experimento 2. Por lo tanto, no podemos determinar si este experimento es mejor que el segundo usando esta matriz como discriminante.

La Tabla 4 muestra una mejora en la *precision* y el *recall*, lo que resulta en un *F1-score* ligeramente superior para la muestra en comparación con los pesos sin entrenar. Esto podría deberse a que los pesos preentrenados ya contienen información relevante sobre las características de los objetos a detectar, lo que puede ayudar a la red a converger más rápidamente y obtener mejores resultados. Sin embargo, es importante tener en cuenta que los resultados pueden variar dependiendo de los pesos preentrenados utilizados y del conjunto de datos.

5.3.4. Experimento 4. Métodos de correlación de escudos (SSIM)

Se han utilizado varios métodos para encontrar similitudes entre las detecciones de la clase "*Escudo*" obtenidas en diferentes procesos, como se explica en la sección 2.4. Se ha descubierto que el *SSIM* ha proporcionado los mejores resultados de todos estos métodos. La subsección 4.4.6 explica cómo funciona este índice. El objetivo es obtener escudos con un alto índice, lo que permite agrupar los escudos que tienen similitudes y realizar operaciones sobre estos grupos.

Se ha demostrado que el uso del *SSIM* es útil para evaluar las similitudes entre las detecciones de escudos. Al identificar y agrupar los escudos con características similares, se abre la posibilidad de realizar diversas operaciones y análisis más detallados sobre estos grupos. Los patrones y tendencias en la presencia y apariencia de los escudos pueden encontrarse más fácilmente con esta técnica. Esto puede ayudar a comprender y clasificar estos elementos de manera más precisa en el contexto del problema discutido.

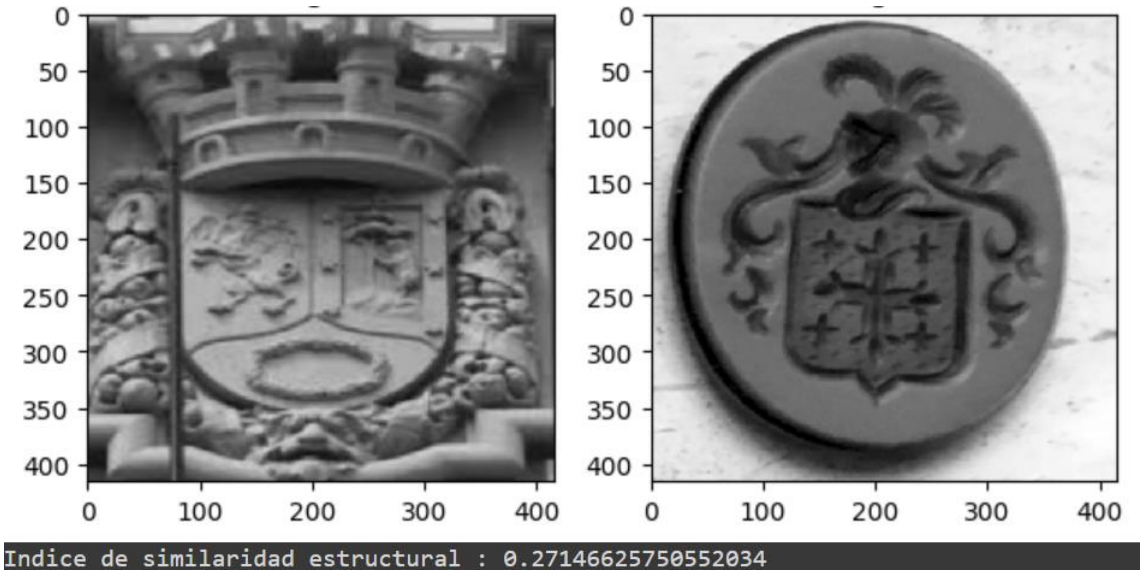

Fig. 27: Ejemplo de escudos con un Índice de Similaridad intermedio

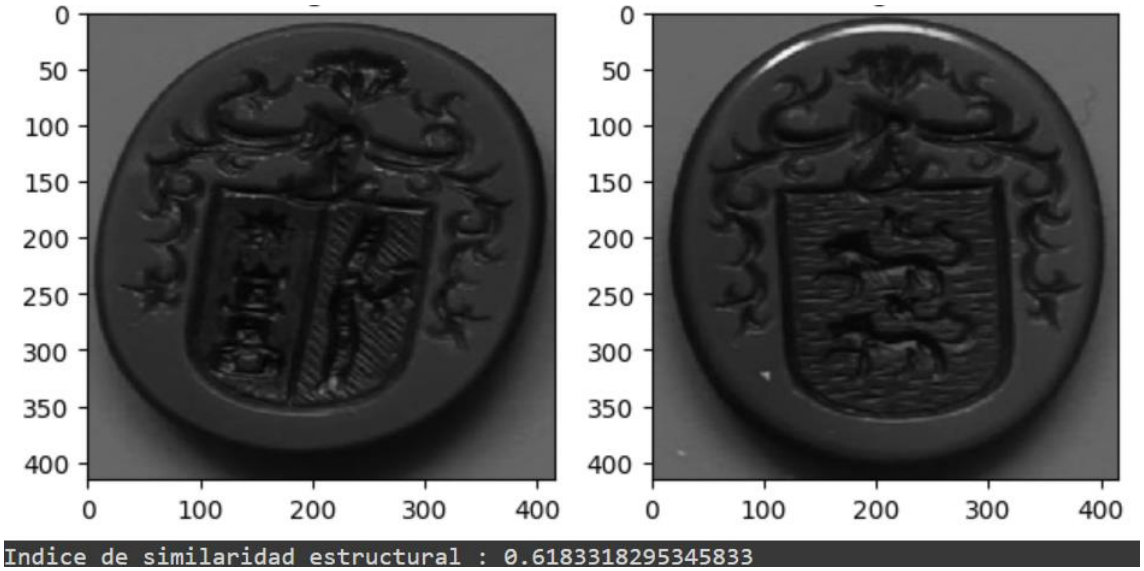

Fig. 28: Ejemplo de escudos con un Índice de Similaridad alto

Debido a las diferencias significativas en la estructura y el contenido de los elementos encontrados, el Índice de Similitud es intermedio, como se muestra en la Figura 27. Estas discrepancias pueden ser causadas por una variedad de factores, como la variabilidad en la apariencia de los escudos, como es el caso, así como la presencia de ruido en las imágenes o la falta de información suficiente para establecer una correspondencia precisa entre los elementos. Se necesitarían enfoques más avanzados que tomen en cuenta la variabilidad inherente en los escudos e incorporen técnicas de procesamiento de imágenes.

Sin embargo, la Figura 28 muestra un valor alto de *SSIM*, lo que indica una fuerte correlación entre las detecciones de escudos. Esto se debe a la presencia de escudos que comparten formas, colores y patrones. Además, es posible que las imágenes utilizadas en este análisis contengan escudos de un mismo tipo o estilo, lo que facilita la identificación de similitudes entre ellas. Es importante destacar que los escudos encontrados comparten rasgos comunes y son candidatos adecuados para operaciones de agrupamiento y análisis conjunto, como lo demuestra un alto *SSIM*. Esto permite el uso de técnicas adicionales, como la extracción de características particulares de los escudos y la exploración de las relaciones semánticas más complejas entre ellos para mejorar la similitud y obtener resultados más consistentes.

5.3.5. Experimento 5. Otros métodos de correlación de escudos

Se han utilizado métodos adicionales para determinar las similitudes entre los escudos, además del *SSIM*, como se detalla en la subsección 2.4.2. Estos métodos no han producido resultados tan notables como los del *SSIM*, pero es importante mencionarlos y analizarlos brevemente.

El primer método utilizado fue el detector de puntos de interés. El detector *SIFT* (*Scale-Invariant Feature Transform*), se utiliza para encontrar características distintivas en una imagen sin verse afectadas por cambios en escala, rotación o iluminación.

El detector *SIFT* extrae puntos clave de la imagen que son invariantes a varios cambios. Estos puntos clave se obtienen descubriendo áreas de alta variación en la intensidad de los píxeles y luego asignándoles un descriptor distintivo que captura información sobre cómo se ven y donde se encuentran en la imagen.

Se utilizó el detector *SIFT* para encontrar puntos de interés en las imágenes que correspondían a las características distintivas de los escudos en el proceso de detección de escudos. Sin embargo, por varias razones, este método no dio resultados satisfactorios. En primer lugar, los escudos en las imágenes pueden variar en tamaño y orientación, lo que dificulta identificar los puntos de interés de manera precisa. Además, la implementación del detector *SIFT* puede requerir una gran cantidad de recursos y puede ser computacionalmente costosa. En las Figuras 29 y 30 se pueden ver ejemplos de escudos que comparten muchos y pocos puntos de interés.

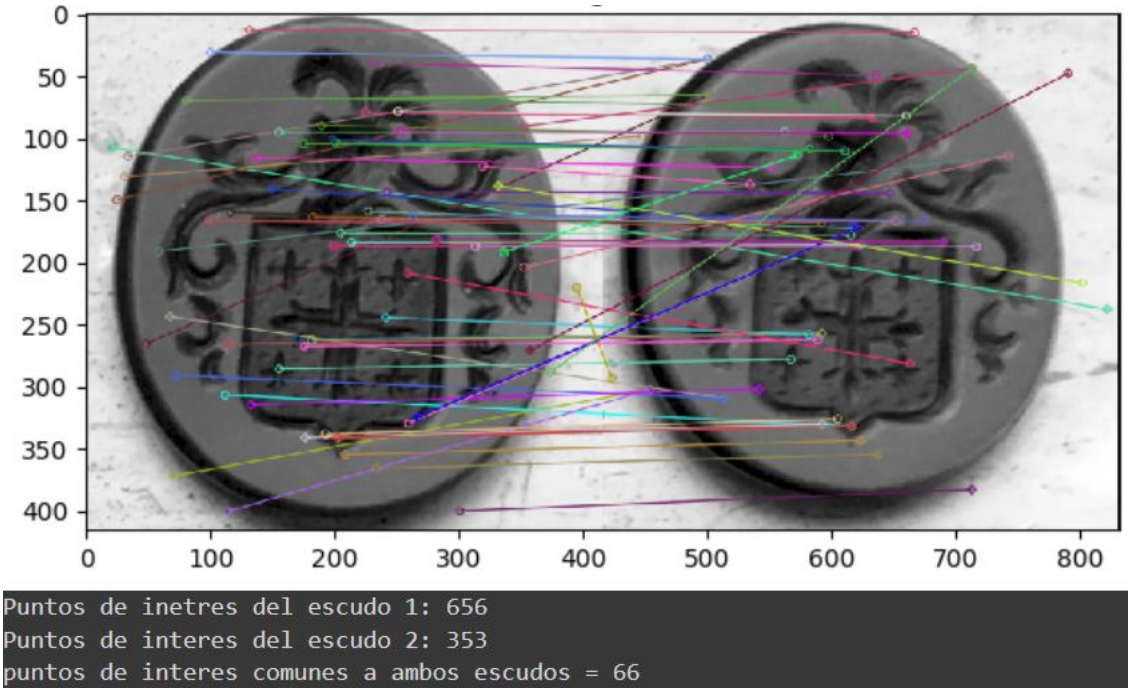

Fig. 29: Escudos con una gran cantidad en común de puntos de interés

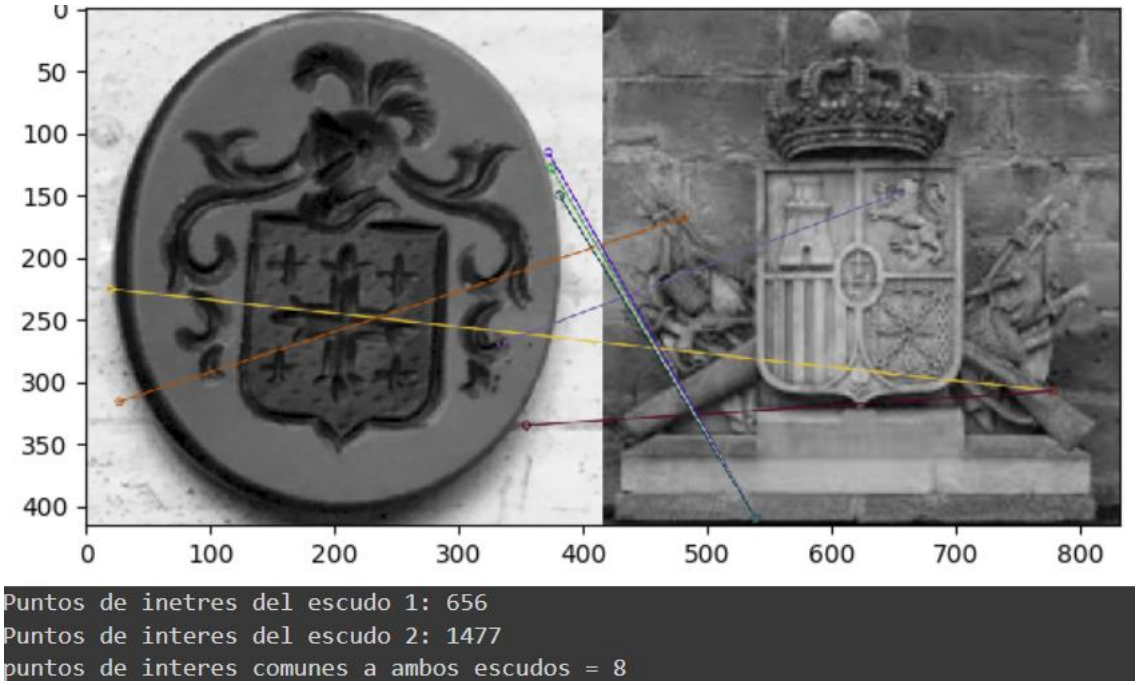

Fig. 30: Escudos con una baja cantidad en común de puntos de interés

El método *absdiff*, de la librería *OpenCV*, también conocido como diferencia absoluta, se utiliza para calcular la diferencia entre dos imágenes, que previamente han tenido que ser normalizadas al mismo tamaño, y destaca las diferencias entre ellas.

El valor absoluto de la diferencia píxel a píxel entre dos imágenes se utiliza para aplicar el método *absdiff*. En otras palabras, la diferencia de intensidad de cada píxel en las dos imágenes se calcula y luego se obtiene el valor absoluto de esa diferencia. El resultado es una nueva imagen que representa las áreas donde las dos imágenes difieren más notablemente.

Para detectar escudos, se utilizó el método *absdiff* para comparar la imagen original con una imagen de referencia con escudo. El objetivo era identificar donde los escudos se distinguían del resto de la imagen y destacar esas diferencias.

No obstante, este método tampoco dio resultados satisfactorios. El método *absdiff* de diferencia de imágenes no pudo capturar de manera precisa y sólida las características distintivas de los escudos, especialmente cuando había variaciones en el tamaño, la orientación y la iluminación. En las Figura 31 se observa un ejemplo de dos escudos 'restados'.

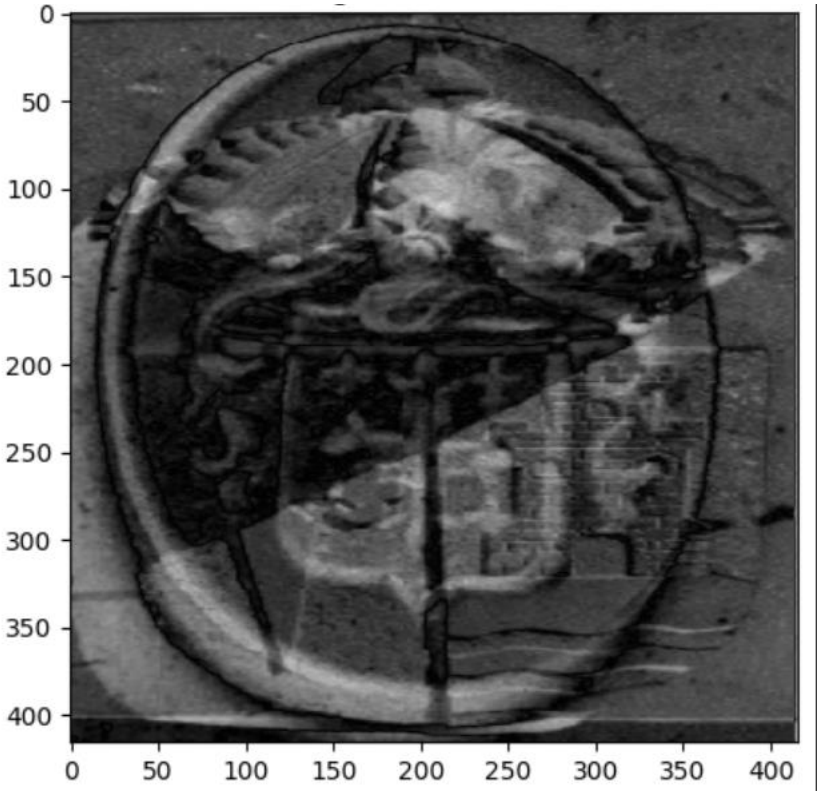

Fig. 31: Escudo resultante de restar dos escudos.

Finalmente, se empleó el método *MatchTemplate* de OpenCV, una técnica de correlación de plantillas que se utiliza para determinar la similitud entre una plantilla de referencia y una región de una imagen.

Se utilizó el método *MatchTemplate* para detectar la presencia de la plantilla de escudo en diferentes áreas de la imagen. El proceso incluyó deslizar la plantilla sobre la imagen en una variedad de ubicaciones y escalas. Además, se utilizó una métrica de correlación para determinar la medida de similitud entre la plantilla y la región correspondiente.

El método *MatchTemplate* produce una imagen de mapa de coincidencia que destaca las áreas donde se descubrió una coincidencia cercana a la plantilla. Estas áreas visibles son las posibles detecciones de escudos en la imagen.

Sin embargo, aunque el método *MatchTemplate* puede ser útil en algunos casos, su rendimiento puede verse afectado por las variaciones en el tamaño, la rotación y la iluminación de los escudos, lo que puede resultar en falsas detecciones o falta de detecciones en situaciones difíciles. En las Figuras 32 y 33 se aprecian escudos con una media de correlación baja y alta, respectivamente.

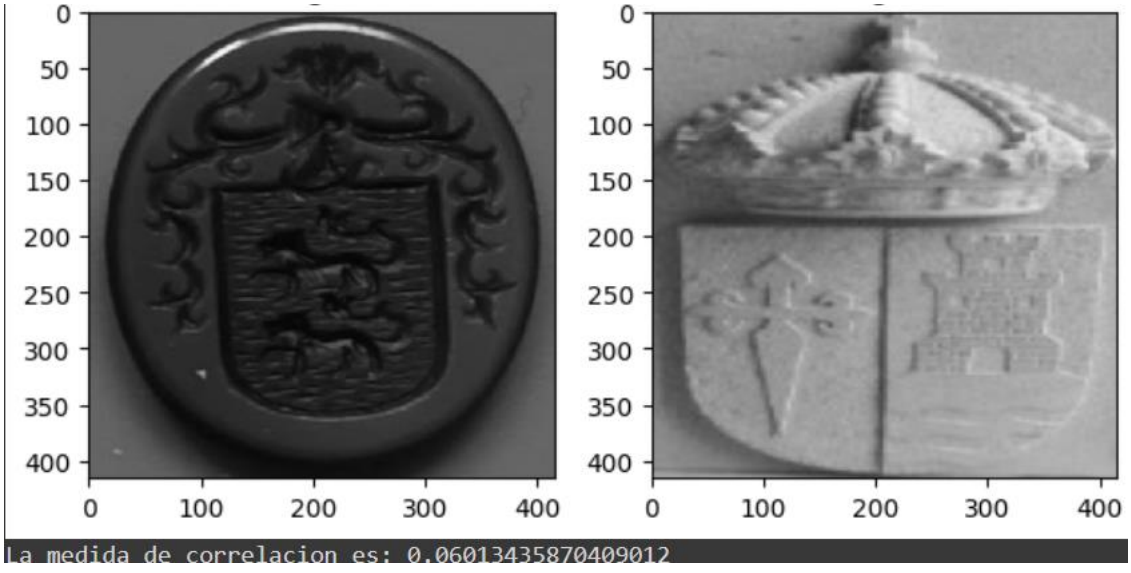

Fig. 32: Escudos con media de correlación baja.

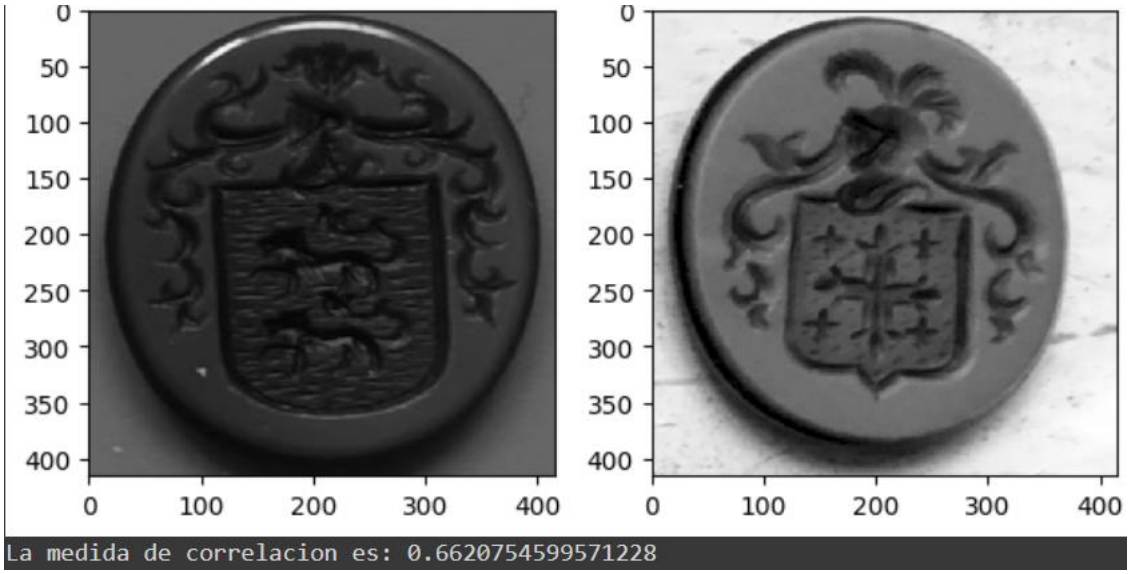

Fig. 33: Escudos con media de correlación alta.

5.3.6. Experimento 6. Mejora de las detecciones de la clase 'Inscripción'

Al igual que con los escudos, se han implementado varias técnicas de postprocesamiento para mejorar las detecciones de la clase "*Inscripción*". Los resultados han sido más precisos y confiables gracias al proceso explicado en la sección 2.3 y a los parámetros utilizados en la subsección 4.4.5.

Se han utilizado varias técnicas durante el postprocesamiento de las detecciones de inscripciones para mejorar su legibilidad y reducir el ruido. La conversión de imágenes a escala de grises, que elimina la información de color innecesaria y agiliza el procesamiento, es una de estas técnicas. Además, se ha utilizado un umbralizado adaptativo utilizando una fórmula *Gaussiana* para dividir las inscripciones del fondo en secciones y destacar sus detalles.

Además, se utilizaron operaciones:

```
blobs = img_umbr1 < img_umbr1.mean
```

Para eliminar las regiones oscuras de la imagen. Las inscripciones presentes en la imagen se pueden identificar y aislar mediante esta operación, mientras se eliminan otras áreas que no son relevantes para el análisis.

Posteriormente, la función `morphology.remove_small_objects(blobs_labels, promedio)` se utilizó para realizar operaciones de etiquetado y eliminación de regiones pequeñas. Esto mejora la precisión de las detecciones al eliminar posibles elementos de ruido o pequeñas áreas no deseadas que pueden haber sido identificadas como inscripciones.

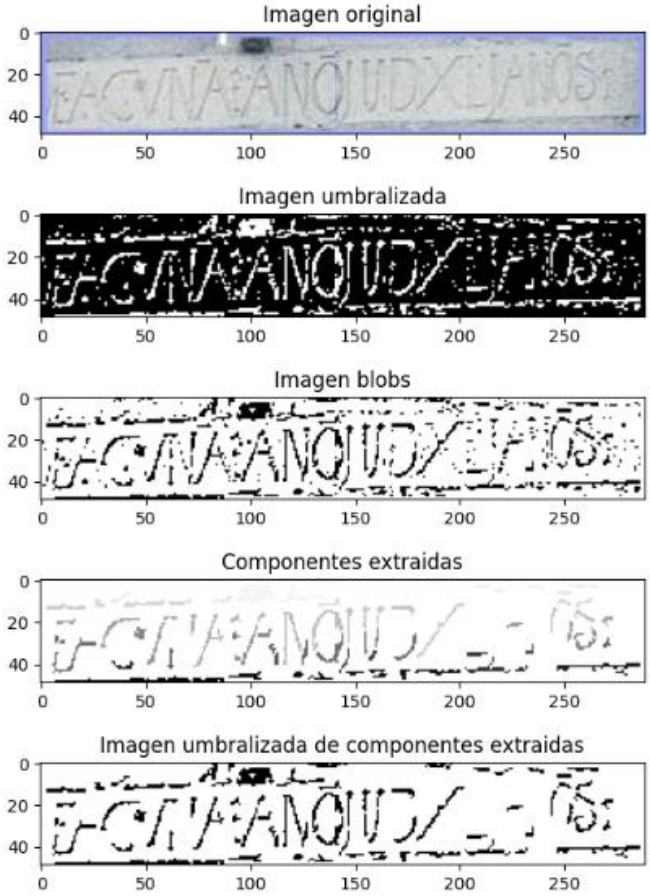

Fig. 34: Pasos de postproceso para la mejora de legibilidad de las inscripciones.

La Figura 34 muestra cómo se ha utilizado el proceso de postprocesamiento a partir del recorte de la imagen original. Cada uno de los pasos de postproceso se aplicó de manera secuencial, lo que resultó en un resultado que redujo significativamente el ruido de la imagen, aunque todavía hay algunas imperfecciones.

La calidad de la imagen ha mejorado a medida que se han implementado los pasos de postproceso. Aunque no se logró el mejor resultado, se eliminó la mayoría del ruido de la imagen inicial.

Es crucial tener en cuenta que el proceso de postprocesamiento es un conjunto de métodos que se utilizan en conjunto para mejorar las detecciones y reducir el ruido. Aunque se ha hecho mucho mejor, es posible que aún queden algunas imperfecciones debido a la complejidad de las imágenes y las características particulares de las inscripciones.

Por tanto, las operaciones de postprocesado en las inscripciones, se pueden resumir en el siguiente pseudocódigo:

// Paso 1: Convertir la imagen a escala de grises
imagen_gris = convertir_a_escala_de_grises(imagen_original)

// Paso 2: Umbralizado adaptativo utilizando máscara gaussiana
umbralizado = umbralizado_adaptativo(imagen_gris, mascara_gaussiana)

// Paso 3: Eliminar las áreas oscuras de la imagen
mascara_binaria = umbralizar_con_umbral_medio(umbralizado)

// Paso 4: Eliminar regiones pequeñas basadas en el tamaño promedio
tamaño_promedio = calcular_tamaño_promedio(umbralizado)
imagen_procesada = eliminar_regiones_pequeñas(mascara_binaria, tamaño_promedio)

// Paso 5: Crear imagen binaria resaltando las áreas de inscripción
imagen_binaria = resaltar_areas_de_inscripcion(imagen_procesada)

// Resultado: imagen_binaria contiene las áreas de inscripción resaltadas

5.3.6. Resumen de los experimentos

Es importante destacar que se han realizado otros experimentos, pero solo estos cinco se han elegido porque son los más relevantes para mejorar la metodología durante el desarrollo del proyecto. Para determinar la configuración ideal para el detector, es necesario analizar estos cinco experimentos. Una comparativa de las métricas más relevantes que se calcularon en los tres primeros experimentos se muestra en la Tabla 5.

	Precision	**Recall**	***F1-score***	**Tiempo computacional por imagen**
Experimento 1	0.89	0.53	0.66	3.9 ms
Experimento 2	0.91	0.65	0.76	3.5 ms
Experimento 3	0.92	0.66	0.77	3,4 ms

Tabla 5: Resumen de los resultados de los tres primeros experimentos de detección usando YOLOv5

Los datos muestran una similitud entre todos los experimentos, aunque se pueden analizar algunas diferencias. La configuración de parámetros del Experimento 3 produce los resultados globales más altos, lo que indica que es recomendable utilizar pesos previamente entrenados y un *batch* igual a -1 y 300 *epochs*. Esta configuración proporciona el mejor valor de *F1-score* y un tiempo de cómputo relativamente bajo.

Al examinar las matrices de confusión de los experimentos, se pueden obtener otras conclusiones. Se puede ver, por ejemplo, que los parámetros de los Experimentos 2 o 3 no se seleccionarían para priorizar la detección de escudos sobre otras categorías, o que se seleccionaría un número de *batch* más bajo si se deseaba priorizar la detección de inscripciones sobre la detección de escudos. Sin embargo, es importante destacar que no se espera un cambio significativo en la elección de estos parámetros para priorizar las diferentes clases porque los valores de *precision* y *recall* son bastante altos en todos los experimentos.

Este proyecto contiene un repositorio de *GitHub* que contiene el código creado, la memoria del proyecto y las imágenes que se han utilizado. Este *GitHub* puede ser accedido a través de la siguiente URL: https://github.com/carpimat/TFG

6. Conclusión

Este capítulo analiza las condiciones del trabajo realizado. Además, se proponen algunos temas de investigación futuros.

6.1. Conclusiones

Se puede decir que los objetivos iniciales se han alcanzado satisfactoriamente. Aunque de tamaño reducido, se ha creado una *BD* que ha permitido obtener resultados bastante aceptables en la detección de objetos de interés en una selección de imágenes de prueba. Además, se aprendió a utilizar la herramienta *Roboflow* para el preprocesamiento, el análisis y la evaluación del rendimiento del modelo utilizado en este problema.

Se pueden sacar algunas conclusiones interesantes del análisis de los experimentos realizados. En primer lugar, se observa que aumentar el tamaño del *batch* mejora el valor de *precision*, pero esto apenas tiene un impacto en la métrica de *F1-score*. Además, los resultados muestran que el primer y el segundo experimento presentan la mayoría de los falsos positivos en la clase de '*Escudo*", mientras que en el primer experimento están más repartidos entre la categoría de '*Escudo'* e '*Inscripción'*. Es importante tener en cuenta que esto puede deberse a la dificultad de definir qué se considera un escudo, ya que la red neuronal puede encontrar difícil diferenciar relieves sobre piedra que no sean manuscritos y que estén aislados de otros relieves. Aunque en algunos casos se producen falsos positivos, después de analizarlos se comprende que son objetos muy similares a escudos. En el caso de las inscripciones, también se pueden identificar ciertos elementos que la red neuronal considera como tal, pero que pueden no serlo. Se propondrán diferentes estrategias en trabajos futuros para abordar este problema.

Los hallazgos indican que YOLOv5 es un modelo adecuado para abordar el problema de detección planteado. A pesar de ser una arquitectura de detección en tiempo real optimizada, ha logrado una precisión satisfactoria. Además, una vez alcanzado el nivel de precisión deseado, esta optimización permite su uso en entornos de producción.

En cuanto al postproceso de inscripciones y escudos, cabe señalar que se ha identificado al *SSIM* como el mejor método de los analizados para el emparejamiento de escudos, y también se ha propuesto un método de trabajo de *Visión Artificial* para mejorar la legibilidad de las inscripciones previamente detectadas

6.2 Trabajo futuro

Los resultados de los experimentos demuestran que hay margen de mejora al trabajo realizado. Se recomienda aumentar la cantidad de imágenes para tener al menos 1000 instancias de cada clase, una vez superadas las restricciones de tiempo. Esto mejorará la capacidad del modelo para generalizar y distinguir entre un escudo y otras inscripciones similares.

La capacidad de realizar la detección de objetos en tiempo real en un dispositivo móvil es una mejora para esta aplicación. Los experimentos se han llevado a cabo en un entorno de procesamiento fuera de línea. La conversión de una aplicación en tiempo real en un dispositivo móvil permitiría utilizar la detección de objetos instantáneamente y en diferentes escenarios, mejorando la experiencia del usuario. Optimizar el modelo y algoritmo para su implementación en dispositivos móviles, teniendo en cuenta las limitaciones de recursos computacionales y energía, así como una interfaz de usuario fácil de comprender y usar. La aplicación se convertiría en una herramienta más práctica y versátil con la implementación exitosa de esta mejora.

Otra mejora que se propone es la capacidad de realizar un análisis más detallado del estado de las inscripciones detectadas. La implementación de esta mejora sería beneficiosa para la conservación del patrimonio histórico, donde la identificación de inscripciones deterioradas o en peligro de desaparecer es crucial. Esta mejora facilitaría la toma de decisiones y optimizaría los recursos destinados a la conservación. Para realizar este postprocesado, se necesitarán herramientas de análisis de imágenes y aprendizaje automático, y un método basado en el reconocimiento de patrones y características particulares de las inscripciones para determinar su estado. Adicionalmente, se requeriría un modelo para realizar la clasificación de manera automática, incluyendo ejemplos de inscripciones en varios estados. La aplicación podría detectar inscripciones y proporcionar información útil para preservar y conservar el patrimonio histórico y cultural.

Los grupos de escudos similares obtenidos durante el postproceso de la clase '*Escudo*' permiten realizar una clasificación más precisa y mejorar la calidad de los escudos detectados. Se propone utilizar una estrategia, basada en realizar operaciones adaptadas a cada grupo de escudos similares, usando métodos específicos para mejorar la calidad y la definición de los escudos. Se podría introducir una segunda red neuronal después de mejorar la calidad de los escudos en grupos similares para identificar formas, colores y tipos específicos de cada grupo.

Una mejora final que se propone para esta aplicación es la adopción de la arquitectura YOLOv8, la cual es una versión más reciente y sofisticada de la arquitectura YOLO, que ha demostrado ser muy efectiva para detectar y clasificar objetos en imágenes. YOLOv8 mejora la precisión y rendimiento en comparación con versiones anteriores, mejorando la capacidad de generalización, velocidad de procesamiento y detección de objetos con resultados más precisos y confiables. YOLOv8 requiere modificaciones en la estructura y configuración del modelo, así como actualización de datos de entrenamiento y pesos preentrenados. Es necesario considerar el hardware y recursos informáticos necesarios para aprovechar sus capacidades. YOLOv8 utilizará una tecnología de detección de objetos más sofisticada y eficiente que aumentará la precisión y calidad de la detección de inscripciones en la aplicación.

Referencias

[1] O. Moro Abadía, M.R. González Morales, "Paleolithic Art: A Cultural History", *Archaeol Res* **21**, 2013, pp. 269–306. URL: https://doi.org/10.1007/s10814-012-9063-8

[2] BOE (Boletín Oficial del Estado), *Ley 16/1985 del Patrimonio Histórico Español*, BOE nº 155, 29/06/1985, pp. 19619-19645. URL: https://www.boe.es/buscar/act.php?id=BOE-A-1985-12666

[3] Y. LeCun, Y. Bengio & G. Hinton, "Deep learning". *Nature*, **521**(7553), 2015, pp. 436-444. URL: https://www.nature.com/articles/nature14539

[4] M. Mazur, "A Step by Step Backpropagation Example". URL: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ (Consultado: 3/05/2023)

[5] I. Goodfellow, Y. Bengio & A Courville, "Deep Learning". *MIT Press*, 2016.

[6] T. Hastie, R. Tibshirani & J. Friedman, "The Elements of Statistical Learning: Data Mining, Inference, and Prediction", 2009, *Springer*. URL: https://web.stanford.edu/~hastie/ElemStatLearn/

[7] M.H. Alkhodari & A.S. Halabi, "Convolutional Neural Networks: An Overview and Application in Radiology", *Academic Radiology* **27**(9), 2020, pp. 1292–1299. URL: https://doi.org/10.1016/j.acra.2020.06.016

[8] J. Redmon & A. Farhadi "Yolov3: An incremental improvement", *arXiv preprint arXiv:1804.02767*, 2018. URL: https://arxiv.org/abs/1804.02767

[9] J. Redmon & A. Farhadi "YOLO9000: Better, faster, stronger", 2017, pp. 7263-7271. DOI: 10.1109/CVPR.2017.690. URL: https://openaccess.thecvf.com/content_cvpr_2017/html/Redmon_YOLO9000_Better_Faster_CVPR_2017_paper.html

[10] G. Jocher "Ultralytics YOLOv5: v5.0.0", *GitHub repository*, 2021, URL: https://github.com/ultralytics/yolov5

[11] A. Bochkovskiy, C-Y. Wang & H-Y.M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection", *arXiv:2004.10934*, 2020. URL: https://arxiv.org/abs/2004.10934

[12] C-Y. Wang, H-Y.M. Liao, I-H. Yeh, Y-H. Wu, P-Y. Chen & J-W. Hsieh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN", *arXiv:1911.11929*, 2019. URL: https://arxiv.org/abs/1911.11929

[13] K. He, X. Zhang, S. Ren & J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition", *arXiv:1406.4729*, 2014. URL: https://arxiv.org/abs/1406.4729

[14] S. Liu, L. Qi, H. Qin, J. Shi & J. Jia, "Path Aggregation Network for Instance Segmentation", *arXiv:1803.01534*, 2018. URL: https://arxiv.org/abs/1803.01534

[15] J. Nilson, T. Akenine-Möller, "Understanding SSIM", *arXiv:2006.13846*, 2020. URL: https://arxiv.org/abs/2006.13846

[16] D.G. Lowe, "Distinctive image features from scale-invariant keypoints**",** *International Journal of Computer Vision* **60**(2), 2004, pp. 91-110. URL: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[17] P. Cunningham, S.J. Delany, "k-Nearest Neighbour Classifiers: 2nd Edition (with Python examples)", *arXiv: 2004.04523*, 2020. URL: https://arxiv.org/abs/2004.04523

[18] Y. Assael, T. Sommerschield, J. Prag, "Restoring ancient text using deep learning: a case study on Greek epigraphy", *arXiv: 1910.0.6262v1*, 2019. URL: https://arxiv.org/abs/1910.06262

[19] A. Willis, Y. Sui, K. Galor, D.H. Sanders, "Estimating Gothic Facade Architecture from Imagery", 2010, pp. 43-48, DOI: 10.1109/CVPR.2010.5543519. URL: https://www.researchgate.net/publication/224165359_Estimating_Gothic_facade_architecture_from_imagery

[20] M. Hansson, “Inscriptions and Images in Secular Buildings: Examples from Renaissance Scania, Sweden, ca. 1450–1658”**,** *International Journal of Historical Archaeology*, **26**, pp. 623–646, 2022. URL: https://doi.org/10.1007/s10761-021-00616-5

[21] “Deep Learning Applied to White Light and Narrow Band Imaging Videolaryngoscopy: Toward Real-Time Laryngeal Cancer Detection” - Scientific Figure on *ResearchGate*. URL: https://www.researchgate.net/figure/YOLOv5-architecture-representation-Color-figure-can-be-viewed-in-the-online-issue_fig2_356562369 Consultado (9/5/23)

[22] J. Redmond, S. Divvala, R. Girshick & A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, 2016, Proc. CVPR. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf

[23] S.M. Aswatha, A.N. Talla, J. Mukhopadhyay & P. Bhowmick “A Method for Extracting Text from Stone Inscriptions Using Character Spotting”, 2014, DOI: 10.1007/978-3-319-16631-5_44. URL: https://link.springer.com/chapter/10.1007/978-3-319-16631-5_44

[24] roboflow, “roboflow”, URL: https://roboflow.com/

[25] G. Jocher, “Tips for best training results”, *GitHub*, 2020. URL: https://github.com/ultralytics/yolov5/wiki/Tips-for-Best-Training-Results.

[26] Z-Q. Zhao, P. Zheng, S-T. Xu & X. Wu, “Object detection with deep learning: A review”, *IEEE Transactions on Neural Networks and Learning Systems* **30**(11), 2019, pp. 3212-3232. DOI: 10.1109/TNNLS.2018.2876865

[27]V.S. Subramanyam, “IOU (Intersection over Union)”, *Analytics Vidhya*, 2021. URL: https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef Consultado (12/05/23)

[28] A. Rosebrock, “Intersection over Union (IoU) for object detection”, *Pyimagesearch*, 2016. URL:https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ Consultado (12/05/23)

[29] M.Everingham, L.V. Gool, C.K.I. Williams, J. Winn & A. Zisserman, “The PASCAL visual object classes (VOC) challenge”, *International Journal of Computer Vision*, **88**(2), 2009, pp. 303-338. URL: https://doi.org/10.1007/s11263-009-0275-4

[30] S. Ren, K. He, R. Girshick & J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks”, *Advances in Neural Information Processing Systems*, 2015, pp. 91–99.

[31] R. Girshick, J. Donahue, T. Darrell & J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, *IEEE conference on computer vision and pattern recognition*, 2014, pp. 580-587.

[32] S. Sarfraz, A. Schumann, A. Eberle & R. Stiefelhagen, "A Taxonomy and Library for Visualizing Learned Features in Convolutional Neural Networks". *IEEE Transactions on Visualization and Computer Graphics*, **25**(1), 2019, pp. 466-476, doi: 10.1109/TVCG.2018.2865240.

[33] S. García, A. Fernández & J. Luengo, (2015), “Evaluación de modelos. In Técnicas de minería de datos”, *Editorial Universidad de Granada*, 2015, pp. 111-130.

[34] A. Géron“Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems”, *O'Reilly Media, Inc*, 2019.