Estudio del uso de redes neuronales 3D para el desarrollo de sistemas inteligentes para la colonoscopia

Miquel Tena Morales

7 de febrero de 2022

Resumen– El cáncer de colon es el segundo tipo de cáncer con mayor mortalidad en España. Detectar a tiempo los pólipos, su lesión precursora, ayuda a mitigar o erradicar su impacto. Nuestro objetivo es ayudar a los profesionales del campo de la medicina a detectar con mayor precisión estos pólipos. Para ello proponemos este estudio realizado con redes neuronales y un algoritmo que añade coherencia temporal. Para el uso de redes neuronales nos hemos basado con la red neuronal convolucional YOLO v5, la última versión de YOLO y un algoritmo que asiste al modelo para que la salida del método sea estable en el espacio y en el tiempo. Obteniendo así un mAP global en el dataset del Hospital Clínic de Barcelona de 0.330 en YOLO v5 y una mejora de hasta un 20 % en F2-Measure integrando el algoritmo de coherencia temporal.

Palabras Clave– Deep learning, machine learning, detección de pólipos, coherencia temporal, red neuronal convolucional

Abstract– Colorectal cancer is the second type of cancer with the highest mortality in Spain. Detecting polyps in time, their precursor lesion, helps mitigate or eradicate their impact. Our goal is to help professionals in the medical field more accurately detect these polyps. For this we propose this study carried out with neural networks and an algorithm that adds temporal coherence. For the use of neural networks we have based ourselves on the YOLO v5 convolutional neural network, the latest version of YOLO and an algorithm that assists the model so that the output of the method is stable in space and time. Thus, obtaining a global mAP in the Hospital Clínic de Barcelona dataset of 0.330 in YOLO v5 and an improvement of up to 20 % in F2-Measure with the temporal coherence algorithm.

Keywords– Deep learning, machine learning, polyp detection, temporal coherence, convolutional neural network

1 Introducción

El cáncer de colon es el segundo tipo de cáncer con mayor mortalidad en España en el año 2020 según la SEOM (Sociedad Española de oncología médica), solamente superado por el cáncer de pulmón [1].
Los pólipos son unas protuberancias que se encuentran en las mucosas del recto, algunos son benignos y otros son malignos y existen varias maneras de clasificarlos, aunque para este proyecto no nos hace falta entrar en detalles, ya que nos centramos solamente en su detección. Estas detecciones se pueden hacer a partir de colonoscopias, que son exploraciones que permiten la visualización del intestino grueso realizadas por profesionales en el campo de la medicina.

La tasa global de pólipos no detectados (de todos los tamaños) en varios estudios realizados en 2012, se estimó que era de un 16.8 % con aproximadamente 149 pacientes totales [2]. Consideramos este porcentaje muy alto, ya que una no detección implica varios problemas relacionados con la salud del paciente, como por ejemplo que el pólipo no detectado evolucione en cáncer, o que la colonoscopia se tendría que repetir en otro momento, lo que demora la detección de posibles pólipos cancerígenos. En cualquiera de los dos casos el paciente sale perdiendo, y más aún si su vida depende de estas detecciones. Es por esto lo que nos motiva a realizar este proyecto: a mejorar la detección de estos pólipos cancerígenos mediante redes neuronales en el tiempo.

• E-mail de contacto: Miquel.Tena@campus.cat
• Mención realizada: Ingenieria de la Computación
• Trabajo tutorizado por: Jorge Bernal del Nozal (Departamento)
• Curso 2021/22

Hace aproximadamente 20 años empezaron a surgir las primeras soluciones computacionales a este problema, pero estas soluciones eran diseñadas para resolver problemas concretos, las cuales utilizaban métodos como la extracción de características locales de las imágenes. Estos métodos son una de las razones por las que estos métodos no generalizaban correctamente en datasets externos.

Debido a este problema, surgieron las soluciones que empleaban algoritmos de machine learning. Los cuales sí son capaces de generalizar a través de múltiples datasets y permiten obtener un gran rendimiento. Aún así queda trabajo para mejorar estos algoritmos, ya que también tienen problemas a la hora de realizar detecciones de pólipos de tamaño reducido.

Es por eso que proponemos una solución basada en algoritmos de deep learning, las cuales nos ayudarán a resolver el problema explicado anteriormente. En este artículo se explican los pasos que se han realizado para la implementación de dicha red y qué herramientas se han usado. Debido a que las redes neuronales, por defecto, tratan las imágenes sueltas. Es decir, a la hora de detectar una imagen en concreto no tendrá en cuenta las detecciones realizadas en imágenes anteriores, lo que generará una salida poco estable y a la vez los profesionales perderán la confianza en el uso de este método. Es por eso que creemos necesario añadir un algoritmo que tenga en cuenta la coherencia temporal, en las que se encuentran los conjuntos de imágenes correspondientes a cada colonoscopia.

1.1. Objetivos

El objetivo principal de este trabajo ha sido desarrollar un sistema de detección de pólipos en imagen de colonoscopia con una salida estable en el tiempo y el espacio. Para conseguirlo nos hemos marcado varios objetivos secundarios:

1. Se tiene que hacer una correcta implementación de una red neuronal que sea capaz de detectar objetos, en este caso hemos optado por YOLO v5 [3].

2. Una vez hayamos realizado un primer experimento, encontrar la configuración que optimiza la detección de pólipos.

3. El siguiente objetivo es implementar un algoritmo que incluya coherencia temporal a las detecciones realizas por el modelo entrenado.

4. Por último estudiar todos los resultados obtenidos para proponer mejoras de cara al futuro.

1.2. Requisitos del proyecto

Para la realización de este proyecto, se necesita un hardware potente, debido a la alta exigencia de las redes neuronales utilizadas. El hardware que ha sido usado por estas redes neuronales ha sido mayoritariamente la GPU (Unidad de procesamiento gráfico). Además, también ha sido de utilidad software que simplifica la complejidad de este proyecto. Presentamos esta información más detallada en las tablas 1 y 2, respectivamente.

	Hardware
CPU	Intel Core i7-9700K @ 3.60GHz
RAM	16 GB (2x8 GB) @ 2133 MHz
GPU	NVIDIA GeForce GTX 1070 8GB
MEMORIA	1 TB HDD + 625GB SSD

TABLA 1: HARDWARE PROPUESTO PARA LA REALIZACIÓN DEL PROYECTO.

Software	**Versión**
CUDA	V10.1.243
Python	3.8.10
SO	Windows 10

TABLA 2: SOFTWARE PROPUESTO PARA LA REALIZACIÓN DEL PROYECTO.

1.3. Metodología

En referencia la metodología de trabajo, hemos usado la metodología Kanban. Escogimos esta metodología porque es la que mejor se adaptaba a nuestro proyecto, con reuniones semanales con el tutor. En cada reunión se acordaba la tarea y/o estudio y el tiempo de realización de dicha tarea. El método Kanban consiste en dividir el trabajo en tareas. Las tareas pueden estar en tres fases: solicitada, en proceso y realizada. En nuestro caso funcionaba de manera cíclica:

1. Se acordaba una reunión con el tutor. En esta reunión se acordaba la tarea/estudio a realizar en una semana vista.

2. En un sprint de una semana (normalmente), trabajábamos en la tarea/estudio. Una vez terminado se acordaba una reunión con el tutor.

3. En esta nueva reunión, se evaluaba el trabajo realizado de la última semana, y se proponían cambios y nuevas tareas.

1.4. Riesgos

Antes de realizar el proyecto se hizo un estudio sobre los posibles riesgos con los que nos podíamos enfrentar durante el desarrollo. Para eso se hizo una tabla definiendo los riesgos y el método de contención de estos. Los riesgos que teníamos en cuenta hacen referencia al dataset usado para el proyecto, recursos hardware y conocimientos previos a la implementación. Una información más detallada se puede encontrar en la tabla de contingencia número 3.

1.5. Planificación

Con tal de realizar correctamente este proyecto, llevamos a cabo un calendario en el cual organizamos por fechas las diferentes fases del trabajo. Así pues, a modo de sprints organizados en semanas de trabajo, hemos estado trabajando en diferentes aspectos del proyecto. En la tabla 4 podemos observar dicha planificación.

Riesgo	Explicación	Probabilidad	Contingencia
No tenemos suficientes datos en la base privada	La base de datos del clinic es pequeña	Media	• Buscar bases de datos públicas en internet. • Data augmentation
No disponer de suficientes recursos hardware	Los recursos hardware de los que disponemos no son suficientes para realizar el entrenamiento	Baja	Pedir al tutor acceso a recursos hardware del CVC.
Problemas a la hora de implementar la LSTM	El desconocimiento y/o falta de tiempo para la implementación de esta red neuronal puede frenar el trascurso del trabajo	Media	Implementar un algoritmo que solucione el problema de la coherencia temporal.

TABLA 3: RIESGOS Y SU CORRESPONDIENTE CONTINGENCIA

Descripción	Fecha inicio	Fecha final
Búsqueda bibliográfica y estado del arte del problema	01/09	10/10
Estudio del dataset y segmentación de imagen	10/10	15/10
Estudio de redes candidatas y primeros experimentos	15/10	20/10
Desarrollo de la solución (Codificación, entrenamiento, predicción)	20/10	20/01
Estudio de los resultados obtenidos	20/10	20/01
Documentación del proyecto	01/09	07/02
Realización del póster final	07/02	13/02

TABLA 4: PLANIFICACIÓN TEMPORAL DEL PROYECTO

2 ESTADO DEL ARTE

Antes de empezar a desarrollar nuestra propuesta, investigamos artículos que tenían un símil con nuestro proyecto y observar que metodologías y herramientas habían usado. Anteriormente hemos hablado de las dos grandes metodologías usadas en estos dos últimos decenios, pero nos centraremos en esas que usan machine learning ya que se acercan más a nuestro proyecto.

Los algoritmos de machine learning son aquellos que construyen un mdelo matemático basado en patrones e inferencia a partir de datos de muestra, conocidos como "training data". El incremento de GPUs (Graphic Processing Unit) y la gran disponibilidad de datasets públicos, han permitido que esta metodologia se aplique en la tarea en lo que a detección de pólipos se refiere.

Se usan varias arquitecturas en relación a la detección de pólipos colorectales. Entre ellas se incluyen métodos de una etapa como SSD y YOLOv3, y métodos de dos etapas: Faster R-CNN, Mask R-CNN y similares.

En julio de 2018, Zheng et al. [4] publicaron una aplicación de un detector de objetos a tiempo real basado en You Only Look Once (YOLO). YOLO es un detector de objetos basado en la regresión que observa la imagen entera una vez para realizar la detección. Utiliza una CNN a la vez para predecir las bounding boxes y la probabilidad a la clase a la que pertenecen.

En setiembre de 2018 Mo et al. [5] utilizo una Faster R-CNN para la detección de pólipos. Faster-RCNN es un detector que reemplaza la ROI (region of interest) con una regional proposal network (RPN), consiguiendo resultados muy competitivos, siendo así una propuesta eficiente para la práctica clínica.

En abril de 2019, Zheng et al. [6] propusieron un metodo para la detección de pólipos y su seguimiento en los frames posteriores. El método consiste de dos partes. Inicialmente detecta y localiza los pólipos con un detector de objetos, U-Net. Con tal de obtener información temporal y hacer un seguimiento de los pólipos en frames utilizaron un flujo óptico. También, para evitar los fallos en el seguimiento causado por el movimiento de la camara, entrenaron un modelo de regresión de movimiento y una CNN, la cual es entrenada en el momento eficientemente utilizando la información de frames anteriores.

En diciembre de 2021, Wan et al. [7] propusieron la implementación del modelo YOLO v5 basado en un mecanismo de atención propia para la detección de pólipos. En el proceso de la extracción de características, añadieron un mecanismo de atención con tal de mejorar la contribución de información buena de características y debilitar las interferencias de canales inservibles.

3 DESARROLLO TECNOLÓGICO

En esta sección se presentan las dos implementaciones que hemos realizado: una implementación de la red neuronal Yolo v5 para la detección de pólipos y nuestro algoritmo que añade coherencia temporal a dicha detección, programado en Python [8]. En esta sección se explica la metodología que se ha seguido para obtener los resultados que se

encuentran al final de este artículo.

3.1. You Only Look Once v5 (YOLO v5)

YOLO [9], la versión abreviada de You Only Look Once (Solo miras una vez en castellano), es un algoritmo que propuso Redmond et al. el cual consiste en una red neuronal end-to-end que predice bounding boxes (cajas que envuelven el objeto detectado) y la probabilidad de la clase a la que pertenece, todo a la vez. La última versión oficial es la tercera aunque han surgido otras dos que no se consideran oficiales.

YOLO v4 [10] fue propuesta por Bochkovskiy et. al. en 2020 con tal de mejorar la anterior versión, YOLO v3. El algoritmo consigue mejores resultados y más rápidos en el estado del arte en lo que a detectores de este estilo se refiere. La arquitectura de YOLO v4 se divide en 4 bloques: Backbone, neck, dense prediction y sparse prediction, como podemos observar en la figura 1. El backbone es la arquitectura de extracción de características, la cual es diferente a YOLO v3. Su nombre es **CSPDarknet53**, que significa Cross-Spatial -Partial connections, el cual es un nuevo backbone que puede mejorar la capacidad de apredizaje de la CNN. Otro modulo adicional del que dispone YOLO v4 respecto YOLO v3, es la spatial pyramid pooling block, que esta contigua a CSPDarnket53 para incrementar el campo receptivo y separar las características contextuales más significativas. Por último, a diferencia de YOLO v3, la versión 4 utiliza la PANet (Path Aggregation Network), con la misión de mejorar el proceso de segmentación de la imagen preservando la información espacial.

Nosotros hemos optado por usar la versión más reciente: YOLO v5 [3], que fue publicado por Glenn Jocher et al. poco después de publicarse la versión 4, utilizando el framework de PyTorch [12] [13].

YOLO v5 se presentaba como una mejora de YOLO v3, implementando el mismo backbone (CSPDarnket53) que YOLO v4 y el mismo cuello (PANet). Las diferencias con la versión 4 de YOLO, se encuentra en la mejora de auto learning bounding box anchors e incluye mosaic data augmentation. Roboflow [14], un equipo de trabajo relacionado con Ultralytics [12], publicó en setiembre de 2021 [15] un articulo en su blog dónde aseguran que "YOLO v5 es más rápido y ligero que YOLO v4" y que "la accuracy de YOLO v5 está a la par con YOLO v4", aunque a día de hoy no hay publicado un artículo científico que soporte dichas afirmaciones.

Decidimos usar esta versión ya que nos llamaba la atención que fuese open-source, adaptada a PyTorch, un framework el cual habíamos visto previamente y, además, nos seducía la idea de que los modelos fuesen ligeros y rápidos, dado que disponemos de un hardware que nos limita el desarrollo del proyecto y en referencia a nuestro objetivo, que se enfoca en ayudar a la detección en colonoscopias a tiempo real, es importante que el cómputo de las imágenes sea ágil con tal de facilitar la inspección al personal médico.

3.1.1. El algoritmo de YOLO

El algoritmo de YOLO funciona igual que un problema de regresión y utiliza una red neuronal convolucional (CNN), muy efectiva en el campo de la visión artificial. Una versión resumida del algoritmo se encuentra a continuación:

1. La idea principal en la que se basa YOLO es en segmentar la imagen de entrada en imágenes más pequeñas, de manera que la imagen queda dividida en una cuadricula de dimensiones S x S [9]. Cada uno de los recuadros que forman esta cuadricula es responsable de la detección de objetos que aparecen en ellos. Es decir, si el centro del objeto se encuentra en alguno de los recuadros, aquel recuadro será el responsable de detectarlo.

2. Cada uno de los recuadros pronostica una bounding box junto con una calificación, que describe la confianza que tiene el modelo de que el objeto que se encuentra en el recuadro pertenezca a cierta clase. YOLO utiliza estas bounding boxes como un problema de regresión con tal de predecir el objeto real.

3. La métrica Intersection over Union (IoU) determina en qué medida estas predicciones corresponden con el objeto original (ground truth). Este método elimina así las bounding box que no corresponden con las características del ground truth (altura, ancho de la imagen).

Un esquema visual del algoritmo básico de YOLO se encuentra en la figura 2.

3.2. Algoritmo propuesto para solucionar la coherencia temporal

Como ya hemos contado anteriormente, una red neuronal convolucional como YOLO, analiza y predice los frames sin tener en cuenta información anterior (predicciones de frames anteriores). Es por eso que hemos optado por incluir un algoritmo a la inferencia del modelo que tiene en cuenta los N frames anteriores. Para cada vídeo se hará lo siguiente:

Supongamos una lista de K frames que representan un vídeo:

- En el caso que en el frame K_i no contenga una detección, se observará si antes se ha detectado un pólipo, haciendo una media entre los N frames anteriores:
  - En caso de ser más del 50 %, se hará la media de las bounding boxes, y el resultado se aplicará en el frame K_i.
  - En caso que la primera media de igual a 50 % o menos, no se hará ningún cambio.
- En el caso que el frame contenga una detección, no se hará nada.

Un ejemplo del algoritmo en Python explicado al detalle se adjunta en la sección 2 del apéndice.

4 EXPERIMENTOS Y MÉTRICAS

En esta sección hablaremos de los datos que hemos usado, así también como los hemos tratado para mejorar los primeros resultados obtenidos. Explicaremos también las métricas empleadas para estudiar los resultados.

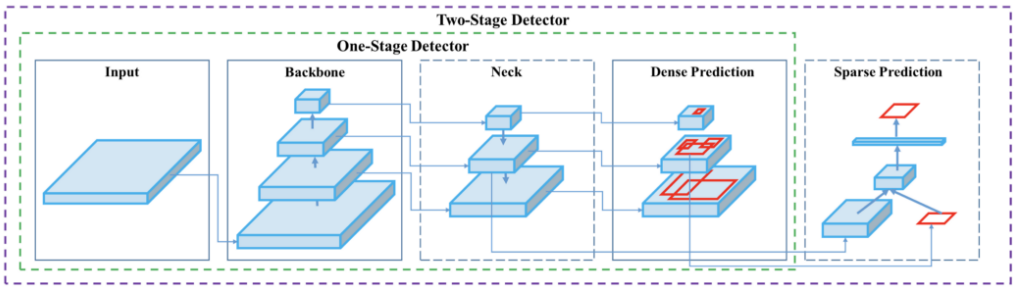


Fig. 1: Arquitectura de YOLO v4 ([11]).

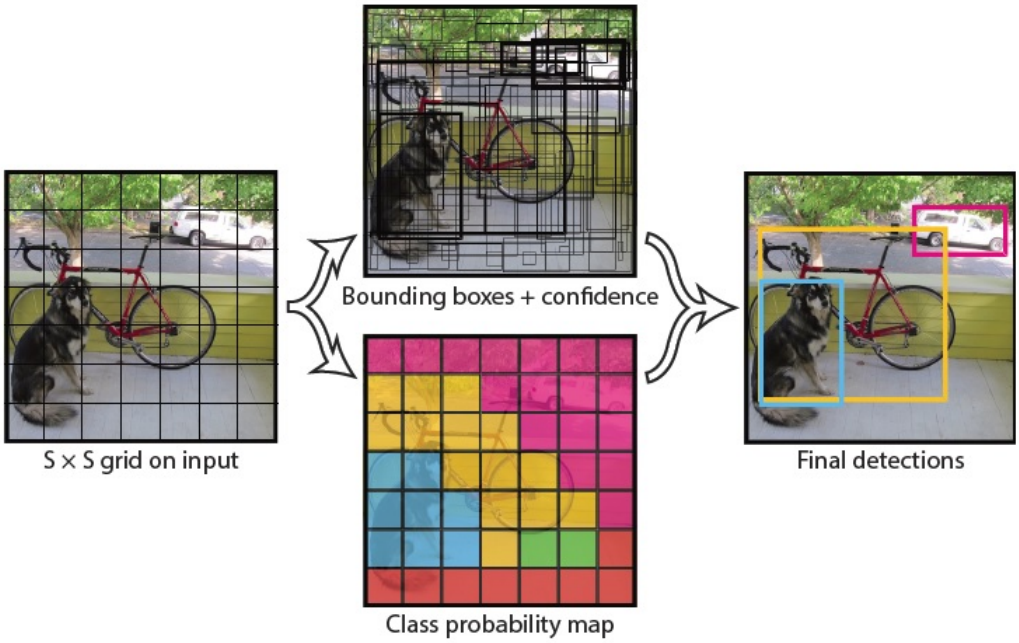
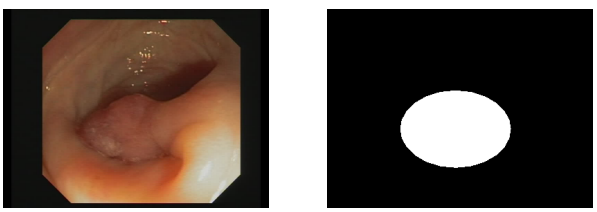


Fig. 2: Pasos para la detección de objetos de YOLO. 1. Imagen cuadriculada (izquierda), 2. Predicción de la bounding box (centro, arriba), 3. Probabilidad de pertenecer a una clase (centro, abajo), 4. Detección final (derecha) ([9]).

4.1. Dataset

El dataset ha sido proporcionado por el Hospital Clínic de Barcelona, y consta de treinta y seis vídeos de colonoscopias. Cada vídeo está desglosado en sus frames correspondientes y, para cada uno de ellos, se cuenta con una máscara binaria la cual nos indica la posición del pólipo. Estas máscaras binarias han estado realizadas por profesionales del campo de la medicina, lo que nos puede asegurar que el ground-truth es correcto. Un ejemplo del dataset descrito se puede observar en la figura 3.



(a) Frame (b) Máscara binaria

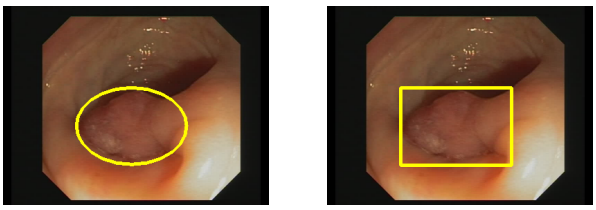
Fig. 3: Representación del dataset proporcionado.

4.1.1. Preparación de los datos

Una vez hemos estudiado el dataset, hay que prepararlo para que funcione en YOLO v5. Para eso hemos necesitado la biblioteca OpenCV [16] de Python, la cual dispone de dos funciones que nos han sido de gran utilidad: findContours y boundingRect. Los ejemplos proporcionados más adelante forman parte del ejemplo de ground-truth de la figura 3.

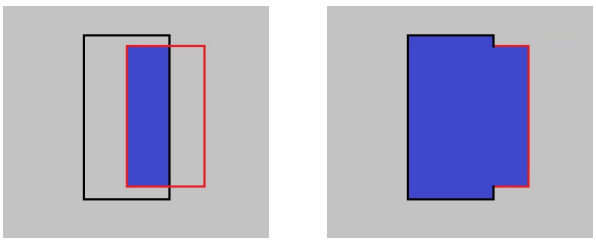
- **findContours:** su función es encontrar los contornos de una imagen binaria pasada por parámetro. Un ejemplo de esta función se puede encontrar en la primera imagen de la figura 4.
- **boundingRect:** su función es transformar los contornos encontrados en la función anterior en un rectangulo. Un ejemplo de esta función se puede encontrar en la segunda imagen de la figura 4.

Una vez se han obtenido las coordenadas del rectángulo, hay que transformarlas al formato que acepta YOLO v5. El formato es el siguiente: [centro x, centro y, ancho, alto], en las cuales el centro x y centro y son coordenadas normalizadas del centro del rectángulo encontrado anteriormente. El ancho y el alto son las coordenadas normalizadas del an-



(a) findContours (b) boundingRect

Fig. 4: Representación de las funciones findContours y boundingRect.



(a) Área de intersección (b) Área de unión

Fig. 5: Ejemplo de intersección y unión entre ground-truth (negro) y la predicción (rojo).

cho y el alto de la imagen. Estas coordenadas se guardan en ficheros de texto con el nombre de la imagen.

4.1.2. Train, validation y test

Como bien hemos dicho antes, el dataset se divide en 36 vídeos. Cada vídeo esta descompuesto por sus frames, imágenes binarias que representan el ground-truth y los ficheros que representan. Por tal de poder comparar resultados con estudios realizados por otras personas, hemos decidido utilizar 18 vídeos para entrenar y validar, y 18 restantes para realizar nuestros test, siguiendo las recomendaciones del GIANA challenge celebrado en la conferencia MICCAI 2021 [17] con el objetivo de poder realizar una comparación justa con otras metodologías.

4.1.3. Data Augmentation

El aumento de datos es un método para incrementar el número de datos que se utilizarán en el entreno de cualquier red neuronal, en caso de que se considere que los datos existentes sean escasos o no lleguen a representar todas las variantes que estos pueden tomar. Es por esto que el aumento de datos es una práctica extendida en nuestro campo.

Una vez empezamos los primeros experimentos, los resultados obtenidos eran bastante pobres. Después de estudiar las probables causas de un rendimiento bajo, nos dimos cuenta que el problema residía en la falta de datos en el entrenamiento. Por eso optamos por aumentar los datos, y lo hicimos utilizando la biblioteca de Python Albumentations [18].

Por tal de aumentar el número de imágenes, se hacen diferentes transformaciones a las mismas. Se hicieron diez transformaciones para cada imagen, entre ellas rotaciones, traslaciones, cambios de color, cambios de gamma y difuminación (mejor deja el nombre en inglés). Todas estas transformaciones pueden realizarse a la vez, según una probabilidad preestablecida.

4.2. Métricas de evaluación

Con tal de evaluar nuestros resultados, necesitamos métricas que nos indiquen lo bien o mal que lo estamos haciendo. En este caso, el objetivo es validar si las bounding boxes proporcionadas por la red coinciden con el ground truth y, para ello, nos apoyaremos en la métrica conocida como Intersection over Union (IoU).

Esta métrica nos servirá para saber cuánto se acerca cuantitativamente el bounding box predicho por el modelo al bounding box del ground-truth. El IoU se calcula mediante la siguiente ecuación, dividiendo el área de intersección entre las dos cajas entre su área de unión. Una representación gráfica del área de intersección y de unión se encuentra en la figura 5. Cuánto más grande sea el IoU, más precisa habrá sido la predicción. En nuestro caso hemos decidido que un 50 % de IoU es suficiente para determinar que se ha hecho una buena predicción.

$$IoU = \frac{\text{Área de intersección}}{\text{Área de unión}}$$

Además, es necesario explicar qué son para nosotros true/false positive y true/false negative para entender el resto de métricas.

- **True Positive (TP):** Consideraremos true positive cuando el modelo/algoritmo haya predicho un pólipo y el ground-truth así lo indique, para evaluar si la predicción concuerda con la del ground-truth se utiliza el IoU.
- **False Positive (FP):** Consideraremos una predicción como false positive en caso de que el modelo haya predicho un pólipo, cuando no lo hay en el ground truth.
- **True Negative (TN):** En caso de que el modelo/algoritmo nos indique que no hay pólipo y así sea en el ground-truth, la predicción se considerará como true negative.
- **False Negative (FN) :** En caso que el modelo/algoritmo nos inidique que no hay pólipo y en el ground-truth nos indique que existe pólipo, la predicción se considerará como false negative. Para evaluar si la predicción concuerda con la del ground-truth se utiliza el IoU. También es falso negativo cuando la evaluación del IoU no supera el treshold seleccionado.

Para evaluar el rendimiento de los diferentes métodos probados, hemos empleado diferentes métricas como Precision, Recall que nos han ayudado a calibrar la configuración del modelo que utiliza YOLO v5.

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

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

Con el fin de poder integrar ambas métricas y tener una idea del funcionamiento global del sistema, hemos empleado también la F-Measure que es una media armónica que combina los valores de precisión y recall. La formula general para un número real β es:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

En nuestro caso hemos utilizado F_1-$Measure$ y F_2-$Measure$, con β = 1 y β = 2, respectivamente.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

$$F_2 = 5 \cdot \frac{\text{precision} \cdot \text{recall}}{(4 \cdot \text{precision}) + \text{recall}}$$

Finalmente, la mean average precision (mAP) nos ayuda a evaluar cómo de bueno es un modelo, en nuestro caso un modelo de detección de objetos. Mean average precision básicamente compara la bounding box predicha con la bounding box del ground-truth (IoU) y nos retorna una calificación. Como más alta sea esta calificación, mejor será el modelo.

Para entender el funcionamiento de esta métrica y cómo afecta el IoU en los resultados, hay que saber de antemano las otras métricas que la conforman y cómo se calculan.

El Average Precision (AP) es una manera de resumir la curva precision-recall en un solo valor que representa el promedio de todas las precisiones. El AP se calcula a partir de la siguiente ecuación.

$$AP = \int_0^1 p(r)dr$$

La curva de Precision-Recall representa el conflicto entre precisión y recuperación. Normalmente,

- los modelos con alta precisión y baja recall producen predicciones muy seguras pero pierden una parte de las instancias.
- los modelos con baja precisión y alta recall pueden encontrar la mayoría de los objetos, pero las predicciones son falsos positivos hasta cierto punto y la confianza disminuye.

Una vez sabemos calcular el AP, calcular el mAP es sencillo:

$$MAP = \frac{\Sigma_{q=1}^{Q} \text{AP(q)}}{Q}$$

Como podemos ver en la formula anterior, el mAP es la media de AP para cada clase (q).

En nuestros resultados podemos ver el mAP referenciado de esta manera: **mAP@0.5** y **mAP@0.5:95**. Esta anotación nos informa del umbral de IoU que estamos usando en cada métrica. En el segundo caso, el mAP se compara promediando sobre el IoU de 0.5 a 0.95 en intervalos de 0.05.

5 RESULTADOS

En esta sección mostraremos los resultados obtenidos después de los experimentos comentados anteriormente. Se mostraran los datos de igual manera que se iban obteniendo, para así entender los cambios realizados durante la realización del proyecto, como también la adición de diferentes métricas que nos han ayudado a mejorar los resultados.

En el caso de entrenar Para comprobar qué configuración es mejor (número de frames a observar anteriormente del frame actual), se han usado las métricas precisión, recall, y F-MEASURE.

5.1. Primeros entrenamientos

Para la detección de pólipos necesitábamos encontrar la configuración que mejor se adaptase a nuestro hardware y necesidades. Para ello realizamos varios experimentos con diferentes configuraciones, siendo las más significativas las que podemos encontrar en la tabla 5. Podemos ver como el data augmentation (DA) mejora nuestros resultados obtenidos en el dataset simple.

Nos dimos cuenta que con pocas epochs y un batch-size bajo (modelo 1) conseguíamos un mAP@0.5:0.95 bastante bueno respecto al de otros investigadores, incluso mejor que el modelo entrenado con 10 epochs y un batch-size de 20 (modelo 2). Decidimos aumentar de epochs y batch-size porque se obtenía un mejor resultado y aprovechamiento de los recursos hardware, según el creador de YOLO v5 [19].

5.1.1. Comparación de modelos

Una vez obtenidos resultados que nos parecían correctos, quedaba estudiar como se comporta cada modelo con cada vídeo que se encuentran en los test. Para ello calculamos el mAP@0.5:0.95 para cada uno, como podemos ver en la figura 6. Se puede encontrar una tabla más detallada (7) con todas las métricas en el apéndice de resultados.

En ella podemos ver como el modelo 1 (azul), obtiene una peor calificación que el modelo 2 (naranja) en la mayoría de vídeos. Es por eso que consideramos que el modelo 2 generaliza mejor que el modelo 1. Esto es debido a la diferencia de configuración entre modelos, dónde modelo 2 ha podido ser entrenado más que modelo 1.

Haciendo un análisis más exhaustivo vídeo a vídeo, podemos ver como el modelo funciona muy bien en vídeos como el vídeo 2, o el 15, dónde añadir coherencia temporal en esos vídeos será prácticamente ineficaz, a diferencia de los vídeos 8 y 18, dónde veremos un incremento notable cuando añadamos coherencia temporal.

5.2. Resultados mejorados con algoritmo de coherencia temporal

Una vez tenemos escogido el modelo que mejor generaliza de los que hemos podido entrenar, toca mejorarlo añadiendo coherencia temporal.

Para ello y poder comparar los resultados con los del modelo anterior, calculamos en F1-Measure y el F2-Measure. Podemos observar en la tabla 6 como a medida que vamos aumentando los frames perdemos precisión, debido a que realizamos la media de bounding boxes que se encuentran en los n-frames anteriores. Aún así, podemos observar una notable mejora en un 20 % del recall, es decir, podemos decir que nuestro algoritmo de coherencia temporal aumenta la detección de pólipos que nuestro modelo no puede realizar.

Con tal de asegurarnos que nuestro método mejora el modelo, podemos ver como también mejoramos en f1-measure y f2-measure, dónde la pérdida de precisión la podemos

DA	epochs	batch-size	precision	recall	mAP@.5	mAP@.5:.95
no	3	4	0.679	0.583	0.598	0.280
si	3	4	**0.751**	**0.615**	**0.689**	**0.371**
no	10	20	0.676	**0.566**	**0.614**	0.297
si	10	20	**0.689**	0.563	0.582	**0.330**

TABLA 5: IMPACTO DE LOS RESULTADOS DE YOLO V5 ASOCIADOS AL DATA AUGMENTATION (DA).

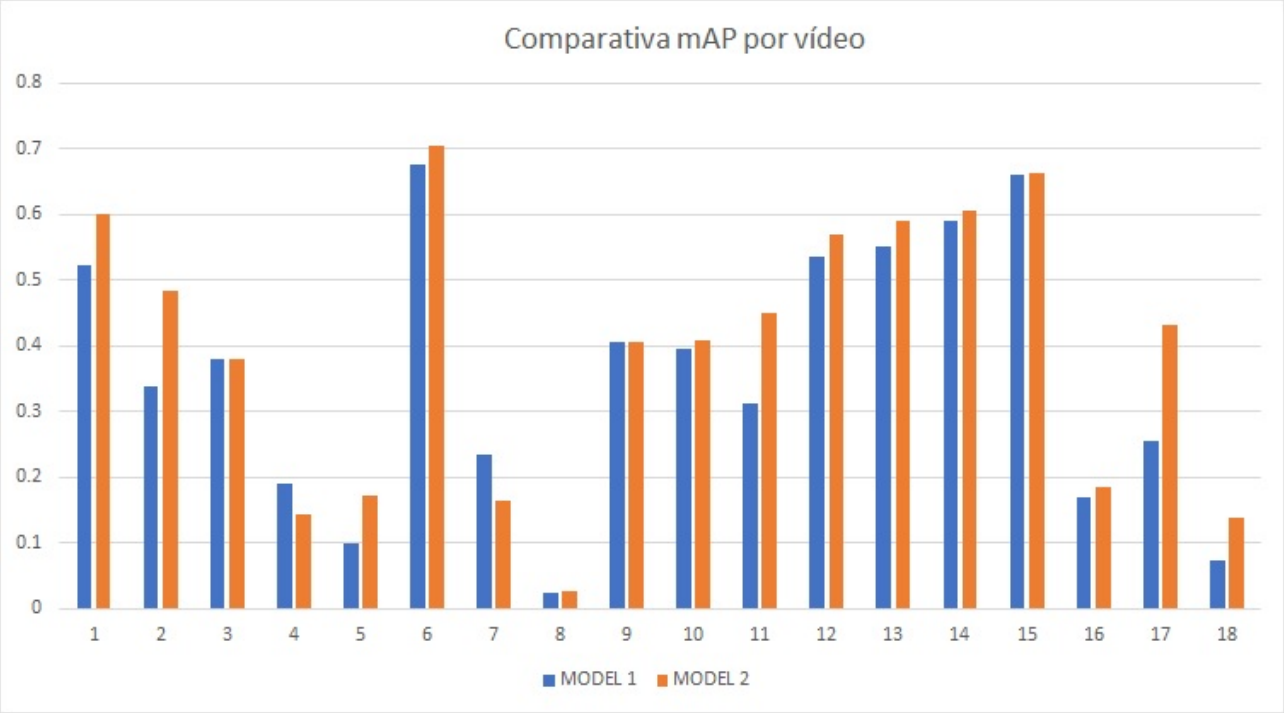


Fig. 6: Comparativa de modelos para cada vídeo.

frames	precision	recall	f1-measure	f2-measure
básico	**0.689**	0.563	0.614	0.582
3	0.633	0.770	**0.695**	0.738
5	0.629	0.771	0.693	0.738
7	0.624	0.774	0.691	0.738
10	0.619	**0.782**	0.691	**0.743**

TABLA 6: COMPARACIÓN DE RESULTADOS DESPUÉS DE APLICAR COHERENCIA TEMPORAL.

compensar con la detección de pólipos que nuestro modelo no era capaz de detectar.

El mismo estudio se realizó para cada vídeo utilizando las métricas f1-measure y f2-measure, los resultados se pueden ver en las figuras 7 y 8 respectivamente.

La línea roja representa como se comporta el modelo con cada vídeo, respecto a los resultados obtenidos con nuestro método representado por las barras de 3, 5, 7 y 10 frame de coherencia temporal. Podemos ver como nuestro método asiste al modelo y mejora las predicciones en la mayoría de vídeos. Podemos considerar por lo tanto nuestro método un éxito. En referencia a los vídeos 1 y 12 los podemos considerar como outliers, es decir, vídeos a los que no les favorece nuestro método.

5.3. Nuestro método implementado en un entorno real

Después de implementar esta solución, nos queda medir si nuestro método es aplicable en un entorno real. Para ello necesitamos cronometrar lo que tarda en hacer inferencia el modelo, y en aplicar la coherencia temporal.

Sabemos que los instrumentos utilizados en las colonoscopias de los datasets usados en este proyecto trabajan a 25 frames per second (FPS), por lo tanto tenemos aproximadamente 0.04 segundos de margen para realizar la detección y nuestro método tarda 0.047 segundos en una GPU Nvidia GForce 1070. Por lo tanto nuestro método no sería aplicable en un entorno real por poco tiempo.

6 CONCLUSIONES

En este proyecto hemos podido implementar un método de detección de pólipos con la última versión de YOLO v5, una de las redes más punteras en detección de objetos en este momento, y, por si fuera poco, mejorarla con nuestro método que añade coherencia temporal en esos casos en los que el modelo de YOLO v5 falla. En este trabajo he aprendido a realizar un estudio utilizando redes neuronales y darme cuenta que no hay que estar satisfecho con los primeros resultados obtenidos, indagar en el método para sacarle el mayor rendimiento. También he podido realizar un estudio

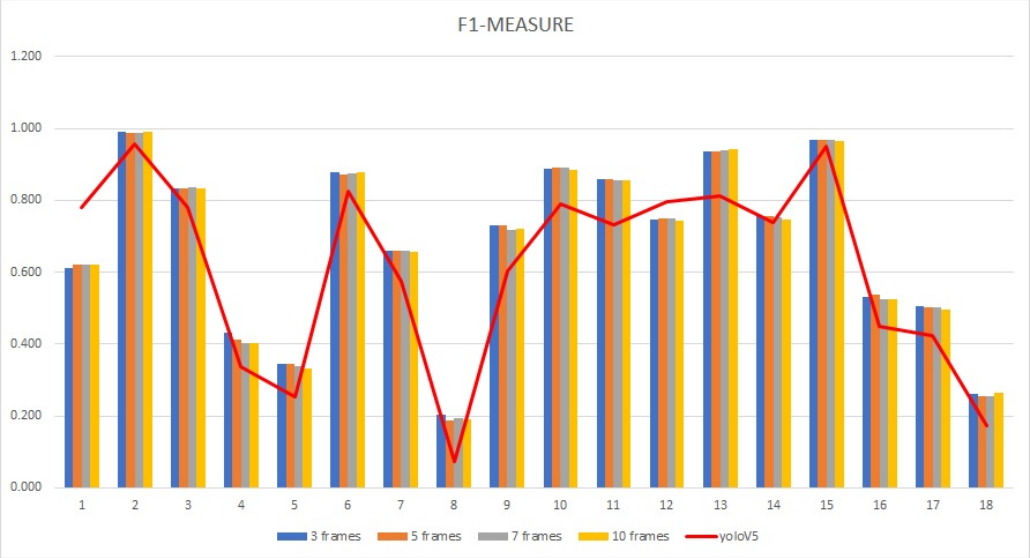


Fig. 7: Comparativa de nuestro método con el modelo básico para cada vídeo. (F1-Measure)

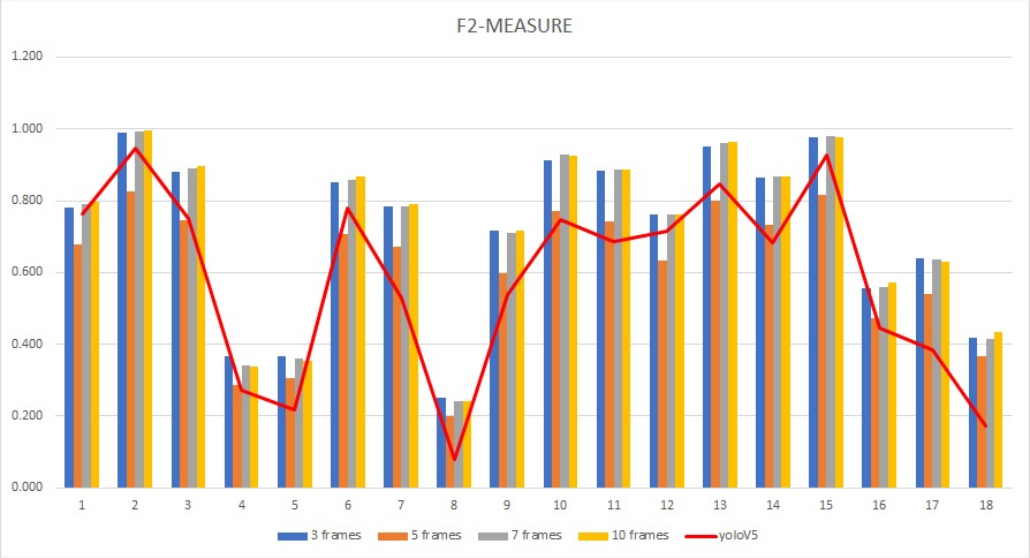


Fig. 8: Comparativa de nuestro método con el modelo básico para cada vídeo. (F2-Measure)

y he desarrollado un sistema que puede servir para salvar la vida de las personas, y eso ha sido una motivación extra para sacar lo mejor de mi. Pero hay algo que se ha quedado por el camino y es el segundo objetivo que nos habíamos marcado, el de implementar una Long-Short Term Memory (LSTM) para implementar el método de coherencia temporal. Debido a la falta de tiempo y de recursos hardware no fue posible, y nos queda como trabajo pendiente la implementación de esta y estudiar si mejora nuestra implementación. Debido a que decidimos comparar resultados con los de otros estudios, no pudimos utilizar bases de datos públicas y creemos que añadir esa información al modelo puede ser muy útil a la hora de mejorarlo. Creemos que estas soluciones pueden causar un impacto positivo en el estudio realizado en este artículo y esperamos poderlo realizar pronto.

AGRADECIMIENTOS

A mi mentor Jorge Bernal del Nozal, quien me propuso este proyecto y me ha hecho ver el lado positivo de los resultados que íbamos obteniendo.

Al Hospital Clínic de Barcelona, por proporcionarnos los datasets.

A mis profesores que he tenido al largo de la carrera, sin ellos no hubiera sido posible escribir cada línea de código y evaluación crítica de los resultados obtenidos, con especial cariño a los profesores de la mención en Computación.

Y por último a mis padres y a mi tía, quienes me animaron a seguir estudiando e hicieron posible la inscripción a esta Universidad.

REFERENCIAS

[1] S. E. de Oncología Médica, *Las cifras del cáncer en España 2021*. 2021.

[2] S. B. Ahn, D. S. Han, J. H. Bae, T. J. Byun, J. P. Kim, and C. S. Eun, “The miss rate for colorectal adenoma determined by quality-adjusted, back-to-back co-

lonoscopies," *Gut and Liver*, vol. 6, no. 1, p. 64–70, 2012.

[3] G. Jocher, A. Stoken, J. Borovec, NanoCode012, ChristopherSTAN, L. Changyu, Laughing, tkianai, A. Hogan, lorenzomammana, yxNONG, AlexWang1900, L. Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, F. Ingham, Frederik, Guilhen, Hatovix, J. Poznanski, J. Fang, L. Y. , changyu98, M. Wang, N. Gupta, O. Akhtar, PetrDvoracek, and P. Rai, "ultralytics/yolov5: v5.0 - yolov5-p6 1280 models, aws, supervise.ly and youtube integrations," Abril 2021.

[4] Y. Zheng, R. Zhang, R. Yu, Y. Jiang, T. W. Mak, S. H. Wong, J. Y. Lau, and C. C. Poon, "Localisation of colorectal polyps by convolutional neural network features learnt from white light and narrow band endoscopic images of multiple databases," *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2018.

[5] X. Mo, K. Tao, Q. Wang, and G. Wang, "An efficient approach for polyps detection in endoscopic videos based on faster r-cnn," *2018 24th International Conference on Pattern Recognition (ICPR)*, 2018.

[6] H. Zheng, H. Chen, J. Huang, X. Li, X. Han, and J. Yao, "Polyp tracking in video colonoscopy using optical flow with an on-the-fly trained cnn," *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, 2019.

[7] J. Wan, B. Chen, and Y. Yu, "Polyp detection from colorectum images by using attentive yolov5," *Diagnostics*, vol. 11, no. 12, p. 2264, 2021.

[8] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[11] H. Ampadu25.00 and H. Ampadu, "Yolov3 and yolov4 in object detection," May 2021.

[12] Ultralytics, "Yolo v5 introduction."

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[14] S. Alexandrova, Z. Tatlock, and M. Cakmak, "Roboflow: A flow-based visual programming language for mobile manipulation tasks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5537–5544, 2015.

[15] J. Nelson, "Yolov5 is here," Sep 2021.

[16] OpenCV, "Open source computer vision library." `https://github.com/opencv/opencv`, 2022.

[17] "Gastrointestinal image analysis challenge - grand challenge."

[18] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, 2020.

[19] G. Jocher, "Yolov5 study: Batch size · issue 2377 · ultralytics/yolov5," Mar 2021.

APÉNDICE

A.1. Resultados extra

video	precision	recall	mAP@.5	mAP@.5:.95	F1-MEASURE	F2-MEASURE
1	0.816	0.750	0.825	0.522	0.782	0.762
2	0.973	0.940	0.964	0.339	0.956	0.946
3	0.837	0.730	0.781	0.380	0.780	0.749
4	0.559	0.240	0.408	0.189	0.336	0.271
5	0.344	0.200	0.212	0.099	0.253	0.218
6	0.914	0.750	0.855	0.677	0.824	0.778
7	0.674	0.500	0.582	0.233	0.574	0.527
8	0.070	0.080	0.047	0.023	0.075	0.078
9	0.763	0.500	0.643	0.406	0.604	0.537
10	0.878	0.720	0.819	0.396	0.791	0.747
11	0.820	0.660	0.762	0.313	0.731	0.687
12	0.978	0.670	0.787	0.536	0.795	0.715
13	0.763	0.870	0.869	0.551	0.813	0.846
14	0.852	0.650	0.775	0.591	0.737	0.682
15	0.996	0.910	0.966	0.660	0.951	0.926
16	0.460	0.440	0.415	0.170	0.450	0.444
17	0.519	0.360	0.356	0.254	0.425	0.383
18	0.179	0.170	0.105	0.074	0.174	0.172

TABLA 7: RESULTADOS POR VÍDEO

A.2. Código utilizado

Adjuntamos el código usado en el algoritmo de coherencia temporal donde:

- video: indica el vídeo el cual se quiere mejorar.
- frame_pred: resultados de las bounding boxes predichas por el modelo. En caso de no haber nada o haber fallado hay un [0].
- n_frames: número de frames el cual se tienen en cuenta para añadir coherencia temporal.
- check_frequency: mira los n_frames anteriores y determina si había pólipo o no.
- bbox_mean: Recibe como parametros los n_frames anteriores y calcula la media de las bounding boxes.
- new_coords: son las nuevas coordenadas del vídeo pasado como parámetro.

```
def algoritmo_coherencia_temporal(video, frames_pred, n):
    new_coords = defaultdict(list)
    n_frames = len(os.listdir(os.path.join(path_gt_videos, str(video))))

    for i in range(1, n+1):
        actual_frame = f"{video:03d}-{i:04d}"
        new_coords[actual_frame] = frames_pred[actual_frame]

    for i in range(n+1, n_frames+1):
        actual_frame = f"{video:03d}-{i:04d}"
        if frames_pred[actual_frame] == [0]:
            frames = [frames_pred[f"{video:03d}-{j:04d}"] for j in range(i-1, i-n-1, -1)]
            if not check_frequency(frames):
                new_coords[actual_frame] = [0]
            else:
                new_coords[actual_frame] = bbox_mean(frames)
        else:
            new_coords[actual_frame] = frames_pred[actual_frame]

    return new_coords
```