UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Aplicación de técnicas de Deep Learning para la clasificación de variedades de polen presentes en imágenes de muestras de miel

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Pérez Martínez, José Luis

Tutor/a: Atienza Vanacloig, Vicente Luis

CURSO ACADÉMICO: 2021/2022

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación de técnicas de Deep Learning para la clasificación de variedades de polen presentes en imágenes de muestras de miel

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: José Luis Pérez Martínez

Tutor: Vicente Luis Atienza Vanacloig

2021-2022

Resumen

El presente trabajo busca implementar un clasificador de mieles basado en el recuento de las partículas de polen presentes en la muestra. Para ello, se empleará técnicas de aprendizaje automático fundamentadas en redes neuronales profundas. La red neuronal se entrenará con imágenes de muestras de mieles a microscopio con las partículas de polen etiquetadas por expertos. Se van a desarrollar las herramientas basadas en dos proyectos de reconocimiento de imágenes de código abierto ya existente: YOLOv5 y Mask R-CNN. Finalmente, se comparará el desempeño de ambos.

Palabras clave: YOLOv5, Mask R-CNN, red neuronal profunda, visión por computador, polen, miel

Abstract

The project aims to implement a honey classifier based on the count of the pollen particles present. To do this, machine learning techniques based on deep neural networks will be used. The neural network will be trained with microscope images of honey samples with pollen particles labeled by experts. The tools will be developed based on two existing open source image recognition projects: YOLOv5 and Mask R-CNN. Ultimately, the performance of both will be compared.

Keywords : YOLOv5, Mask R-CNN, deep neural network, computer vision, pollen, honey

Tabla de contenidos

Índice de tablas

Índice de ilustraciones

1. Introducción

1.1 Motivación

La visión por computador es una herramienta que se está incorporando cada vez más en la vida cotidiana y en la industria. Desde alarmas inteligentes a sistemas de control de calidad, pasando por vehículos autónomos, hay infinidad de campos de aplicación emergiendo, con las correspondientes oportunidades laborales. Adicionalmente, en el ámbito del procesamiento de imágenes, gran parte de los conocimientos que se pueden adquirir y desarrollar para un proyecto se puede transferir a otros problemas, de ámbitos distintos, pero técnicamente similares. Ésta es la principal razón que me ha motivado a trabajar en este proyecto.

Dentro del ámbito de la visión por computador, resultan de especial interés las técnicas basadas en redes neuronales profundas y segmentación. Éstas han demostrado su buen desempeño en las principales competiciones de visión artificial como ImageNet[1], siendo la técnica más exitosa desde 2012. También en la industria están ganando notoriedad estas técnicas, especialmente en los ámbitos mencionados anteriormente: seguridad, calidad y sistemas autónomos.

A nivel personal, he trabajado con anterioridad en el desarrollo de hardware y software de bajo nivel orientado a vehículos terrestres o aéreos con sistemas de conducción asistida o autónoma. Creo que la conducción autónoma va a continuar ganando relevancia en los próximos años y ésta va a estar muy ligada a la visión por computador. A pesar de que este proyecto no esté enfocado directamente en la temática de conducción, gran parte de los conocimientos de este proyecto se pueden transferir a ese ámbito.

En general, los contenidos abordados durante el Grado no profundizan mucho en la visión por computador y trabajar en un proyecto de este tipo para el Trabajo de Fin de Grado me puede servir para ahondar en el tema.

1.2 Objetivos

El objetivo final de este proyecto es crear un sistema basado en visión por computador capaz de clasificar mieles monoflorales mediante el análisis del contenido polínico en imágenes de microscopio de muestras de las mieles en cuestión convenientemente preparadas. Se trata de automatizar una técnica de clasificación tradicional, basada en el hecho de que las abejas se impregnan con el polen de las flores de las especies que visitan durante su recolección de néctar para la elaboración de la miel. La clasificación de la miel

[1] www.image-net.org

se basará en la determinación del porcentaje de presencia del polen de la especie botánica dominante.

Adicionalmente, se pretende que el software desarrollado sirva como herramienta (o, en su defecto, como base para desarrollar una herramienta) para futuros trabajos en el ámbito. Para ello, se desarrollará también un entorno que permita desplegar la herramienta de forma sencilla y estable, tanto para analizar nuevas muestras como para reentrenar la red neuronal o validar los resultados. Así mismo, se preparará la documentación del código y un manual de usuario de acuerdo con los estándares y prácticas habituales para programas de código abierto (*open source*).

Existen varios proyectos de código abierto orientados a la detección de objetos. En este proyecto se va a trabajar con dos de ellos, con aproximaciones significativamente diferentes: YOLO y Mask-RCNN. También es el objetivo de este proyecto poder decidir qué aproximación es más adecuada para este problema específico.

1.3 Impacto Esperado

El proceso de clasificación de mieles por residuo polínico (i.e.: por las moléculas de polen presentes en la miel), ya sea en investigación o en procesos de control de calidad a nivel industrial, son realizados por expertos que tienen que evaluar las muestras tomadas una a una. Junto al hecho de la escasez de expertos suficientemente cualificados y certificados, está el problema de la laboriosidad del procedimiento, que requiere la localización y reconocimiento de al menos 500 ó 600 granos de polen en el sedimento seco de miel.

El objetivo de este proyecto sería facilitar (si no automatizar) este proceso mediante una herramienta que permita realizar una clasificación preliminar de las muestras.

Los potenciales usuarios finales de la herramienta desarrollada en este proyecto serían los propios investigadores del grupo de Visión por Computador del Departamento de Informática de Sistemas y Computadores (DISCA) de la UPV y, en caso de resultar de interés, los investigadores y expertos del Departamento de Tecnología de Alimentos (DTA) que trabajan en la materia y con los que el grupo de visión ha estado colaborando, en el marco de un proyecto denominado *Polenet*.

1.4 Metodología

La metodología empleada, aun sin profundizar en la implementación de metodologías ágiles (AGILE[2]) por no tratarse de un proyecto colaborativo, sí que intenta aprovechar los conceptos clave de estas metodologías como son los ciclos de desarrollo cortos e

[2] https://en.wikipedia.org/wiki/Agile_software_development

incrementales, evaluación continua en cada ciclo y reportes periódicos y frecuentes del estado del proyecto, en este caso al tutor.

Como pasos previos al desarrollo del trabajo, se ha realizado una búsqueda de información sobre el ámbito específico (técnicas de clasificación de mieles, trabajos previos en el reconocimiento de pólenes, estrategias y criterios empleados, etc.) y general (procesamiento de imágenes, metodologías de clasificación, técnicas de aprendizaje, uso de redes neuronales, etc.), así como un sondeo de las herramientas y tecnologías disponibles para afrontar este tipo de problemas. Con la información disponible y los conocimientos iniciales, se eligieron las dos herramientas más prometedoras: YOLO y Mask-RCNN.

A partir de este punto se inicia un proceso iterativo, en el que en cada iteración se alterna entre las dos herramientas escogidas. Partiendo de una versión simplificada del problema, en la que solo se trabaja con un pequeño subconjunto del *dataset* de muestras de mieles, y con pocos pólenes a detectar, se va incrementando la complejidad del problema tras cada iteración hasta llegar a la versión final del problema, en la que entrenando el sistema con todas las imágenes disponibles en el *dataset* de muestras se intenta clasificar la miel empleando todas las clases de pólenes registradas.

Cada ciclo del proyecto se compone de una fase de entrenamiento de la red bajo los nuevos términos, una validación de la red entrenada y una evaluación de los resultados obtenidos. Dada la complejidad que tiene el refinamiento de la red, inherente a la gran cantidad de parámetros que se pueden ajustar, es difícil establecer de antemano qué parámetros se van a modificar en cada iteración, de modo que se decidirán en base a la evaluación de los resultados de la iteración previa.

Se irá manteniendo la simetría en la evolución de las dos herramientas tanto como sea posible, de modo que se pueda ir comparando el progreso de los resultados de cada una. Si se determina que una de las herramientas es consistentemente peor que la otra, será posible descartarla y centrar el desarrollo en la otra en fases más tempranas.

Al terminar los ciclos de desarrollo se hará una evaluación final más extensa, empleando muestras nunca utilizadas en el entrenamiento o la evaluación previamente y se compararán los resultados con la clasificación realizada por el experto.

Por último, se documentará todo el software desarrollado y se creará un manual de usuario que facilite el uso de la herramienta.

1.5 Convenciones

- Las palabras extranjeras se escribirán en *cursiva*.
- Las fórmulas se escribirán en *Cambria Math Cursiva*.
- El código fuente se escribirá en `Courier New`.

2. Estado del arte

El contexto tecnológico de este proyecto presenta dos variantes: el estado del arte en cuanto a técnicas de clasificación de mieles y el estado del arte en cuanto a las técnicas de detección y clasificación de objetos basadas en visión por computador.

2.1 Clasificación de mieles

A la hora de clasificar la miel, existen varios criterios: por su origen botánico, por su presentación -estado en el que se encuentra la miel-, por su forma de producción -tradicional o ecológica- o por su destino -consumo o industria-. De entre ellos, el de principal interés para el consumidor, así como el que plantea el mayor reto a la hora de identificar, es el primero. El origen botánico consiste en clasificar las mieles en monoflorales, poliflorales, multiflorales o mielatos, en función de las flores o plantas de las que las abejas han extraído el néctar. Según las plantas que dan origen a la miel, y el porcentaje que representan de ésta, se determina la calidad (y, en última instancia, el precio) de la miel resultante. Es, por tanto, de gran interés tanto para el productor como para el consumidor poder establecer cuál ha sido el origen de la miel.

La clasificación de mieles por su origen se basa en la melisopalinología, una rama de la palinología que consiste en identificar los granos de polen presentes en las mieles para determinar su procedencia. Para determinar el origen botánico se combinan métodos fisicoquímicos -por ejemplo, la quimiometría de parámetros como la conductividad eléctrica, el Hidroximetilfurfural (HMF), la acidez (pH) y el color-, palinológicos y análisis sensorial (Serrano, Villarejo, Espejo, & Jodral, 2004).

En el campo palinológico, la principal técnica para determinar los pólenes que componen la muestra de miel consisten en realizar una cromatografía. Alternativamente, se realiza un análisis a microscopio de la muestra y un experto hace la clasificación y conteo de los pólenes que identifica. La cromatografía es una técnica compleja, que requiere equipo y unas instalaciones especializadas. Por su parte, el análisis a microscopio no requiere una gran inversión en equipo, pero es un proceso laborioso por parte del experto que limita en gran medida el escalado.

No obstante, si es posible emplear técnicas de visión por computador para automatizar el proceso de conteo y clasificación de las muestras, esta opción puede ganar mayor interés. Hay trabajos previos que abordan la clasificación automática de pólenes (Kaya, Pinar, Erez, & Fidan, 2013) (Holt & Bennett, 2014) e incluso se han propuesto competiciones de *machine learning* centradas en esta temática[3].

[3] https://iplab.dmi.unict.it/pollenclassificationchallenge

2.2 Clasificación de objetos mediante Visión por computador

Desde los inicios de la computación, se ha intentado sacar provecho a su capacidad de cálculo para tareas de clasificación. En la década de 1950, las primeras aproximaciones de lo que ahora se llama *machine learning* estaban basados en modelos probabilísticos, como el Clasificador Bayesiano Ingenuo o la Regresión Logística.

Durante esa época empezaron a surgir las primeras redes neuronales, aunque por aquel entonces se encontraban muy limitadas por la dificultad de entrenarlas de forma eficiente, limitando en gran medida el tamaño de las redes. Esto fue así hasta que en los 80 se empezó a utilizar el algoritmo de propagación hacia atrás (*Backpropagation*) como un método eficiente de actualizar los pesos de la red.

Sin embargo, otros métodos eclipsaron a las redes neuronales en los años sucesivos. En los 90, los métodos de kernels como la Máquina de Soporte Vectorial (*Support Vector Machine*, SVM) (Cortes & Vapnik, 1995) fueron las técnicas más utilizadas en la clasificación por ordenador. Entre los años 2000 y 2010, los métodos de kernels fueron reemplazados por árboles de decisión. En especial, el algoritmo de Bosques Aleatorios (*Random Forest*) (Breiman, 2001) es suficientemente robusto y general como para aplicarlo a una gran variedad de problemas. Adicionalmente, en 2014 se empezó a utilizar técnicas de potenciación del gradiente (*gradient boosting*) con las que es posible incrementar aún más el rendimiento de cualquier modelo de *machine learning* (Alexey & Alois, 2013). En la actualidad, la combinación de árboles de decisión con potenciación de gradiente es una de las mejores técnicas, si no la mejor, para tratar problemas que no sean de percepción.

El auge de las redes neuronales volvió a principios de la década de 2010. Los modelos de redes neuronales profundas empezaron a poder entrenarse de forma eficiente en tarjetas gráficas (*Graphic Processor Unit*, GPU) y pronto superaron en rendimiento a las demás técnicas de *machine learning* (Krizhevsky, Sutskever, & Hinton, 2012). Desde 2012, las redes neuronales convolucionales profundas (*convnets*) son la técnica principal para tareas de visión por computador.

Las técnicas de aprendizaje profundo están basadas en la acumulación de varias capas de representación. Los datos de entrada pasan por varias capas, cada una de las cuales aplica transformaciones relativamente sencillas. Las transformaciones pueden ir desde la rotación o el cambio de paleta de la imagen a la localización de esquinas o de contornos (para los casos de imágenes), o cualquier otra transformación más abstracta. Cada una de estas capas mejora, aunque sea ligeramente, la representación de los datos, de forma que con la acumulación de múltiples capas es posible, en última instancia, clasificar los datos de entrada. En la Figura 1 se puede ver cómo la imagen original pasa por varias capas, cada una de las cuales crea un conjunto de nuevas representaciones y las pasan a la siguiente, para finalmente clasificar la imagen como un dígito en la última capa (capa de salida).

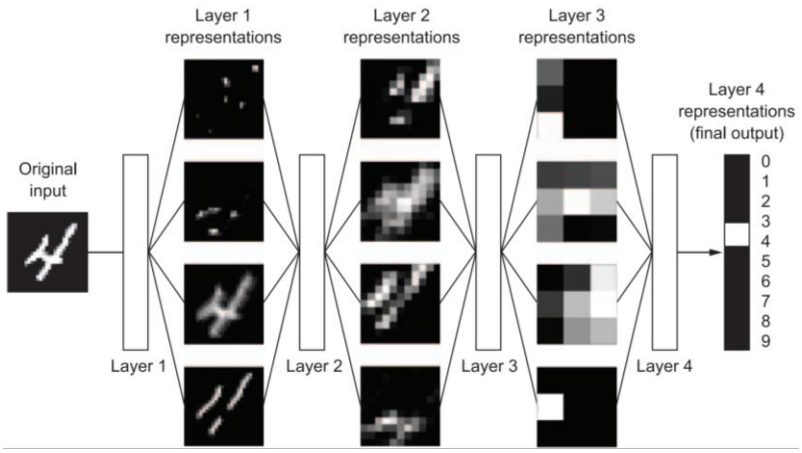

Figura 1: Representaciones profundas aprendidas por un modelo de clasificación de dígitos (Chollet, 2018)

La "profundidad" viene dada por el número de capas que tiene el modelo, que en la actualidad puede ir desde decenas hasta centenas. En las redes neuronales profundas, cada capa está caracterizada por un conjunto de valores numéricos denominados "pesos" que definen las transformaciones que realizan. El proceso de "entrenamiento" de la red neuronal profunda (Figura 2) consiste en:

1) Tomar una entrada X.
2) Utilizar la red neuronal, pre-entrenada o con pesos (*weights*) aleatorios, para obtener una predicción Y'.
3) Comparar la predicción Y' con el valor real Y mediante una función de pérdida (*loss function*) para obtener una puntuación de la calidad de la predicción (*loss score*).
4) Retroalimentar la red neuronal en base a los resultados para actualizar los pesos de cada capa. Este último paso se realiza mediante un optimizador basado en propagación hacia atrás y descenso del gradiente.
5) Repetir el proceso hasta que termine el entrenamiento. Los criterios más habituales para dar por finalizado el entrenamiento son por alcanzar una puntuación de pérdida suficientemente pequeña o por superar el límite de iteraciones de entrenamiento impuesto. Los valores de ambos casos dependen de las características del caso específico.

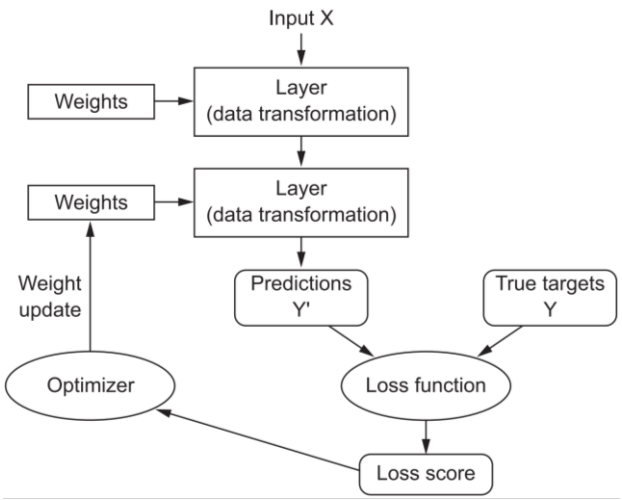

Figura 2: Diagrama de flujo de una red neuronal profunda (Chollet, 2018)

En la actualidad, Python es el lenguaje de programación más popular a la hora de implementar redes neuronales. Esto se debe, por un lado, a lo sencillo que resulta trabajar con este lenguaje, y por otro, a que ofrece potentes librerías de forma gratuita para trabajar con redes neuronales. Los ejemplos más notorios son TensorFlow[4] y PyTorch[5].

Sin embargo, en este trabajo no se va a construir una red neuronal desde cero, sino que se va a partir desde proyectos de código abierto de clasificación de imágenes. En concreto, se van a utilizar dos proyectos que tienen aproximaciones distintas:

- Mask R-CNN (He, Gkioxari, Dollár, & Girshick, 2017): Está basado en la idea original de R-CNN (*Regions with Convolutional Neural Networks features* o Regiones con características de Redes Neuronales Convolucionales) (Girshick, Donahue, Darrell, & Malik, 2014), que propone dividir la imagen en regiones de interés (*region of interest, ROI*) y aplicar la red neuronal sobre cada una de ellas. Esta idea se mantiene en versiones posteriores -Fast R-CNN (Girshick, Fast R-CNN, 2015) y Faster R-CNN (Ren, He, Girshick, & Sun, 2015)-, en las que se mejora la velocidad de entrenamiento y detección de la red. Mask R-CNN añade una capa adicional: una máscara que define con precisión el contorno del objeto.

Figura 3: Ejemplo de detección con Mask R-CNN

- YOLOv5 (Jocher, y otros, 2022): Está basado en el método de "solo miras una vez" (*You Only Look Once*, YOLO) (Redmon, Divvala, Girshick, & Farhadi, 2015). El algoritmo trabaja de forma similar a R-CNN, dividiendo la imagen en ROI, pero el algoritmo de YOLO solo visita cada región una vez, mejorando en gran medida la velocidad del proceso. El algoritmo fue mejorado por el autor[6] hasta la versión YOLOv3 (Redmon & Farhadi, YOLOv3: An Incremental Improvement, 2018) y más adelante por el equipo de Ultralytics[7] hasta la versión YOLOv5.

[4] https://www.tensorflow.org
[5] https://pytorch.org
[6] https://pjreddie.com/darknet/yolo
[7] https://github.com/ultralytics

Figura 4: Ejemplo de detección con YOLOv5

2.3 Crítica al estado del arte

A pesar de que hace un tiempo que ya se están empleando técnicas de visión por computador para la clasificación de pólenes (Daood, Ribeiro, & Bush, 2016) (d. Geus, Barcelos, Batista, & d. Silva, 2019), la mayoría se omiten el proceso de segmentación de las muestras. Normalmente la segmentación (Figura 6) se realiza manualmente, con alguna herramienta informática que ayude a extraer los granos de polen (Figura 7). El proceso de visión por computador se centra en la clasificación de las muestras de polen ya extraídas de la imagen completa (Figura 8).

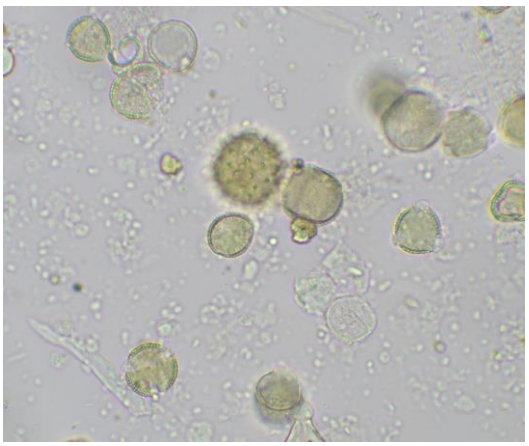

Figura 5: Muestra de miel a microscopio

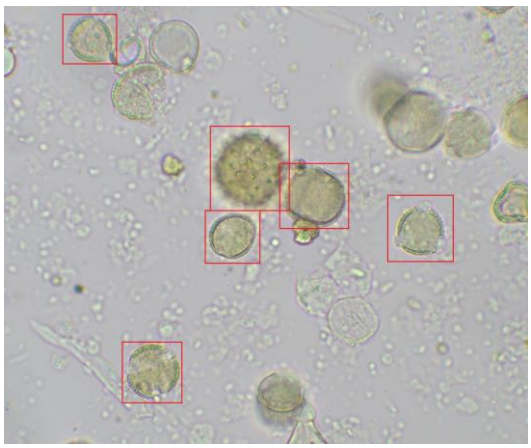

Figura 6: Selección de granos de polen

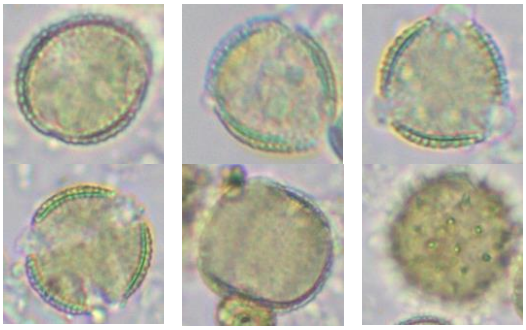

Figura 7: Extracción de los granos de polen

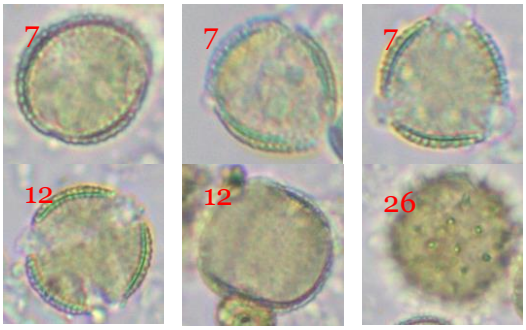

Figura 8: Clasificación de los granos de polen

La detección de objetos en imágenes complejas ha estado evolucionando mucho en los últimos tiempos, especialmente motivado por los sistemas de conducción asistida de vehículos y las cámaras de videovigilancia inteligentes. La intención de este proyecto es extender las técnicas y tecnologías ya existentes empleadas en otros contextos al ámbito de la detección y clasificación de pólenes, con el objetivo de mejorar la automatización de este proceso.

Sí que hay algunos trabajos previos que abordan el problema desde la fase de segmentación. Es especialmente interesante el trabajo del equipo de la Universidad de Extremadura (Gallardo-Caballero, y otros, 2019). Sin embargo, en este trabajo emplean vídeo que captura las imágenes a microscopio a distintas alturas, permitiendo trabajar con el volumen. En el caso de este trabajo sólo se dispone de imágenes planas, en las que no todas las partículas de polen están enfocadas.

Adicionalmente, en lugar de desarrollar un software específico y una red neuronal propia desde cero, se van a emplear los recursos disponibles de código abierto.

2.4 Propuesta

El problema que se intenta abordar en este proyecto es no solo clasificar instancias de polen, como en los casos citados previamente, sino ser capaz de detectarlas en la imagen completa de una muestra. En otras palabras, se trata de un problema de detección, segmentación y clasificación.

Dado que existen varias aproximaciones para afrontar este problema, se va a realizar un trabajo de exploración, probando las dos técnicas de código abierto más conocidas: Mask R-CNN y YOLOv5.

Para trabajar con Mask R-CNN, se va a partir del proyecto de código abierto de Matterport[8], que es una implementación para Keras y TensorFlow. En el caso de YOLOv5, se usará como base el repositorio ofrecido por los propios desarrolladores de Ultralytics[9], una implementación basada en PyTorch.

[8] https://github.com/matterport/Mask_RCNN
[9] https://github.com/ultralytics/yolov5

3. Análisis del problema

3.1 Métricas y evaluación

En el ámbito de la visión por computador, la evaluación de resultados se realiza exponiendo la red neuronal a un conjunto de imágenes que no hayan sido utilizadas en el entrenamiento. Luego, se puntúa la eficacia de la red comparando los objetos detectados en cada imagen con los que realmente se encontraban en ella (*Ground Truth*, GT).

Durante la fase de entrenamiento de la red, la principal métrica para evaluar el proceso de aprendizaje es la **función de pérdida** (*loss*), que puntúa el error que se estima en la red respecto al GT del conjunto de entrenamiento. El cálculo exacto de la función de pérdida depende de la implementación, pero suele estar basado en el error cuadrático medio.

A la hora de estimar la función de pérdida, se tienen en cuenta el error de diversos factores como el error de las cajas (*bounding box*), el error en la detección de los objetos o el error en la clasificación de los objetos. En cualquier caso, el objetivo del entrenamiento es siempre minimizar esta función de pérdida y monitorizar la evolución de ésta a lo largo del entrenamiento permite determinar si la red se está entrenando adecuadamente o no.

Adicionalmente, se evalúa esta misma función de pérdida en un conjunto de datos de validación, distinto al de entrenamiento, para detectar comportamientos anómalos como el *overfitting* y el *underfitting*.

El uso de estas métricas está ya integrado en los procesos de entrenamiento tanto de Mask R-CNN como de YOLOv5, aunque su implementación varía ligeramente entre un modelo y otro:

- **YOLOv5**: Evalúa por separado las funciones de pérdida de los *bounding box*, de la detección de objetos y de la clasificación de los objetos detectados.
- **Mask R-CNN**: Calcula una función de pérdida única empleando una media ponderada de las distintas funciones de pérdida. Cabe destacar que Mask R-CNN introduce una función de pérdida adicional dedicada al error en las máscaras.

Tras el proceso de entrenamiento, se valida la red con un conjunto de imágenes distinto al de entrenamiento. Durante la validación, en un caso simple de clasificación binaria (es decir, cuando hay un único tipo de objeto y se trata de determinar si el objeto a clasificar es o no es de ese tipo) se define el término *positivo* cuando el objeto ha sido clasificado como del tipo considerado y *negativo* cuando se ha clasificado como no perteneciente a esa clase de objeto. De esta forma, en cada clasificación se pueden dar cuatro situaciones:

- **Verdadero positivo** (*True Positive*, TP): El clasificador ha determinado un positivo, y ello coincide con el conocimiento del GT.
- **Verdadero negativo** (*True Negative*, TN): El clasificador ha determinado un negativo, y esto coincide con el conocimiento del GT.

- **Falso positivo** (*False Positive*, FP): El clasificador ha determinado un positivo, pero según el GT debería ser negativo.
- **Falso negativo** (*False Negative*, FN): El clasificador evalúa un negativo, pero según el GT debería ser positivo.

Los dos primeros casos son casos de predicciones correctas, mientras que los dos últimos son predicciones erróneas. En el caso de clasificación en múltiples clases, que es el que corresponde a nuestro problema de clasificación de pólenes, y en particular en problemas de *clasificación de instancias*, como los que afrontan YOLO y Mask-RCNN, como ya se ha dicho, no sólo debe resolverse un problema de clasificación multiclase, sino que, al mismo tiempo, debe hacer una detección de objetos en la imagen (determinar la localización de los objetos que han de ser clasificados). En este escenario, las cuatro situaciones anteriores son habitualmente reinterpretadas de la forma que se recoge en la Tabla 1:

Caso	**Descripción**
TP	El objeto detectado en la imagen coincide con un objeto de la misma clase en la misma ubicación de la imagen en el GT.
TN	El fondo de la imagen se detecta como tal. En general, no se detecta explícitamente el fondo como "fondo", sino que se considera de esta manera todo lo que no se ha detectado explícitamente como un objeto. Al no realizar detecciones explícitas de dicho fondo, esta métrica no contabiliza.
FP	La IA ha detectado un objeto, pero cuya clase no coincide con la de ningún objeto en esa ubicación de la imagen en el GT.
FN	La IA no ha detectado un objeto que se encontraba en el GT.

Tabla 1: Descripción de métricas para detección de objetos en imágenes

A partir de estos valores es posible extraer otras métricas derivadas de interés estadístico que ayudan a entender y evaluar el desempeño de la red neuronal. Ejemplos de estas métricas derivadas son:

- **Precisión** (*precision*): Ratio de objetos clasificados correctamente sobre todos los detectados

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

- **Recubrimiento** (*recall*): Ratio de objetos clasificados correctamente sobre todos los presentes en el GT

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

Para ambos casos es posible trabajar a nivel global, sin hacer distinciones entre clases (*micro average*) o a nivel de clase y calcular los valores de precisión y recubrimiento como los promedios de las clases (*macro average*). En este trabajo, salvo que se indique lo contrario, se trabajará con *micro average*.

Otra métrica que se empleará para evaluar la red neuronal es la **matriz de confusión**. La matriz de confusión, o matriz de error, es una representación visual de las detecciones y clasificaciones realizadas por la red. Es una matriz cuadrada de $C \times C$, siendo C el número de clases a clasificar, aunque en el caso de detecciones en imágenes complejas,

como se da aquí, se suele añadir una clase extra para representar el fondo. Las filas representan las clases detectadas por la red, mientras que las columnas hacen referencia a las clases en el GT. Para cada celda (i, j), se indica los objetos de la clase c_i que han sido clasificado como clase c_j.

Por ejemplo, en la Figura 9, la celda ("*Brasicáceas*", "*Cistrus sp.*") tiene un valor de 0.04. Esto quiere decir que el 4% de los objetos etiquetados como "Brasicáceas" en el GT han sido clasificados (erróneamente) como "Citrus sp." por la red neuronal.

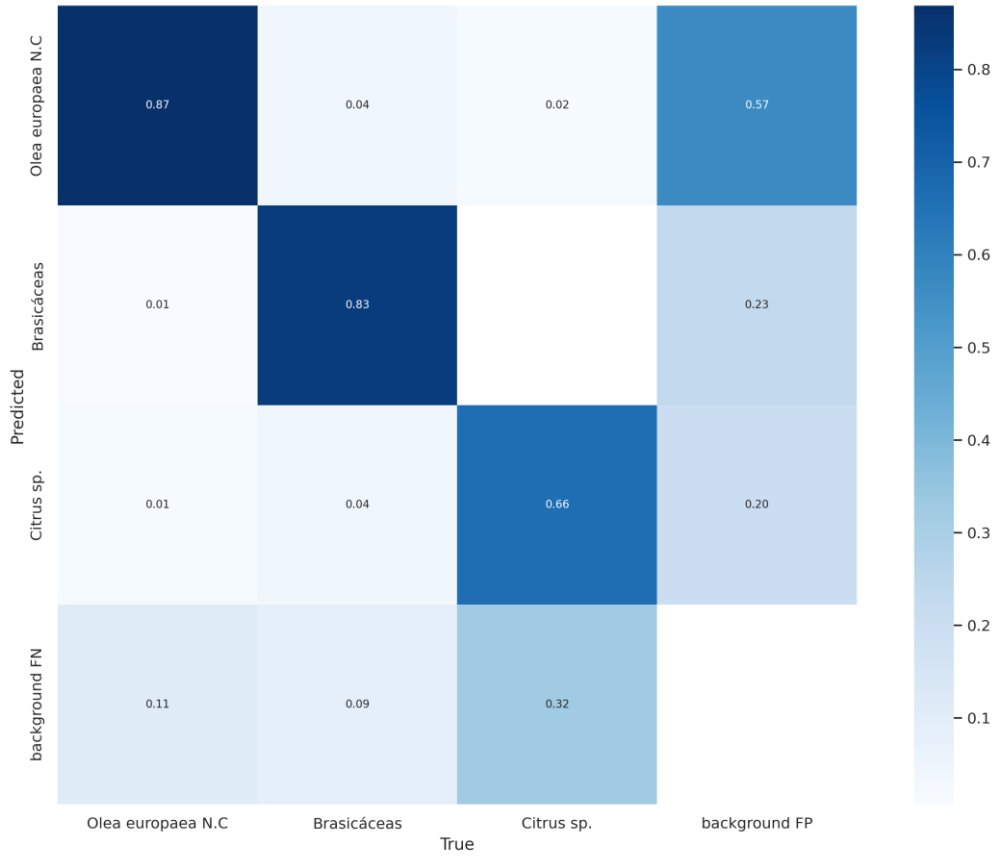

Figura 9: Matriz de confusión

Es, por lo tanto, una forma alternativa de expresar los TP, TN, FP y FN. También se puede representar también la precisión y el recubrimiento añadiendo los valores marginales de la matriz. De hecho, así es como se representará la matriz de confusión en este trabajo (Figura 10), mostrando así toda la información relevante de forma condensada y visual.

Figura 10: Matriz de confusión con valores absolutos y marginales

Cabe destacar que todas estás métricas están sujetas a una dimensión adicional: la confianza de la detección. La red neuronal determina un nivel de confianza para cada objeto detectado que representa lo "segura" que está de que el objeto detectado se haya detectado correctamente. Establecer el valor umbral a partir del cual se da por válida una detección depende del problema concreto: incrementar el valor umbral disminuye los FP, pero aumenta los FN.

Es habitual utilizar la **precisión media promedio** (*mean Average Precission*, mAP) para situaciones en las que hay múltiples objetos en cada imagen.

3.2 Especificación de requisitos

Para considerar exitoso este proyecto, deberá cumplir una serie de requisitos. Dichos requisitos se pueden clasificar en tres tipos: de desarrollo, de uso y de resultados.

Requisitos de desarrollo

Los requisitos de desarrollo hacen referencia a las herramientas, programas, lenguajes, recursos y técnicas que se van a usar para desarrollar el proyecto. La mayoría de estos

requisitos tienen como objetivo acotar la complejidad del proyecto, que de otra forma puede superar largamente el alcance de un TFG. Otros vienen dados por limitaciones en la disponibilidad de licencias u otros recursos. Los requisitos que cumplir son los siguientes:

- Lenguaje de programación Python.
- Uso de herramientas de reconocimiento de imagen existentes y de código abierto.
- Trabajar con los *datasets* de Polenet. Esto es, ser capaz de trabajar con los XML que contienen la información de las muestras y las etiquetas, así como manejarse en la organización de directorios de Polenet.
- Ejecutarse en Linux.
- Fácil de instalar, desplegar y ejecutar.
- Completamente documentado de acuerdo con los estándares de Python (Docstring) para facilitar el trabajo futuro.
- Incluir un manual de uso de la herramienta.

Requisitos de uso

Estos requisitos describen cómo se debe usar la herramienta resultante, así como las características que debe soportar. No será necesario que la herramienta final esté terminada a nivel comercial, pero si deberá cumplir una serie de funcionalidades:

- La herramienta tiene que permitir: 1) entrenar o reentrenar la red neuronal; 2) validar el desempeño de la red; 3) realizar detecciones sobre imágenes nuevas.
- En el entrenamiento y la validación de la red, las imágenes y el GT debe proveerse siguiendo el formato de Polenet.
- Para la detección, puede proveerse una imagen o un conjunto de imágenes.
- El entrenamiento de la red debe proveer como resultados un fichero de pesos para la red y un conjunto de métricas que permitan evaluar el desempeño de la red y del proceso de entrenamiento. Como mínimo, la evolución de la función de pérdida y la evolución del mAP.
- La evaluación de la red deberá generar, como mínimo, una matriz de confusión para analizar el desempeño.
- La detección de partículas de polen en imágenes nuevas deberá generar una versión de las imágenes con el *bounding box* de los objetos detectados y su clase asignada, así como una lista con las estadísticas de todas las detecciones.
- Las imágenes pueden estar en formato JPG o PNG.

Requisitos de resultados

Los requisitos de los resultados determinan los criterios de evaluación y aceptación de la herramienta.

- La función de pérdida de las redes entrenadas debe ser monótonamente decreciente (sin *overfitting*) y estabilizarse en torno a un valor al final del entrenamiento (sin *underfitting*).

- El mAP tanto al final del entrenamiento como en la evaluación deberá ser mayor a 0.5 para el mAP@50. En otros trabajos similares (Gallardo-Caballero, y otros, 2019) se ha logrado mAP@50 superiores a 0.9. Aunque este proyecto cuenta con limitaciones adicionales y es solo un estudio preliminar, valores de mAP@50 se considerarían como un desempeño deficiente de la red, o incluso de la herramienta en general.

3.3 Modelado conceptual

El diseño conceptual de la aplicación puede ser representado por el diagrama de la Figura 11. El diagrama incluye los siguientes elementos:

- **Aplicación** (**azul**): Conjunto de elementos que conforman la aplicación para Polenet.
- **Dataset de Polenet** (**gris**): Se dispone de una base de datos de imágenes de mieles vistas a microscopio en formato JPG, así como el etiquetado de las imágenes y otros metadatos de la muestra recogidos en un fichero XML.
- **Subconjuntos del *dataset*** (**amarillo**): La aplicación trabaja con varios conjuntos de imágenes, según el caso de uso. Cada conjunto debe ser disjunto para el correcto desempeño de la herramienta, como es habitual en cualquier problema de aprendizaje automático. Los conjuntos en última instancia son un listado de las imágenes a emplear para cada caso de uso, a saber: entrenamiento, evaluación o detección. Los conjuntos de entrenamiento y evaluación deben ser subconjuntos el *dataset* (i.e.: estar etiquetados). El conjunto de detección puede pertenecer o no al *dataset*.
- **Parámetros** (**rojo**): La aplicación requerirá un conjunto de parámetros e hiperparámetros que permitan configurar el funcionamiento de la aplicación. Como mínimo será necesario uno para determinar el caso de uso. Cada caso de uso puede requerir unos parámetros específicos.
- **Pesos** (**morado**): Los pesos específicos de cada capa de la red neuronal profunda es lo que determina su comportamiento. El fichero con los pesos será uno de los *outputs* del entrenamiento y será un fichero de entrada necesario tanto en la evaluación como en la detección.
- **Resultados** (**verde**): Cada caso de uso generará un conjunto de resultados. Estos resultados se pueden englobar en tres grupos:
    - **Estadísticas**: Se generará un registro de datos estadísticos que representan los resultados de cada caso de uso y permiten evaluarlos. Por ejemplo: en el entrenamiento se recoge el resultado de la función de pérdida tras cada ciclo de entrenamiento; en la evaluación se recoge el mPA; en la validación se recoge las clases detectadas en las imágenes y su probabilidad.
    - **Gráficos**: Para los casos de uso de entrenamiento y validación se generarán adicionalmente representaciones gráficas de los valores estadísticos. Un ejemplo es la generación de la matriz de confusión.

- **Imágenes clasificadas**: En el caso de la detección, se generará una copia de las imágenes que incluya los *bounding box* de los objetos detectados.

El diagrama también incluye el flujo de datos en función del caso de uso: entrenamiento (**negro**), evaluación (**verde**) y detección (**rojo**).

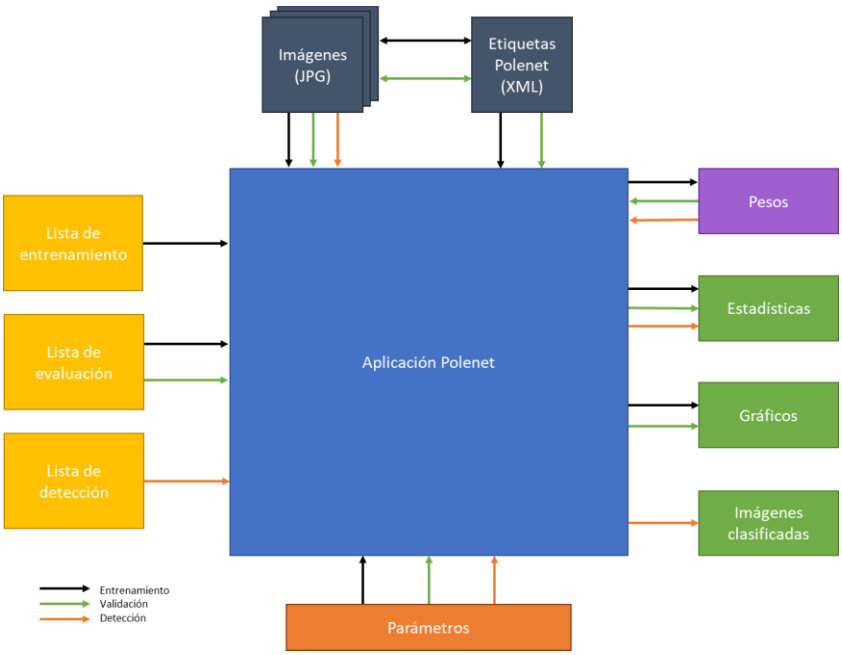

Figura 11: Diagrama conceptual de la aplicación de Polenet

4. Diseño de la solución

4.1 Arquitectura del sistema y tecnología utilizada

El desarrollo de la aplicación para Polenet, tanto en el caso basado en YOLOv5 como en el de Mask R-CNN, se realizará con una arquitectura basada en Docker[10]. Docker es un sistema de emulación de sistemas operativos, conceptualmente similar a una máquina virtual (VM), pero que en lugar de cargar un sistema operativo completo y trabajar sobre un hipervisor (*Hypervisor*) que hace de intermediario con el sistema operativo huésped, utiliza el kernel del sistema operativo (Linux) para reconstruir sobre él las características del SO emulado (Figura 12). Escapa del objetivo de este trabajo describir cómo funciona Docker, pero se puede encontrar más información sobre las características, modo de funcionamiento, detalles técnicos y otros temas de interés en su página de documentación propia[11].

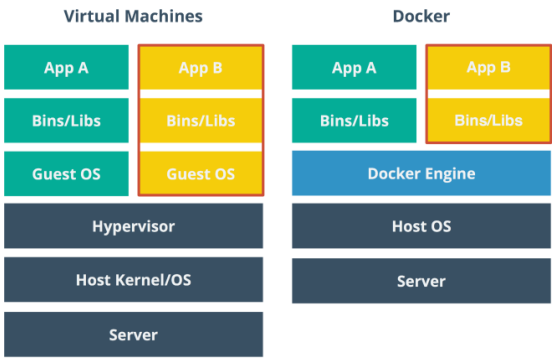

Figura 12: Máquinas Virtuales vs contenedores Docker

El interés de la arquitectura Docker es debido a las ventajas que ofrece: un despliegue sencillo, con un entorno controlado, fácil de mantener y potencialmente fácil de portar entre distintas máquinas e incluso distintos sistemas operativos, siempre y cuando soporten Docker.

El diseño de la arquitectura es el descrito por la Figura 13, con los siguientes elementos:

- **Contenedor Polenet**: Contenedor Docker que contiene todas las herramientas, librerías y aplicaciones para el correcto funcionamiento de la aplicación Polenet. El contenido concreto de este contenedor se discutirá más adelante.
- **Contenedor Nvidia**[12]: Contenedor desarrollado por Nvidia que incluye los drivers necesarios para trabajar con sus tarjetas gráficas desde los contenedores Docker. Para trabajar con tarjetas gráficas de otros fabricantes sería necesario utilizar un contenedor equivalente para los drivers de la gráfica que se esté empleando. En última instancia es posible trabajar sin utilizar una GPU dedicada, en detrimento del rendimiento de los procesos.

[10] https://www.docker.com
[11] https://docs.docker.com
[12] https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/overview.html

- **Docker Engine**[13]: Motor de Docker que gestiona el kernel y las imágenes para desplegar los contenedores Docker.
- **Sistema Operativo**: El desarrollo del trabajo se ha basado en el sistema operativo Ubuntu[14]. Cualquier sistema operativo basado en Linux soporta por defecto Docker, que requiere un kernel Linux. Sin embargo, otros sistemas operativos como Windows o Mac también dan soporte a Docker en sus versiones más recientes.
- **Dataset Polenet**: El conjunto de imágenes (JPG) y ficheros de etiquetas (XML) que componen el *dataset* de Polenet se alojará en el sistema operativo huésped. El *dataset* completo puede ser muy pesado en términos de memoria, y es inmutable desde el punto de vista de la aplicación, así que no hay motivos para incluirlo dentro del contenedor de Polenet. Al crear el contenedor se creará una pasarela que permita el acceso desde la aplicación al directorio en el que se encuentre el *dataset* en el huésped.

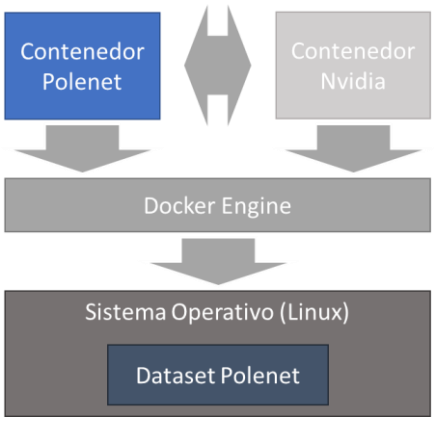

Figura 13: Arquitectura Docker para la aplicación Polenet

Para facilitar el acceso, desarrollo, mantenimiento y supervisión del código de las aplicaciones desarrolladas, se utilizará Gitlab[15] para alojar el código y la documentación desarrollados.

Es habitual el uso de la tecnología GIT para el desarrollo colaborativo de proyectos, especialmente en los casos de código abierto. Las dos herramientas más utilizadas de GIT son Github y Gitlab. Ambos casos son de registro y acceso gratuito (fuera del ámbito empresarial) y ofrecen utilidades muy similares. La principal razón por la que se ha decidido usar Gitlab para alojar el proyecto en lugar de Github es porque el primero es, en sí, de código abierto. Esto quiere decir que en un futuro sería posible levantar una instancia privada de Gitlab en un servidor propio y migrar de forma directa a él.

Se utilizará un repositorio GIT para cada una de las implementaciones (YOLOv5 y Mask R-CNN) que incluirá todo lo necesario para poder generar la imagen Docker correspondiente a la implementación, además de una guía de instalación y uso (README).

[13] https://docs.docker.com/engine
[14] https://ubuntu.com
[15] https://gitlab.com

4.2 Diseño detallado

El diseño detallado difiere entre la implementación de YOLOv5 y Mask R-CNN, ya que viene condicionado por las características que ofrecen y las que requieren cada una de las herramientas usadas como base.

YOLOv5

YOLOv5 de Ultralytics ofrece una implementación ya empaquetada en una imagen Docker que se puede descargar directamente de DockerHub[16]. Esta imagen ya incluye todas las librerías y demás dependencias necesarias para el correcto funcionamiento de la herramienta.

Ultralytics también incluye integraciones con otros entornos de interés en el ámbito de computación en la nube, como Google Collab[17], Kaggle[18], Amazon Web Services[19] o Google Cloud Platform[20], pero en este proyecto solo se va a trabajar en local. Estas integraciones pueden ser de interés para futuros trabajos.

El funcionamiento de YOLOv5 se basa en tres scripts de Python, uno para cada caso de uso (entrenamiento, validación y detección), que conforman la API de la herramienta. En paralelo con distintos conjuntos de parámetros que permiten adaptar la herramienta a las necesidades del usuario, YOLOv5 requiere que se definan los siguientes componentes:

- Configuración de la CNN: Un fichero en formato YAML que describe las características de la red neuronal convolucional utilizada, incluyendo su tamaño, en número de clases de salida, el *backbone* y el *head*.
- Información del *dataset*: Un fichero en formato YAML definiendo el *dataset*: número y nombre de las clases, ubicación del *dataset* y los conjuntos de entrenamiento y validación. Estos últimos se pueden definir de varias formas. La forma más conveniente y la utilizada en este trabajo es mediante un fichero de texto que liste todas las imágenes pertenecientes a cada conjunto (train.txt y test.txt).
- Listado de imágenes de entrenamiento y de validación: Los ficheros listando las imágenes de cada conjunto, mencionados en el punto anterior (train.txt y test.txt).
- Imágenes del *dataset*: El conjunto de imágenes del *dataset*. La ubicación de las imágenes debe coincidir con la especificada en los listados.
- Etiquetas del *dataset*. YOLOv5 requiere que las etiquetas de las imágenes del *dataset* estén en un formato muy concreto: cada imagen debe tener su propio fichero de etiquetas, en la misma ubicación y con el mismo nombre que la imagen a la que hacen referencia (cambiando la terminación de la extensión del archivo,

[16] https://hub.docker.com/r/ultralytics/yolov5
[17] https://colab.research.google.com/github/ultralytics/yolov5/blob/master/tutorial.ipynb
[18] https://www.kaggle.com/ultralytics/yolov5
[19] https://github.com/ultralytics/yolov5/wiki/AWS-Quickstart
[20] https://github.com/ultralytics/yolov5/wiki/GCP-Quickstart

de .jpg/.png a .txt). El fichero de etiquetas debe incluir una línea por cada clase presente en la imagen, con el siguiente formato:

```
class x_center y_center width height
```

donde `class` es el número de la clase tal y como están ordenadas en el fichero de información del *dataset* (empezando en 0), `x_center` e `y_center` son las coordenadas del centro del *bounding box* del objeto y `width` y `height` son el ancho y alto del *bounding box*, respectivamente. Tanto las coordenadas como el alto y el ancho están normalizados en el rango [0-1].

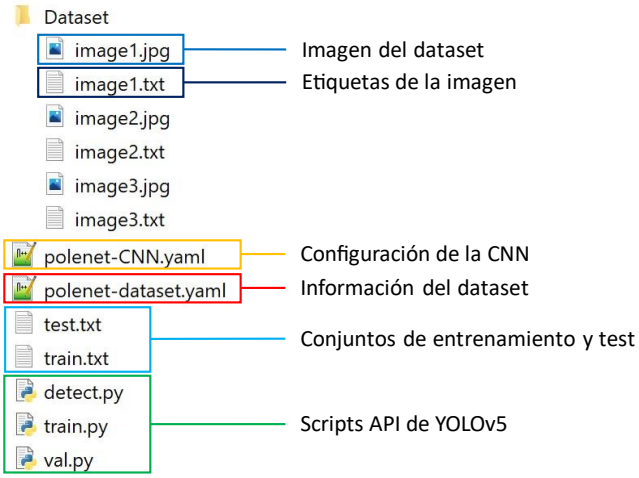

Figura 14: Ejemplo de estructura YOLOv5

En el apartado de resultados, YOLOv5 permite generar tanto los gráficos de las métricas más relevantes (función de pérdida, mAP, precisión, recubrimiento, matriz de confusión, etc.) como generar un fichero CSV con los resultados para un procesamiento posterior. En general, la propia herramienta ofrece facilidades para generar cualquiera de los archivos que se han clasificado como relevantes en el modelado conceptual.

Para poder trabajar con el *dataset* de Polenet en YOLOv5, será necesario añadir a la arquitectura del programa un script que genere los ficheros requeridos (etiquetas de las imágenes, configuración de la CNN, información del *dataset* y los conjuntos de entrenamiento y de test) a partir de la información contenida en los XML de Polenet.

También se desarrollará un script para generar una matriz de confusión personalizada, que incluya más información numérica y los valores marginales. Se usará de base el proyecto de Yassine Ait Jedi[21] desarrollado para Mask R-CNN, adaptado para YOLOv5.

Por último, para facilitar el proceso de detección en YOLOv5, se modificarán los ficheros originales para permitir que se pueda proveer una lista de imágenes como parámetro de entrada en `train.py`, en lugar de administrar las imágenes de una en una.

[21] https://github.com/Altimis/Confusion-matrix-for-Mask-R-CNN

Mask R-CNN

La implementación de Mask R-CNN de Matterport es una adaptación de la versión original a Python 3 empleando Keras y TensorFlow. Esta implementación no incluye una integración con otras herramientas, como en el caso de YOLOv5. Además, está desarrollado para la versión 1.x de TensorFlow, sin actualizaciones en los últimos años. Esto puede causar problemas de integración en los entornos actuales por varios motivos.

Por un lado, la librería TensorFlow se vincula estáticamente a los drivers de la tarjeta gráfica. Esto quiere decir que una versión antigua de TensorFlow, compilada para unos drivers antiguos, no será compatible con tarjetas gráficas más modernas. En concreto, las versiones 1.x de TensorFlow no son compatibles con las tarjetas gráficas de NVIDIA de la generación 20xx en adelante.

Por otro lado, las versiones más modernas de TensorFlow, las 2.x, introducen cambios significativos en la API, de modo que las llamadas que se hacen a ella desde la implementación de Mask R-CNN están obsoletas.

Para afrontar este problema hay varias aproximaciones:

1. Portar el código de Mask R-CNN a TensorFlow 2.x. Esta es posiblemente la mejor solución a largo plazo, pero también es la que requiere mayor esfuerzo, ya que implica revisar, rehacer y validar toda la herramienta. Aunque es algo que puede ser interesante para trabajo futuro, escapa del alcance de este proyecto.
2. Descargar el código fuente de TensorFlow 1.x y compilarlo para los últimos drivers. Aunque esta opción es viable, es relativamente delicada, ya que hay muchas dependencias que pueden hacer que, aun funcionando en un equipo, no sea fácilmente portable. Además de ser un proceso delicado y muy manual, sería necesario repetirlo para futuros drivers que sean necesarios para nuevas gráficas.
3. Usar una imagen Docker base de NVIDIA-TensorFlow[22]. NVIDIA ofrece un conjunto de imágenes preparadas con diversos componentes. En el caso concreto de TensorFlow, es posible encontrar una imagen que incluye la versión 1.x de TensorFlow compilado para los últimos controladores de NVIDIA. La idea es la misma que en el punto anterior, tener una versión de TensorFlow 1.x compilada para los últimos drivers, con la ventaja de que este proceso ya hecho para todos los drivers más actuales de NVIDIA. En este caso, cuando sea necesario actualizar los drivers por una versión más moderna, solo es necesario descargar la versión más actual de la imagen Docker y crear un nuevo contenedor.

Dado que en este proyecto estamos trabajando con Docker para garantizar un entorno controlado y estable, la tercera opción soluciona los problemas de compatibilidad casi sin esfuerzo adicional. En Mask R-CNN no se dispone de una imagen Docker ya preparada, pero podemos generar una propia. Las imágenes Docker necesitan partir de una imagen ya existente, por lo que basta con indicar que nuestra imagen partirá de la de NVIDIA-Tensorflow 1.x para solucionar los problemas de compatibilidad.

Una vez solucionados los problemas de compatibilidad, nos centramos en el uso de la herramienta de Mask R-CNN en sí. Mask R-CNN, a diferencia de YOLOv5, no ofrece la API a través de diversos scripts para cada caso de uso, sino que expone diversas

[22] https://catalog.ngc.nvidia.com/orgs/nvidia/containers/tensorflow

funciones. Las más relevantes son `model.train` y `model.detect`. Mask R-CNN no ofrece una llamada explicita para el caso de validación, por lo que la evaluación de resultados deberá desarrollarse. Sí que ofrece una librería con diversas utilidades que facilitan el cálculo de métricas habituales, como el mAP, la precisión o el recubrimiento.

Las funciones del API de Mask R-CNN trabajan alrededor de dos clases: `Config` y `utils.Dataset`. Para utilizar la herramienta es necesario implementar las interfaces de ambas clases para que carguen la información de nuestro *dataset*.

Para más información sobre las características y el uso de la implementación de Mask R-CNN de Matterport se puede consultar el repositorio oficial (Abdulla, 2017).

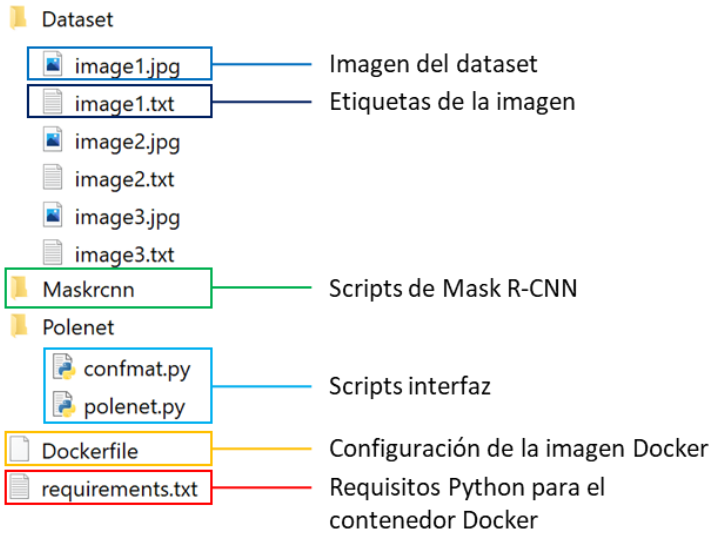

Figura 15: Ejemplo de estructura Mask R-CNN

En cuanto a los resultados, todo el proceso de gestión de las imágenes, las máscaras, los *bounding box*, las estadísticas y los gráficos es necesario gestionarlo en nuestra aplicación como parte de la integración Mask R-CNN, ya que la API solo proporciona los datos, sin generar ningún fichero adicional. Para este proyecto se ha preparado la generación de los gráficos de pérdida en el entrenamiento, las estadísticas de TF, FN y FP en la validación y la matriz de confusión (basado en la de Yassine Ait Jedi) en la validación y las imágenes con los *bounding box* en la detección. Adicionalmente, los resultados de cada caso de uso se guardarán en formato JSON para facilitar la automatización en procesos posteriores para la clasificación de mieles.

5. Desarrollo de la solución propuesta

En la descripción del desarrollo de las propuestas en este apartado se parte de un entorno que cumple todos los prerrequisitos necesarios para este proyecto y todas las dependencias y herramientas básicas ya instaladas (drivers necesarios, Python, Docker, GIT, etcétera). Se va a tratar el desarrollo de cada una de las herramientas (YOLOv5 y Mask R-CNN) de forma independiente.

El *dataset* de imágenes de mieles está compuesto por 15 colecciones de muestras que forman parte del conjunto de muestras de Polenet. Las imágenes de estos *datasets* están parcialmente etiquetadas (i.e.: algunos elementos presentes en las imágenes están clasificados, pero no todos ellos) y segmentadas mediante *bounding box* por expertos. El mismo *dataset* será utilizado para YOLOv5 y Mask R-CNN.

5.1 YOLOv5

El primer paso para trabajar con YOLOv5 es instalar la herramienta. En lugar de descargar el repositorio y construir la imagen Docker mediante el Dockerfile que incluye, se opta por descargar directamente la imagen Docker disponible en DockerHub. Con Docker instalado en el equipo, esto se realiza mediante el comando:

```
$ docker pull ultralytics/yolov5
```

En paralelo, se prepara un directorio `PROJ` en la máquina huésped en la que se incluirán todos los ficheros relacionados con el proyecto para compartirlos con el contenedor Docker. En PROJ se crea otro subdirectorio `Dataset` en el que se guarda una copia del *dataset* de Polenet.

Con estos elementos listos, se crea el contenedor Docker de YOLOv5, compartiendo la carpeta `PROJ`:

```
$ docker run --ipc=host -it -v ~/PROJ:/usr/src/app/PROJ --gpus all
  ultralytics/yolov5:latest
```

Notar que se emplea la opción `--gpus all` para permitir el uso de la GPU desde el contenedor Docker. También es importante señalar la opción `-it` para usar el contenedor de forma interactiva. Para más información sobre las opciones del comando `docker run`, se puede consultar el manual de Docker.

En este punto ya nos encontramos dentro del contenedor Docker de YOLOv5, completamente funcional. En el directorio `/usr/src/app/` se encuentran todos los elementos del repositorio de Ultralytics y el entorno ya cuenta con todas las dependencias instaladas. Ya es posible utilizar la herramienta de YOLOv5 tal y como se

describe en su sitio web. Adicionalmente, en el mismo directorio, se encontrará la carpeta `PROJ` con los mismos elementos que en nuestro sistema huésped fuera del contenedor Docker.

El siguiente paso, de cara a un posible trabajo colaborativo futuro, pero también para poder trabajar con control de versiones y otras utilidades que ofrece GIT, es preparar un repositorio `polenet-yolov5`[23] en GitLab y clonarlo en la carpeta `PROJ`. Aquí se irá colocando todo lo que se vaya desarrollando para el proyecto.

Una vez se tienen todos los elementos preparados, el primer trabajo que hay que realizar es adaptar el formato de las etiquetas de Polenet al requerido por YOLOv5. Las etiquetas del *dataset* de Polenet están contenidas en varios ficheros XML -uno para cada grupo- mientras que YOLOv5 requiere las etiquetas en un formato muy concreto, tal y como se define en el apartado 4.2. YOLOv5 también requiere una serie de ficheros adicionales, tal y como se indica en ese mismo apartado, para describir la red neuronal y el *dataset*.

Para generar estos ficheros se desarrolla el script gen_labels.py, que ofrece las siguientes funciones:

- `getData`: Obtiene la información del *dataset* de Polenet a partir de una lista con los ficheros XML.
- `showData`: Muestra un resumen de los pólenes etiquetados en el *dataset* de Polenet.
- `createTrainTestList`: Crea los ficheros de `train.txt` y `test.txt`, que contienen la lista de imágenes a usar en el entrenamiento y en la validación de la red neuronal, respectivamente.
- `createCNNConf`: Crea el fichero `polenet-YOLOv5l.yaml`, que contiene la descripción de la red neuronal convolucional que se va a emplear.
- `createDataConf`: Crea el fichero `polenet.yaml`, que describe el número, el ID y el nombre de las clases del *dataset*, así como la ubicación de las imágenes o, en su defecto, de las listas de entrenamiento y validación.
- `createLabels`: Genera los ficheros de etiquetas requeridos por YOLOv5 para todas las imágenes del *dataset*.
- `main`: Ofrece una interfaz de usuario que permite ejecutar el script directamente y seleccionar las funcionalidades descritas arriba mediante parámetros de entrada.

El desarrollo de este script está pensado para poder ser integrado dentro de otras herramientas más adelante, dando acceso a las principales funcionalidades, pero también permite ser ejecutado directamente en consola, ofreciendo las funcionalidades mediante opciones en los parámetros de llamada del script. Para generar todos los ficheros necesarios para trabajar con el *dataset* de Polenet en YOLOv5 basta con ejecutar:

```
$ python gen_labels.py -d <xmls> -C <clases> -S <tamaño> -t
  <ratio_train_test> -lc
```

[23] https://gitlab.com/joseluis.perezmartinez.94/polenet-yolov5

Donde `<xmls>` es el o los ficheros XML con la información del dataset de Polenet, `<clases>` son los IDs de las clases que se quieren emplear, `<tamaño>` define la complejidad de la red neuronal convolucionaly `<ratio_train_test>` es un número que define la proporción del dataset que se va a emplear para entrenamiento y para validación. Un ejemplo concreto podría ser el siguiente:

```
$ python gen_labels.py -d 21012.xml -C 7 12 17 -S l -t 0.8 -lc
```

Este ejemplo generaría los ficheros `train.txt`, `test.txt`, `polenet-YOLOv5l.yaml` y `polenet.yaml` a partir de la información del dataset de Polenet contenido en el fichero `21012.xml`, empleando solo las clases 7, 12 y 17 y usando el 80% de las imágenes para entrenamiento y el 20% para validación.

Es importante notar que el uso del script gen_labels.py, aunque no hace uso directo de la GPU ni de YOLOv5, y aunque la carpeta PROJ es virtualmente la misma en el huésped como en el contenedor Docker, la generación de ficheros debe realizarse dentro del contenedor, ya que las rutas de los ficheros implicados no tienen por qué coincidir en ambos contextos.

Una vez generados todos los elementos necesarios, ya es posible utilizar YOLOv5. El primer paso es entrenar la red, lo cual se puede hacer mediante el script `train.py` ubicado en `/usr/src/app/` dentro del contenedor Docker.

```
$ python train.py
```

El script train.py admite gran variedad de parámetros para configurar su funcionamiento, recogidos en la documentación de Ultralytics. Al final del proceso de entrenamiento, se generan varios ficheros y gráficos que ilustran el proceso de entrenamiento, y dos ficheros de pesos: `best.pt` y `last.pt`. Estos ficheros contienen los pesos de la red neuronal de la mejor y de la última iteración de entrenamiento, respectivamente. A lo largo del proyecto, salvo que se indique lo contrario, se empleará siempre `best.pt` para la validación y detección, renombrado como `polenet.pt`.

Tras finalizar el entrenamiento de la red, se procede a validar la calidad de la red neuronal obtenida. En este caso se empleará el script `val.py` de YOLOv5.

```
$ python val.py --weights polenet.pt
```

El proceso termina generando una serie de gráficos de diversas métricas para la evaluación de los resultados, así como algunos ejemplos del desempeño como en la Figura 16. Para poder procesar los resultados de la evaluación, se utiliza el parámetro `--save-json`, que genera un fichero JSON con las predicciones de la red neuronal.

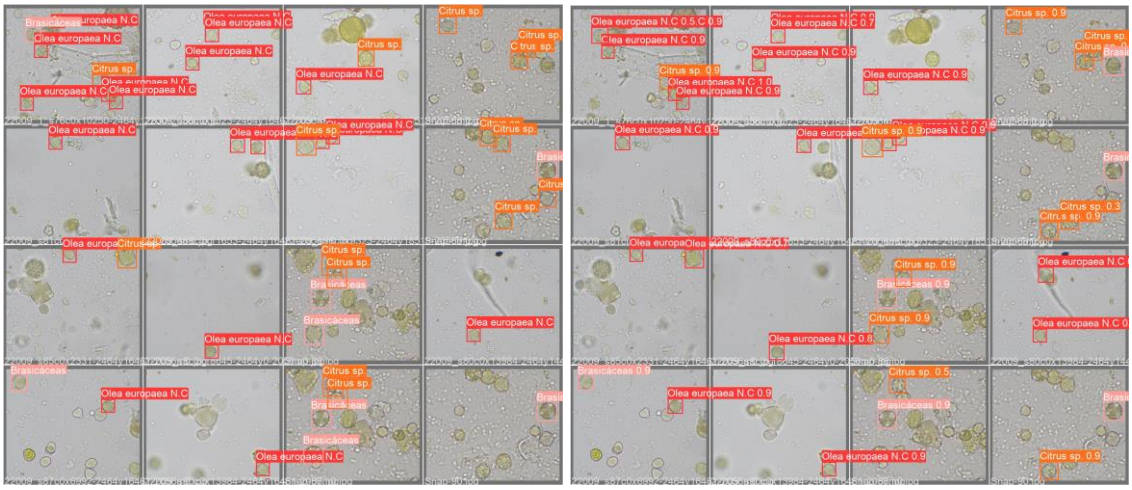

Figura 16: Comparativa entre las etiquetas y las predicciones de 16 imágenes. Izquierda las etiquetas del Ground Truth. Derecha las predicciones y su nivel de confianza.

La forma más sencilla y visual de analizar el desempeño de la red neuronal a la hora de realizar detecciones es mediante una matriz de confusión. Aunque YOLOv5 ya genera su propia versión de la matriz de confusión de forma automática (Figura 9), ésta carece de bastante información que puede ser de interés, como los valores absolutos y marginales de la matriz. En su lugar se va a generar una versión propia que incluya más información de interés, basada en la desarrollada por Yassine Ait Jedi para Mask R-CNN (Figura 10). Se parte del fichero de utilidades utils.py de su repositorio[24] y se desarrolla el script `gen_conf_mat.py` que adapta la idea de Mask R-CNN al formato de YOLOv5. `gen_conf_mat.py` ofrece las siguientes funciones:

- `loadGroundTruth`: Carga el GT del *dataset* utilizado en la validación mediante los XML de Polenet a un diccionario Python.
- `loadPredictions`: Carga las predicciones de la red neuronal del fichero JSON generado por YOLOv5 a un diccionario Python.
- `genConfMat`: A partir del GT y las predicciones de la red neuronal genera una matriz de confusión personalizada.
- `main`: Ofrece una interfaz de usuario que permite ejecutar el script directamente y seleccionar las funcionalidades descritas arriba mediante parámetros de entrada.

De nuevo, el script está diseñado tanto para poder integrarse como una librería dentro de otras aplicaciones futuras como para poder utilizarse directamente mediante la ejecución del script. Un ejemplo de uso del script puede ser:

```
$ python val.py --data polenet.yaml --weights polenet.pt --save-
  json

$ python gen_conf_mat.py -d 21012.xml -v test.txt -r polenet-
  predictions.json
```

[24] https://github.com/Altimis/Confusion-matrix-for-Mask-R-CNN/blob/master/utils.py

En este ejemplo se realiza la validación de la red neuronal mediante `val.py` y los resultados se guardan en `polenet-predictions.json`. Luego, `gen_conf_mat.py` toma los resultados, así como la información del *dataset* y del subconjunto empleado para test y genera la matriz de confusión personalizada.

Por último, para el caso de uso de detección no se ha desarrollado ningún programa específico, pero sí que se han modificado un par de ficheros de YOLOv5 para facilitar su uso. El script API de YOLOv5 para la detección de objetos mediante la red neuronal entrenada, `detect.py`, admite distintos tipos de ficheros de entrada para las imágenes a detectar (JPG, PNG, etcétera) y de hecho también admite formatos de vídeo como MP4, ya que también puede procesar vídeos. También admite otros tipos de elementos, como URLs o ficheros TXT para tratar streaming de vídeos. Dado que en este proyecto no interesa el procesado de vídeos, y aun menos en casos de streaming, se va a modificar cómo se trata los ficheros TXT para que en su lugar se considere como una lista de imágenes, de forma análoga a como se hace en los casos de entrenamiento y validación. Los ficheros modificados son `detect.py` y `datasets.py`. La detección de pólenes se realiza con el comando:

```
$ python detect.py --weights polenet.pt --source image_list.txt
```

El script generará una versión de las imágenes procesadas con los *bounding box* de los objetos detectados y opcionalmente un fichero CSV con las detecciones. Este CSV puede ser cargado en Excel para trabajar posteriormente con él.

Todos los scripts desarrollados en este apartado están documentados conforme a las convenciones de Docstring para Python. Además, es posible consultar el modo de empleo de los scripts mediante el comando `--help`. Se puede encontrar más información en el fichero README[25] del repositorio.

5.2 Mask R-CNN

El desarrollo la herramienta Mask R-CNN comienza, al igual que en el caso de YOLOv5, instalando la herramienta. A diferencia de YOLOv5, Mask R-CNN no cuenta con una imagen Docker preparada con el entorno. En su lugar se descarga el contenido del repositorio GIT en el directorio `Mask_RCNN`, directamente bajo el directorio raíz del proyecto (`PROJ`).

Una vez instalado el repositorio, se preparan dos ficheros para preparar el entorno en un contenedor Docker: un fichero Dockerfile para generar la imagen Docker y un fichero `requirements.txt` con todas las dependencias necesarias para Python. La imagen con la que se va a trabajar estará basada en la de Nvidia para TensorFlow 1 y Python 3. También se prepara un directorio `polenet` en el que se ubicará las utilidades propias desarrolladas para el proyecto de Polenet. Con estos elementos se genera la imagen Docker mediante el comando:

[25] https://gitlab.com/joseluis.perezmartinez.94/polenet-yolov5/-/blob/main/README.md

```
$ docker build -t mask-rcnn:tf1-py3 .
```

Y a continuación el contenedor mediante:

```
$ docker run --ipc=host -it -v /path/to/dataset:/usr/src/app/dataset -
  -gpus all mask-rcnn:tf1-py3
```

Se emplea la opción `--gpus all` para permitir el uso de la GPU desde el contenedor Docker y la opción `-it` para usar el contenedor de forma interactiva. Para más información sobre las opciones del comando `docker run`, se puede consultar el manual de Docker.

Dentro del contenedor de Mask R-CNN que se ha creado, todos los elementos del proyecto se encuentran en el directorio el directorio `/usr/src/app/`. En concreto se encontrarán los ficheros `Dockerfile` y `c`, junto a las carpetas `Mask-RCNN`, `polenet` y `dataset`. Esta última se ha vinculado de forma externa en la creación del contenedor al directorio en el equipo huésped en el que se encuentra toda la información del *dataset* de Polenet, a fin de no sobredimensionar el tamaño del contenedor. El Dockerfile con el que se ha creado la imagen, junto con el fichero `requirements.txt`, gestionan la configuración del entorno y la instalación de las dependencias, de modo que en el contenedor ya está todo preparado para utilizar Mask R-CNN.

Como se ha mencionado en el punto 4.2, para utilizar Mask R-CNN es necesario desarrollar una clase que implemente las interfaces de `Config` y `utils.Dataset` de acuerdo a las características de la red neuronal que se quiere emplear y del dataset, respectivamente. También es necesario desarrollar la herramienta que utilice la API que ofrece Mask R-CNN. Todo ello se ha hecho en el script `polenet.py`, que incluye:

- `PolenetConfig`: Implementación de la interfaz `Config`. La clase define la configuración de la red neuronal.
- `PolenetDataset`: Implementación de la interfaz `utils.Dataset`. Carga la información del *dataset* de Polenet y genera los conjuntos de entrenamiento y de validación.
- `PolenetCallback`: Implementa la interfaz `keras.callbacks.Callback` de la librería de Keras para realizar callbacks durante los procesos de entrenamiento, validación y detección. Es una utilidad que ofrece la API de Mask R-CNN y que en este proyecto se emplea para monitorizar la función de pérdida y el mAP, así como para generar registros.
- `visualize`: Genera una versión de la imagen recibida con los *bounding box* del GT sobre ella.
- `getData`: Obtiene la información del *dataset* de Polenet a partir de una lista con los ficheros XML.
- `showData`: Muestra un resumen de los pólenes etiquetados en el *dataset* de Polenet.
- `displayResults`: Basada en la propuesta realizada por un usuario en el repositorio de Matterport[26], genera una versión de la imagen recibida con los *bounding box* de la detección.

[26] https://github.com/matterport/Mask_RCNN/issues/134#issuecomment-424297149

- `train`: Entrena la red neuronal en el *dataset* de Polenet.
- `evaluate`: Evalúa la red neuronal entrenada en el *dataset* de Polenet.
- `detect`: Analiza una o varias imágenes para detectar y clasificar las partículas de polen presentes.
- `main`: Ofrece una interfaz de usuario que permite ejecutar el script directamente y seleccionar las funcionalidades descritas arriba mediante parámetros de entrada.

El script puede ejecutarse directamente en consola (con los parámetros adecuados) o integrarse en otra aplicación como una librería haciendo uso de las funciones mencionadas. En este proyecto se va a emplear directamente el script, dejando la integración con otras herramientas para posibles futuras ampliaciones de la herramienta.

Mask R-CNN está diseñado para trabajar tanto con *bounding box* como con máscaras, que definen con precisión el contorno de los objetos. El *dataset* de Polenet solo incluye información de los *bounding box*, no de las máscaras. Para superar esta limitación, se han probado 4 modelos "ingenuos" de generación automática de las máscaras:

- Máscara cuadrada: La máscara coincide con el bounding box (Figura 17). Esta aproximación es la equivalente a no contar con una máscara.
- Máscara circular: La máscara es un círculo inscrito en el bounding box (Figura 18).
- Máscara octagonal circunscrita: La máscara es un octógono que circunscribe al círculo inscrito en el bounding box (Figura 19).
- Máscara octagonal inscrita: La máscara es un octógono inscrito en el círculo inscrito en el bounding box (Figura 20).

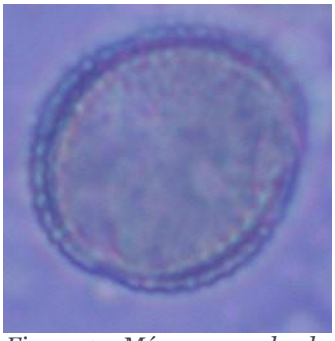

Figura 17: Máscara cuadrada

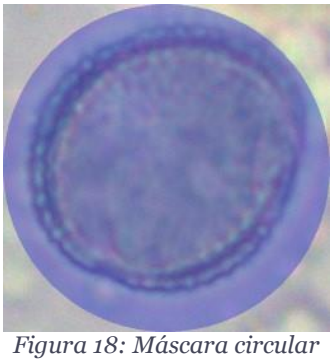

Figura 18: Máscara circular

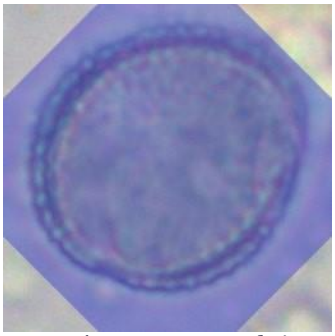

Figura 19; Máscara octogonal circunscrita

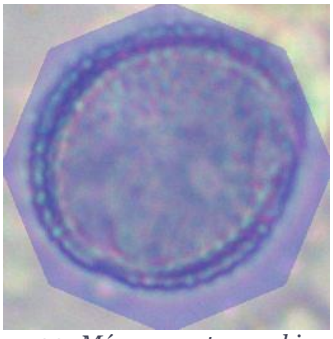

Figura 20: Máscara octogonal inscrita

Es posible escoger la máscara que se desea emplear al entrenar la red mediante el parámetro de entrada `--mask-shape`.

Cada caso de uso genera un fichero de logs y un fichero JSON con los resultados del proceso. El entrenamiento genera varios ficheros `.h5` con los pesos de la red neuronal, uno por cada iteración. Salvo que se indique lo contrario, a lo largo de este proyecto se empleará siempre el de la última iteración, que se renombrará como `polenet.h5`. En la validación de la red neuronal se ha integrado la generación de la matriz de confusión mediante el script `confmat.py`, basado en el desarrollado por Yassine Ait Jedi.

Para entrenar la red neuronal de Mask R-CNN, basta con ejecutar el script `polenet.py` con la opción `train`:

```
$ python polenet.py train -d <xmls>
```

La opción `-d <xmls>` es obligatoria y permite seleccionar uno o varios de los ficheros XML que contienen la información del *dataset* de Polenet. Una vez se tiene el fichero de pesos, se puede evaluar la calidad de la red neuronal de para Mask R-CNN mediante la opción `val`:

```
$ python polenet.py val -d <xmls> -w polenet.h5
```

El script también permite detectar granos de polen en imágenes que no forman parte del dataset de Polenet mediante la opción `detect`:

```
$ python polenet.py detect -d <xmls> -w polenet.h5 -i <input>
```

El parámetro `-i <input>` puede ser una imagen, una serie de imágenes o un fichero de texto con la lista de imágenes a procesar. El proceso de detección genera los resultados en un fichero JSON para permitir su posterior procesado por otras herramientas, pero

también permite generar una versión de las imágenes con los *bounding box* y las máscaras localizadas por la red neuronal (Figura 21).

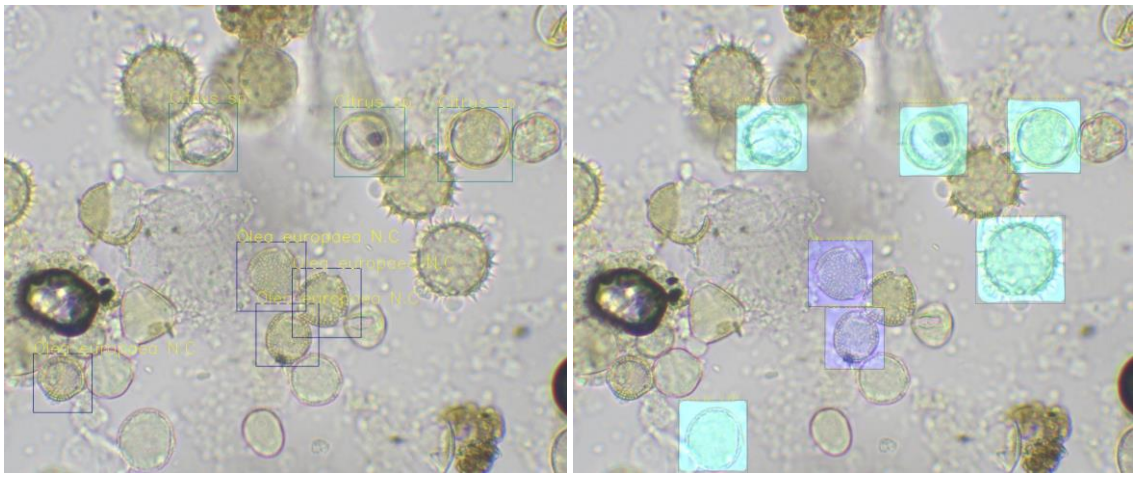

Figura 21: Comparativa entre las etiquetas y las predicciones de Mask R-CNN con máscaras cuadradas. Izquierda las etiquetas. Derecha las predicciones y su nivel de confianza.

El script desarrollado en este apartado, `polenet.py`, está documentados conforme a las convenciones de Docstring para Python. Además, es posible consultar el modo de empleo del script mediante el comando `--help`. Se puede encontrar más información en el fichero README[27] del repositorio.

[27] https://gitlab.com/joseluis.perezmartinez.94/polenet-maskrcnn/-/blob/master/README.md

6. Pruebas

Una vez se ha preparado tanto el entorno para YOLOv5 como para Mask R-CNN, se ha procedido a realizar una serie de experimentos para comprobar cómo se comporta cada una de las implementaciones de las *convnets*.

La idea es empezar con un subconjunto del dataset de Polenet y unas pocas clases de pólenes, para ir incrementando progresivamente el tamaño del dataset y el número de clases de pólenes. También se va a ir alternando entre la implementación de YOLOv5 y de Mask R-CNN, repitiendo el mismo experimento en ambos entornos y comparando los resultados.

El dataset inicial se compone de 3 muestras, indicadas en la Tabla 2. En cuanto a los pólenes se han escogido los 3 más abundantes en las muestras: *olea europaea* (625), *citrus sp.* (278) y *brasicáceas* (104). En total se disponen de 352 imágenes que contienen alguno de los pólenes escogidos etiquetados, de las cuáles el 80% (282) se usan para entrenar y el 20% (70) para validación.

	Muestra	**Miel**
1	22009 Azh	Azahar
2	Azahar 21334	Azahar
3	Azahar_21181	Azahar

Tabla 2: Muestras del dataset inicial

El siguiente experimento consiste en, utilizando el mismo dataset, ampliar el conjunto de clases hasta 10. En este caso los pólenes con los que se trabajan son: *olea europaea* (625), *citrus sp.* (278), *brasicáceas* (104), *cistus sp.* (102), *echium sp.* (99), *quercus sp.* (85), *helianthus annuus* (73), *asteráceas* (48), *eucaluptus sp.* (47) y *desconocido contable* (45).

A partir de este punto las siguientes clases cuentan con muy poca representación, así que en el último experimento se emplearán todas las muestras del dataset de Polenet de las que se disponía en el momento de realizar el experimento (Tabla 3). En este caso se emplearán todas las clases que tengan al menos 500 muestras etiquetadas en el dataset. Esto da como resultado las siguientes 12 clases: *brasicáceas* (3781), *olea europaea* (1663), *eucalyptus sp.* (1758), *onobrychis sp.* (1243), *echium sp.* (1074), *citrus sp.* (854), *quercus sp.* (842), *rosmarinus officinalis* (827), *thymus sp.* (760), *helianthus annuus* (757), *cistus sp.* (616) y *castanea sativa* (513).

	Muestra	**Miel**
1	20655	Mil flores
2	21012	Eucalipto
3	21158	Romero
4	21320_girasol	Girasol
5	21333_RO & ALM	Romero
6	21339_EUC_Helianthus	Eucalipto
7	21351_BRASSICA	Mil flores/Colza
8	Miel Cantueso	Cantueso
9	22009 Azh	Azahar

10	Azahar 21334	Azahar
11	Azahar_129	Azahar
12	22023 Tomillo	Tomillo
13	Azahar 21334	Azahar
14	Azahar limon 21354	Azahar
15	Azahar_252	Azahar
16	Azahar_21181	Azahar
17	I-22023 AZ	Azahar
18	Onobrychis 21385	Mil flores
19	Romero 21350	Romero
20	Romero_33	Romero
21	Romero_161	Romero
22	Tom 21348	Tomillo

Tabla 3: Muestras del dataset ampliado

En los apartados siguientes se va a comentar los resultados obtenidos para estos experimentos en cada uno de los entornos.

6.1 YOLOv5

El entorno de trabajo de YOLOv5 está dentro del contenedor Docker dedicado, tal y como se describe en el apartado 4. Los experimentos descritos aquí se realizan todos dentro de este entorno.

En todos los experimentos se va a emplear la misma arquitectura para la red neuronal: el mismo *backbone*, el mismo *head*, la misma amplitud y la misma profundidad. Lo único que se va a modificar es la capa de salida de acuerdo con el número de clases en cada experimento. Además, para cada experimento se van a entrenar la red desde cero y se va a utilizar la configuración de hiperparámetros (incluyendo *data augmentation*) que viene por defecto en la configuración de YOLOv5.

6.1.1 3 Clases

Como se describe en el apartado 5.1, el entorno de YOLOv5 necesita preparar unos ficheros con la información del dataset para que la herramienta pueda trabajar con ella. Este trabajo lo realiza uno de los scripts que se han desarrollado, y que en este caso se ejecuta con el siguiente comando en consola:

```
$ python gen_labels.py -d "../Datasets/22009 Azh/22009 Azh.xml"
  "../Datasets/Azahar 21334/Azahar 21334.xml"
  "../Datasets/Azahar_21181/Azahar_21181.xml" -C 7 12 17 -S 1 -t
  0.8 -lc
```

Este comando ya genera todos los ficheros de etiquetas necesarios, así como los YAML con la descripción del dataset y de la red neuronal a utilizar. A continuación, se procede con el entrenamiento de la red.

Entrenamiento

El entrenamiento se inicia utilizando el script `train.py`, incluido con la herramienta de YOLOv5. Para esta prueba inicial se utiliza el número de iteraciones recomendada por defecto para YOLOv5 (300):

```
$ python train.py --cfg polenet-YOLOv5l.yaml --data polenet.yaml
  --device 0 --project polenet --epochs 300
```

Al finalizar el entrenamiento obtenemos el fichero de pesos de la red neuronal entrenada y varias gráficas que evalúan el proceso de entrenamiento, de las cuales vamos a utilizar la de la Figura 22 para analizar el desempeño. Esta figura contiene los siguientes elementos:

- *Train/box_loss*: Función de pérdida del *bounding box* de los objetos detectados en el entrenamiento. Mide el error del *bounding box* de la detección frente al del GT.
- *Val/box_loss*: Función de pérdida del *bounding box* de los objetos detectados en la validación.
- *Train/obj_loss*: Función de pérdida de la detección de objetos en el entrenamiento. Mide el error producido por no detectar objetos que se encontraban en el GT o por detectar objetos que no están etiquetados.
- *Val/obj_loss*: Función de pérdida de la detección de objetos en la validación.
- *Train/cls_loss*: Función de pérdida de la clasificación de objetos en el entrenamiento. Mide el error a la hora de clasificar los objetos detectados.
- *Val/cls_loss*: Función de pérdida de la clasificación de objetos en la validación.
- *Metrics/precision*: Precisión de la red neuronal a lo largo del entrenamiento.
- *Metrics/recall*: Recubrimiento de la red neuronal a lo largo del entrenamiento.
- *Metrics/mAP_0.5*: Evolución de la Precisión Promedio media para un IoU umbral del 50% a lo largo del entrenamiento.
- *Metrics/mAP_0.5:0.95*: Evolución de la Precisión Promedio media para umbrales del 50% al 95% a lo largo del entrenamiento.

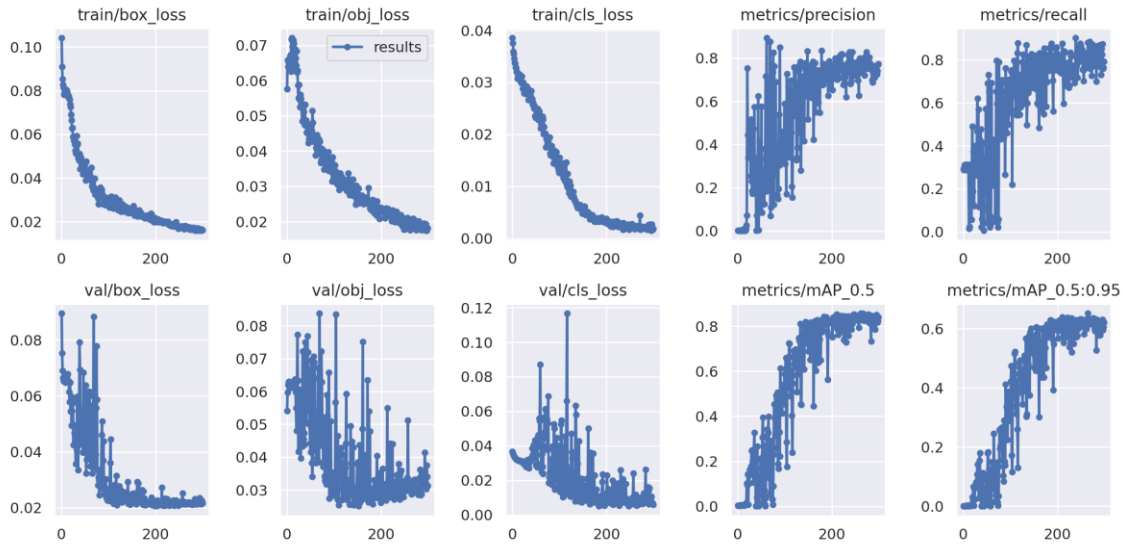

Figura 22: Métricas del proceso de entrenamiento de la red neuronal YOLO para 3 clases

En principio, la función de pérdida de los *bounding box* no es muy relevante para nuestro caso de uso, ya que nos interesa más la correcta detección y clasificación de las partículas de polen que la precisión con la que se ajuste el *bounding box*. Aun así, se puede ver que el error desciende a lo largo del entrenamiento hasta que se estabiliza al final, tanto para el conjunto de entrenamiento como para el de validación.

La función de pérdida de la detección de objetos sufre bastante volatilidad a lo largo del entrenamiento para el conjunto de validación. Esto puede deberse a que no todas las instancias de las clases están etiquetadas en el dataset, solo un subconjunto. Además, hay muchos elementos que no están etiquetados, ya sea porque no lo estaban originalmente o porque eran elementos de las clases que se han descartado para este experimento. Esto puede llevar a que la red entrenada detecte pólenes correctamente, pero que no estén etiquetados en el GT y den error en la función de pérdida. A pesar de eso, parece que tiende a estabilizarse y minimizarse al final del entrenamiento. En las últimas instancias termina con un valor superior a lo estimado en el entrenamiento, con lo que se puede esperar algunas limitaciones en la capacidad de generalizar del modelo en este aspecto.

La función de pérdida de la clasificación de objetos también presenta cierto nivel de volatilidad, aunque en este caso es más achacable a la naturaleza del problema. Las diferencias entre los distintos tipos de pólenes pueden ser difíciles de extraer. Éste es el parámetro que va a ser más interesante seguir cómo evoluciona conforme se vayan añadiendo más clases. Al final del entrenamiento esta función de pérdida se estabiliza y alcanza el mínimo para el conjunto de evaluación, con valores cercanos a los de entrenamiento.

Por su parte mAP@0.5 al final del entrenamiento se estabiliza por encima de 0.8, que en principio es un valor aceptable teniendo en cuenta el tamaño del dataset y las limitaciones de éste. Este valor y el mAP@[0.5:0.95] lo usaremos como valor de referencia para comparar los resultados entre experimentos.

Validación

La validación se realiza mediante el script `val.py` ofrecido con YOLOv5:

```
$ python val.py --data polenet.yaml --weights weights/polenet.pt
  --device 0 --save-json
```

Al finalizar la validación, se generan una serie de gráficos que ilustran los resultados con algunas métricas. En concreto es de interés la curva PR (Figura 23), que ilustra el desempeño general de la red, así como para cada una de las clases. Esta gráfica también incluye en la leyenda el valor del mAP@50 general y de cada clase.

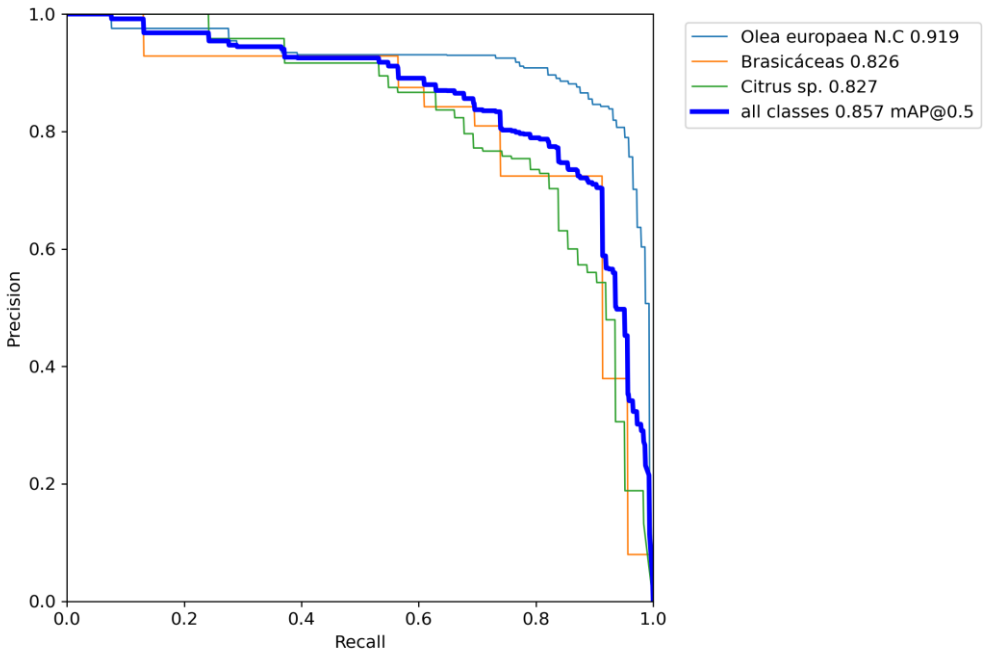

Figura 23: Curva PR de la red neuronal YOLO para 3 clases

La red neuronal entrenada tiene un mAP@0.5 de 0.857, pero se puede observar que es más eficaz a la hora de trabajar con la clase *olea europaea* (0.919 mAP@0.5) que con las otras dos. Este comportamiento entra dentro de lo esperable, ya que el dataset disponía de muchas más muestras de esta clase que de las otras.

Adicionalmente se genera un fichero json con los resultados de la validación, que emplearemos para generar la matriz de confusión con el programa que se ha desarrollado (`gen_conf_mat.py`) :

```
$ python gen_conf_mat.py "../Datasets/22009 Azh/22009 Azh.xml"
  "../Datasets/Azahar 21334/Azahar 21334.xml"
  "../Datasets/Azahar_21181/Azahar_21181.xml" -C 7 12 17 -v
  test.txt -r polenet_predictions.json -s 0.5
```

La matriz de confusión se ha generado para un valor de confianza umbral de 0.5 y se ha transcrito a la Tabla 4 para mejor legibilidad.

		Real				**Total**
		Fondo	Olea europaea	Brasicáceas	Citrus sp.	
Detectado	Fondo		19 7.5%	4 1.6%	23 9.0%	**46** **0.0%** **100.0%**
	Olea europaea	15 5.9%	124 48.6%		1 0.4%	**140** **88.6%** **11.4%**
	Brasicáceas	6 2.3%	1 0.4%	18 7.1%		**25** **72.0%** **28.0%**
	Citrus sp.	4 1.6%	1 0.4%	1 0.4%	38 14.9%	**44** **86.4%** **13.6%**
Total		**25** **0.0%** **100.0%**	**145** **85.5%** **14.5%**	**23** **78.3%** **21.7%**	**62** **61.3%** **38.7%**	**255** **70.6%** **29.4%**

Tabla 4: Matriz de confusión de la red neuronal YOLO para 3 clases

En la matriz de confusión se puede observar de nuevo que *olea europaea* es la clase con mejor precisión (88.6%) y mejor recubrimiento (85.5%). También se puede observar que la clase *citrus sp.* tiene un recubrimiento bastante inferior a la de las otras clases (61.3%) aun cuando su precisión es bastante alta (86.4%). Lo más probable es que con las muestras de entrenamiento de las que dispone a la red neuronal le cueste distinguir las partículas de *citrus sp.* del fondo (i.e. otras partículas no aprendidas por la red). Esto se podrá comprobar en los siguientes experimentos.

En general este experimento muestra que todos los componentes de la herramienta de YOLOv5 se comporta como se esperaba y permite establecer unos valores de partida razonables para tomar como referencia en los siguientes experimentos.

6.1.2 10 Clases

Para este experimento de nuevo se ejecuta el scipt para generar las etiquetas y demás ficheros necesarios, esta vez para 10 clases

```
$ python gen_labels.py -d "../Datasets/22009 Azh/22009 Azh.xml"
  "../Datasets/Azahar 21334/Azahar 21334.xml"
  "../Datasets/Azahar_21181/Azahar_21181.xml" -C 7 8 11 12 17 21
  24 26 37 100 -S 1 -t 0.8 -lc
```

Entrenamiento

El entrenamiento se inicia de nuevo con el mismo comando que para 3 clases. Se establece las iteraciones máximas en 500 para dar más tiempo de aprendizaje a la red neuronal:

```
$ python train.py --cfg polenet-YOLOv5l.yaml --data polenet.yaml
  --device 0 --project polenet --epochs 500
```

Las métricas de este entrenamiento se recogen en la Figura 24.

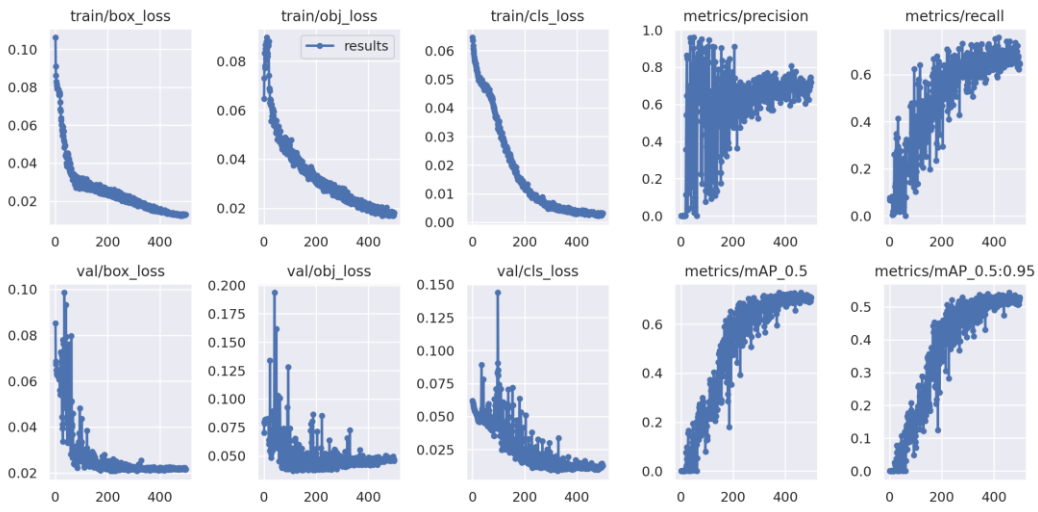

Figura 24: Métricas del proceso de entrenamiento de la red neuronal YOLO para 10 clases

En este caso se puede observar que la volatilidad de la función de pérdida para la detección de objetos ha aumentado significativamente respecto a las 3 clases, alzando valores de error máximos de casi 0.2, frente al 0.09 de error máximo que presenta en el entrenamiento. Incluso cuando al final el error converge y se estabiliza, el error al final del entrenamiento es alto (~0.05) en comparación con el error en el entrenamiento (~0.02) o incluso con el del experimento de 3 clases (~0.3).

También la función de pérdida de la clasificación presenta más volatilidad durante el entrenamiento, aunque en este caso sí que parece que al final del entrenamiento converge a un error similar al observado para 3 clases.

La precisión refleja también esta volatilidad presente en la primera mitad del entrenamiento. Al final converge a un valor alrededor del 0.7, frente a la precisión de 0.8 de la red para 3 clases. También el recubrimiento al final del entrenamiento es una décima más baja que en la red de 3 clases.

Por último, el mAP@0.5 y el mAP@[0.5:0.95] acaban convergiendo alrededor de 0.7 y 0.5, respectivamente. Esto es también una décima menos para cada uno en comparación con los resultados para 3 clases.

En general las métricas indican que la calidad de la red neuronal ha empeorado respecto a la versión con solo 3 clases. En general este comportamiento era esperado, ya que las clases introducidas en este experimento tienen mucha menor representación en el *dataset*, teniendo algunas clases menos de 50 muestras.

Validación

Los datos del proceso de entrenamiento ya indican peores métricas para la red neuronal de este experimento frente a la versión con 3 clases. A continuación, se va a estudiar el impacto que tiene esto directamente sobre el desempeño de la red y comprobar si el hecho de manejar más clases, aun con peor calidad, ayuda a mejorar la red.

En la curva PR (Figura 25) se puede observar un claro deterioro en el comportamiento general de la red para todas las clases, con un mAP@0.5 de 0.730 frente al 0.857 que presentaba con 3 clases. Sin embargo, la capacidad de detección de las clases que ya estaban presentes anteriormente sí que ha mejorado ligeramente: de 0.919 a 0.928 para *olea europaea*, de 0.826 a 0.828 para *brasicáceas* y de 0.827 a 0.838 para *citrus sp.*.

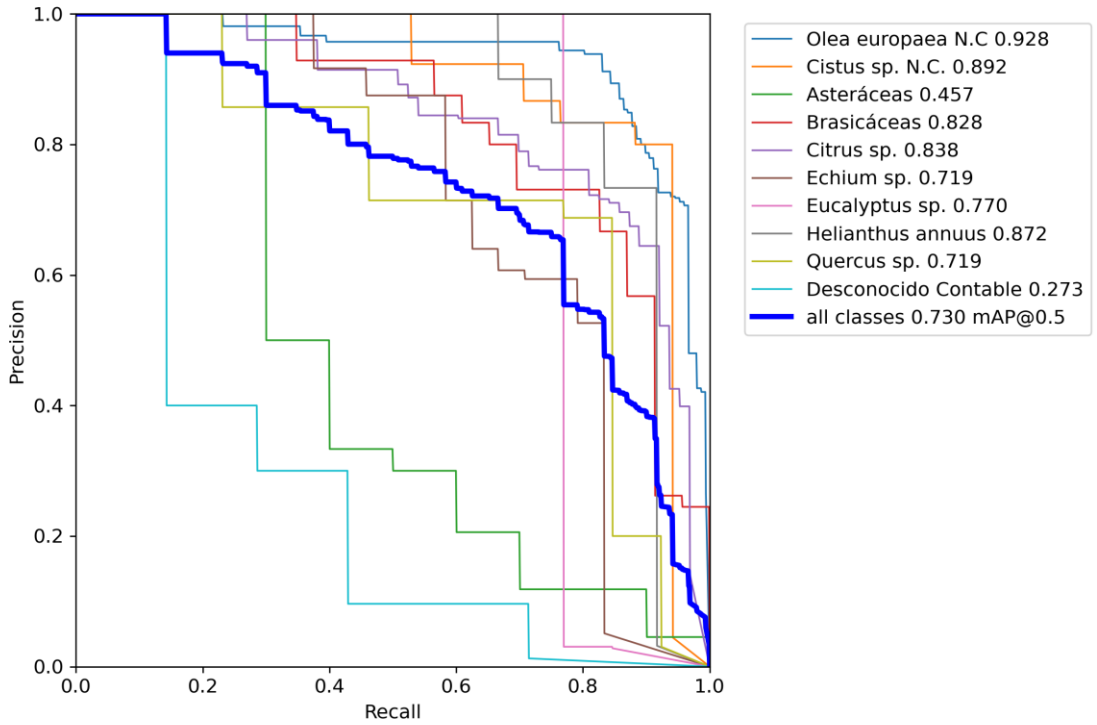

Figura 25: Curva PR de la red neuronal YOLO para 10 clases

En general se puede observar que las clases que peor detecta la red neuronal son las que tienen menor representación en el dataset, siendo especialmente dramático para los casos de *astaráceas* (0.457 mAP@0.5) y *desconocido contable* (0.273 mAP@0.5), que además presentan bastante variabilidad entre las muestras. La excepción parece ser la clase *eucaliptus sp.*, que a pesar de tener menos de 50 muestras tiene una de las mejores puntuaciones con un mAP@0.5 de 0.770. La justificación de este fenómeno seguramente se pueda explicar a través de su característica forma triangular (Figura 26), que permite distinguirlo fácilmente de otros pólenes.

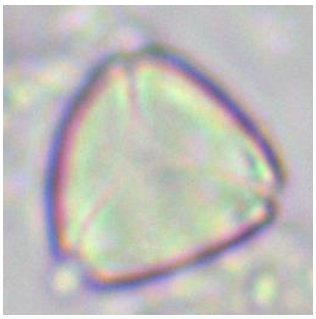

Figura 26: Partícula de polen de Eucalyptus sp.

La matriz de confusión respalda las observaciones realizadas en la curva PR. Las clases con más representación en general se comportan mejor que las que tienen menos. También se puede observar que con el tamaño del dataset que se está empleando, el GT de validación cuenta con muy pocos elementos para algunas clases, con menos de 20 muestras en muchos casos. Esto aumenta mucho la variabilidad y limita la efectividad y las conclusiones que se pueden extraer.

		Real											
		Fondo	Ol.	Cis.	As.	Br.	Cit.	Ec.	Eu.	He.	Qu.	D.C.	**Total**
Detectado	Fondo		23 6.1%	1 0.3%	5 1.3%	1 0.3%	15 4.0%	10 2.7%	3 0.8%	3 0.8%	3 0.8%	3 0.8%	**67** **0.0%** **100.0%**
	Olea europaea	9 2.4%	124 32.7%		2 0.5%		2 0.5%						**137** **90.5%** **9.5%**
	Cistus sp.	4 1.1%		16 4.2%									**20** **80.0%** **20.0%**
	Asteráceas	3 0.8%			3 0.8%								**6** **50.0%** **50.0%**
	Brasicáceas	8 2.1%				20 5.3%	1 0.3%					1 0.3%	**30** **66.7%** **33.3%**
	Citrus sp.	9 2.4%				1 0.3%	44 11.6%		1 0.3%				**55** **80.0%** **20.0%**
	Echium sp.	4 1.1%						14 3.7%					**18** **77.8%** **22.2%**
	Eucalyptus sp.	-							10 2.7%				**10** **100.0%** **0.0%**
	Helianthus annuus	2 0.5%								9 2.4%			**11** **81.8%** **18.2%**
	Quercus sp.	3 0.8%					2 0.5%				10 2.6%		**15** **66.7%** **33.3%**
	Desconocido Contable	6 1.6%				1 0.3%						3 0.8%	**10** **30.0%** **70.0%**
	Total	**48** **0.0%** **100.0%**	**147** **84.3%** **15.7%**	**17** **94.1%** **5.9%**	**10** **30.0%** **70.0%**	**23** **87.0%** **13.0%**	**64** **68.7%** **31.3%**	**24** **58.3%** **41.7%**	**14** **71.4%** **28.6%**	**12** **75.0%** **25.0%**	**13** **76.9%** **23.1%**	**7** **42.9%** **57.1%**	**379** **66.7%** **33.3%**

Tabla 5: Matriz de confusión de la red neuronal YOLO para 10 clases

6.1.3 12 Clases

En el experimento de 12 clases se van a emplear todas las muestras de las que se disponen. Para ello se ha preparado un fichero `dataset.txt` con la lista de los *datasets*. Se ejecuta el script utilizando este fichero para indicar los *datasets* a utilizar:

```
$ python gen_labels.py -d "../Datasets/dataset.txt" -C 12 24 7
  33 21 17 37 40 43 26 8 14 -S 1 -t 0.8 -lc
```

Entrenamiento

El entrenamiento en este caso se fija de nuevo en 300 iteraciones. Al tener un dataset mayor no es necesario iterar sobre él tantas veces:

```
$ python train.py --cfg polenet-YOLOv5l.yaml --data polenet.yaml
  --device 0 --project polenet --epochs 300
```

Las métricas de este entrenamiento se recogen en la Figura 27. Los errores convergen rápidamente y en unas 100 iteraciones ya se ha alcanzado los errores mínimos. El algoritmo de YOLOv5 detiene el entrenamiento si no observa una mejora tras 100 ciclos,

por lo que a pesar de que se habían programado 300 iteraciones, el entrenamiento se ha detenido tras poco más de 200.

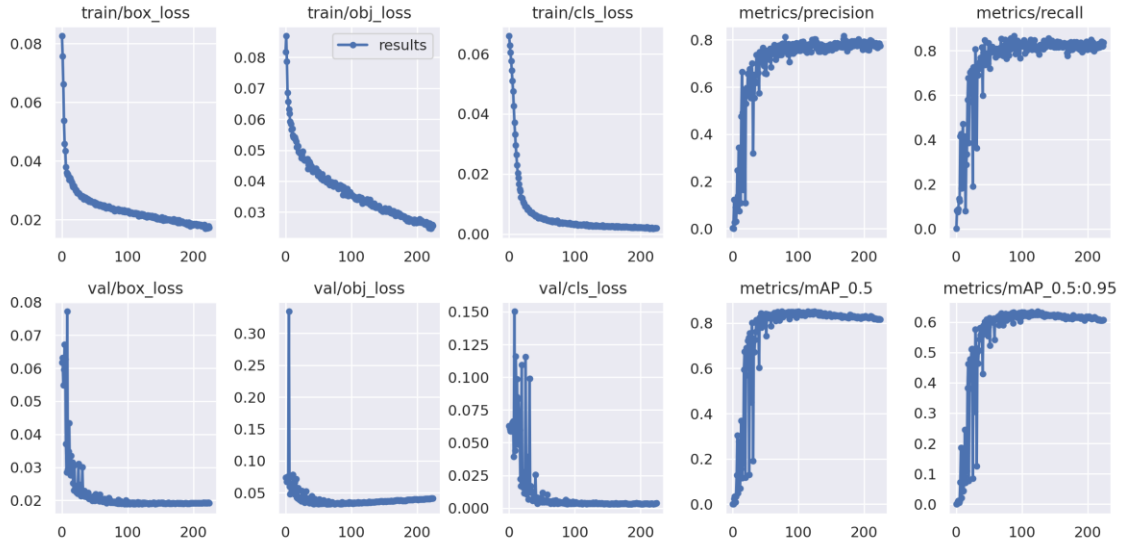

Figura 27: Métricas del proceso de entrenamiento de la red neuronal YOLO para 12 clases

Las funciones de pérdida de la detección de objeto y de clasificación son mucho más estables durante el entrenamiento en este experimento, y los valores finales son algo mejores que en el experimento de 10 clases. Sí que parece que puede darse un ligero *overfitting* en la función de pérdida de la detección de objetos, pero seguramente no tenga un impacto muy relevante en el desempeño de la red.

Las gráficas de mAP@0.5 y mAP@[0.5:0.95] también reflejan menor variabilidad y un desempeño mejor que en el caso de 10 clases, con valores similares a los que conseguía la red neuronal con solo 3 clases. También refleja un ligero descenso del rendimiento de la red durante la segunda mitad del entrenamiento, posiblemente relacionado con el ligero *overfitting* mencionado antes. En cualquier caso, YOLOv5 ya escoge la versión de los pesos que ha obtenido mejores resultados, por lo que no hay que preocuparse.

La precisión y el recubrimiento también confirman esta mejoría en el proceso de entrenamiento de la red y el rendimiento esperado de ella, mejorando incluso los resultados con 3 clases.

Validación

La curva PR en este experimento (Figura 28) es vuelve a ser similar a la de 3 clases. El mAP@0.5 de todas las clases es 0.850, cercano al 0.857 obtenido para 3 clases y significativamente mejor que el 0.730 conseguido por la red para 10 clases. Estos resultados corroboran la importancia de aumentar el tamaño del *dataset* de entrenamiento conforme se van introduciendo más clases al modelo.

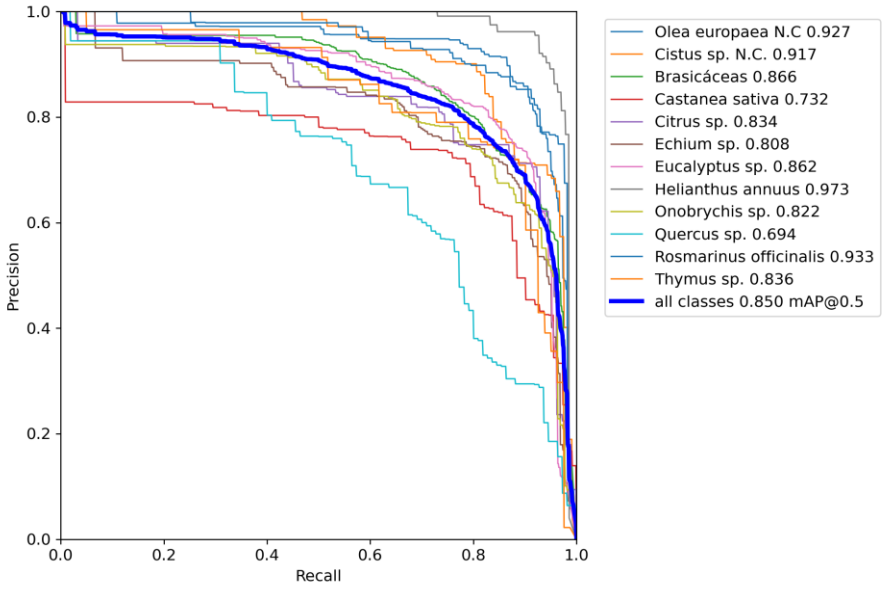

Figura 28: Curva PR de la red neuronal YOLO para 12 clases

Analizando los resultados individuales de cada clase, en este experimento se da la situación antiintuitiva de que algunas de las clases con menos muestras, *cistus sp.* (616) y *helianthus annuus* (757), tienen algunos de los mAP@0.5 más altos: 0.917 y 0.973, respectivamente. Algunas de las posibles explicaciones podrían ser el hecho de que las partículas de estos pólenes sean especialmente características y fáciles de reconocer por la red neuronal (Figura 29) o que para estos casos prácticamente todas las apariciones estén etiquetadas en el dataset, mientras que otras clases más abundantes tienen más muestras no etiquetadas, lo que afecta a la precisión de sus modelos.

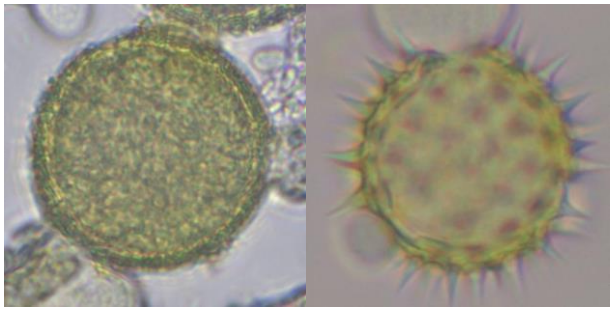

Figura 29: Izquierda, una muestra de Cistus sp. Derecha, una muestra de Helianthus Annuus.

En el otro extremo, las clases que resultan más problemáticas para el modelo de la red neuronal son *castanea sativa* (0.732 mAP@50) y *quercus sp.* (0.694 mAP@0.5). La clase *castanea sativa* es, de hecho, la que tiene menor representación en el dataset de las clases seleccionadas (513). Sin embargo, los resultados de las clases *cistus sp.* y *helianthus annuus* hacen sospechar que con al menos 500 muestras debería ser suficiente para que la red neuronal pueda trabajar bien con la clase.

Algo que tienen en común ambas clases es que no son componentes principales de las mieles cuyas muestras conforman el dataset. Esto, junto a que las partículas de polen son más difíciles de reconocer, puede que el proceso de etiquetado haya sido más laxo para estás clases, dejando un ratio mayor de apariciones sin etiquetar en comparación con otras clases.

Por último, puede que se deba simplemente a que las características de estas clases le sean más difíciles de extraer a la red neuronal. La matriz de confusión (Tabla 6: Matriz de confusión de la red neuronal YOLO para 12 clases) puede aportar más información en ese sentido.

		Real													
		Fondo	Ol.	Cis.	Br.	Ca.	Cit.	Ec.	Eu.	He.	On.	Qu.	Ro.	Th.	**Total**
Detectado	Fondo		35 0.9%	26 0.6%	127 3.1%	19 0.5%	54 1.3%	48 1.2%	75 1.8%	8 0.2%	46 1.1%	34 0.8%	11 0.3%	9 0.2%	**492** **0.0%** **100.0%**
	Olea europaea	69 1.7%	513 12.5%		9 0.2%										**591** **86.8%** **13.2%**
	Cistus sp.	14 0.3%		114 2.8%						1 0.0%					**129** **88.4%** **11.6%**
	Brasicáceas	185 4.5%	5 0.1%		617 15.1%										**807** **76.5%** **23.5%**
	Castanea sativa	51 1.3%				56 1.4%									**107** **52.3%** **47.7%**
	Citrus sp.	48 1.2%					212 5.2%								**260** **81.5%** **18.5%**
	Echium sp.	70 1.7%						199 4.9%							**269** **74.0%** **26.0%**
	Eucalyptus sp.	233 5.7%			1 0.0%		1 0.0%		367 9.0%						**602** **61.0%** **39.0%**
	Helianthus annuus	13 0.3%								146 3.6%					**159** **91.8%** **8.2%**
	Onobrychis sp.	58 1.4%			1 0.0%						176 4.3%		2 0.0%		**237** **74.3%** **25.7%**
	Quercus sp.	54 1.3%										88 2.1%			**142** **62.0%** **38.0%**
	Rosmarinus officinalis	36 0.9											158 3.9%		**194** **81.4%** **18.6%**
	Thymus sp.	30 0.7												72 1.8%	**102** **70.6%** **29.4%**
	Total	**861** **0.0%** **100.0%**	**553** **92.8%** **7.2%**	**140** **81.4%** **18.6%**	**755** **81.7%** **18.3%**	**75** **74.7%** **25.3%**	**267** **79.4%** **20.6%**	**247** **80.6%** **19.4%**	**442** **83.0%** **17.0%**	**155** **94.2%** **5.8%**	**222** **79.3%** **20.7%**	**122** **72.1%** **27.9%**	**171** **92.4%** **7.6%**	**81** **88.9%** **11.1%**	**4091** **66.4%** **33.6%**

Tabla 6: Matriz de confusión de la red neuronal YOLO para 12 clases

El error de clasificación más frecuente es entre las clases *olea europaea* y *brasicáceas*, con un total de 14 errores (0.3% de todas las detecciones). En principio es un error razonable ya que, por un lado, son las dos clases más abundantes, y por otro, son relativamente similares (Figura 30).

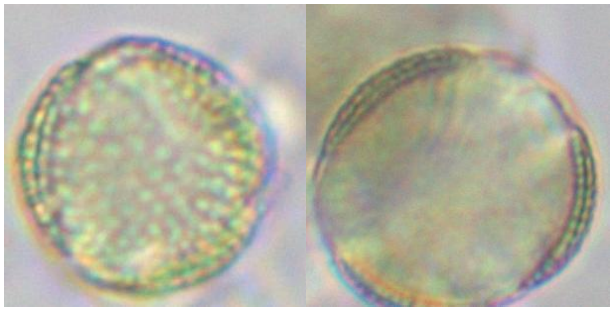

Figura 30: Izquierda, una muestra de Olea Europaea. Derecha, una muestra de Brasicáceas.

En los valores marginales de la matriz de confusión se puede apreciar que la red neuronal ofrece un recubrimiento mínimo del 72.1% de las clases que componen el modelo y hasta un 94.2% para el caso de *helianthus annuus* (84.1% de promedio). Por su parte la precisión es algo menor, con una precisión mínima del 52.3% para *castanea sativa* y de hasta el 91.8% para *helianthus annuus* (66.4% de promedio). Estos resultados refuerzan la hipótesis la falta de precisión del modelo, principalmente motivado por el etiquetado incompleto, es uno de los principales factores limitando la calidad del clasificador.

Sin embargo, para el caso de uso en el que está encapsulado este proyecto, la clasificación de mieles, el factor más importante es el recubrimiento, aun cuando la precisión sigue siendo un factor relevante.

En la clasificación de mieles por polimetría manual no se clasifican todas las instancias de pólenes de las muestras. En su lugar se clasifica un número de elementos elegidos al azar, y se hace el recuento sobre ellos. En esta situación prima la precisión sobre el recubrimiento.

Por el contrario, la clasificación automatizada no tiene que seleccionar un subconjunto de instancias al azar, sino que puede reconocer todas las que encuentre el modelo. En esta situación se corre el riesgo de que se favorezcan las clases que al modelo le resulten más fáciles de reconocer y acaben apareciendo con una frecuencia superior a la real. Para mitigar este efecto, lo ideal es que el modelo detecte todos los objetos de las muestras y les asigne una todas las clases posibles, aunque sea con un nivel de certidumbre bajo. En otras palabras, maximizar el recubrimiento e intentar lograr la mejor precisión en ese contexto.

En nuestro modelo actual no se incluye todavía todas las clases posibles, por lo que hay muchos elementos que no puede clasificar correctamente en ningún caso y no tiene sentido reducir al mínimo el nivel de confianza umbral, pero sí que puede ser interesante reducirlo a un nivel que mejore más el recubrimiento sin comprometer demasiado la precisión. En la matriz de confusión de la Tabla 7 se ha reducido el nivel de confianza umbral del 50% utilizado hasta ahora a 30%.

Con esta configuración el recubrimiento promedio del modelo llega a 87.4% (+3.3) a cambio de que la precisión promedio se reduzca a 65.1% (-1.3). Esta versión del modelo, con un valor de confianza umbral fijado en 30%, será el que se utilizará en las siguientes pruebas de detección.

		Real												
		Fondo	Ol.	Cis.	Br.	Ca.	Cit.	Ec.	Eu.	He.	On.	Qu.	Ro.	Th.	**Total**
Detectado	Fondo		26 0.6%	24 0.6%	92 2.1%	13 0.3%	48 1.1%	38 0.9%	47 1.1%	6 0.1%	34 0.8%	23 0.5%	10 0.2%	8 0.2%	**369** **0.0%** **100.0%**
	Olea europaea	85 2.0%	526 12.0%		10 0.2%		4 0.1%				1 0.0%	1 0.0%			**627** **83.9%** **16.1%**
	Cistus sp.	22 0.5%		116 2.7%						1 0.0%					**139** **83.4%** **16.6%**
	Brasicáceas	250 5.7%	5 0.1%		656 15.0%										**911** **72.0%** **28.0%**
	Castanea sativa	64 1.5%				62 1.42%			1 0.0%						**127** **48.8%** **51.2%**
	Citrus sp.	57 1.3%			4 0.1%		216 4.9%			1 0.0%		1 0.0%			**279** **77.4%** **22.6%**
	Echium sp.	83 1.9%						209 4.8%							**292** **71.6%** **28.4%**
	Eucalyptus sp.	290 6.6%			1 0.0%		1 0.0%	2 0.0%	396 9.1%						**690** **57.4%** **42.6%**
	Helianthus annuus	20 0.5%								148 3.4%					**168** **88.1%** **11.9%**
	Onobrychis sp.	74 1.7%			1 0.0%						187 4.3%		2 0.0%		**264** **70.8%** **29.2%**
	Quercus sp.	83 1.9%					4 0.1%					97 2.2%			**184** **52.7%** **47.3%**
	Rosmarinus officinalis	45 1.0%											159 3.6%		**204** **77.9%** **22.1%**
	Thymus sp.	42 1.0%												73 1.7%	**115** **63.5%** **36.5%**
	Total	**1115** **0.0%** **100.0%**	**557** **94.4%** **5.6%**	**140** **82.9%** **17.1%**	**764** **85.9%** **14.1%**	**75** **82.7%** **17.3%**	**273** **79.1%** **20.9%**	**249** **83.9%** **16.1%**	**444** **89.2%** **10.8%**	**156** **94.9%** **5.1%**	**222** **84.2%** **15.8%**	**122** **79.5%** **20.5%**	**171** **93.0%** **7.0%**	**81** **90.1%** **9.9%**	**4369** **65.1%** **34.9%**

Tabla 7: Matriz de confusión de la red neuronal YOLO para 12 clases con un nivel de confianza umbral del 30%

6.1.4 Clasificación de mieles

El experimento de detección y clasificación de mieles consiste en utilizar la red neuronal entrenada para 12 clases, con una confianza umbral del 30%, para detectar y clasificar los pólenes presentes en cada una de las muestras que conforman el dataset de Polenet individualmente (41 muestras en total). La detección se realizará utilizando el script `detect.py` que proporciona la herramienta YOLOv5 y que se ha adaptado para estos experimentos:

```
$ python detect.py --weights weights/polenet.pt --conf-thres 0.3
  --device 0 --data polenet.yaml --nosave --source
  <directorio_muestra>
```

`<directorio_muestra>` es la carpeta que contiene todas las imágenes de la muestra. El comando se repetirá para cada una de las muestras empleando el directorio apropiado.

Las ejecuciones generarán un fichero `summary.txt` con el recuento de todas las clases detectadas.

Es importante resaltar que una parte importante de estas muestras ha sido utilizada en el entrenamiento de red, ya que en este punto no quedaban muchas imágenes de las muestras monoflorales que no se hayan utilizado en entrenamiento o validación. Es por ello que los resultados no son un reflejo real del nivel de desempeño del modelo, pero sí que puede servir como prueba de concepto y para hacerse una idea de cómo trabaja. En su lugar, se han combinado las imágenes disponibles que aún no se habían empleado en la muestra etiquetada como *New*. Aunque esta no es una muestra real, permitirá valorar la red neuronal al trabajar con imágenes nunca vistas. Además, al combinar imágenes de distintos tipos de mieles, la muestra resultante presenta más variedad en los pólenes, formando un patrón más complejo que resulta interesante para poner a prueba el modelo.

En la Figura 31 se hace una comparativa de la presencia de cada tipo de polen en cada una de las muestras del dataset, comparando los porcentajes según las etiquetas (*Ground Truth*) y los detectados por la red neuronal (*Detected*).

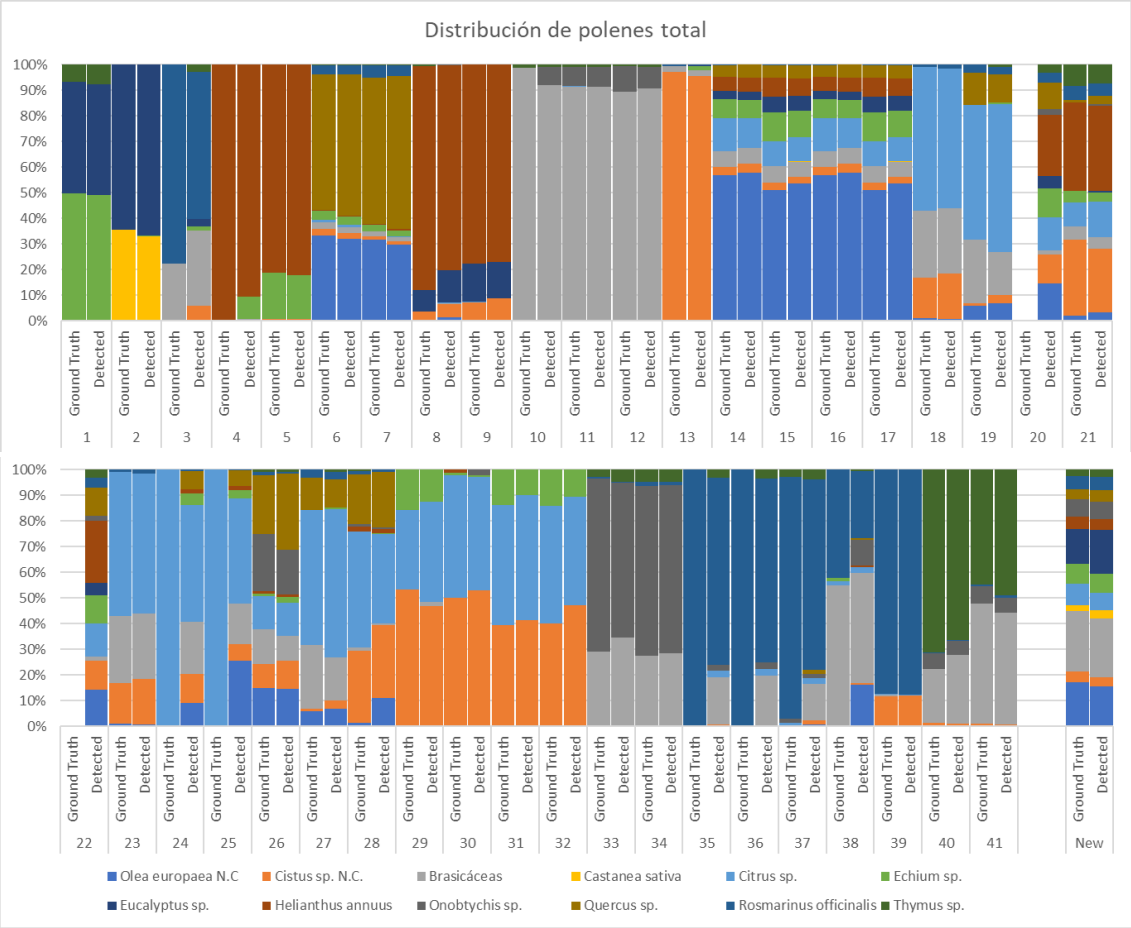

Figura 31: Distribución de los pólenes etiquetados y detectados. Los pólenes etiquetados (Ground Truth) y los detectados (Detected) se muestran lado a lado para cada muestra

Notar que las muestras 20 y 22 no tienen columna Ground Truth. Esto se debe a que el *dataset* de estas muestras no incluía etiquetas.

A la hora de clasificar las mieles, hay algunas partículas que no se consideran para el recuento de granos (No Contables). Entre las 12 clases que maneja nuestro modelo hay 2 que caen en esta categoría: *olea europaea* y *cistus sp*. En la Figura 32 se ha realizado la misma representación que en la Figura 31, pero eliminando estas partículas no contables. De este modo cada columna es la representación de la miel.

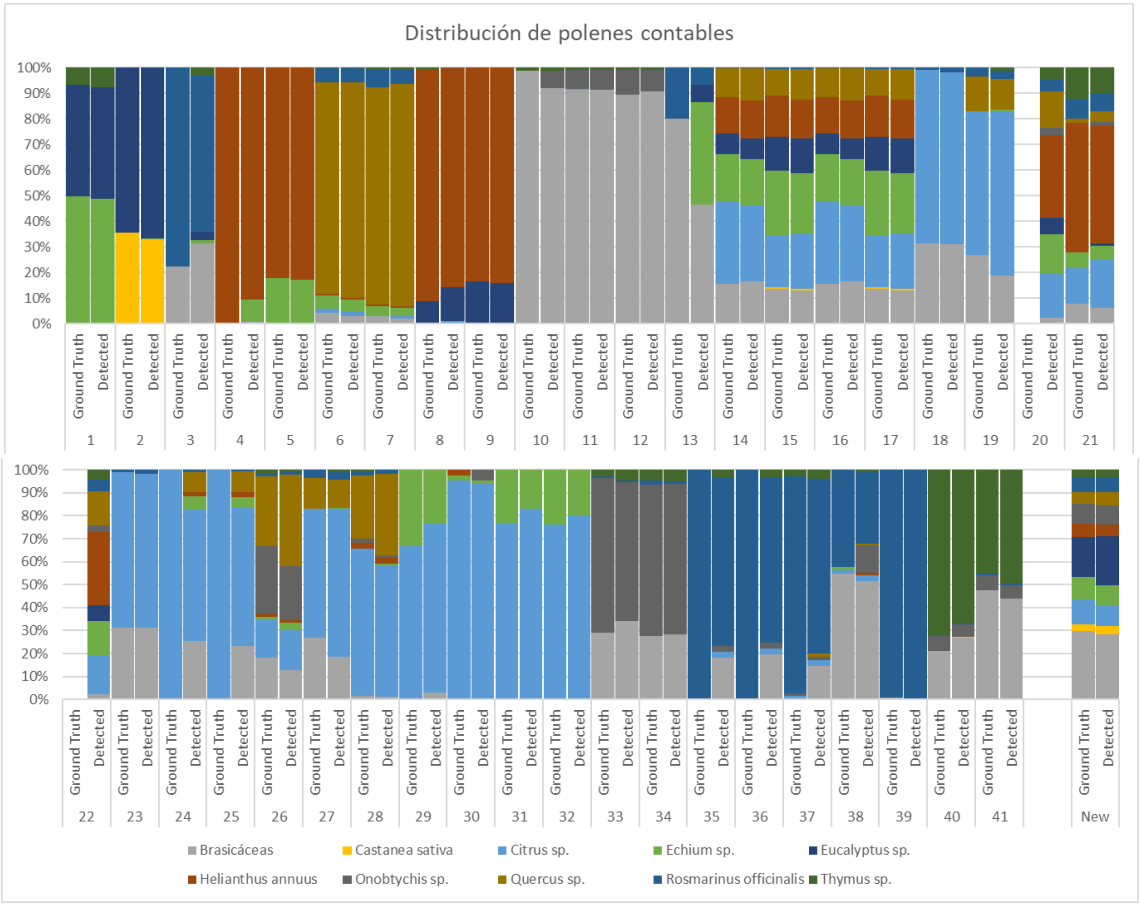

Figura 32: Distribución de los pólenes contables etiquetados y detectados. Los pólenes etiquetados (Ground Truth) y los detectados (Detected) se muestran lado a lado para cada muestra

Una vez obtenido el recuento se ha comparado la abundancia de cada polen para determinar el tipo de miel, siguiendo el criterio establecido para el dataset de Polenet (Tabla 8).

Miel	**Polen**	**Porcentaje mínimo**
Azahar	Citrus sp.	10%
Castaño	Castanea sativa	75%
Chupamieles	Echium sp.	45%
Esparceta	Onobrychis sp.	45%
Eucalipto	Eucalyptus sp.	75%
Girasol	Helianthus Annuus	45%
Romero	Rosmarinus Officinalis	10%
Tomillo	Thymus sp.	10%

Tabla 8: Presencia mínima de polen en la miel para la clasificación monofloral.

En la Tabla 9 se hace una comparativa entre la clasificación de las muestras realizada por un experto, la clasificación basada únicamente en polimetría en función de las clases etiquetadas en el *dataset* y la clasificación basada en polimetría automatizada empleando

el clasificador de YOLOv5 entrenado para 12 clases. La clasificación experta puede estar basada en más criterios que la polimetría, como puede ser el color o el sabor, entre otros. Po su parte, la clasificación por polimetría manual está basada únicamente en el recuento de pólenes, para una comparación más justa con la herramienta, con lo que puede discrepar de la asignación oficial del experto. Los casos en los que más de un tipo de miel cumpla el requisito mínimo de polen, se muestran todos los candidatos, ya que cualquiera de las clasificaciones se consideraría correcta.

Muestra	**Clasificación experta**	**Clasificación por polimetría manual**	**Clasificación por polimetría basada en YOLOv5**
1	Mil flores	Chupamieles	Chupamieles
2	Eucalipto	Eucalipto	Eucalipto
3	Romero	Romero	Romero
4	Girasol	Girasol	Girasol
5	Girasol	Girasol	Girasol
6	Romero	Mil flores	Mil flores
7	Romero	Mil flores	Mil flores
8	Eucalipto	Girasol	Girasol
9	Eucalipto	Girasol	Girasol
10	Mil flores/Colza	Colza	Colza
11	Mil flores/Colza	Colza	Colza
12	Mil flores/Colza	Colza	Colza
13	Cantueso	Colza/Romero	Colza
14	Azahar	Azahar	Azahar
15	Azahar	Azahar	Azahar
16	Azahar	Azahar	Azahar
17	Azahar	Azahar	Azahar
18	Azahar	Azahar	Azahar
19	Azahar	Azahar	Azahar
20	Tomillo	-	Azahar
21	Tomillo	Girasol/Tomillo	Azahar/Girasol
22	Tomillo	-	Azahar
23	Azahar	Azahar	Azahar
24	Azahar	Azahar	Azahar
25	Azahar	Azahar	Azahar
26	Azahar	Azahar	Azahar
27	Azahar	Azahar	Azahar
28	Azahar	Azahar	Azahar
29	Azahar	Azahar	Azahar
30	Azahar	Azahar	Azahar
31	Azahar	Azahar	Azahar
32	Azahar	Azahar	Azahar
33	Mil flores	Esparceta	Esparceta
34	Mil flores	Esparceta	Esparceta
35	Romero	Romero	Romero
36	Romero	Romero	Romero
37	Romero	Romero	Romero
38	Romero	Romero/Colza	Romero/Colza
39	Romero	Romero	Romero
40	Tomillo	Tomillo	Tomillo
41	Tomillo	Tomillo/Colza	Tomillo

Nueva muestra	-	Mil flores	Mil flores

Tabla 9: Clasificación de las muestras de mieles según el criterio de un experto, por polimetría manual o mediante polimetría basada en YOLOv5

La única discrepancia que se observa entre la clasificación por polimetría manual y mediante YOLOv5 es en la muestra 21. Esta muestra está compuesta de 168 imágenes, pero con solo 95 instancias etiquetadas en total. La densidad de etiquetado en el *dataset* es relativamente baja. Por su parte, el clasificador de YOLOv5 ha identificado un total de 248 instancias. Esta discrepancia, junto a lo cerca que está el recuento de los principales pólenes (*thymus sp.*, *helianthus annuus* y *citrus sp.*) de los umbrales mínimos de sus respectivas mieles (tomillo, girasol y azahar) en ambos casos han propiciado esta clasificación incorrecta. Este caso es un ejemplo de una muestra que requeriría introducir criterios adicionales al polimétrico para designar el origen monofloral.

Como nota adicional, la muestra 13 está clasificada como miel de cantueso, cuyo componente principal es el polen *lavandula stoechas*. Este polen no forma parte de los seleccionados para el experimento y no se incluye ni en la clasificación por polimetría manual ni en la basada en YOLOv5.

6.2 Mask R-CNN

El entorno de trabajo de Al igual que en YOLOv5, el entorno de trabajo de Mask R-cNN se encuentra dentro de un contenedor Docker dedicado. Los experimentos descritos en este apartado se han realizado dentro de este contenedor.

En todos los experimentos se va a emplear la misma arquitectura para la red neuronal: el mismo *backbone* y el mismo *head*, solamente modificando la capa de salida de acuerdo con el número de clases de cada experimento. Además, para cada experimento se van a entrenar la red desde cero y se va a utilizar la configuración de hiperparámetros que viene por defecto en la configuración de Mask R-CNN.

Cada experimento se va a repetir reentrenando la red utilizando distintas máscaras generadas automáticamente, como se menciona en el apartado 5.2, comparando el impacto de cada máscara y determinar cuál es la más adecuada.

Para el caso de Mask R-CNN no se va a realizar el experimento de 10 clases, ya que el mismo experimento en YOLOv5 ha confirmado que la candidad de muestras etiquetadas para la mayoría de las clases son insuficientes para entrenar la red y la volatilidad limita tanto el rendimiento de la red neuronal como las observaciones y conclusiones que se pueden extraer. En su lugar se va a realizar directamente el experimento con 12 clases.

6.2.1 3 Clases

A diferencia de YOLOv5, Mask R-CNN no necesita preparar el entorno ni generar ficheros para trabajar con el *dataset* de Polenet, ya que el script desarrollado ya se encarga de cargar los datos directamente de los ficheros XML con la información de las etiquetas. Se procede directamente a entrenar la red neuronal.

Se ha tenido en consideración que Mask R-CNN tiene que extraer información no solo de los *bounding box*, sino también de las máscaras. Dado que éstas no son perfectamente ajustadas, sino que se han generado automáticamente con formas geométricas, y el *dataset* de este experimento no es muy grande, se ha optado por emplear el 90% del *dataset* en el entrenamiento para maximizar las posibilidades de una buena generalización, dejando solo un 10% para la validación.

Entrenamiento

El entrenamiento se inicia utilizando el script que hemos desarrollado para Mask R-CNN: `polenet.py`. Para esta prueba inicial se utiliza el número de iteraciones recomendada por defecto para Mask R-CNN (30):

```
$ python polenet.py train -d "../Datasets/22009 Azh.xml"
  "../Datasets/Azahar 21334.xml" "../Datasets/Azahar_21181.xml"
  -C 7 12 17 -S <mascara> --logs ../resultados -r 0.9 -e 30;
```

El comando se repite modificando el parámetro `<mascara>` para cada una de las máscaras preparadas: `square` para la máscara cuadrada, `circle` para la máscara circular, `cir_oct` para la máscara octogonal circunscrita e `ins_oct` para la máscara octogonal inscrita.

Validación

La validación también se realiza mediante el script `polenet.py`, modificando los parámetros de llamada:

```
$ python polenet.py val -d "../Datasets/22009 Azh.xml"
  "../Datasets/Azahar 21334.xml" "../Datasets/Azahar_21181.xml"
  -C 7 12 17 -w <pesos> -r 0.9 -o ../resultados --logs
  ../resultados
```

Se repite el comando con los distintos ficheros de pesos `<pesos>`, cada uno correspondiente a una máscara.

Debido a la decisión de emplear un mayor porcentaje del *dataset* en el entrenamiento, no quedan muchas muestras para la validación. Los resultados obtenidos son más volátiles, pero aún se pueden extraer conclusiones interesantes para este primer experimento en cuanto al desempeño de la red.

Mask R-CNN, a diferencia de YOLOv5, no genera gráficos o métricas por sí mismo. En su lugar, se guardan los resultados y el registro de la ejecución, mediante los cuales se genera la matriz de confusión con la herramienta `confmat.py` integrada en `polenet.py`. De este modo, se genera una matriz de confusión para cada una de las máscaras (Tablas 10, 12, 13 y 14).

Square

La red neuronal basada en una máscara cuadrada, en adelante *Square*, es por sus características la más adecuada para comparar el desempeño de una red basada en Mask R-CNN con YOLOv5. La máscara cuadrada recubre la misma área que el bounding box, por lo que no aporta información adicional y viene a ser el modelo equivalente a Faster R-CNN, es decir, Mask R-CNN sin máscara.

		Real				
		Fondo	Olea europaea	Brasicáceas	Citrus sp.	**Total**
Detectado	Fondo		2 1.6%	4 3.2%	5 4.1%	**11** **0.0%** **100.0%**
	Olea europaea	17 13.8%	59 48.0%		1 0.8%	77 **76.6%** **23.4%**
	Brasicáceas	1 0.8%		2 1.6%		**3** **66.7%** **33.3%**
	Citrus sp.	16 13.0%			16 13.0%	**32** **50.0%** **50.0%**
	Total	**34** **0.0%** **100.0%**	**61** **96.7%** **3.28%**	**6** **33.3%** **66.7%**	**22** **72.7%** **27.3%**	**123** **62.6%** **37.4%**

Tabla 10: Matriz de confusión de la red neuronal Mask R-CNN para 3 clases (máscara cuadrada)

El recubrimiento de las clases oscila entre el 33.3% y el 96.7%, con un recubrimiento general del 86.5%. Por su parte la precisión en cada clase va del 50% hasta el 76.6%, con una precisión promedio del 62.6%. Como cabía esperar, hay mucha variabilidad en los resultados. Las grandes diferencias entre clases se pueden justificar en la diferencia de instancias de cada una, con el caso extremo de la clase *brasicáceas*, que solo tiene 6 instancias etiquetadas entre todas las imágenes de validación disponibles.

Precisamente esta clase es la que presenta peor recubrimiento con un 33.3%. Sin embargo, en valor absoluto son solo 4 instancias las que no se han detectado. Puede deberse simplemente a "mala suerte" y no ser una representación real de la capacidad de recubrimiento de la red neuronal para esta clase. Esto se comprobará más adelante con un *dataset* mayor.

Comparando los resultados con los del experimento de YOLOv5 para 3 clases, Mask R-CNN muestra una capacidad de recubrimiento mayor, con un promedio de 86.5% frente al 78.3% de YOLOv5. Por el contrario, YOLOv5 presenta una mejor precisión: 70.6% contra 62.6%.

Cabe destacar, como ya se ha mencionado previamente en el apartado de YOLOv5, que la precisión puede ser engañosa, debido a la cantidad de instancias no etiquetadas del *dataset*. Una forma de cuantificar cuánto del error de precisión se puede atribuir a etiquetado incompleto se puede estudiar la cantidad de falsos positivos etiquetados como fondo (no etiquetados) y los falsos positivos etiquetados como otras clases:

	Mask R-CNN	**YOLOv5**
FP de fondo	34 (27.6%)	25 (9.8%)
FP de clase	1 (0.8%)	4 (1.6%)
FN	11 (8.9)	46 (18.0%)

Tabla 11: Errores de detección para 3 clases en Mask R-CNN square *y YOLOv5*

Como se puede observar en la Tabla 11, del 37.4% de detecciones incorrectas que Mask R-CNN 12 (9.8%) son errores seguros, pero 34 (27.6%) no están garantizados y podrían

tratarse de detecciones correctas, pero no etiquetadas en el *dataset*. En comparación, YOLOv5 tiene garantizadas 50 detecciones incorrectas (19.6%), mientras que 25 (9.8%) podrían ser correctas no etiquetadas.

En conclusión, Mask R-CNN *Square* presenta un mejor recubrimiento que YOLOv5 y, aun cuando la precisión en principio es peor, podría acabar siendo mejor con las muestras del *dataset* completamente etiquetadas. Lo que sí es seguro es que la variabilidad para Mask R-CNN es mayor que en YOLOv5, así que estas primeras observaciones no se podrán confirmar hasta que se realicen con *dataset* más grandes.

CirOct

La red neuronal basada en una máscara octogonal circunscrita, en adelante *CirOct*, emplea una máscara octogonal con cuatro de los lados tangentes al cuadrado del *bounding box*, tal y como se ha ilustrado en la Figura 19.

El recubrimiento promedio de la red es del 89.9%, con un rango para las clases de entre el 72.7% y el 96.7%. En cuanto a la precisión, alcanza un promedio del 64.5%, con valores de entre 47.1% y 83.3% para las clases.

		Real				
		Fondo	Olea europaea	Brasicáceas	Citrus sp.	**Total**
Detectado	Fondo		2 1.6%	1 0.8	6 4.8%	**9** **0.0%** **100.0%**
	Olea europaea	16 12.9%	59 47.6%			**75** **78.7%** **21.3%**
	Brasicáceas	1 0.8%		5 4.0%		**6** **83.3%** **16.7%**
	Citrus sp.	18 14.5%			16 12.9%	**34** **47.1%** **52.9%**
	Total	**35** **0.0%** **100.0%**	**61** **96.7%** **3.3%**	**6** **83.3%** **16.7%**	**22** **72.7%** **27.3%**	**137** **64.5%** **35.5%**

Tabla 12: Matriz de confusión de la red neuronal Mask R-CNN para 3 clases (máscara octagonal circunscrita)

Es interesante notar también que, a diferencia de con la máscara cuadrada, en este modelo no se clasifica incorrectamente ninguna partícula de polen. Todos los errores vienen bien de no detectar una instancia presente en el *dataset* o de detectar una partícula no etiquetada en él.

Partiendo de que las pocas muestras disponibles implican cierto nivel de volatilidad e incertidumbre, la red *CirOct* muestra una mejoría tanto en el recubrimiento como en la precisión en relación con la red *Square*. Esto es un reflejo de lo que cabe esperar de Mask R-CNN. La máscara octogonal se aproxima más a la silueta auténtica de las partículas de polen, que son casi circulares en las 3 clases de este experimento. Mask R-CNN trabaja mucho mejor cuando puede sacar provecho de la información de las máscaras.

InsOct

La red neuronal basada en una máscara octogonal inscrita, en adelante *InsOct*, emplea una máscara octogonal con cuatro de los vértices en el centro de los lados del *bounding box*, como se muestra en la Figura 20.

La red alcanza un recubrimiento promedio del 91.0% y una precisión promedio del 65.3%, superando a las redes *Square* y *CirOct* en ambas métricas. Además, al igual que en la red *CirOct*, no se producen errores de clasificación entre clases. Todos los errores son detectando fondo como una partícula o no detectando partículas presentes en el *dataset*.

		Real				
		Fondo	Olea europaea	Brasicáceas	Citrus sp.	**Total**
Detectado	Fondo		2 1.6%	2 1.6	4 3.2%	**8** **0.0%** **100.0%**
	Olea europaea	19 15.3%	59 47.6%			**78** **75.6%** **24.4%**
	Brasicáceas			4 4.0%		**4** **100.0%** **0.0%**
	Citrus sp.	16 12.9%			18 14.5%	**34** **52.9%** **47.1%**
	Total	**35** **0.0%** **100.0%**	**61** **96.7%** **3.3%**	**6** **66.7%** **33.3%**	**22** **81.8%** **18.2%**	**137** **65.3%** **34.7%**

Tabla 13: Matriz de confusión de la red neuronal Mask R-CNN para 3 clases (máscara octagonal inscrita)

Analizando por clases, el recubrimiento de la red *InsOct* toma valores de entre el 66.7% y el 96.7%, donde el valor más bajo se trata de la clase *brasicáceas* que, como ya se ha mencionado previamente, solo cuenta con 6 muestras y los resultados son susceptibles a la volatilidad. La precisión por clases va desde el 52.9% al 100%. De nuevo, el 100% se da para las *brasicáceas*, por lo que no es tan significativo. Lo que sí es digno de mención es que se mejora la precisión de la clase *citrus sp.* respecto a los otros modelos, superando por primera vez el 50%.

La máscara octogonal circunscrita es la que más reduce el tamaño del *bounding box*, buscando ajustarse más a la forma de la partícula de polen. Para las clases en las que se está trabajando en este experimento parece que es la aproximación que mejor funciona. Es razonable que sea así, ya que todas partículas que se emplean tienen una forma casi circular. En futuros experimentos se verá si la tendencia se confirma al introducir más clases que puedan tener formas no tan circulares.

Circle

La red neuronal basada en una máscara circular, en adelante *Circle*, emplea una aproximación algo distinta dentro de Mask R-CNN, al no tratarse de un polígono. Intuitivamente se espera que sea la que presente mejores resultados, ya que la mayoría de las partículas de polen tienen forma quasi-circular y las máscaras funcionan mejor cuanto más se ajusten a la forma del objeto. Sin embargo, los resultados de la validación muestran el efecto contrario. El recubrimiento es ahora solo del 68.5% y la precisión cae hasta el 44.5%.

		Real				
		Fondo	Olea europaea	Brasicáceas	Citrus sp.	**Total**
Detectado	Fondo		19 13.9%	1 0.7%	2 1.5%	**22** **0.0%** **100.0%**
	Olea europaea	6 4.4%	36 26.3%			**42** **85.7%** **14.3%**
	Brasicáceas	1 0.7%		5 3.6%		**6** **83.3%** **16.7%**
	Citrus sp.	41 29.9%	6 4.3%		20 14.6%	**67** **29.8%** **70.2%**
	Total	**48** **0.0%** **100.0%**	**61** **59.0%** **41.0%**	**6** **83.3%** **16.7%**	**22** **90.9%** **9.1%**	**137** **44.5%** **55.5%**

Tabla 14: Matriz de confusión de la red neuronal Mask R-CNN para 3 clases (máscara circular)

Resulta difícil justificar el pobre desempeño de la red neuronal para esta máscara. La superficie que cubre la máscara circular es un punto intermedio entre la que cubre la máscara octogonal circunscrita y la octogonal inscrita, y la forma es relativamente similar. Sin embargo, la red neuronal *Circle* muestra unos resultados significativamente peores que cualquiera de las otras dos.

Analizando las detecciones realizadas (Figura 33) se ha comprobado que, mientras que con las otras redes neuronales la máscara detectada sí que mantiene más o menos la forma con la que se ha entrenado, en la red *Circle* no se conserva nada parecido.

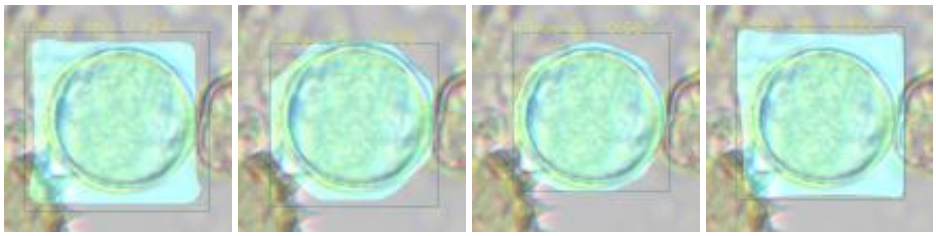

Figura 33: Detecciones con Mask R-CNN utilizando distintas máscaras. De izquierda a derecha: cuadrada, octogonal circunscrita, octogonal inscrita y circular

Se ha revisado el código, pero no parece haber ningún problema en la parte desarrollada para este proyecto. Parece deberse a que la implementación de Mask R-CNN de Matterport para Keras/TensorFlow, o quizás por incompatibilidad u obsolescencia de alguna librería, no soporta máscaras definidas como círculos o elipses. La solución de este problema requeriría entrar a modificar el código de las herramientas originales a un nivel más bajo del deseado para el alcance de este proyecto. En su lugar, para los siguientes experimentos se emplearán solo máscaras definidas por polígonos.

6.2.2 12 Clases

En el experimento de 12 clases para Mask R-CNN se ha empleado el *dataset* completo de Polenet. Dado que este *dataset* sí que contiene una cantidad significativa de imágenes y elementos etiquetados, se ha empleado el 80% del dataset para entrenamiento y el 20% para la validación, al igual que en YOLOv5.

A raíz de los resultados de 3 clases que parecen indicar que la máscara circular no se está aplicando bien, solo se realizará este experimento para las máscaras poligonales: cuadrada, octogonal circunscrita y octogonal inscrita.

Adicionalmente, en estos experimentos se van a introducir algunos parámetros de *data augmentation* simples, como rotaciones y otras transformaciones simples de las imágenes, reproduciendo los parámetros por defecto de *data augmentation* que tiene YOLOv5.

Entrenamiento

En Mask R-CNN es necesario ajustar el número de iteraciones de entrenamiento en función del número de imágenes disponibles en el dataset y el número de imágenes que se cargan en cada iteración. La documentación disponible no deja muy claro cómo estimar el número de iteraciones adecuadas al usar data augmentation. A diferencia de YOLOv5, Mask R-CNN no termina el entrenamiento de forma automática cuando detecta que no está habiendo mejoría, por lo que es importante encontrar un equilibrio adecuado entre el número de iteraciones y la mejoría de la red neuronal. Para ello, se ejecuta el entrenamiento para la máscara cuadrada con un límite de 1000 iteraciones. Basándose en los resultados del entrenamiento para 1000 iteraciones (Figura 34), se decide que 500 iteraciones son suficientes para obtener unos resultados prácticamente idénticos en un tiempo de ejecución mucho menor.

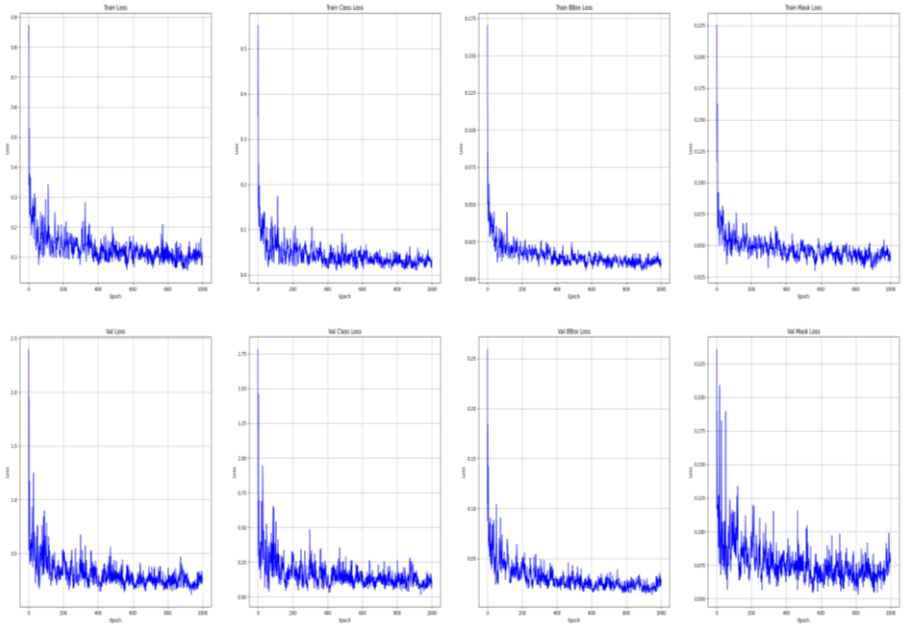

Figura 34: Métricas del proceso de entrenamiento de la red neuronal Mask R-CNN para 12 clases (máscara cuadrada)

Se repite el entrenamiento para las máscaras restantes (`cir_oct` e `ins_oct`) con 500 iteraciones:

```
$ python polenet.py train -d ../Datasets/dataset.txt -C 12 24 7
  33 21 17 37 40 43 26 8 14 -S <mascara> --logs ../resultados -r
  0.8 -e 500
```

Empleando como métrica principal la función de pérdida en la validación (`Val Loss`), tanto con la máscara cuadrada tras 500 y 1000 iteraciones, como con la máscara octogonal circunscrita (Figura 35) y la máscara octogonal inscrita (Figura 36) tras 500 iteraciones, el error ronda alrededor del 0.3, por lo que se esperan desempeños similares en todos los casos.

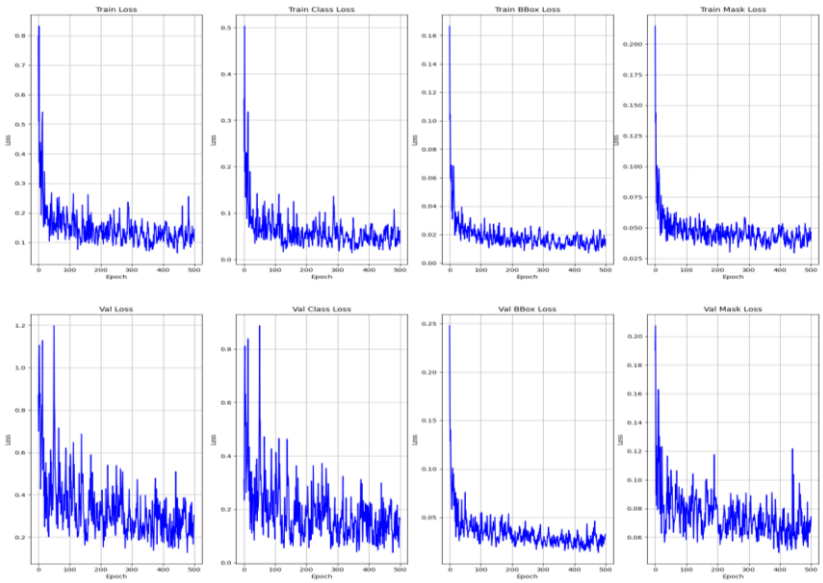

Figura 35: Métricas del proceso de entrenamiento de la red neuronal Mask R-CNN para 12 clases (máscara octagonal circunscrita)

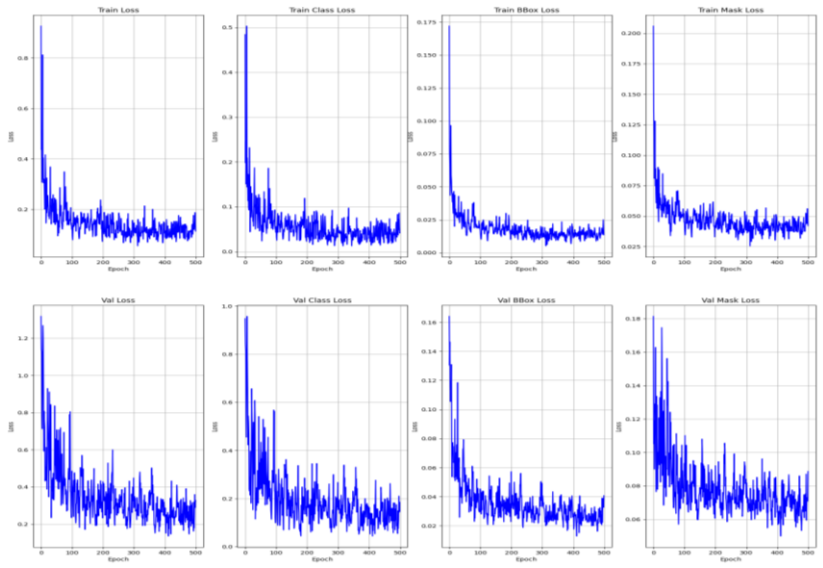

Figura 36: Métricas del proceso de entrenamiento de la red neuronal Mask R-CNN para 12 clases (máscara octagonal inscrita)

Validación

La validación también se realiza de nuevo mediante el script `polenet.py`:

```
$ python polenet.py val -d "../Datasets/dataset.txt" -C 12 24 7
  33 21 17 37 40 43 26 8 14 -w <pesos> -r 0.8 -o ../resultados -
  -logs ../resultados
```

A pesar de que se dispone de conjuntos de pesos más entrenados para la máscara cuadrada, se va a emplear siempre los pesos obtenidos tras 500 iteraciones de entrenamiento para comparar las distintas máscaras en igualdad de condiciones. El *dataset* de validación para estos experimentos contiene más de 1200 imágenes y más de 3100 elementos etiquetados, con un mínimo de 100 elementos en cada clase. La volatilidad de los resultados de estos experimentos es mucho menor que en el caso de las 3 clases, por lo que será posible realizar observaciones más concluyentes. Además, incluye partículas de polen con formas más variadas, lo que permitirá evaluar mejor el impacto de las distintas máscaras.

Square

La red neuronal *Square* presenta un recubrimiento promedio del 81.9% y una precisión promedio del 59.5%. Esto implica un deterioro en ambas métricas frente a los resultados para 3 clases (86.5% y 62.6%, respectivamente).

		Real													
		Fondo	Ol.	Cis.	Br.	Ca.	Cit.	Ec.	Eu.	He.	On.	Qu.	Ro.	Th.	**Total**
Detectado	Fondo		83 2.0%	18 0.4%	77 1.9%	22 0.5%	35 0.9%	13 0.3%	80 2.0%	5 0.1%	62 1.5%	109 2.7%	20 0.5%	26 0.6%	**548** **0.0%** **100.0%**
	Olea europaea	106 2.6%	257 6.3%												**363** **70.8%** **29.2%**
	Cistus sp.	37 0.9%		111 2.7%			1 0.0%								**149** **74.5%** **25.5%**
	Brasicáceas	367 9.0%	1 0.0%		683 16.8%		2 0.0%				4 0.1%				**1057** **64.6%** **35.4%**
	Castanea sativa	41 1.0%				80 2.0%									**121** **66.1%** **33.9%**
	Citrus sp.	24 0.6%	1 0.0%				121 3.0%							1 0.0%	**147** **82.3%** **17.7%**
	Echium sp.	112 2.8%						210 5.2%							**326** **64.4%** **35.6%**
	Eucalyptus sp.	155 3.8%							316 7.8%						**471** **67.1%** **32.9%**
	Helianthus annuus	26 0.6%								139 3.4%					**165** **84.2%** **15.8%**
	Onobrychis sp.	48 1.2%									172 4.2%		2 0.0%		**222** **77.5%** **22.5%**
	Quercus sp.	34 0.8%										66 1.6%			**100** **66.0%** **34.0%**
	Rosmarinus officinalis	42 1.0%											132 3.2%		**174** **75.9%** **24.1%**
	Thymus sp.	85 2.1%					2 0.0%				1 0.0%		1 0.0%	128 3.1%	**217** **59.0%** **41.0%**
	Total	**921** **0.0%** **100.0%**	**342** **75.1%** **24.9%**	**129** **86.0%** **14.0%**	**760** **89.9%** **10.1%**	**100** **80.0%** **20.0%**	**161** **75.2%** **24.8%**	**223** **94.2%** **5.8%**	**400** **79.0%** **21.0%**	**144** **96.5%** **3.5%**	**239** **72.0%** **28.0%**	**175** **37.7%** **62.3%**	**155** **85.2%** **14.8%**	**155** **82.6%** **17.4%**	**4060** **59.5%** **40.5%**

Tabla 15: Matriz de confusión de la red neuronal Mask R-CNN para 12 clases (máscara cuadrada)

Analizando por clases, la *olea europaea* es la más afectada, perdiendo un 21.6% en el recubrimiento y un 5.8% en la precisión. Sin embargo, la clase *brasicáceas* presenta una

gran mejoría en cuanto a recubrimiento (del 33.3% al 89.9%) y *citrus sp.* mejora también de forma significativa en la precisión (del 50.0% al 82.3%).

La clase con peor desempeño en términos de recubrimiento es *quercus sp.*, muy por debajo de los resultados de las otras clases con apenas un 37.7% de instancias detectadas. No se da casos de elementos clasificados incorrectamente, sino simplemente no detectados. Comparando distintas muestras de *quercus sp.* se puede observar que puede haber bastante variación dentro de la clase. Sin embargo, no está claro que esto sea suficiente para justificar el pobre desempeño de Mask R-CNN. YOLOv5 también presenta dificultades a la hora de reconocer partículas de *quercus sp.*, siendo ésta la clase con peor recubrimiento, pero alcanzando un recubrimiento del 72.1%.

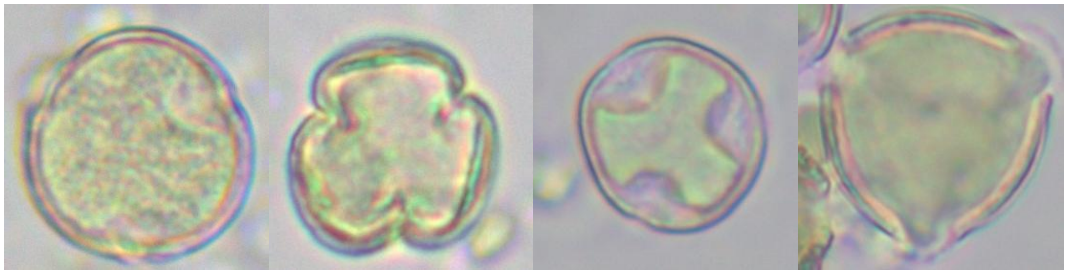

Figura 37: Distintas muestras de Quercus sp.

La peor precisión la presenta la clase *thymus sp.*, con apenas un 59.0%. Esto no se corresponde con los resultados de YOLOv5 y parece tratarse de una peculiaridad de Mask R-CNN. Más concretamente, parece una peculiaridad de la máscara cuadrada ya que, como se observará más adelante, no se mantiene la tendencia con las otras máscaras. Quizás se pueda justificar con la forma hexagonal que tienden a tomar las partículas (Figura 38).

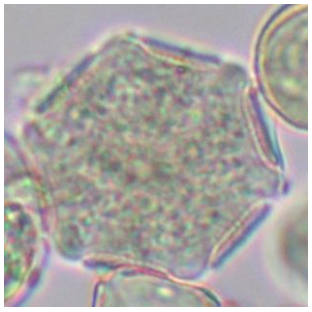

Figura 38: Muestra de Thymus sp.

En el lado opuesto, la mejor precisión (84.2%) y el mejor recubrimiento (96.5%) la presenta la clase *helianthus annuus*, reafirmando la tendencia mostrada en YOLOv5. Estos resultados confirman la clase como la más sencilla de detectar y clasificar para las redes neuronales, lo cual puede dar problemas, paradójicamente. Al ser significativamente más fácil de detectar y clasificar correctamente que las demás clases, puede resultar en una sobreestimación de la cantidad de partículas presentes en las muestras, en proporción con el resto. Sería interesante monitorizar la clase en este sentido en futuros experimentos, aunque eso queda fuera del alcance de este trabajo.

Repitiendo el análisis del error de precisión (Tabla 16), observamos que del 40.5% de detecciones incorrectas, 564 (13.9%) son errores seguros en base al GT, mientras que 921

(22.7%) podrían tratarse de detecciones correctas no etiquetadas. No obstante, a diferencia del experimento con 3 clases, Mask R-CNN supera a YOLOv5 en ambos campos, que solo presenta 512 (12.5%) errores garantizados y 861 (21%) posibles elementos no etiquetados.

	Mask R-CNN	**YOLOv5**
FP de fondo	921 (22.7%)	861 (21.0%)
FP de clase	16 (0.4%)	20 (0.5%)
FN	548 (13.5%)	492 (12.0%)

Tabla 16: Errores de detección para 12 clases en Mask R-CNN square *y YOLOv5*

En conjunto, parece que la red neuronal de YOLOv5 supera en rendimiento general a Mask R-CNN Square trabajando con 12 clases.

CirOct

La red neuronal *CirOct*, con la máscara octogonal circunscrita, presenta un recubrimiento promedio del 71.8% y una precisión promedio del 54.8%. El comportamiento de la red es peor que empleando una máscara cuadrada, en contraposición a los resultados obtenidos para 3 clases.

		Real													
		Fondo	Ol.	Cis.	Br.	Ca.	Cit.	Ec.	Eu.	He.	On.	Qu.	Ro.	Th.	**Total**
Detectado	Fondo		22 0.6%	25 0.6%	79 1.9%	31 0.8%	71 1.8%	47 1.2%	216 5.5%	19 0.5%	62 1.6%	98 2.5%	47 1.2%	102 2.61%	**816** **0.0%** **100.0%**
	Olea europaea	198 5.1%	320 8.2%%		2 0.1%	3 0.1%									**523** **61.2%** **38.8%**
	Cistus sp.	35 0.9%		104 2.7%											**139** **74.8%** **25.2%**
	Brasicáceas	389 10.0%			682 17.5%		2 0.1%				9 0.2%			1 0.0%	**1083** **63.0%** **37.0%**
	Castanea sativa	22 0.6%				69 1.8%									**91** **75.8%** **24.2%**
	Citrus sp.	16 0.4%					85 2.2%							1 0.0%	**102** **83.3%** **16.7%**
	Echium sp.	59 1.5%						176 4.5%	4 0.1%						**239** **73.6%** **26.4%**
	Eucalyptus sp.	30 0.8%							180 4.6%						**210** **85.7%** **14.3%**
	Helianthus annuus	24 0.6%								125 3.2%					**149** **83.9%** **16.1%**
	Onobrychis sp.	73 1.9%									168 4.3%		2 0.1%	2 0.1%	**245** **68.6%** **31.4%**
	Quercus sp.	56 1.4%										77 2.0%			**133** **57.9%** **42.1%**
	Rosmarinus officinalis	12 0.3%											106 2.7%		**118** **89.8%** **10.2%**
	Thymus sp.	7 0.2%												49 1.3%	**56** **87.5%** **12.5%**
	Total	**921** **0.0%** **100.0%**	**342** **93.6%** **6.4%**	**129** **80.6%** **19.4%**	**760** **89.7%** **10.3%**	**100** **69.0%** **31.0%**	**161** **52.8%** **47.2%**	**223** **78.9%** **21.1%**	**400** **45.0%** **55.0%**	**144** **86.8%** **13.2%**	**239** **70.3%** **12.7%**	**175** **44.0%** **56.0%**	**155** **68.4%** **31.6%**	**155** **31.6%** **68.4%**	**3904** **54.8%** **45.2%**

Tabla 17: Matriz de confusión de la red neuronal Mask R-CNN para 12 clases (máscara octagonal circunscrita)

Analizando por clases se puede observar que además de la clase *quercus sp.*, que ya presentaba un comportamiento pobre para la red *Square*, ahora también aparecen las clases *eucalyptus sp.* y *thymus sp.* con un recubrimiento por debajo del 50%. Sin embargo, estas dos últimas sí que presentan algunas de las precisiones más altas del modelo, indicando que a la red le cuesta detectar instancias de estas clases.

Es interesante el caso del *thymus sp.*, que en la red *Square* presentaba un recubrimiento alto pero la peor precisión del modelo, mientras que en la red *CirOct* tiene el recubrimiento más bajo, pero una de las precisiones más altas.

La red *CirOct* también presenta una varianza entre clases mayor que en la red *Square*, tanto en la precisión ([57.9%, 89.8%] frente a [59.0%, 84.2%]) como en recubrimiento ([31.6%, 93.6%] frente a [37.7%, 96.5%]). Es deseable minimizar la varianza entre clases para evitar favorecer unas clases frente a otras.

InsOct

La red neuronal *InsOct* que emplea la máscara octogonal inscrita presenta un recubrimiento promedio del 73.6% y una precisión promedio del 55.5%. Se mejora en ambas métricas a la red *CirOct*, pero aún sigue siendo superior la red *Square*.

		Real													
		Fondo	Ol.	Cis.	Br.	Ca.	Cit.	Ec.	Eu.	He.	On.	Qu.	Ro.	Th.	**Total**
Detectado	Fondo		88 2.1%	7 0.2%	113 2.9%	55 1.4%	39 1.0%	45 1.1%	135 3.4%	4 0.1%	65 1.6%	96 2.4%	36 0.9%	71 1.8%	**750** **0.0%** **100.0%**
	Olea europaea	85 2.1%	256 6.5%				1 0.0%								**342** **74.8%** **25.2%**
	Cistus sp.	67 1.7%		122 3.1%			2 0.1%								**191** **63.9%** **36.1%**
	Brasicáceas	387 9.8%			647 16.3%		6 2%				15 0.4%			3 0.1%	**1058** **61.1%** **38.9%**
	Castanea sativa	8 0.2%				45 1.1%									**53** **84.9%** **15.1%**
	Citrus sp.	54 1.4%	2 0.1%				113 2.9%							1 0.0%	**170** **66.5%** **33.5%**
	Echium sp.	73 1.8%						178 4.5%	4 0.1%						**255** **69.8%** **30.2%**
	Eucalyptus sp.	97 2.4%							261 6.6%						**358** **72.9%** **27.1%**
	Helianthus annuus	40 1.0%								140 3.5%					**180** **77.8%** **22.2%**
	Onobrychis sp.	55 1.4%									158 4.0%		1 0.0%		**214** **73.8%** **26.2%**
	Quercus sp.	74 1.9%										79 2.0%			**153** **51.6%** **48.4%**
	Rosmarinus officinalis	15 0.4%											116 2.9%		**131** **88.5%** **11.5%**
	Thymus sp.	18 0.5%									1 0.0%		2 0.1%	80 2.0%	**101** **79.2%** **20.79%**
	Total	**973** **0.0%** **100.0%**	**342** **74.8%** **25.2%**	**129** **94.6%** **5.4%**	**760** **85.1%** **14.9%**	**100** **45.0%** **55.0%**	**161** **70.2%** **29.8%**	**223** **79.8%** **20.2%**	**400** **65.2%** **34.8%**	**144** **97.2%** **2.8%**	**239** **66.1%** **33.9%**	**175** **45.1%** **54.9%**	**155** **74.8%** **25.2%**	**155** **51.6%** **48.4%**	**3956** **55.5%** **44.5%**

Tabla 18: Matriz de confusión de la red neuronal Mask R-CNN para 12 clases (máscara octagonal inscrita)

En el análisis por clases se observa que, al igual que en los otros modelos, la clase *quercus sp.* es la que presenta más dificultades, tanto en el recubrimiento como en la precisión. Las otras clases que presentan un recubrimiento especialmente bajo son *thymus sp.*, de forma similar a la red *CirOct*, y *castanea sativa*, que por el contrario presenta una de las precisiones más altas del modelo.

Los valores entre clases oscilan en el rango [45.0%, 97.2%] para el recubrimiento y [51.6%, 88.5%] para la precisión. Estos rangos suponen una variabilidad superior mayor a la de la red *Square* para la precisión, con valores en el rango [57.9%, 89.8%], pero menor en cuanto al recubrimiento, en el rango [37.7%, 96.5%].

A pesar de que el recubrimiento promedio de la red InsOct es menor que el de la red Square, la menor variabilidad en el recubrimiento puede resultar incluso más interesante para la clasificación de mieles. Esto se estudiará en el siguiente apartado.

6.2.3 Clasificación de mieles

El experimento de detección y clasificación de mieles consiste en utilizar la red neuronal entrenada para 12 clases, con una confianza umbral del 30%, para detectar y clasificar los pólenes presentes en una muestra con imágenes no utilizadas en el entrenamiento. El experimento se repite para cada una de las máscaras para comparar el desempeño de cada modelo. La muestra que se va a clasificar está generada artificialmente combinando imágenes de todas las muestras del dataset de Polenet, a fin de tener una muestra compleja y con una presencia significativa de todas las clases. La detección se realizará utilizando el script `polenet.py` con el comando `detect`:

```
$ python polenet.py detect -d "../Datasets/dataset.txt" -C 12 24 7 33
  21 17 37 40 43 26 8 14 -w <pesos> -r 0.8 -i
  <directorio_muestra> -o ../resultados --logs ../resultados
```

`<directorio_muestra>` es la carpeta que contiene todas las imágenes de la muestra. El comando se repetirá con los ficheros de pesos de los modelos entrenados con cada una de las máscaras. Las ejecuciones generarán un fichero `summary.txt` con el recuento de todas las clases detectadas.

La Figura 39 ilustra las proporciones de los pólenes detectados por cada uno de los modelos en comparación con el *Gound Truth*. Se ha empleado el error cuadrático medio de los ratios de cada clase para establecer y comparar la calidad de cada modelo, estableciendo como más preciso el modelo *Square* (RMS de 1.83%), seguido de InsOct (RMS de 2.53%) y de *CirOct* (RMS de 3.94%).

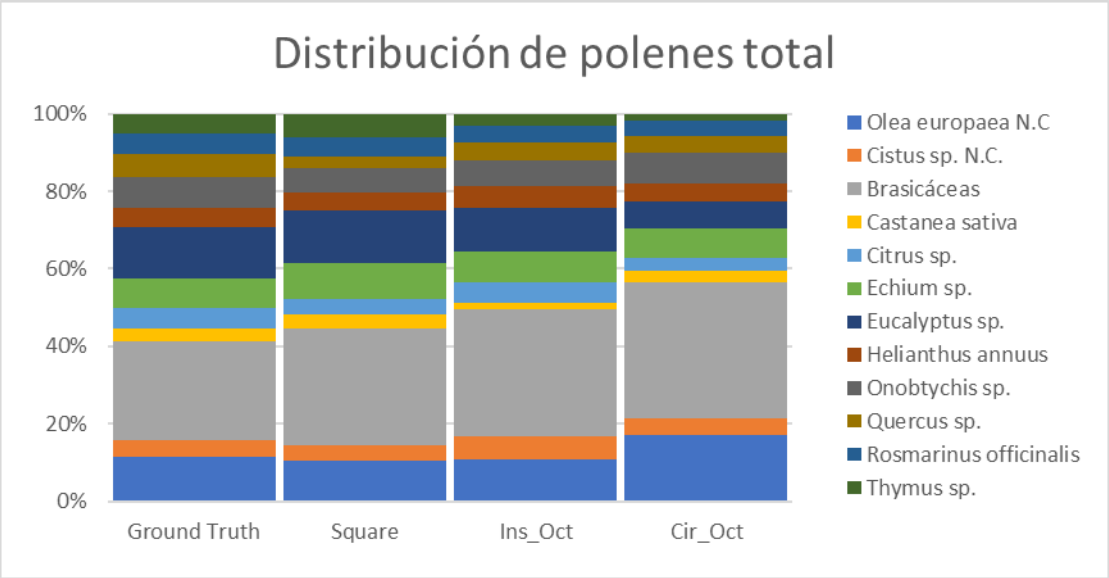

Figura 39: Distribución de los pólenes etiquetados y detectados por los modelos basados en distintas máscaras

La clase más sobreestimada para todos los modelos es *brasicáceas*, cuyo ratio ha aumentado un +4.62% en el modelo *Square*, hasta un +9.59% para el modelo *CirOct*. La clase más subestimada es *quercus sp.* en el modelo *Square,* con un -3.02%, mientras que para los modelos *InsOct* y *CirOct* es *eucalyptus sp.*, con un -2.24% y un -6.61% respectivamente.

En la Figura 40 se muestra la misma comparativa, pero empleando únicamente los pólenes contables, es decir, los que se tienen en cuenta a la hora de clasificar la miel. En otras palabras, cada columna es la representación visual de la miel de la muestra.

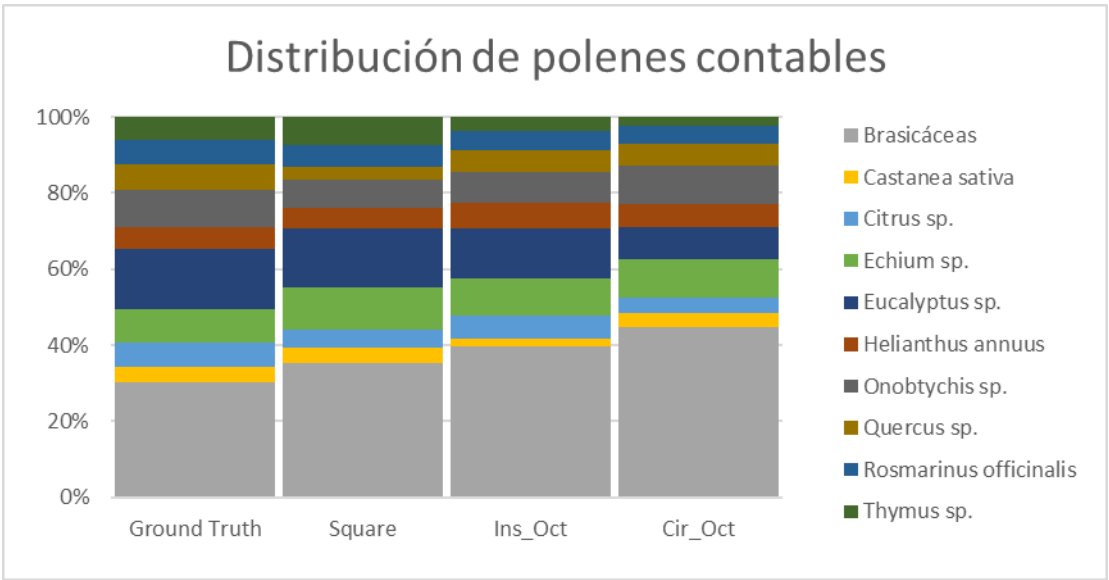

Figura 40: Distribución de los pólenes etiquetados y detectados por los modelos basados en distintas máscaras para pólenes contables

Realizando las mismas métricas basadas en los ratios de los pólenes las conclusiones no cambian mucho, siendo la red *Square* la que presenta un menor error cuadrático medio (RMS de 2.24%), seguido de *InsOct* (RMS de 3.32%) y de *CirOct* (RMS de 5.34%). Los

errores en los ratios de los pólenes oscilan en este caso en el rango [-3.63%, +4.98%] para *Square*, en el rango [-2.53%, +9.33%] para *InsOct* y entre [-7.27%, +14.39%] para *CirOct*.

Las métricas de este experimento señalan la máscara cuadrada como la más adecuada para clasificar los pólenes, dentro de las máscaras autogeneradas. También muestran la tendencia a la sobreestimación de las clases con alto recubrimiento, en la que destaca el caso de *brasicáceas*. Por el contrario, las clases con un recubrimiento bajo y alta precisión tienden a subestimarse.

7. Conclusiones

Los resultados recogidos en los apartados 6.1.4 para YOLOv5 y 6.2.3 para Mask R-CNN confirman que es posible emplear estos sistemas de redes convolucionales profundas de código abierto para la detección de pólenes y clasificación de mieles, con unos resultados razonablemente positivos. En ese sentido, todavía hay margen de mejora en el refinamiento de los modelos para ambas herramientas, como se comentará en el apartado 8.

Las herramientas desarrolladas incluyen las funcionalidades necesarias para entrenar y validar la red neuronal en ambos modelos (YOLOv5 y Mask R-CNN), así como realizar detecciones de muestras nuevas y generar información de los análisis, tal y como se ha realizado a lo largo de este trabajo, con lo que se puede considerar una base sólida sobre la que se pueden desarrollar futuras aplicaciones que mejoren el nivel de automatización en la clasificación de mieles por análisis polimétrico. Además, en ambos casos se han desarrollado empleando Dockers y el código fuente y la documentación está accesible en los respectivos repositorios, siguiendo los principios de desarrollo de código abierto. La instalación puede realizarse de forma sencilla y el mantenimiento y la actualización del código fuente puede realizarse de forma colaborativa y transparente.

Para comparar los modelos basados en YOLOv5 y en Mask R-CNN, se han examinado los ratios asignados a cada partícula de polen con las proporciones establecidas mediante detección manual en el *Ground Truth*.

Se han seleccionado los modelos de YOLOv5 con un valor de confianza umbral del 30% (YOLOv5@0.3) y del 50% (YOLOv5@0.5). Los modelos seleccionados de Mask R-CNN implementan cada uno una máscara distinta: Square (máscara cuadrada), InsOct (máscara octogonal inscrita) y CirOct (máscara octogonal circunscrita).

La Figura 41 compara lado a lado las proporciones detectadas por cada uno de los modelos (*Detected*) con los valores de sus respectivas referencias (*Ground Truth*). La Figura 42 hace la misma comparativa, pero excluyendo los pólenes que no se tienen en cuenta a la hora de establecer las proporciones y determinar el tipo de miel (pólenes no contables).

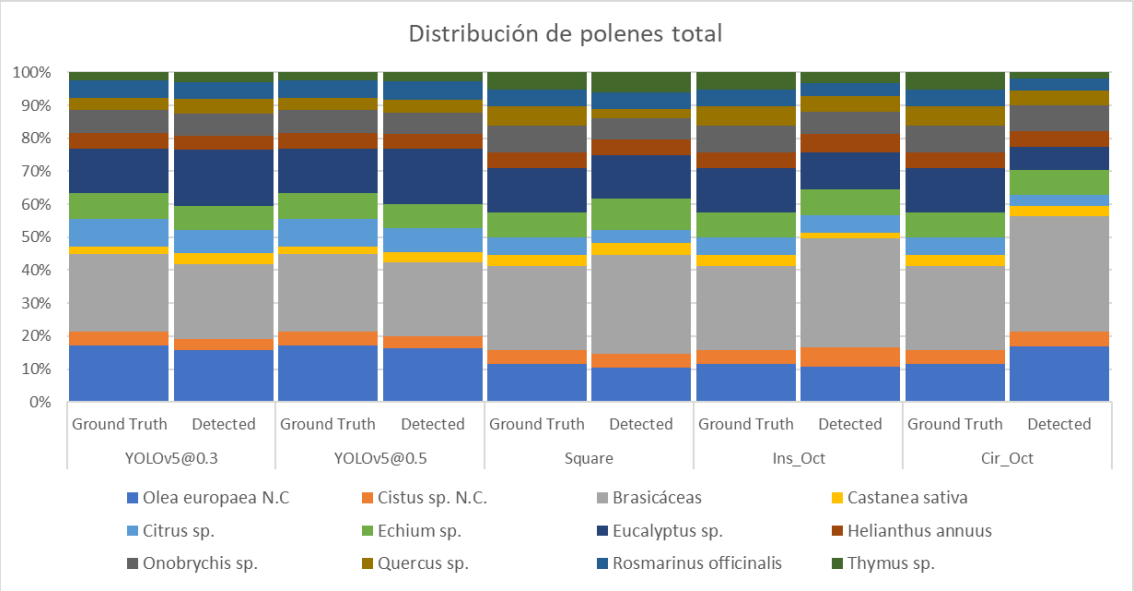

Figura 41: Comparativa de las distribuciones de pólenes detectados por los distintos modelos basados en YOLOv5 y Mask R-CNN

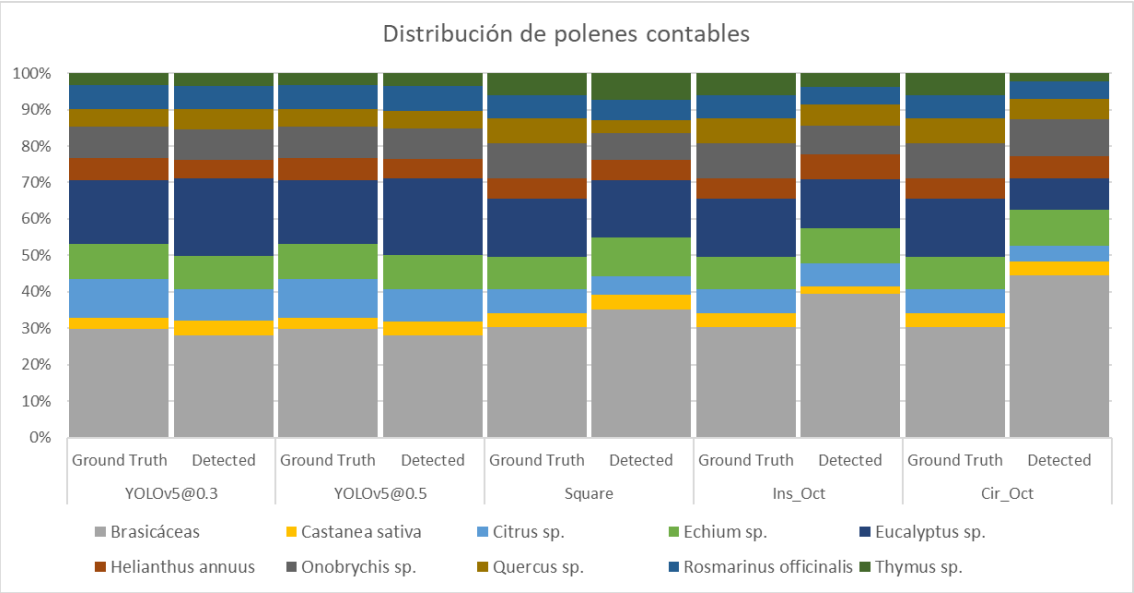

Figura 42: Comparativa de las distribuciones de pólenes contables detectados por los distintos modelos basados en YOLOv5 y Mask R-CNN

En la Tabla 19 se recoge el error en las proporciones relativas de cada una de las clases para cada uno de los modelos de red neuronal. En ella se puede observar la diferencia de comportamiento entre YOLOv5 y Mask R-CNN: mientras YOLOv5 tiende a sobreestimar la presencia de *eucalyptus sp.* y subestimar la clase *citrus sp.*, Mask R-CNN favorece la clase *brasicáceas* en detrimento de *quercus sp.* o *thymus sp.*

	YOLOv5		Mask R-CNN		
	@ 30%	**@ 50%**	**Square**	**InsOct**	**CirOct**
Olea europaea	-1.44%	-0.70%	-1.13%	-0.80%	+5.47%
Cistus sp.	-0.83%	-0.72%	-0.08%	+1.63%	+0.18%
Brasicáceas	-0.70%	-1.06%	+4.62%	+7.52%	+9.59%
Castanea sativa	+0.87%	+0.67%	+0.09%	-1.70%	-0.41%
Citrus sp.	-1.41%	-1.17%	-1.21%	-0.09%	-2.09%
Echium sp.	-0.35%	-0.18%	+1.81%	+0.48%	+0.26%
Eucalyptus sp.	+3.61%	+3.08%	±0.00%	-2.24%	-6.61%
Helianthus annuus	-0.59%	-0.38%	-0.13%	+0.79%	±0.00%
Onobrychis sp.	-0.22%	-0.24%	+1.69%	-1.34%	-0.08%
Quercus sp.	+0.85%	+0.20%	-3.02%	-1.09%	-1.56%
Rosmarinus officinalis	-0.16%	+0.14%	-0.24%	-1.11%	-1.37%
Thymus sp.	+0.39%	0.34%	+0.98%	-2.05%	-3.38%

Tabla 19: Desviación de las detecciones respecto al GT de cada modelo de red neuronal por clases.

A la hora de establecer cuál es el mejor modelo para la clasificación de mieles se ha recurrido al error cuadrático medio de las estimaciones de las clases, recogidas en la Tabla 20. La tabla también incluye el error mínimo y máximo de cada modelo.

	YOLOv5		Mask R-CNN		
	@ 30%	**@ 50%**	**Square**	**InsOct**	**CirOct**
Error mínimo	-1.44%	-1.17%	-3.02%	-2.24%	-6.61%
Error máximo	+3.61%	+3.08%	+4.62%	+7.52%	+9.59%
Error RMS	1.31%	1.07%	1.83%	2.53%	3.94%

Tabla 20: Error mínimo, error máximo y error cuadrático medio de cada modelo de red neuronal.

De acuerdo con las métricas recogidas, se puede concluir que YOLOv5 ofrece un rendimiento superior a Mask R-CNN a la hora de clasificar mieles, siendo la versión con un nivel de confianza umbral del 50% el mejor modelo. Dentro de Mask R-CNN, el mejor rendimiento lo ofrece la máscara cuadrada, dando a entender que introducir máscaras de forma artificial generándolas automáticamente no ayuda al modelo cuando hay más variedad de formas y no se ajustan bien.

8. Trabajos futuros

Aun cuando se han podido cumplir los objetivos base del proyecto y se ha podido profundizar en el comportamiento de ambas implementaciones de las *convnets*, incluso más de lo planeado originalmente, aún existen múltiples aspectos que se pueden mejorar. En concreto, hay dos ramas principales en las que se puede trabajar en un futuro inmediato en relación con este proyecto.

La primera rama está directamente relacionada con las herramientas. Aun cuando se han implementado todas las funcionalidades básicas para entrenar, validar y detectar, algunas funcionalidades adicionales de usabilidad ayudarían en el desarrollo de experimentos. Algunos ejemplos en ese sentido pueden ser la automatización del proceso de detección y clasificación de las mieles o, en el caso concreto de Mask R-CNN, mejorar el proceso de detección para permitir el análisis de imágenes de forma masiva (actualmente limitado a pequeños sets de imágenes).

También están actualmente limitadas las métricas generadas durante los procesos de entrenamiento y detección. YOLOv5 es el que ofrece más información actualmente, presentando gran variedad de métricas y permitiendo ampliarlas y mostrarlas en tiempo real y a través de la web mediante la integración con herramientas externas que ofrece, tal y como se menciona en el apartado 4.2. Por su lado, Mask R-CNN no ofrece de por sí gráficos y solo algunas métricas por consola. Aun cuando ya se han introducido algunos elementos como las funciones de pérdida y la matriz de confusión, sería deseable incorporar más elementos para equipararlo a YOLOv5.

Por último, en los experimentos realizados este trabajo no se ha entrado a personalizar los hiperparámetros y la configuración del *data augmentation*, que pueden ayudar a mejorar los resultados de las redes tanto para YOLOv5 como para Mask R-CNN. La gran variedad de opciones posibles hace difícil establecer la configuración óptima y se estima que el impacto no sería muy relevante, pero puede ganar especial relevancia al introducir clases con poca representación. En específico, YOLOv5 ofrece una utilidad basada en algoritmos genéticos que permite optimizar de forma automatizada los hiperparámetros, aunque el proceso es lento y computacionalmente costoso.

La segunda rama trata de los experimentos a realizar. En este trabajo se han incluido los más significativos de los realizados, pero aún quedan pendientes algunos que son de gran interés.

Un ejemplo es la clasificación de múltiples muestras de mieles con Mask R-CNN, comparando los resultados de la clasificación con la versión manual (como se ha hecho en YOLOv5 en el apartado 6.1.4). Este experimento no se ha realizado debido a las limitaciones técnicas que presenta Mask R-CNN a la hora de trabajar con imágenes masivamente en la detección, como se ha comentado previamente. E incluso sería necesario repetirlo para YOLOv5, con muestras completamente nuevas que reflejen realmente la calidad de la detección.

También está pendiente ampliar el número de clases dentro del modelo, hasta incluir todas las clases presentes en el *dataset* y que representan todos los elementos que se

pueden encontrar en una muestra de miel. Este sería el objetivo final de los modelos, pero para poder realizar esto y tener resultados razonables sería necesario ampliar el *dataset* para que los elementos con menos presencia aún tengan suficiente representación para que la red neuronal pueda aprender a identificarlos.

Otro experimento que sería interesante es emplear un *dataset* completamente etiquetado (i.e. con todas las partículas presentes asignadas a una clase) para entrenar las redes. De este modo se podría evaluar el impacto que tiene en el desempeño de la red el haber sido entrenada con falsos FP y/o falsos TN.

9. Referencias

Abdulla, W. (2017). Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow. (Gitlab, Ed.) *GitHub repository*. Obtenido de https://github.com/matterport/Mask_RCNN

Alexey, N., & Alois, K. (2013). Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics, 7*. doi:10.3389/fnbot.2013.00021

Breiman, L. (2001). Random Forests. *Machine Learning, 45*, 5-32. doi:10.1023/A:1010933404324

Chollet, F. (2018). *Deep Learning with Python*. New York, United States of America: Manning.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Mach Learn, 20*, 273-297. doi:10.1007/BF00994018

d. Geus, A. R., Barcelos, C. A., Batista, M. A., & d. Silva, S. F. (2019). Large-scale Pollen Recognition with Deep Learning. *2019 27th European Signal Processing Conference (EUSIPCO)*, (págs. 1-5). doi:10.23919/EUSIPCO.2019.8902735

Daood, A., Ribeiro, E., & Bush, M. (10 de Diciembre de 2016). Pollen Grain Recognition Using Deep Learning. *Advances in Visual Computing. ISVC 2016. Lecture Notes in Computer Science, 10072*. doi:10.1007/978-3-319-50835-1_30

Gallardo-Caballero, R., García-Orellana, C. J., García-Manso, A., González-Velasco, H. M., Tormo-Molina, R., & Macías-Macías, M. (2019). Precise Pollen Grain Detection in Bright Field Microscopy Using Deep Learning Techniques. *Sensors 2019*. doi:10.3390/s19163583

Girshick, R. (2015). Fast R-CNN. doi:10.48550/ARXIV.1504.08083

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (Junio de 2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 580-587.

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. doi:10.48550/ARXIV.1703.06870

Holt, K. A., & Bennett, K. D. (27 de Mayo de 2014). Principles and methods for automated palynology. *New Phytologist*, 735-742. doi:10.1111/nph.12848

Jocher, G., Chaurasia, A., Borovec, J., Nanocode012, Kwon, Y., TaoXie, . . . Minh, M. T. (22 de Febrero de 2022). *Ultralytics*. doi:10.5281/zenodo.6222936

Kaya, Y., Pinar, S. M., Erez, M. E., & Fidan, M. (15 de Febrero de 2013). An expert classification system of pollen of Onopordum using a rough set approach.

Review of Palaeobotany and Palynology, 189, 50-56. doi:10.1016/j.revpalbo.2012.11.004.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional. *Advances in Neural Information Processing Systems, 25*, 1097-1105.

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. doi:10.48550/ARXIV.1804.02767

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. doi:10.48550/ARXIV.1506.02640

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. doi:10.48550/ARXIV.1506.01497

Serrano, S., Villarejo, M., Espejo, R., & Jodral, M. (2004). Chemical and physical parameters of Andalusian honey: classification of Citrus and Eucalyptus honeys by discriminant analysis. *Food Chemistry, 87*(4), 619-625. doi:10.1016/j.foodchem.2004.01.031

Anexos

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				x
ODS 2. **Hambre cero.**		x		
ODS 3. **Salud y bienestar.**			x	
ODS 4. **Educación de calidad.**				x
ODS 5. **Igualdad de género.**				x
ODS 6. **Agua limpia y saneamiento.**				x
ODS 7. **Energía asequible y no contaminante.**				x
ODS 8. **Trabajo decente y crecimiento económico.**				x
ODS 9. **Industria, innovación e infraestructuras.**			x	
ODS 10. **Reducción de las desigualdades**.				x
ODS 11. **Ciudades y comunidades sostenibles**.				x
ODS 12. **Producción y consumo responsables.**				x
ODS 13. **Acción por el clima.**				x
ODS 14. **Vida submarina.**				x
ODS 15. **Vida de ecosistemas terrestres.**				x
ODS 16. **Paz, justicia e instituciones sólidas.**				x
ODS 17. **Alianzas para lograr objetivos.**				x

Este TFG está más estrechamente relacionado con el segundo objetivo de Desarrollo Sostenible, “Hambre cero”, en particular con la meta 2.a:

*“Aumentar las inversiones, incluso mediante una mayor cooperación internacional**, en la infraestructura rural, la investigación agrícola** y los servicios de extensión**, el desarrollo tecnológico** y los bancos de genes de plantas y ganado a fin de mejorar la capacidad de producción agrícola en los países en desarrollo, en particular en los países menos adelantados”*

Esto guarda relación al impacto esperado de este TFG, encaminado al desarrollo de la tecnología que facilite la clasificación automática de mieles monoflorales. Actualmente, la clasificación por métodos polínicos solo puede realizarse en centros acreditados, que deben disponer de profesionales especializados en esta laboriosa tarea, por lo que los análisis son costosos y poco accesibles. La implantación de estos sistemas automáticos facilitaría el acceso de los productores y comercializadores a estos servicios de certificación de la calidad de sus productos.

También se puede vincular levemente a los objetivos 3 “Salud y bienestar” y 9 “Industria, innovación e infraetructuras”, ya que el desarrollo de un clasificador de partículas de polen puede tener uso tanto en el control de alergias como en sistemas industriales de control de calidad de la miel, aunque los específicos de este proyecto no están enfocados en ese sentido.

Glosario

Bosques Aleatorios (*Random Forest*): Estructuras basadas en árboles de decisión que contienen un gran número de decisiones muy especializadas para posteriormente ensamblar los resultados.

Clasificador Bayesiano Ingenuo (*Naive Bayes*): Clasificador basado en el teorema de Bayes asumiendo que las características en los datos de entrada son independientes.

Cuadro delimitador (*bounding box*): Rectángulo mínimo que circunscribe la instancia de un objeto en una imagen.

Descenso de gradiente: Algoritmo que estima dónde una función genera sus valores más bajos (i.e.: mínimos locales) mediante aproximación numérica siguiendo la dirección del gradiente, es decir, la dirección en la que la función disminuye más rápidamente.

Falso Negativo (*False Negative*): Objeto que no ha sido detectado.

Falso Positivo (*False Positive*): Fondo detectado como un objeto, o un objeto clasificado de forma incorrecta.

Función de pérdida (*loss function*): Función diseñada para cuantificar la desviación de una estimación del valor real.

Intersección sobre Uniones (*Intersection over Union*): Ratio entre la intersección y la unión de dos áreas. El máximo (1) se da cuando las dos áreas coinciden perfectamente, mientras que el mínimo (0) se da cuando las áreas son disjuntas. En segmentación de instancias se emplea con el BB de la detección y el del GT para determinar la calidad de la detección.

Máquina de Soporte Vectorial (*Support Vector Machine*): Algoritmo de clasificación basado en la definición de la frontera entre las diferentes categorías. El algoritmo consiste en dos pasos: 1) Transformar el conjunto de datos en una representación en la que las fronteras entre las categorías sean fáciles de representar; 2) Una buena frontera que maximice la distancia a los datos de cada categoría.

Matriz de confusión (*confusion matrix*): También llamada matriz de error. Es una representación visual de las detecciones y clasificaciones realizadas por la red. Se representa como una matriz cuadrada de $C \times C$, siendo C el número de clases a posibles. En el caso de segmentación de instancias, se suele añadir una clase extra para representar el fondo. Las filas representan las clases detectadas por la red, mientras que las columnas hacen referencia a las clases en el GT. Para cada celda (i, j), se indica los objetos de la clase c_i que han sido clasificado como clase c_j.

Melisopalinología: Rama de la palinología que busca identificar los tipos de mieles y ubicarlos geográficamente en base a la presencia de granos de polen específicos.

Overfitting: Fenómeno que se da cuando el proceso de aprendizaje en IA se ajusta demasiado al conjunto de datos de entrenamiento, causando que el modelo no generalice apropiadamente para datos nuevos y, en última instancia, empeorando su desempeño.

Palinología: Disciplina de la botánica dedicada al estudio, principalmente morfológico, del polen y las esporas.

Precisión (*precision*): Ratio de objetos clasificados correctamente (verdaderos positivos) sobre todos los detectados:

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

Precisión Promedio (*Average Precision*): Área bajo la curva precisión-recubrimiento para un IoU umbral concreto. En este caso se emplea la notación $AP@IoU$ o AP_{IoU}. Por ejemplo, AP_{50} es el área bajo la curva precisión-recubrimiento aceptando como válidas detecciones con un IoU mayor o igual a 0,5.

También puede referirse al promedio de las áreas bajo la curva precisión-recubrimiento para un rango de IoUs. El caso más utilizado es el rango de 0,5 a 0,95, con incrementos de 0,05, que se denota como $AP@[0.5{:}\ 0.95]$ o $AP_{[0.5:0.95]}$.

Precisión Promedio media (mean Average Precision): Media del AP para todas las clases. En algunos contextos esto es lo que se considera la Precisión Promedio (AP), lo que puede llevar a confusión.

Propagación hacia atrás (*Backpropagation*): Algoritmo que calcula el gradiente de la función de pérdida respecto a los pesos de la red tras cada predicción. Este gradiente se emplea para actualizar los pesos de la red de forma que se minimice el error de forma continua y progresiva para el conjunto de entrenamiento.

Recubrimiento (*recall*): Ratio de objetos clasificados correctamente (verdaderos positivos) sobre todos los presentes en el *dataset*:

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

Regresión Logística (*Logistic Regression*): Algoritmo de clasificación basado en la predicción de una variable categórica en función de las variables independientes dadas.

Región de Interés (*Region of Interest*): Subsección de la imagen en la que se aplican las técnicas de detección de objetos, típicamente redes neuronales convolucionales, en problema de segmentación de instancias.

Underfitting: Fenómeno que se da cuando el modelo utilizado para aprendizaje automático no es capaz de capturar la tendencia subyacente de los datos. Se da al intentar utilizar un modelo demasiado simple a unos datos con un nivel de complejidad subyacente mayor. Por ejemplo, emplear un modelo lineal a unos datos polinómicos. El modelo resultante es incapaz de predecir apropiadamente nuevos datos.

Verdad básica (*ground truth*): Información que se asume como cierta, basada en observaciones directas. En *machine learning* es la información base que se utiliza para entrenar el sistema.

Verdadero Negativo (*True Negative*): Fondo que no ha sido clasificado incorrectamente como una clase. No se contabiliza en el caso de segmentación de instancias.

Verdadero Positivo (*True Positive*): Objeto detectado y clasificado de forma correcta.

Abreviaturas y acrónimos

AP	Precisión Promedio (*Average Precision*)
BB	Cuadro delimitador (*Bounding Box*)
Convnet	Red neuronal convolucional (convolutional neural network)
FN	Falso negativo (*False Negative*)
FP	Falso positivo (*False Positive*)
GPU	Tarjeta gráfica (*Graphic Processor Unit*)
GT	Verdad básica (*Ground Truth*)
IA	Inteligencia Artificial
IoU	Intersección sobre Unión (*Intersection over Union*)
JPG/JPEG	*Joint Photographic Experts Group*
mAP	Precisión Promedio media (*mean Average Precision*)
SVM	Máquina de Soporte Vectorial
PNG	Gráficos de Red Portable (*Portable Network Graphics*)
R-CNN	Regiones con características de Redes Neuronales Convolucionales (*Regions with Convolutional Neural Networks features*)
RMS	Media cuadrática (*Root Mean Square*)
ROI	Región de Interés (*Region of Interest*)
TFG	Trabajo de Fin de Grado
TN	Verdadero negativo (*True Negative*)
TP	Verdadero positivo (*True Positive*)
XML	Lenguaje de Marcado Extensible (*Extensible Markup Language*)
YOLO	Solo Miras Una Vez (*You Only Look Once*)