Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

ÁREA DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

TÉCNICAS DE PROCESAMIENTO DE IMAGEN PARA LA SEGMENTACIÓN Y LOCALIZACIÓN DE CÉLULAS A PARTIR DE VÍDEOS

D. FERNÁNDEZ CRUCHAGA, Miguel
TUTOR: D. DÍAZ BLANCO, Ignacio
COTUTOR: D. ENGUITA GONZÁLEZ, José María

Junio 2022

Agradecimientos

Me gustaría agradecer al Grupo de Supervisión y Diagnóstico de Procesos Industriales (GSDPI) del área de Ingeniería de Sistemas y Automática (ISA) de la universidad de Oviedo por proporcionarme un servidor para el entrenamiento del modelo realizado en este proyecto y al Instituto de Investigación Sanitaria del Principado de Asturias (ISPA) por la cesión de los videos de cultivos celulares utilizados en este proyecto. A todos los profesores que me han ayudado a formarme en esta etapa como estudiante de Ingeniería Electrónica Industrial y Automática en la Escuela Politécnica de Ingeniería de Gijón y en especial a mis tutores de este Trabajo Fin de Grado por su paciencia infinita y ayuda constante. Por último, a mi familia, que se ha esforzado por educarme y formarme como persona de la mejor forma posible.

Índice

Lista de Figuras

1. Introducción y objetivos:

1.1.- MOTIVACIÓN

El movimiento celular colectivo y coordinado es la base de procesos esenciales que ocurren en todos los organismos multicelulares. Hasta ahora ha sido difícil definir las pautas que rigen estos movimientos con suficiente claridad, mostrando eficazmente la dinámica de la interacción célula-célula de un modo comprensible para el investigador [1].

Por su parte, la migración celular colectiva ejerce una labor básica durante los procesos de morfogénesis, cicatrización de heridas y renovación de tejidos en el adulto, estando implicada también en la propagación del cáncer. Las células que migran de forma colectiva incrementan la eficacia de su movimiento y esto apunta a que se produce una interacción celular que genera un entorno óptimo para una migración dirigida eficiente [2].

Estos hechos plantean que el estudio del movimiento celular es una estrategia importante en la comprensión de los mecanismos implicados en el cáncer. Poder analizar de forma automática los movimientos y trayectorias de las células en cantidades masivas de datos podría ayudar a entender mejor como estos están relacionados con el desarrollo de los distintos tipos de tumores y con ello poder diagnosticar y tratar a cada paciente de una forma más precisa e individualizada.

En la última década, el desarrollo de técnicas de aprendizaje automático para el análisis de imagen, especialmente redes convolucionales profundas, ha sido sorprendente tanto en la clasificación de imágenes, como especialmente en las tareas de detección, segmentación y clasificación de objetos en imágenes y vídeos. Estos avances abren nuevas vías relacionadas con la aplicación de estas técnicas en el análisis del movimiento celular, siendo la principal motivación de este TFG.

1.2.- INTELIGENCIA ARTIFICIAL Y DETECCIÓN DE OBJETOS POR ORDENADOR

La inteligencia artificial es el presente y futuro de numerosos ámbitos, siendo incontables las aplicaciones que puede tener en ellos. Actualmente, está debajo de aplicaciones tan cotidianas como cualquier traductor lingüístico de internet, aplicaciones como Google Maps, el auto corrector del teclado de nuestro teléfono o la recomendación de vídeos que podemos ver en YouTube, y en otras no tan cotidianas, como en la sanidad, permitiendo a los médicos detectar enfermedades en pacientes cuando ni siquiera son visibles en radiografías por el ojo humano. En un futuro lo podremos ver por las calles en vehículos autónomos totalmente conectados, y en las ciudades adaptando semáforos y límites de velocidad para controlar el tráfico y reducir las emisiones contaminantes.

La inteligencia artificial, definida por John McCarthy, es la ciencia e ingeniería de hacer máquinas inteligentes, especialmente programas informáticos inteligentes [3]. Por su parte, el aprendizaje profundo o *deep learning* es una técnica específica dentro del abanico del aprendizaje automático, denominado comúnmente como *machine learning*, y comprende una serie de técnicas para el aprendizaje de modelos con sucesivas capas a partir de un conjunto de datos o *dataset* [4].

De esta forma, las redes neuronales profundas o *deep neural networks* están pensadas para copiar el funcionamiento del cerebro humano y, después de un entrenamiento, ser capaces de detectar los patrones de objetos aprendidos en nuevos datos. Son capaces de hacerlo gracias a que contienen un elevado número de capas que forman una jerarquía composicional capaz de realizar aprendizajes complejos [5].

Los datos con los que un modelo *deep learning* trabaja pueden ser imágenes, sonidos, textos o vídeos, entre otros. En problemas de clasificación, en general, es imprescindible que aquello que sea objeto de ser identificado deba ser primero localizado y etiquetado. Por ejemplo, si se quiere entrenar a un modelo d*eep learning* para detectar rosas en imágenes de campos de flores, se le debe proporcionar además de un conjunto de imágenes de tamaño suficiente, un archivo por imagen o conjunto de imágenes dónde se identifique mediante coordenadas la localización de dichas rosas [6].

La red profunda, a partir de las imágenes de entrada y las etiquetas proporcionadas, es capaz de sintonizar sus pesos, minimizando un error [ej. de clasificación y/o regresión] mediante el descenso del gradiente utilizando el algoritmo *backpropagation*, que permite el cómputo iterativo del gradiente de los pesos de la red [5]. En la última década han surgido *frameworks* específicos para *deep learning* como TensorFlow/Keras, Theano, PyTorch y otros muchos que permiten simplificar el diseño y entrenamiento de arquitecturas profundas, explotando la potencia de cálculo que ofrecen las tarjetas gráficas GPU. Esto ha llevado a una explosión de literatura y aplicaciones que han supuesto en la última década un avance sin precedentes, con aplicaciones en múltiples campos, entre los que se incluye la biomedicina.

1.3.- CÉLULAS SANAS Y CANCERÍGENAS

Una célula es la unidad biológica que puede vivir por sí sola y gracias a ella se forman los organismos vivos y tejidos del cuerpo [7]. Las células cancerosas son aquellas que tienen comportamientos anormales con respecto a las células sanas o normales. Las células corrientes se forman únicamente cuando reciben una señal para hacerlo y se dejan de multiplicar, o se destruyen, cuando se lo indica otra señal, mientras que una célula cancerosa no obedece esas señales e invade áreas cercanas y se disemina a otras áreas del cuerpo [8].

Hoy en día se realizan exámenes de células sospechosas de actividad tumoral, a partir de biopsias, para la detección de cáncer en individuos que no hayan presentado síntomas todavía. Estos varían en función del tipo de cáncer que se quiera analizar. Su tratamiento dependerá del tipo de cáncer y de lo avanzado que esté, pudiendo ser tratado con cirugía, radioterapia, quimioterapia o inmunoterapia.

1.4.- CULTIVOS CELULARES

Los cultivos celulares son las técnicas que permiten mantener un entorno controlado para el crecimiento “in vitro” de células [9]. Las células de los cultivos celulares que aparecen en los vídeos utilizados en este proyecto son todas ellas células derivadas del carcinoma epidermoides de cabeza y cuello humanos. Para este trabajo se han utilizado vídeos de cultivos con células SCC38, SCC40 y SCC42B. Las células SCC38 y SCC42B

son células derivadas de carcinomas epidermoides de laringe y las células SCC40 derivan de un carcinoma epidermoide de cavidad oral.

1.5.- DESCRIPCIÓN DE LOS CAPÍTULOS DE ESTA MEMORIA

La estructura de los capítulos siguientes de esta memoria es la siguiente. En el capítulo 2, se detalla el estado del arte de las técnicas de segmentación de células mediante ordenador y de las arquitecturas *deep learning*. En el capítulo 3, se presentan los pasos seguidos para la adecuación del modelo *deep learning* escogido al *dataset* de este trabajo. En el capítulo 4, se explican los resultados obtenidos tras los entrenamientos y se hace un análisis de las bondades y limitaciones del modelo creado. En el capítulo 5, se concluye esta memoria describiendo las contribuciones de este trabajo y cómo continuar a partir de él en el futuro. Por último, en el anexo (capítulo 6), se incluye una guía en detalle sobre la realización de algunas de las tareas de este proyecto, así como una estimación del coste del proyecto.

1.6.- MARCO DEL TRABAJO Y OBJETIVOS

El objetivo de este trabajo fin de grado es estudiar la viabilidad del uso de técnicas de aprendizaje profundo para la identificación de células individuales en vídeos de cultivos celulares. Es el paso previo necesario para poder después evaluar el *tracking* de las células con algoritmos diseñados para ello, identificando el comportamiento de las células cancerosas con respecto a las normales a partir de la forma en la que se mueven.

Proporcionadas las coordenadas de las células en los vídeos, se calcularían y analizarían posteriormente parámetros que permitan caracterizar y cuantificar su movimiento, tales como la distribución de velocidades, existencia de movimientos coordinados, giros, etc. Estos descriptores pueden alimentar a algoritmos de aprendizaje automático, incluido aprendizaje profundo, para clasificar el tipo de movimiento, por ejemplo, entre canceroso o normal, pudiendo incluso determinar diferencias cualitativas con potencial relevancia biomédica, como realizar un diagnóstico más personalizado del cáncer y determinar un tratamiento más individualizado para cada paciente.

2. Estado del arte

2.1.- TÉCNICAS DE SEGMENTACIÓN APLICADAS A LA DETECCIÓN DE CÉLULAS.

Al abordar un problema de detección de células en imágenes microscópicas lo primero que se debe tener en cuenta es que no hay un método único que pueda segmentar las células correctamente en la totalidad de los casos. El diferente conjunto de características que tiene cada tipo de imagen hoy no lo permite todavía. Dicha complejidad ya era señalada por Erik Meijering [10]: *La segmentación automatizada de imágenes para el análisis celular es generalmente un problema difícil debido a la gran variabilidad (diferentes microscopios, coloraciones, tipos de células, y densidades de célula) y a la complejidad de los datos (posiblemente time-lapse, adquiridos a diferentes longitudes de onda, utilizando múltiples microscopios, y conteniendo un gran número de células)*

El objetivo en la segmentación de células es, en cualquier caso, dividir una imagen entre el fondo y las células individuales. El método más básico para realizar esto se conoce por umbralización, denominado también *thresholding*, y es aquel que establece un umbral en el brillo de la imagen, asignando los píxeles que poseen valores por encima de este valor a una región y el resto a otra. Se trata, de esta forma, de encontrar el valor del brillo del borde de la célula considerando los valores por debajo de este como el fondo y por encima los píxeles que pertenecen a la célula. Este método, por sí solo, es válido únicamente cuando las células son homogéneas, están muy bien separadas entre sí, tienen un brillo similar y el ruido de la imagen es bajo [10].

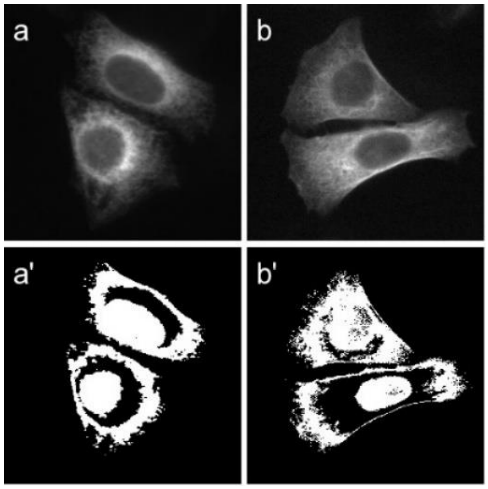

Figura 2.1.- Método de segmentación por Thresholding aplicado en imágenes de células [28].

El método de umbralización se puede complementar con medidas cuantitativas como el color, posición o textura para definir una regla más fiable al separar los píxeles entre una región u otra. Si el problema a resolver se vuelve más complejo y las células están en contacto, los métodos anteriores servirían para separar las regiones de células con respecto al fondo, pero no valdrían para separar las células entre sí.

Otras formas clásicas de abordar la segmentación de células son: la detección de características derivadas de la intensidad utilizando el filtrado lineal de imágenes, como por ejemplo el filtro laplaciano de Gauss [10]; el filtrado morfológico con filtros no lineales como los filtros de erosión o dilatación [10]; el enfoque de acumulación de regiones con algoritmos como *watershed*; o mediante procedimientos que ajustan modelos deformables a los datos de la imagen con algoritmos como *level-sets* [10].

Para poder segmentar en los casos en que las células están en contacto entre sí mediante enfoques clásicos se podría aplicar el algoritmo matemático denominado *watershed* a todos los píxeles de la imagen. Para entender este algoritmo se debería pensar en el brillo de la imagen y entenderlo como un paisaje con montañas, las células, y con valles, que representaría al fondo. De esta manera, el objetivo del algoritmo sería encontrar los picos de brillo de la imagen, suavizando primero la imagen como paso de preprocesamiento para tener solo un único píxel máximo por célula y entonces se podría aplicar el algoritmo de *watershed* pensándolo como agua fluyendo desde la cima de estas

montañas y asentándose en las regiones oscuras entre las células en contacto. Con ello se conseguiría separar correctamente las células en contacto [11].

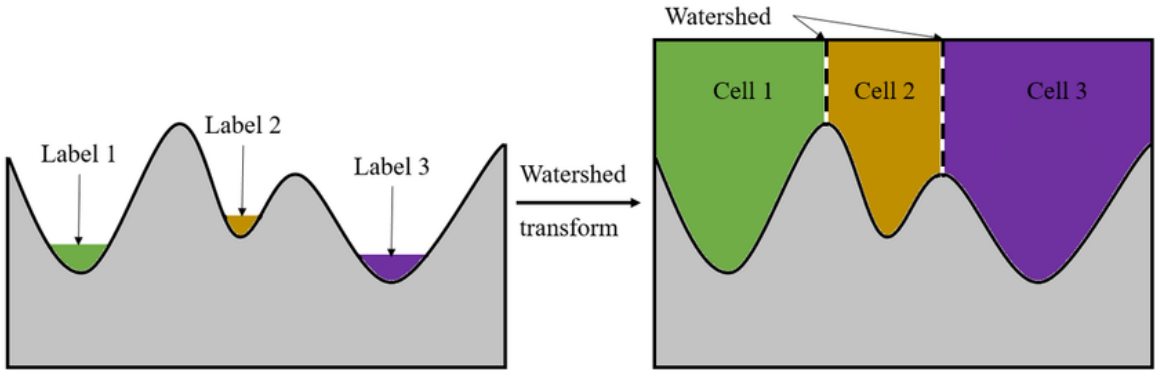

Figura 2.2.- Transformación mediante el algoritmo *watershed* [11].

El algoritmo *watershed* deja de ser adecuado cuando en las imágenes no se ha empleado el marcado fluorescente del núcleo y la intensidad de la imagen no refleja adecuadamente la cantidad de materia presente.

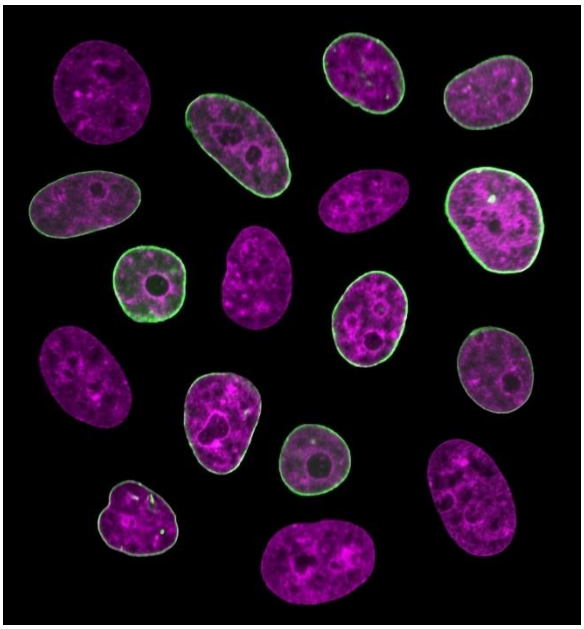

Figura 2.3.- Núcleos de células humanas con marcados fluorescentes [12].

En la actualidad, en estos casos en los que por la complejidad de la imagen los métodos clásicos no son suficientes por sí solos se puede optar por utilizar técnicas no supervisadas de *machine learning* como paso de preprocesamiento de la imagen. Estas técnicas engloban a todos los métodos de *machine learning* que trabajan con datos que no

han sido etiquetados previamente, como los algoritmos de agrupamiento, de detección de anomalías, reglas de asociación, inferencia probabilística o reducción de dimensionalidad como los *autoencoders* [13]. Como Yuanpi, Fuyong Xing y Xiaoshuang Shi relatan en [14]: *los métodos no supervisados se basan en la heurística y no pueden generalizarse bien a diferentes modalidades de imágenes de microscopía. Además, la falta de homogeneidad del fondo, la morfología irregular de las células y el hecho de que las células se toquen suponen un reto adicional para estos métodos no supervisados. Para abordar estos problemas, los métodos basados en el aprendizaje supervisado también han atraído una atención considerable debido a su prometedor rendimiento.*

Por otra parte, se puede optar por técnicas supervisadas de *machine learning* o por técnicas de *machine learning* de aprendizaje activo [13]. En las técnicas supervisadas se requiere un paso previo manual de etiquetado de células en las imágenes que forman el conjunto de entrenamiento o *training set*, mientras que en los sistemas de aprendizaje activo se requiere que el usuario, iterativamente, proporcione etiquetas cuando se le requiera. Para ello, por ejemplo, se le puede requerir que etiquete ejemplos de células en una parte de la imagen durante la etapa del entrenamiento y el método las predice para el resto de la imagen. Repitiendo esto iterativamente y corrigiendo los errores se entrenaría al modelo que, después, podría detectar automáticamente él solo, sin supervisión, grandes cantidades de imágenes. Debido al requerimiento de participar activamente durante el entrenamiento, se les llama también sistemas semiautomatizados. La mayor duración y el requerimiento de una persona en todo momento supervisando el entrenamiento en ella hace que estos métodos no sean tan atractivos para la segmentación de células como los métodos automatizados basados en técnicas supervisadas de *machine learning* [13].

Dentro del abanico de las técnicas supervisadas de *machine learning* destacan hoy en día las basadas en redes neuronales profundas o *deep neural networks*. Estas redes se componen de un número elevado de capas e intentan emular el funcionamiento de un cerebro humano. Necesitan de un gran número de imágenes en las que previamente se han anotado manualmente la localización de las regiones u objetos de interés para poder ser entrenadas. Reducir el número de imágenes previamente etiquetadas necesarias para entrenar un modelo *deep learning* es uno de los retos del *machine learning* en la actualidad. Uno de los enfoques para lograr esto es la técnica de *transfer learning*, que se basa en el aprovechamiento de lo

aprendido por un modelo para realizar el aprendizaje de otra red en un contexto relacionado, pero diferente, con mucho menos esfuerzo.

2.2.- DEEP LEARNING

Una red neuronal se compone de neuronas artificiales como elemento más básico de procesamiento de la red, organizándose estas a su vez en capas. Las neuronas se conectan entre sí mediante conexiones de fuerza variable denominadas pesos. Cuando una red neuronal analiza los registros individuales y realiza una predicción incorrecta esta ajusta los pesos, sucesivamente en el entrenamiento hasta realizar predicciones muy precisas. Llegado este momento, la red es capaz de predecir correctamente en casos en los que se desconoce el resultado y se ha finalizado el entrenamiento [15].

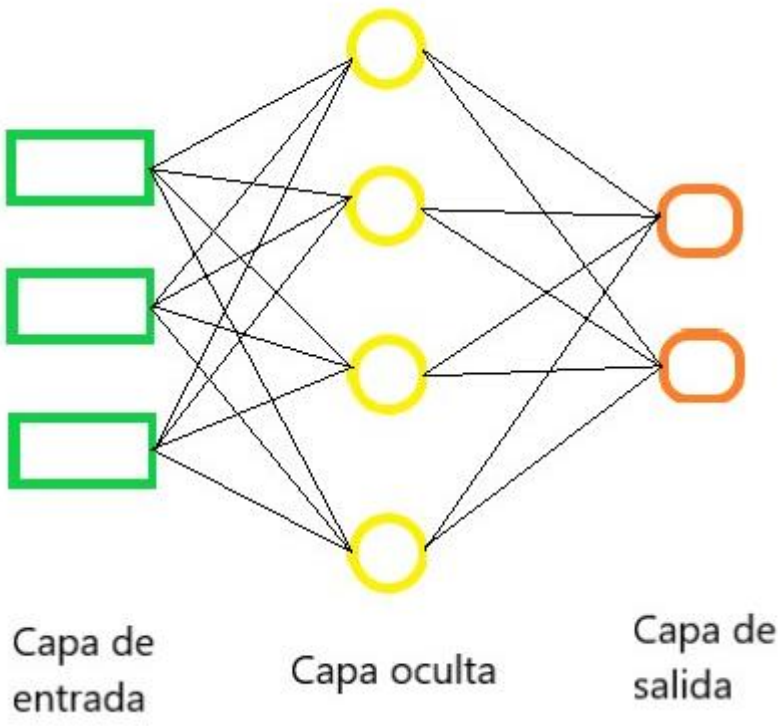

Figura 2.4.- Capas de una red neuronal.

El termino *deep learning* o aprendizaje profundo hace referencia al tipo de redes neuronales artificiales que contienen gran cantidad de capas, siendo muy grande la profundidad de la red. La arquitectura de red define la estructura de un modelo de d*eep learning* y el objetivo de su diseño. Esta determinará:

- La capacidad del modelo para alcanzar precisiones altas en problemas complejos.
- El objeto para predecir por el modelo y lo que espera como entrada y salida.
- La combinación de capas y el flujo de datos a través de ellas.

Las arquitecturas más comunes son:

- Red profunda *feedforward*: esta fue la primera forma de red neuronal artificial y la más sencilla ideada. En esta red no hay realimentación y la información fluye únicamente de la entrada a la salida de la red [16].
- Red neuronal recurrente (RNN): Consisten en redes neuronales *feedforward* extendidas incluyendo conexiones retroalimentadas [16]. Se utilizan para realizar predicciones futuras. Las redes LSTM (Memoria a Largo-Corto Plazo) son un tipo de redes neuronales recurrentes de uso para datos de secuencias y señales [17].
- *Autoencoders*: consisten en un tipo de redes *feedforward* cuyo objetivo en el entrenamiento es replicar o reconstruir en la capa de salida el dato que entro en la capa entrada con la mayor precisión posible. Estas primero codifican el dato de entrada en las primeras capas para reducirlo de tamaño más tarde y finalmente decodificarlo generando una imagen reconstruida [17].
- Redes neuronales convolucionales (CNN): Se asocian normalmente con imágenes como datos de entrada y consisten en redes con múltiples capas que procesan y extraen las características de los datos de entradas [17].

Las redes neuronales convolucionales están formadas por innumerables capas siguiendo un patrón de convolución que se repite una y otra vez. La convolución consiste en una operación dónde se aplica un conjunto de filtros a cada imagen de entrenamiento, activando cada filtro diferentes características sobre la imagen. Estos filtros son capaces de variar características simples como el brillo o los bordes hasta otras más complejas como los patrones que definen un objeto que se encuentra en ella [18].

Las redes neuronales convolucionales tienen la capacidad de aprender a detectar las distintas características de una imagen. La imagen de salida de esta operación se convierte en la imagen de entrada para la siguiente capa. Las redes neuronales convolucionales se

componen por una capa de entrada, una de salida y varias capas intermedias ocultas. El objetivo de alterar las imágenes mediante filtros es el de aprender las características o patrones que hay en estas [18].

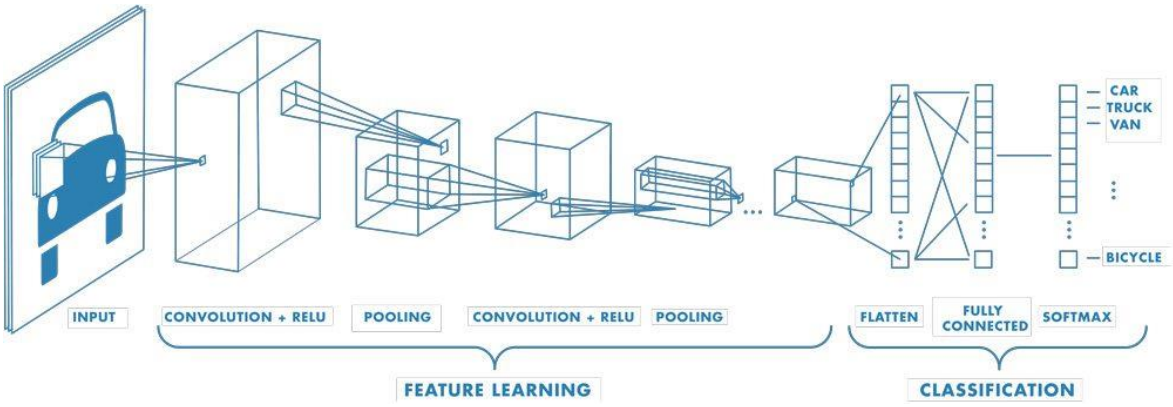

Figura 2.5.- Capas de una red neuronal convolucional [19].

Además de las capas convolucionales, en las CNNs hay otras capas como las ReLU (Unidad lineal rectificada) o de activación o las *pooling*. Las capas ReLU incrementan la velocidad y eficacia de los entrenamientos asignando a cero los valores negativos y manteniendo los positivos, mientras las capas *pooling* se encargan de simplificar los datos de salida disminuyendo el número de parámetros que la red necesita aprender al decrementar la tasa de muestro no lineal [19].

El fin del modelo de d*eep learning* encargado de la detección de objetos en imágenes puede ser identificar un solo tipo de objetos o varios en cada imagen. Por ejemplo, identificar coches en imágenes frente a identificar las diferentes marcas de coches en las imágenes. Para el segundo caso el modelo tendrá que ser capaz, además de localizar los objetos, de clasificarlos. (Ver figura 2.6 y 2.7)

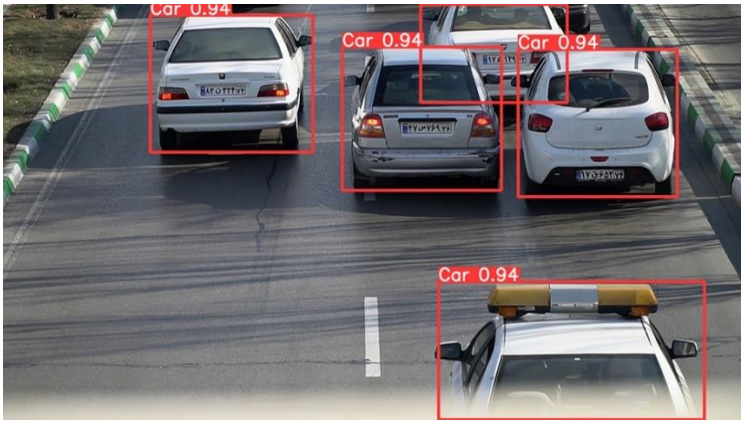

Figura 2.6.- Detección de coches por un modelo *deep learning*.

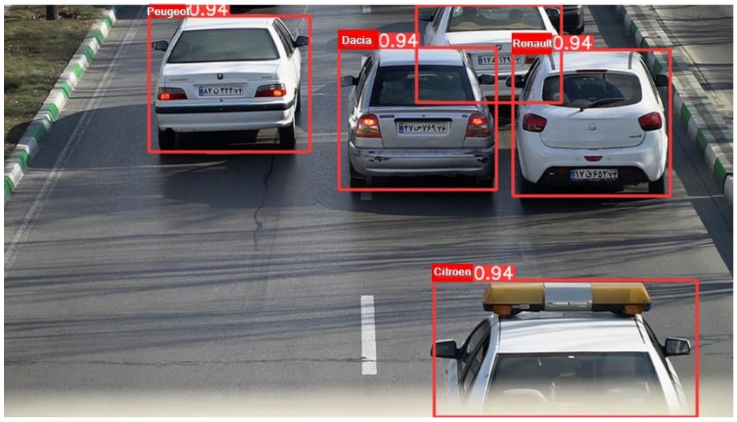

Figura 2.7.- Detección y clasificación de coches por marcas.

En el análisis de las arquitecturas de *deep learning* más utilizadas en la segmentación de células se deben de tener en cuenta los modelos DetectNet, Faster R-CNN, Darknet (con *backbone* ResNeXt) y Darknet (con *backbone YOLOv3)* [20].

DetectNet es una red derivada de GoogLeNET modificada para la detección de objetos realizando simultáneamente las técnicas de clasificación y regresión. Contiene capas de datos que se encargan de tomar las imágenes de entrenamiento y etiquetas y una capa de transformación que realiza un proceso de *data augmentation* para aumentar el número de datos de entrenamiento en cada época. Finalmente, la red GoogLeNet FCN realiza la extracción de características y la predicción de cuadros delimitadores y clases de objetos [21].

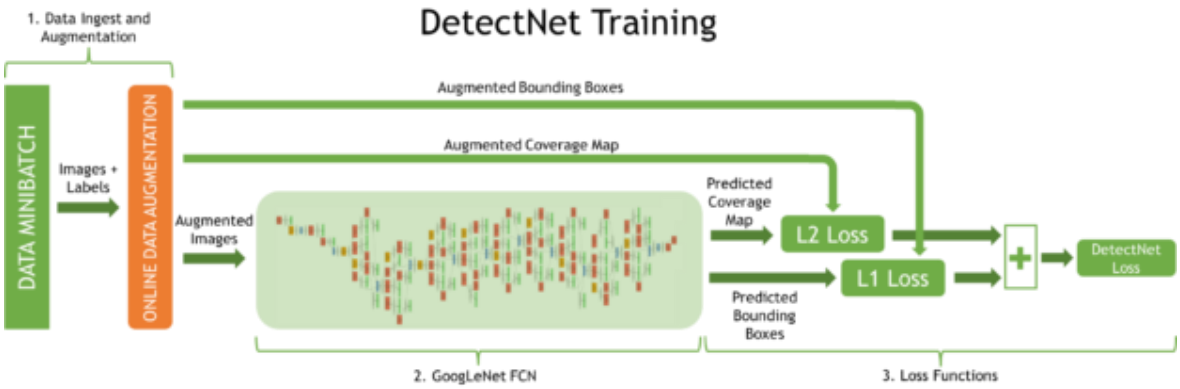

Figura 2.8.- Estructura de entrenamiento de DetectNet [21].

YOLO (*You Only Look Once*) es una red neuronal convolucional que se caracteriza por predecir los cuadros delimitadores, conocidos también como cajas o *bounding boxes*, de la imagen completa y de una sola evaluación. Predice simultáneamente todos los cuadros delimitadores en todas las clases y las probabilidades de clase de estos. Lo hace dividiendo la imagen en una cuadrícula y haciendo responsable a la celda dónde se ubique el centro del objeto de su detección [22] [23].

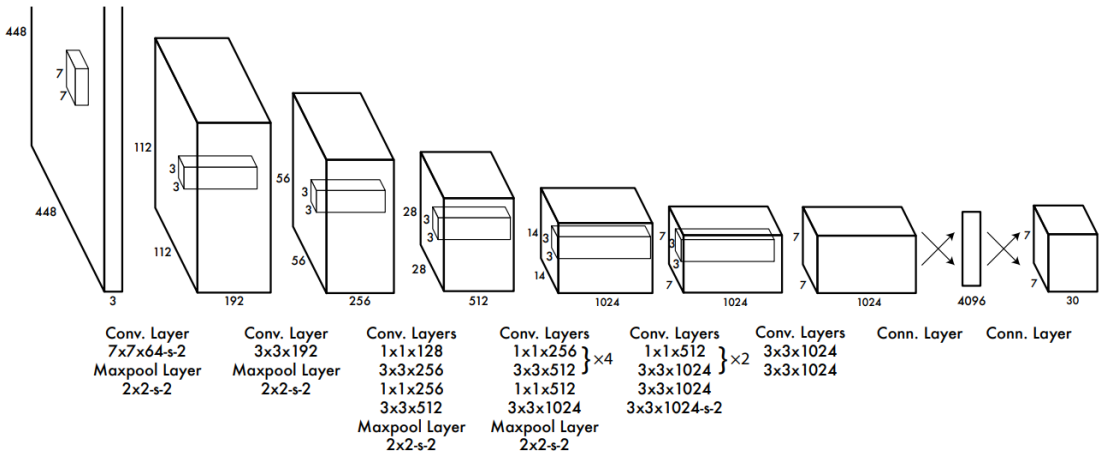

Figura 2.9.- Arquitectura YOLO [23].

El modelo Darknet con *backbone* o extractor de características ResNeXt utiliza el mismo entorno que YOLO, pero con un número y tipo diferente de capas. ResNeXt es un

rediseño de ResNet que se ha construido repitiendo un bloque de construcción que agrega un conjunto de transformaciones con la misma topología. Este diseño da como resultado una arquitectura homogénea de múltiples ramas en la que se tienen que configurar unos pocos hiperparámetros simplemente [24].

Por su parte, el modelo Faster R-CNN se compone de dos módulos, el primero, una red convolucional profunda que se encarga de decir al segundo, un módulo de detección Fast R-CNN, dónde mirar. Exactamente, el primer módulo propone en la imagen de entrada un conjunto de objetos rectangulares con una puntuación que el módulo de detección utiliza después [25].

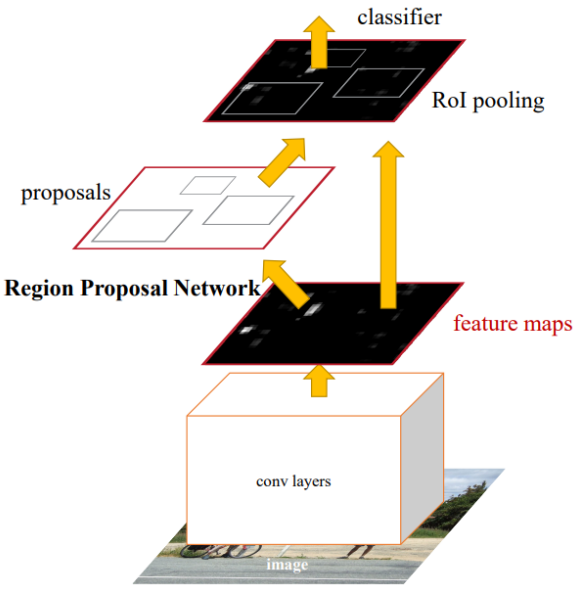

Figura 2.10.- Arquitectura Fast R-CNN [25]

Estos modelos han sido analizados y comparados entre sí y los resultados sitúan a Faster R-CNN cómo el mejor modelo actualmente, seguido del modelo YOLOv3 [20].

Model	F1-score	Precision	Recall
DetectNet	56.88%	53.04%	61.33%
Faster RCNN	**65.83%**	**60.73%**	**71.88%**
Darknet (custom ResNeXt)	55.70%	60.55%	51.56%
Darknet (YOLOv3-SPP)	57.31%	57.20%	57.42%

Figura 2.11.- Comparación rendimiento modelos DetectNet, Faster R-CNN y YOLOv3 [20].

Para poder entender los resultados de la comparación de los tres modelos hay que comprender primero a que se le llama *f1-score*, *precision* y *recall.*

Figura 2.12.- Valores en la predicción de dos clases de objetos en una imagen.

La precisión de un modelo sobre una clase de objetos es el porcentaje de aciertos con respecto número total de predicciones de esa clase. El *recall* por su parte es el porcentaje de aciertos sobre una clase con respecto al valor real de objetos de esa clase. Por último, el F1-

score de una clase es el resultado de multiplicar el valor de precisión, *recall* de esa clase y por dos, combinando de esta manera ambos valores. [26]

Al fijarse ahora en la figura 2.11 se ve que el modelo Faster R-CNN es el modelo más fiable al predecir que un punto pertenece a una clase, valor de precisión más alto, y el que mejor puede detectar a esa clase, valor de *recall* más alto. También se observa que le sigue el modelo YOLOv3 en su implementación original en el *framework* denominado Darknet. La razón por la cual se escogió la red YOLO, en lugar de Faster R-CNN, se explica en la sección 3.3.

3. Materiales y técnicas

3.1.- MATERIAL UTILIZADO PARA LA OBTENCIÓN DE LOS VÍDEOS DE CULTIVOS CELULARES

Las imágenes de microscopía se realizaron por parte del Instituto de Investigación Sanitaria del Principado de Asturias (ISPA) en un microscopio Zeiss AxioObserver Z1 (Carl Zeiss, Alemania) con un objetivo Plan-Apochromat 40X/1.3 (NA = 1.3, distancia de trabajo = 0.21 mm) de lente de aceite y una cámara AxioCamMRm (Carl Zeiss).

Los vídeos se adquirieron en el sistema Cell Observer que permite el estudio de células in vivo, está motorizado en los ejes x, y, z, equipado con un microscopio invertido AxioObserver z1, un sistema Apotome para la obtención de secciones ópticas, un incubador para el mantenimiento de temperatura, humedad y CO2 apropiados para estudios de células vivas, software ZenBlue y estación de trabajo. El cultivo celular se mantuvo a 37ºC con 5% CO2. ZENBlue es un software modular de análisis y procesamiento de imágenes con funcionalidad básica para la adquisición de imágenes y definiciones de microscopio, procesamiento elemental de imágenes y anotaciones, y módulos para tareas específicas.

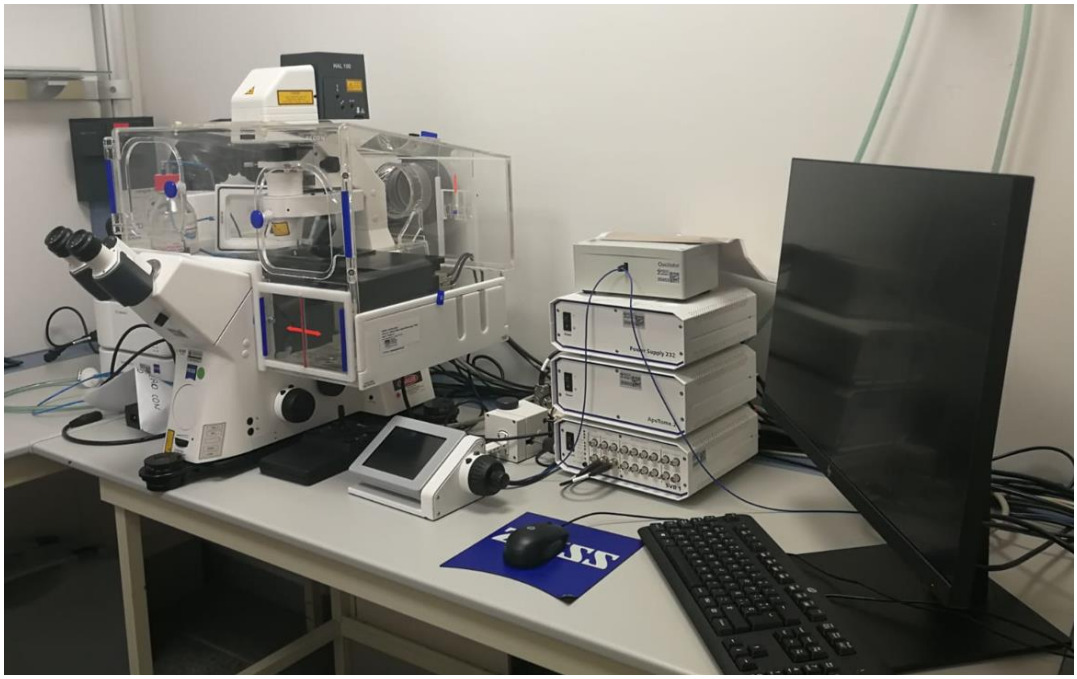

Figura 3.1.- Material utilizado en la grabación de los vídeos de cultivos celulares.

3.2.- IMPORTANCIA DEL USO DE GPUS

Para el entrenamiento de redes neuronales convolucionales el uso de GPUs es un requisito indispensable para realizar esta tarea de forma eficiente. Una GPU es un procesador con memoria dedicada que realiza operaciones de coma flotante requeridas para el procesamiento de gráficos. Las redes neuronales convolucionales realizan miles de millones de operaciones en su entrenamiento y a diferencia de las CPUs, que realizan las operaciones una detrás de otra, las GPUs son capaces de realizar varias operaciones al mismo tiempo. Esto supone una ventaja crucial en el entrenamiento de redes tan grandes como las convolucionales. Por esta razón, fue necesario para este proyecto contar con una GPU de memoria suficiente para entrenar una red profunda convolucional.

Google Colab es una herramienta online que permite utilizar las CPUs, GPUs y TPUs que Google pone a la disposición de cualquier usuario para entrenar un modelo *deep learning.* La versión gratuita está pensada para un uso interactivo y no permite realizar por ello entrenamientos extensos, caducando las sesiones si no se realiza ninguna acción más en la plataforma cada cierto tiempo y teniendo que empezar desde cero cargando el modelo y los archivos en el servidor cada vez que esta caduca.

Debido a esto se decidió realizar los entrenamientos en un servidor Linux con sistema operativo Debian 11 y GPU de 6 GB del grupo GSDPI de la universidad de Oviedo. El modelo de este servidor es Dell Precision Tower 3620 y el modelo de la tarjeta gráfica añadida es Gigabyte RTX2080 Ti Turbo Modelo TurboFun.

3.3.- ELECCIÓN DE LA ARQUITECTURA DEEP LEARNING

Un requisito de este trabajo es la utilización de Python como lenguaje de programación y su principal objetivo la búsqueda de un método de segmentación sencillo de implementar y muy reproducible.

YOLOv3, a diferencia de la arquitectura Faster R-CNN, es un modelo muy accesible y fácilmente reproducible debido a la amplia cantidad de recursos que existen en internet y a la variedad de *frameworks* a los que ha sido adaptado. Por ello, se ha optado por la red neuronal Yolov3 como método de segmentación. Se ha optado por la versión YOLOv3- Tiny frente a otras más recientes como la 4 o la 5 o frente a la versión 3 original debido a las

características de la GPU del servidor utilizado en el entrenamiento. Para poder entrenar estas otras versiones más potentes haría falta un servidor con una GPU con más memoria de la utilizada [27].

3.4.- FLUJO DE TRABAJO

El flujo de trabajo adoptado en la realización de los experimentos de este proyecto se muestra en la figura 3.2.

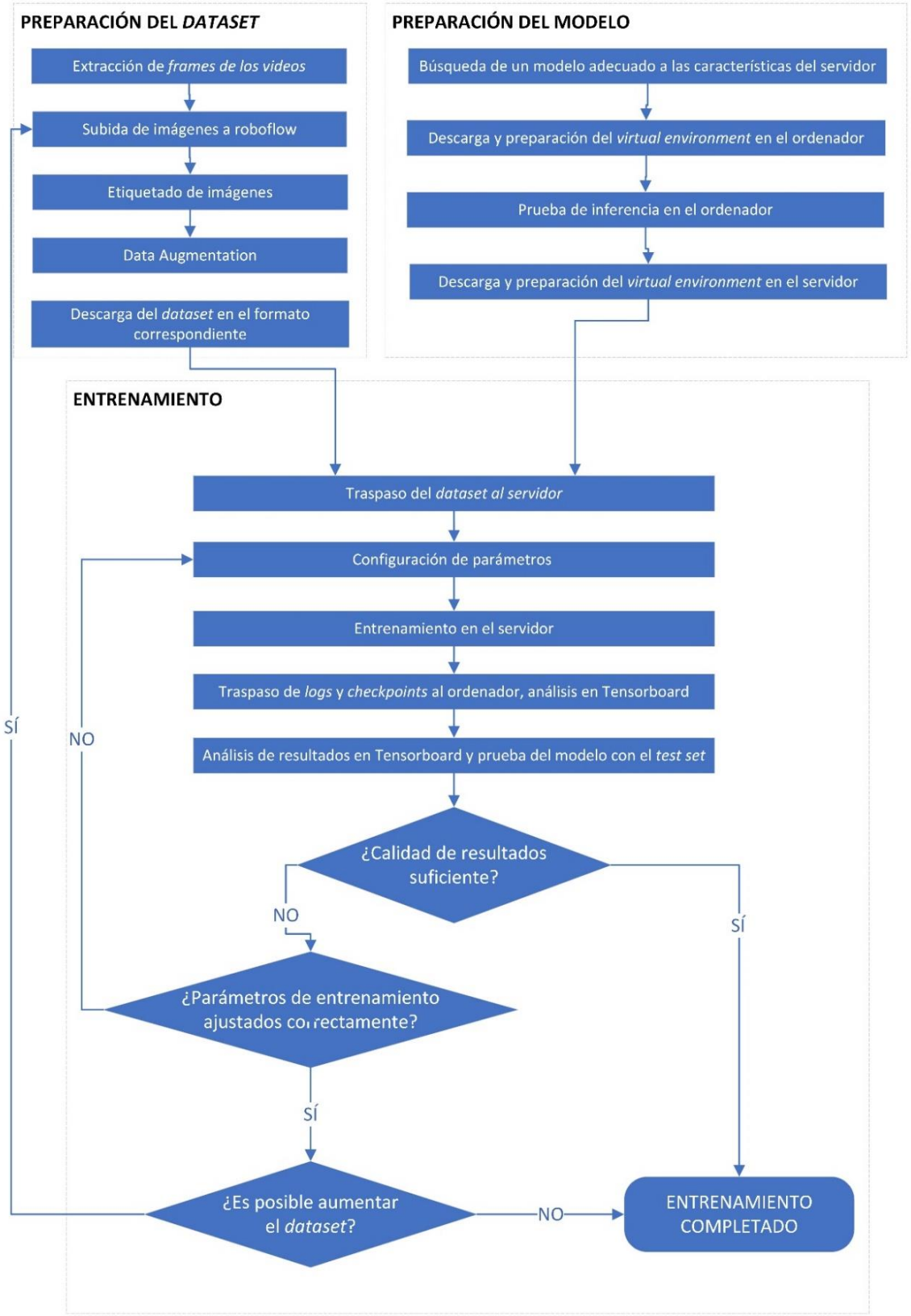

Figura 3.2.- Diagrama de flujo de trabajo seguido en este trabajo.

3.5.- DATASET

Para poder entrenar un modelo *deep learning* se requiere hacer un trabajo previo de preparación del conjunto de datos o *dataset* a utilizar en el entrenamiento, así como la búsqueda de un modelo *deep learning* que se adecúe a las características y librerías del servidor que utilicemos en el entrenamiento.

Una vez terminados los pasos anteriores, se pasaría entonces a adecuar los parámetros del modelo a nuestro caso concreto, regulándolos a medida que se van obteniendo resultados de los entrenamientos como si de un sistema de lazo cerrado de realimentación se tratara.

Los pasos que se han seguido para la creación del *dataset* de este proyecto son:

- Obtención de imágenes
- Etiquetado de imágenes
- *Data augmentation*

3.5.1.- Obtención de imágenes

Lo primero que se debe de hacer antes de poder entrenar un modelo de *deep learning* es obtener un conjunto de datos con los que pueda detectar patrones en el entrenamiento. La extensión del conjunto de datos dependerá del objeto a segmentar y de las características del modelo *deep learning* elegido, no cumpliéndose siempre que un conjunto de datos mayor proporcione un resultado final mejor. Un conjunto de datos demasiado grande no solo no mejoraría la calidad del resultado, sino que ralentizaría los tiempos de entrenamiento lo cual se traduciría en un mayor coste final. Por ello, es muy importante seleccionar un número adecuado de imágenes, así como una calidad de imagen suficiente para realizar el entrenamiento de la forma más eficiente posible.

Dependiendo del objeto a segmentar cada imagen albergará un único objeto o varios y estos estarán, o no, en contacto entre sí. Si cada imagen contiene varios objetos significará que necesitaremos menor número de imágenes para la misma calidad de resultado en la segmentación final, siendo lo importante el número de objetos etiquetados y no tanto el número de imágenes. Si están en contacto varios objetos entre sí, se complicará la operación de segmentado enormemente debido a que el modelo, además de ser capaz de diferenciar

entre fondo y objeto en las imágenes, tendrá que ser capaz de detectar los bordes de los objetos en contacto entre sí y separarlos.

Las imágenes de este trabajo albergan una media de 100 células por imagen estando muchas en grupos y visualmente en contacto entre sí. Estas imágenes se han obtenido de los fotogramas o *frames* del video `SCC42B-4.avi` (ver apartado 6.1. del anexo).

3.5.2.- Etiquetado de imágenes

El siguiente paso, una vez obtenido un número suficiente de imágenes, es etiquetar los objetos que nos interesan de ellas. En este trabajo no se realiza una labor de clasificación y simplemente se busca la localización y segmentación de las células.

Para el etiquetado de las imágenes se ha utilizado la herramienta Roboflow, una plataforma donde se puede etiquetar fácilmente los objetos, además de ser posible aumentar mediante técnicas de *data augmentation* el conjunto de imágenes y exportarlo en diferentes formatos (ver apartado 6.2 del anexo).

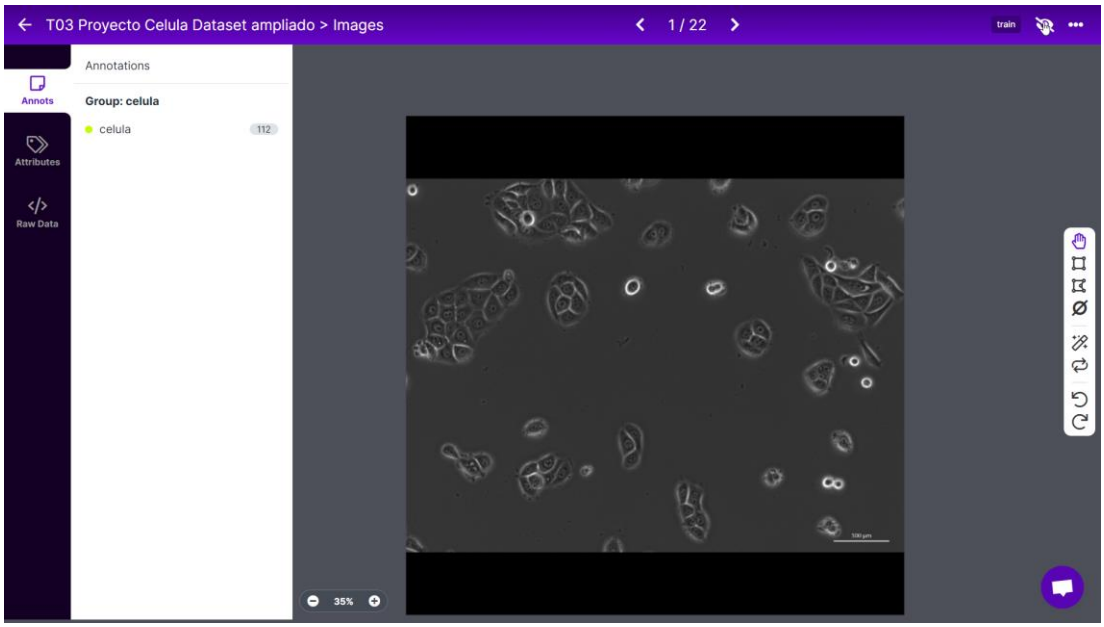

Figura 3.3.- Plataforma Roboflow para etiquetado de imágenes.

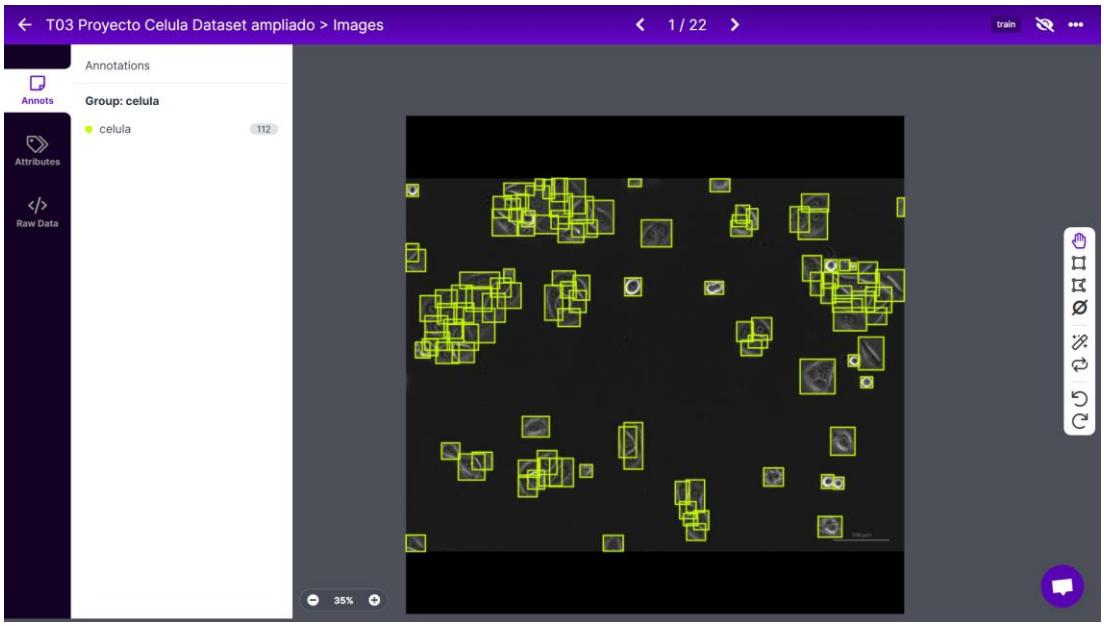

Figura 3.4.- Imagen con células etiquetadas mediante la plataforma Roboflow.

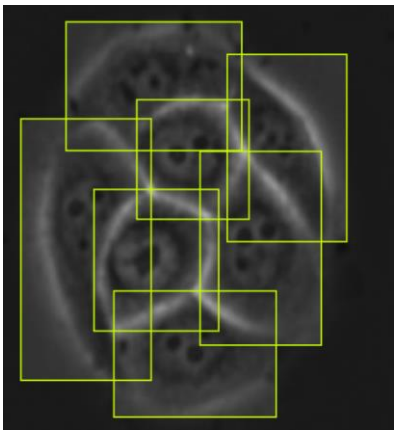

Figura 3.5.- Imagen de células en grupo etiquetadas con Roboflow.

El etiquetado de los objetos depende del modelo de *deep learning* escogido primeramente y segundo del objeto a segmentar. Se puede hacer mediante cuadrados o cajas o mediante formas geométricas más complejas que den un segmentado más preciso de los bordes del objeto. Para YOLOv3-Tiny, dado que utiliza cajas o *bounding boxes* en los entrenamientos, lo más adecuado es usar estos también para el etiquetado, dado que, aunque

utilicemos formas más precisas en el etiquetado, YOLOv3-Tiny las aproximará a la *bounding box* más parecida.

Al terminar la fase de etiquetado, se tendrá un conjunto de datos etiquetados. En este trabajo se han etiquetado 22 imágenes con una media de 100 células por imagen, es decir, un total aproximado de 2200 células etiquetadas.

3.5.3.- Data augmentation

Gracias al *data augmentation*, entendido como el aumento del conjunto de datos mediante transformaciones en las imágenes, podemos aumentar fácilmente nuestro conjunto de datos sin tener que realizar ninguna labor añadida de etiquetado.

Estas técnicas consisten en crear imágenes artificiales aplicando distintos efectos al conjunto de imágenes etiquetadas dando lugar a versiones modificadas de las mismas. Algunos ejemplos son: voltear las imágenes, rotarlas un ángulo determinado, aplicar *zoom* sobre ellas, subir o bajar el brillo de las mismas o desenfocarlas.

Roboflow además del método clásico de d*ata augmentation* explicado previamente, da la posibilidad de realizar un segundo tipo de *data augmentation* el cual consiste en realizar una transformación distinta sobre cada *bounding box* de cada imagen. En este trabajo se han utilizado y combinado ambas alternativas.

Las transformaciones utilizadas en este trabajo son:

- *Flip*: Transformación que voltea las imágenes en el eje horizontal o en el eje vertical.
- *Rotate*: Rotación de las imágenes de forma aleatoria entre dos ángulos indicados.
- *Crop*: Hacer zoom y recortar las imágenes de forma aleatoria hasta un porcentaje indicado.
- *Shear*: se trata de variar la perspectiva con la que se ve una imagen mediante una inclinación simultánea en el eje horizontal y vertical.
- *Brightness*: Aclarar y oscurecer aleatoriamente en un porcentaje determinado las imágenes.

- *Exposure*: Variar la exposición gamma de una imagen para hacerla más clara u oscura.
- *Blur*: Añadir desenfoque gaussiano de forma aleatoria en las imágenes hasta un desenfoque máximo indicado.
- *Noise*: Añadir un porcentaje de ruido determinado a las imágenes.

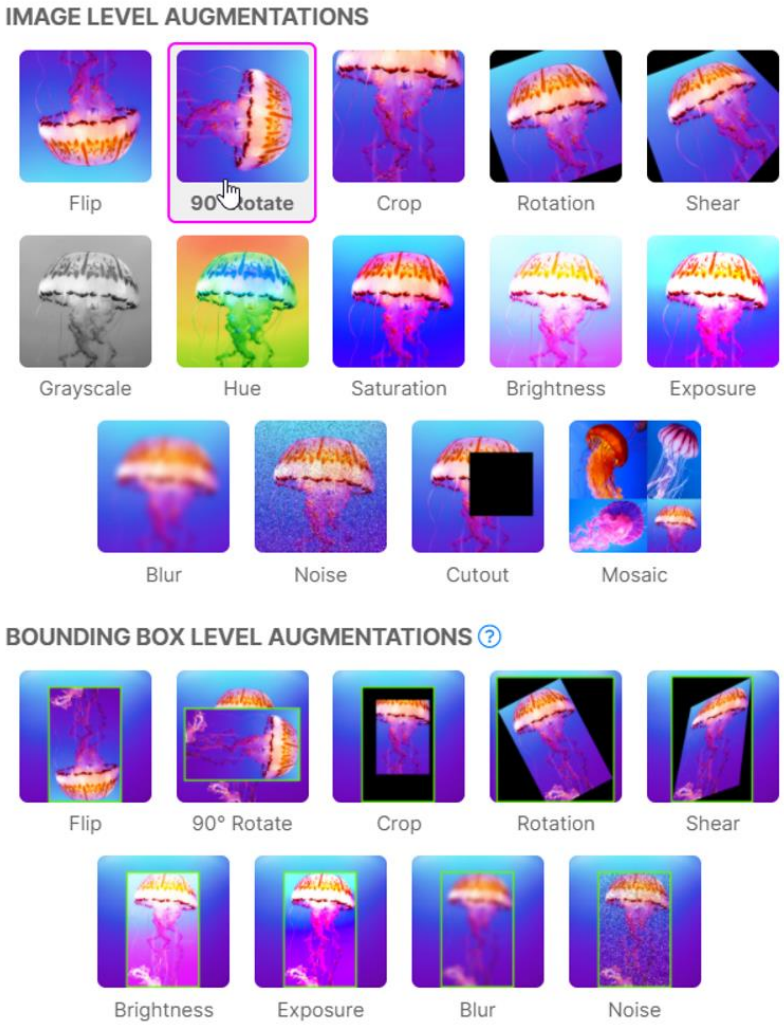

Figura 3.6.- Técnicas de *data augmentation* disponibles en la plataforma Roboflow.

3.5.4.- Partición del dataset.

El conjunto de imágenes que forman el d*ataset* debe dividirse en dos partes antes de realizar el entrenamiento. La primera es la llamada *training set* y será la que utilice el modelo para aprender los patrones y características de las imágenes en el entrenamiento, mientras

que la segunda, denominada *validation set*, se empleará en evaluar al final de cada época del entrenamiento la precisión del modelo al intentar predecir y localizar los objetos en imágenes previamente etiquetadas.

En este trabajo se ha hecho la partición estándar, y más utilizada en *machine learning*, del ochenta por ciento del total de imágenes para el *training set* y del veinte por ciento para el *validation set.* Para el *testset*, conjunto de imágenes utilizados para evaluar el modelo una vez terminado el entrenamiento, se han utilizado el resto de los vídeos (`SCC38.avi`, `SCC40.avi`, `SCC42B-1.avi`, `SCC42B-2.avi`, `Scene9SCC42B.avi`) no empleados en la creación del *dataset* y también los 30 fotogramas no empleados del video `SCC42B-4.avi`.

3.5.5.- Formato de dataset y estructura de carpetas.

Dependiendo del modelo *deep learning* escogido y del *framework*, entendiendo este como la interfaz que nos permite incrementar la eficiencia en el entrenamiento de un modelo *deep learning*, en el que trabajemos para su entrenamiento tendremos que exportar nuestro *dataset* en uno u otro formato.

El modelo *deep learning* YOLO fue desarrollado en un *framework* denominado Darknet que es una red neuronal escrita en C cuyo formato de etiquetado consiste en un archivo de texto por imagen dónde en cada línea se indica: la clase del objeto etiquetado, mediante un número que hace referencia a la línea del archivo `names.txt` dónde está escrito el nombre de la clase del objeto etiquetado; y sus coordenadas, mediante cuatro números que indican por orden la posición del centro del objeto en el eje X y en el eje Y, y el ancho y el alto del mismo. [`x_center, y_center, width, height`]

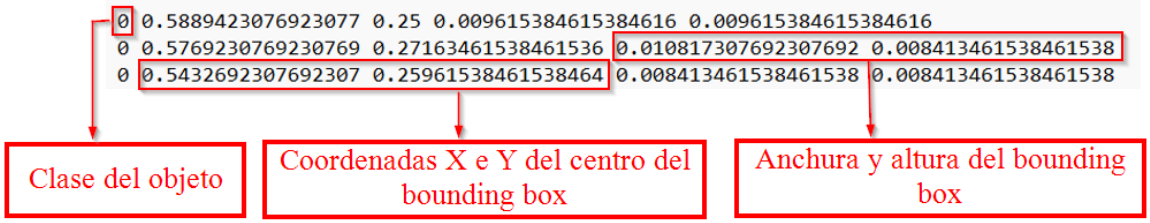

Figura 3.7.- Formato de YOLOv3 (Darknet) para el etiquetado de objetos.

El archivo de texto dónde se indican los nombres de las clases de objetos empleados en la fase de etiquetado será un archivo de texto que contendrá cada nombre de clase en una línea distinta. Para distinguir entre los datos que se empleen en el entrenamiento frente a los datos de validación se separan estos junto a los archivos de etiquetado correspondientes en dos carpetas con nombre '`train`' y '`valid`'.

En este trabajo debido a las características del servidor utilizado en el entrenamiento se utiliza una versión de YOLOv3 adaptada al *framework* Tensorflow en su versión 2.4.1. Para este modelo en concreto se debe utilizar un formato de *dataset* en el cual las imágenes se separan entre dos carpetas `train` y `valid`, y las etiquetas en dos archivos de texto, uno para el conjunto de imágenes de la carpeta `train`, denominado `Dataset_train.txt`, y otro para el conjunto de imágenes de la carpeta `valid`, `Dataset_valid.txt`. En cada línea de estos archivos está escrita la ruta de la imagen a la que hace referencia seguido por el número de clase del objeto etiquetado y las coordenadas, separando con un espacio cada conjunto de cifras correspondientes al etiquetado de cada objeto.

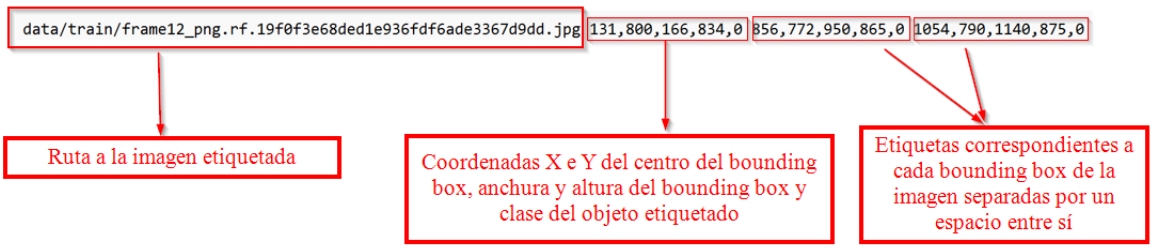

Figura 3.8.- Formato de etiquetado utilizado en este proyecto.

Por último, el archivo de texto que contiene los nombres de las clases empleadas en el etiquetado de los objetos en cada imagen se denomina `Dataset_names.txt` (En este proyecto este archivo solo contiene la clase 'células') En la estructura de carpetas de este proyecto, estos tres archivos de texto al igual que las carpetas `train` y `valid` se localizan en la carpeta `data`.

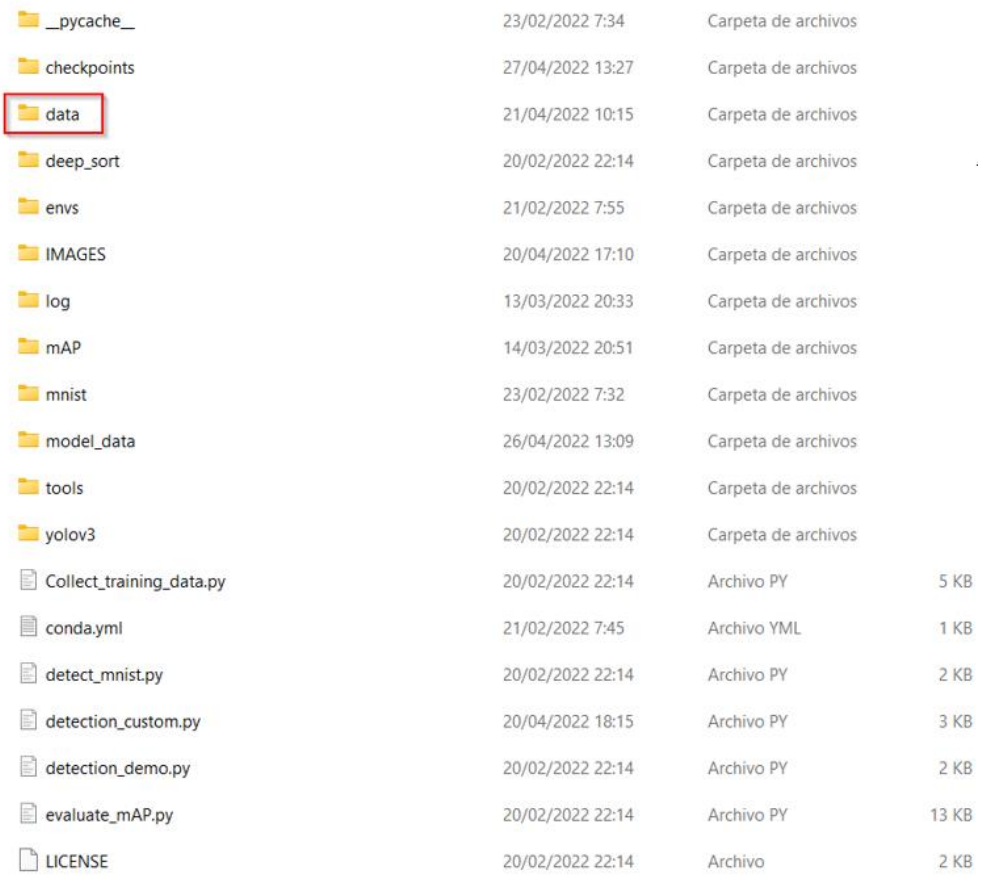

Figura 3.9.- Estructura de carpetas del modelo *deep learning* utilizado en este trabajo.

3.6.- PREPARACIÓN DEL MODELO YOLOV3-TINY

3.6.1.- Prueba de inferencia.

Este paso consiste en realizar una primera prueba de inferencia con el *dataset* que trae el modelo por defecto, en este caso el *dataset* 'mnist', para comprobar que el modelo funciona correctamente en el ordenador y se ha configurado correctamente el entorno virtual. Este paso es conveniente hacerlo antes de lanzar el modelo a entrenar en el servidor para identificar más fácilmente cualquier problema que pueda aparecer (ver apartado 6.4 del anexo).

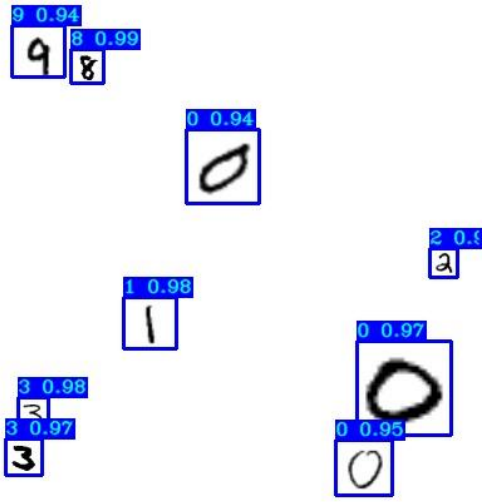

Figura 3.10.- Prueba de inferencia en el ordenador con el *dataset* 'mnist'

3.6.2.- Elección de parámetros de entrenamiento.

En el archivo de configuración del modelo denominado `configs.py` y presente en la carpeta `yolov3` se seleccionaron los parámetros *input_size*, tamaño de entrada de las imágenes a entrenar, *train_epoch*, épocas del entrenamiento, *train_batch_size* y *test_batch_size*, número de imágenes que se pasa al modelo en cada iteración de aprendizaje y de validación.

Como tamaño de entrada se seleccionó primeramente 416, que es la resolución estándar de entrada de YOLOv3-Tiny (416 x 416 píxeles) y después se probó a aumentarlo al doble de resolución, 832. Se eligieron 200 épocas de entrenamiento y tamaño de *batch* 4 primeramente, tanto para el entrenamiento como para la validación, probando después a subirlo a 16.

3.6.3.- Entrenamiento en servidor.

El entrenamiento del modelo se realizó como se menciona anteriormente en un servidor Linux con sistema operativo Debian 11 del grupo de investigación GSDPI del área de Ingeniería de Sistemas y Automática de la universidad de Oviedo. Para la ejecución de programas se utiliza una conexión SSH mediante el programa Putty y para la transferencia de datos se utiliza una conexión SFTP mediante el programa Filezilla (ver apartado 6.5 del

anexo). Se transfirió primero mediante Filezilla el *dataset* y el archivo de configuración `config.py` a las correspondientes carpetas del servidor y después, mediante el programa Putty, se ha creado y activado el entorno virtual necesario para el entrenamiento de este modelo (ver apartados 6.3 y 6.5 del anexo). Por último, se lanza el entrenamiento.

```
miguelferc@gsdpi.edv.uniovi.es's password:
Linux gsdpi 4.19.0-20-amd64 #1 SMP Debian 4.19.235-1 (2022-03-17) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Jun  5 18:24:36 2022 from 156.35.225.247
miguelferc@gsdpi:~$ ls
dataset  datos  log.txt  TensorFlow-2.x-YOLOv3
miguelferc@gsdpi:~$ cd TensorFlow-2.x-YOLOv3
miguelferc@gsdpi:~/TensorFlow-2.x-YOLOv3$ source activate celulas
(celulas) miguelferc@gsdpi:~/TensorFlow-2.x-YOLOv3$ nohup python train.py >> log.txt &
```

Figura 3.11.- Comandos utilizados para lanzar el entrenamiento en el servidor Linux.

4. Resultados

4.1.- GOOGLE COLAB

En una primera iteración se utilizó la plataforma Google Colab para probar a entrenar el modelo YOLOv4 con el primer *dataset* que se etiquetó y ver los resultados. La primera iteración fue con un *dataset* de únicamente 4 fotogramas (con una media de 100 células por fotograma) en el que se habían etiquetado los núcleos de las células, en vez del contorno de la célula como se hizo posteriormente, y la segunda fue con un *dataset* ya de 24 fotogramas con también los núcleos de las células etiquetados.

Resultó una primera buena toma de contacto, en la que se constató que el modelo YOLOv4 era una buena herramienta para trabajar con el *dataset* de este proyecto. En el segundo *dataset*, tras etiquetar los 24 fotogramas en la plataforma Supervisely, se aumentaron posteriormente a 144 fotogramas con técnicas de *data augmentation*.

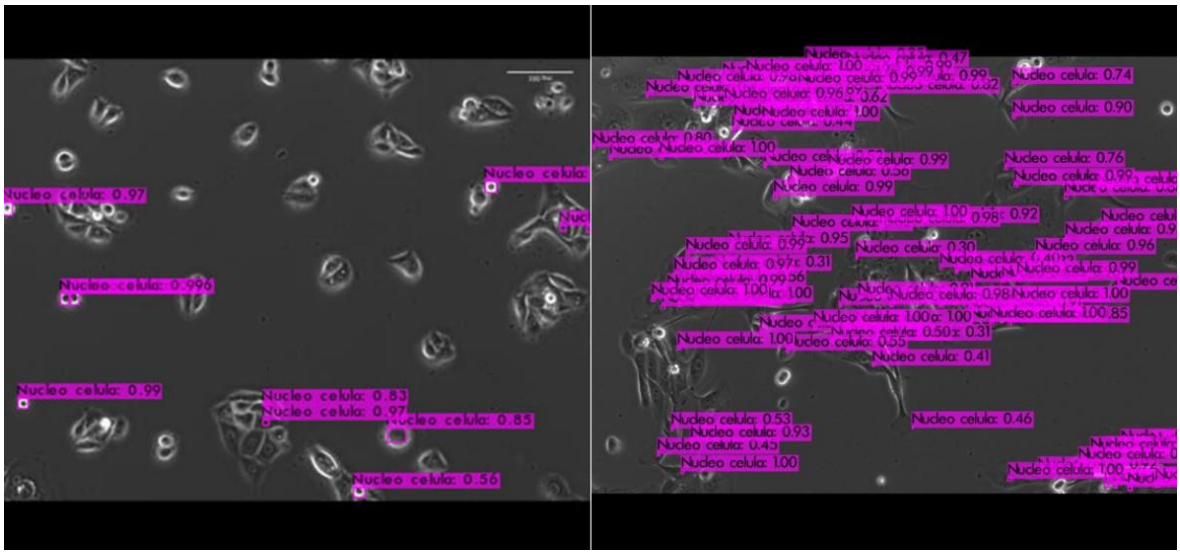

Figura 4.1.- Resultado entrenamiento en Google Colab con *dataset* de 4 fotogramas en la primera imagen y de 144 fotogramas en la segunda.

A diferencia de Roboflow, Supervisely deja la responsabilidad en el propio usuario en la fase de *data augmentation* de seleccionar por sí mismo los parámetros de *resize* y *noise* entre otros, siendo más difícil aplicar una variabilidad aleatoria para cada imagen o *bounding*

box entre dos valores dados. Por la mayor complejidad de Supervisely se decidió utilizar Roboflow para este trabajo.

4.2.- ANÁLISIS DE ENTRENAMIENTOS

Para poder comparar los entrenamientos se utilizó la plataforma Tensorboard que permite realizar gráficas con los datos obtenidos durante el entrenamiento. El valor de pérdida de validación o *validation loss* indica cómo de mala ha sido la predicción después de cada época en las imágenes que componen el *validation set.* Cuanto más cercano a cero sea su valor, más perfecta será la predicción en la fase de validación. Se comparó por tanto el parámetro *validation loss* para decidir la mejor estrategia de entrenamiento para este modelo y *dataset*.

En el primer entrenamiento realizado, de 200 épocas, *batch* 4 y tamaño de entrada 416, el mejor resultado se obtuvo tras una hora y tres minutos y medio de entrenamiento en la época 142. El parámetro de *batch* indica el número de imágenes que se pasa al modelo en cada iteración de aprendizaje. El resultado de *validation loss* obtenido en el primer entrenamiento es de 127,2. El segundo entrenamiento se realizó con un *batch* de 16 para comprobar la influencia de este parámetro en los resultados del entrenamiento, y se obtuvo, en el mismo punto dónde se había obtenido el mejor resultado en el primer entrenamiento, un *validation loss* de 138.7. Como se observa en la figura 4.1 la calidad del resultado es mayor durante todas las épocas para el entrenamiento realizado con un *batch* de 4.

Posteriormente se realizó un entrenamiento con *batch* 4 y tamaño de entrada de imagen de 832 para comparar el rendimiento en el entrenamiento. Como se observa en la figura 4.2, los resultados de este entrenamiento fueron peores en todo el rango de épocas realizadas en comparación a un tamaño de entrada de 416.

Por esta razón, se decide conservar el parámetro de *batch* en 4, tamaño de entrada en 416 y no se decide ampliar el número de épocas de entrenamiento debido a una estabilización de los resultados obtenidos a partir de la época 142, como se observa en la figura 4.1. La precisión media obtenida en esta época, dónde se obtienen los mejores resultados de entrenamiento, es de 0.727, para un valor de *batch* 4 y tamaño de imagen 416. El valor de

Recall no es útil para este modelo dado que únicamente se encarga de localizar células y no de clasificarlas.

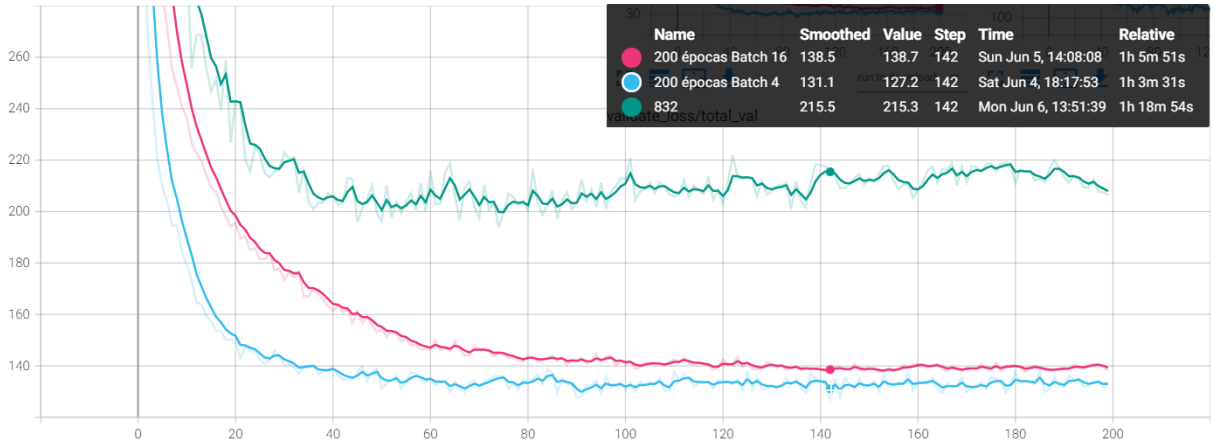

Figura 4.2.- *Validation loss* en cada época en los entrenamientos de *batch* 4 y 16 con tamaño entrada 416, y de batch 4 con tamaño de entrada 832.

4.3.- DETECCIÓN DE CÉLULAS EN VÍDEOS DE MUESTRA

A continuación, se muestran los resultados obtenidos en la detección de células en cada vídeo de muestra de este proyecto con el modelo YOLOv3-Tiny entrenado 200 épocas con un valor de *batch* igual a 4, un tamaño de entrada de imagen de 416 y un *dataset* de 154 fotogramas y una media de 100 células por fotograma. Los resultados que se presentan son de varias líneas celulares, SCC38, SCC40, SCC42B-1, SCC42B-2 y SCC42B-4. Se presentan primero fotogramas de cada video y después secuencias de 9 fotogramas para tres de los videos. Como se observa en las imágenes, la clase de los *bounding boxes* se ha modificado de 'células' a 'c' para tener una visualización más clara de la detección de las células en los resultados que se presentan a continuación. Por otro lado, se adjunta la carpeta `Resultados` dónde se encuentran los resultados de la detección de células en algunos fotogramas de cada video y en el video `SCC42B-2.avi` (debido a la restricción de 150 MB como tamaño máximo de los ficheros que se permiten adjuntar no se ha podido incorporar todos los fotogramas y videos a la carpeta de `Resultados`).

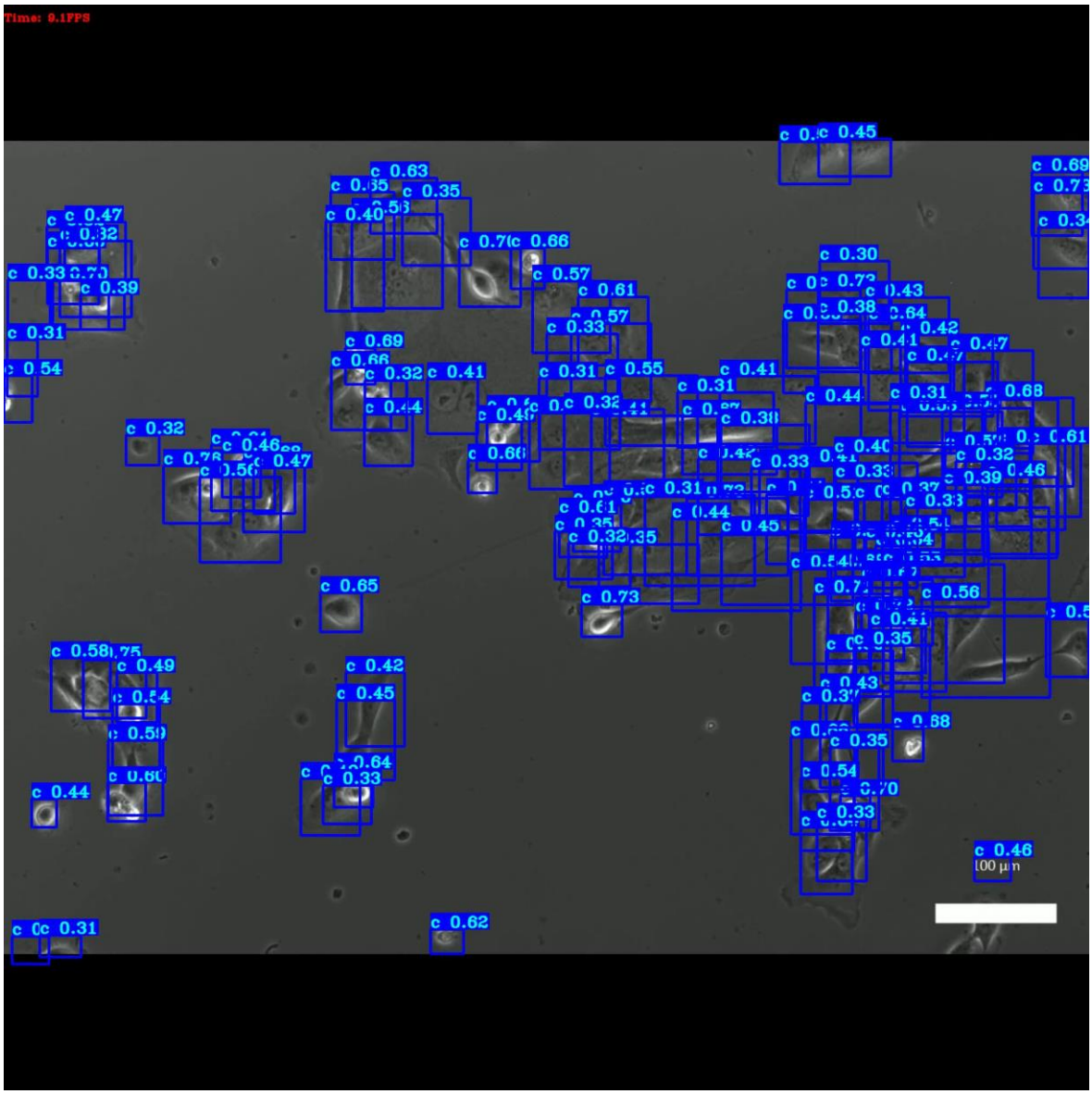

Figura 4.3.- Detección de células en un *frame* del video `SCC38.avi` por el modelo.

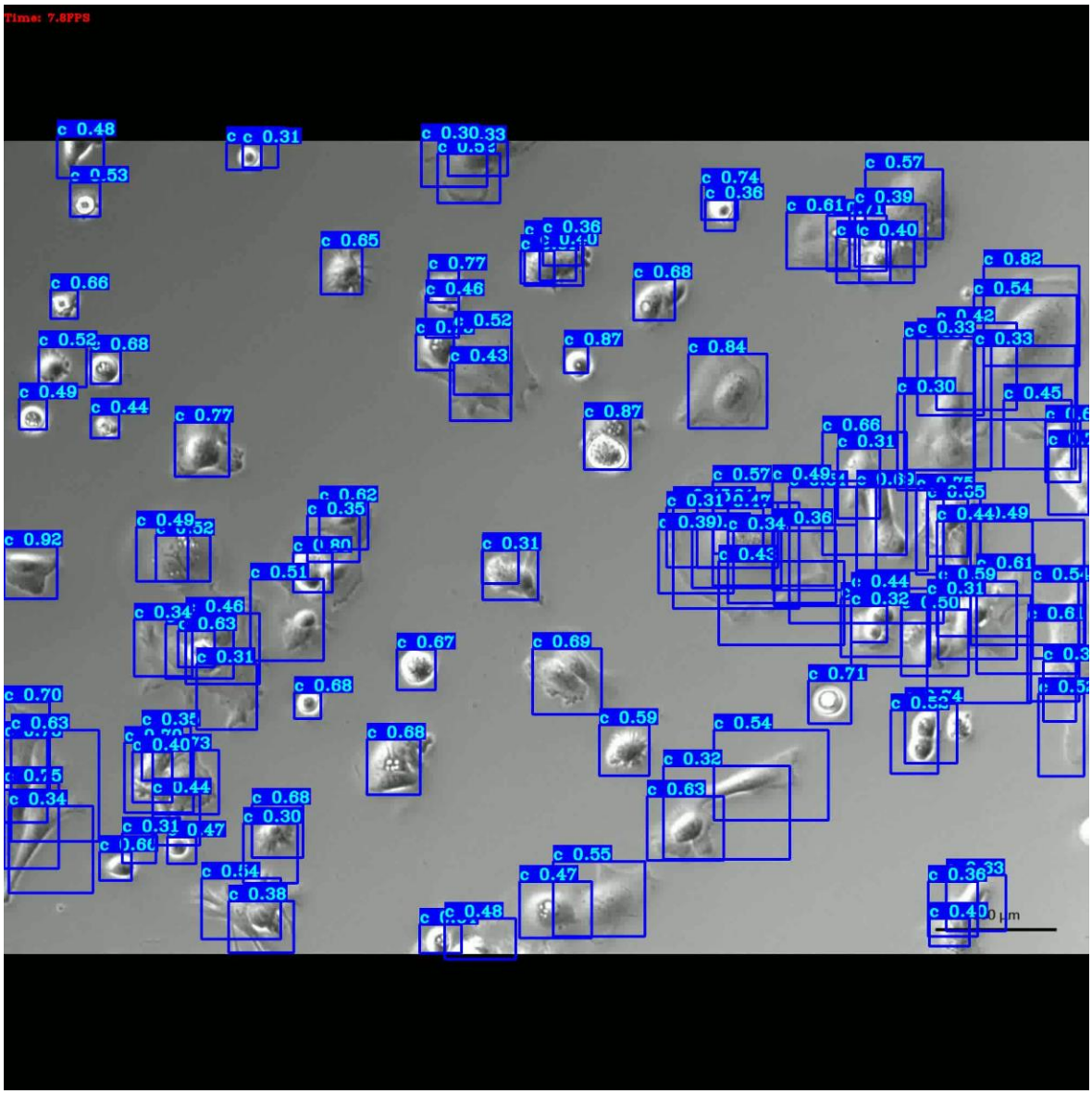

Figura 4.4.- Detección de células en un *frame* del video `SCC40.avi` por el modelo.

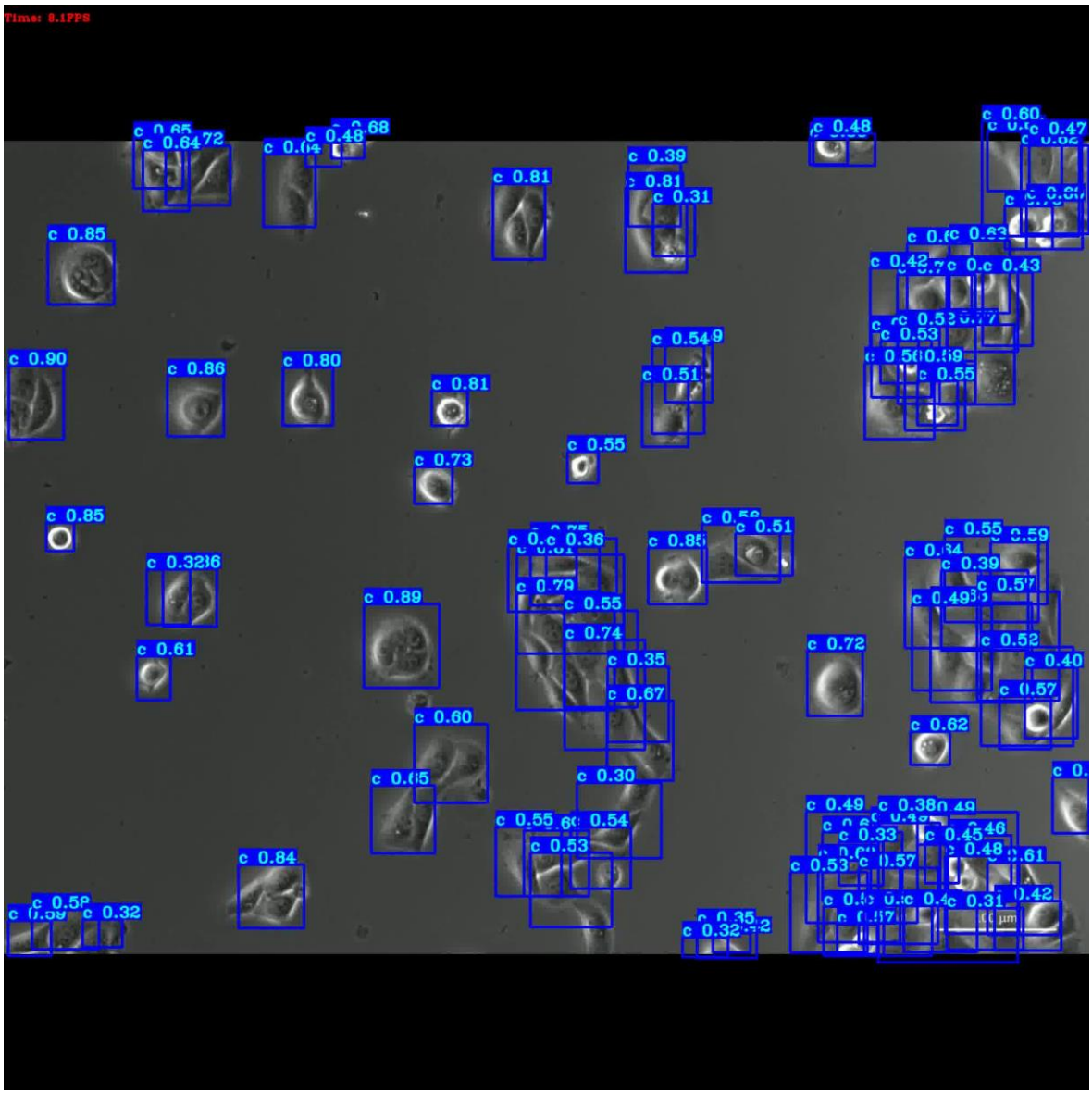

Figura 4.5.- Detección de células en un *frame* del video `SCC42B-1.avi` por el modelo.

Figura 4.6.- Detección de células en un *frame* del video `SCC42-B-2.avi` por el modelo.

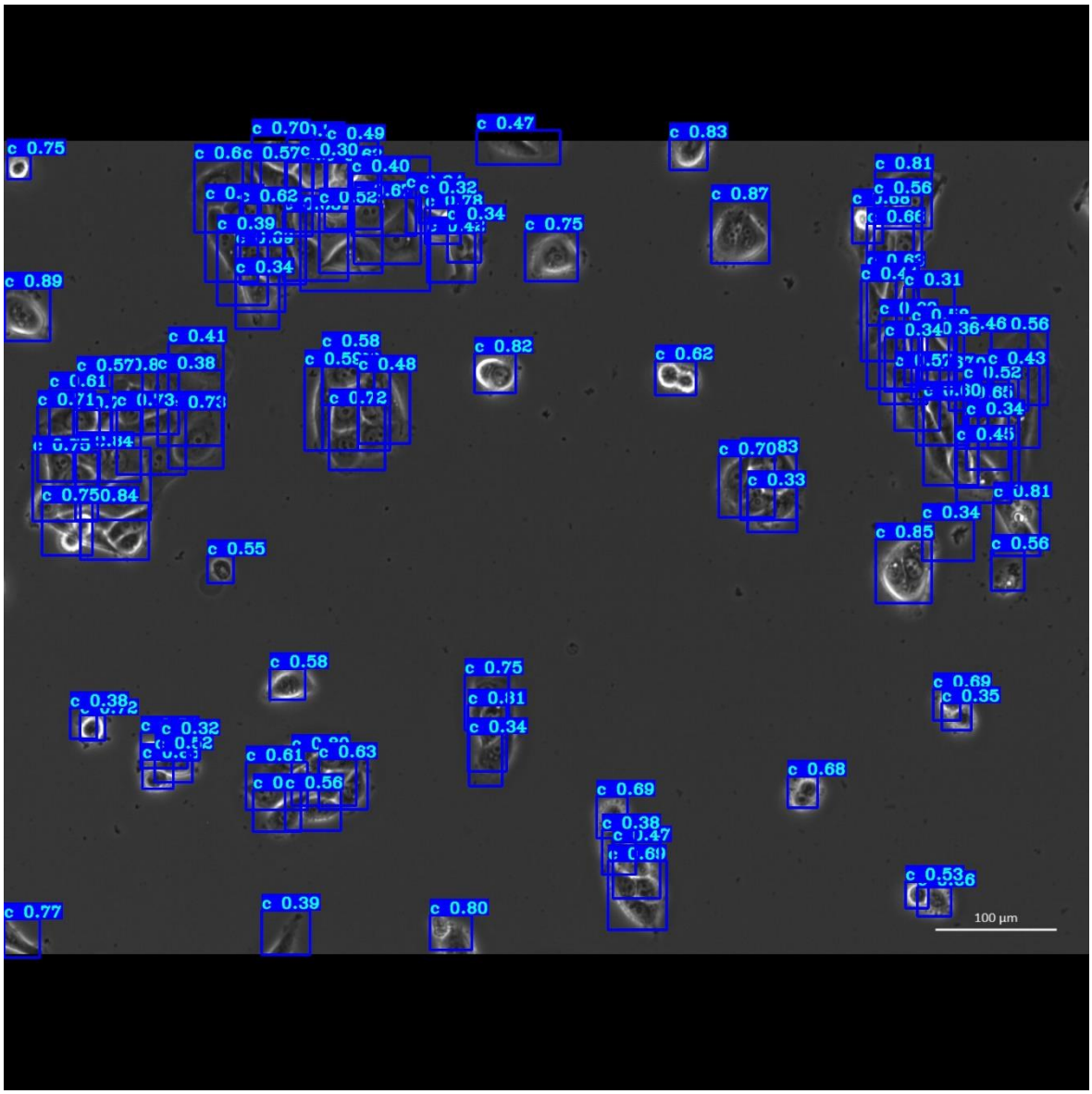

Figura 4.7.- Detección de células en un *frame* del video `SCC42B-4.avi` por el modelo.

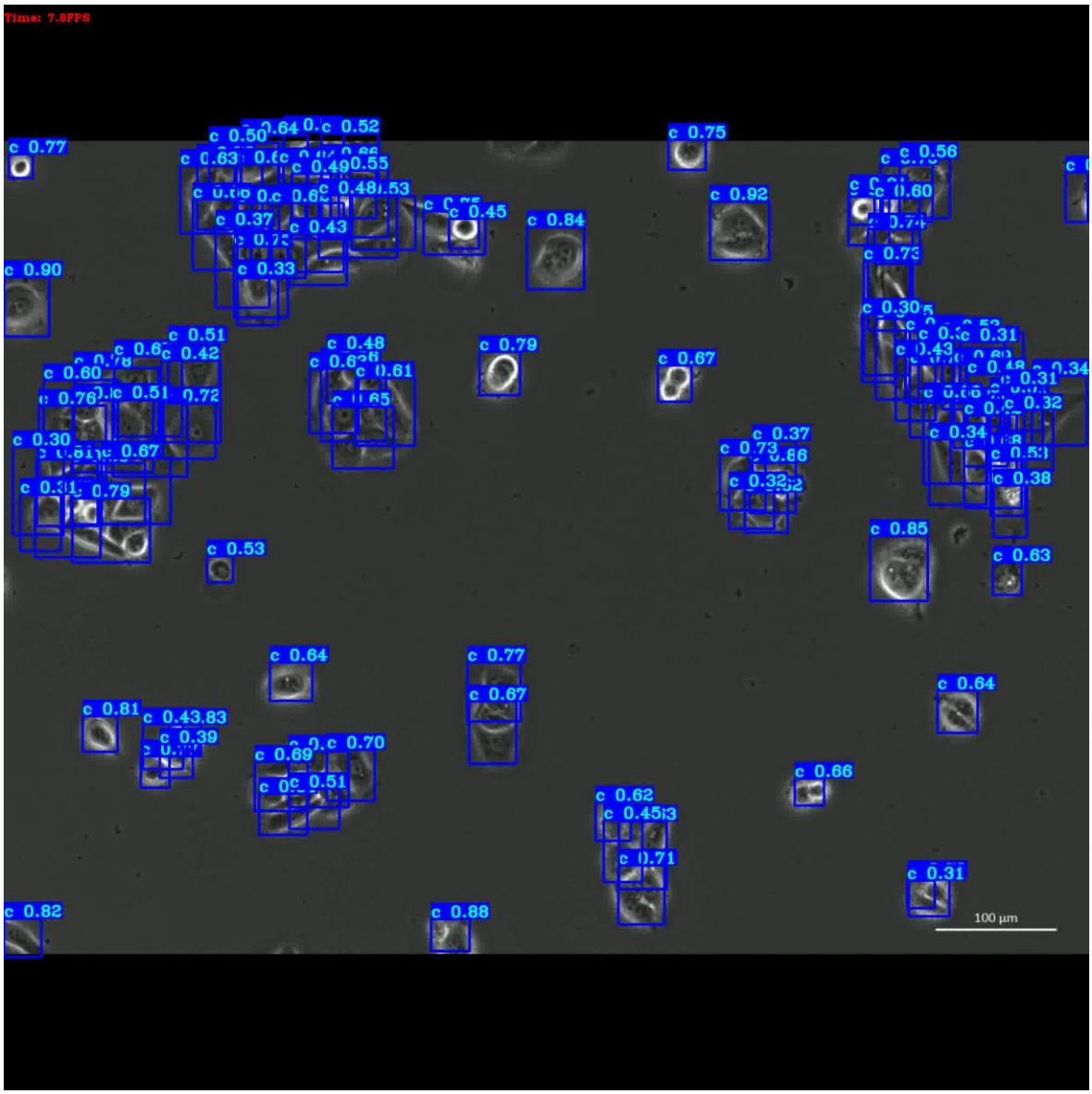

Figura 4.8.- Detección de células en un *frame* del video ‘Scene 9 SCC42B’ por el modelo.

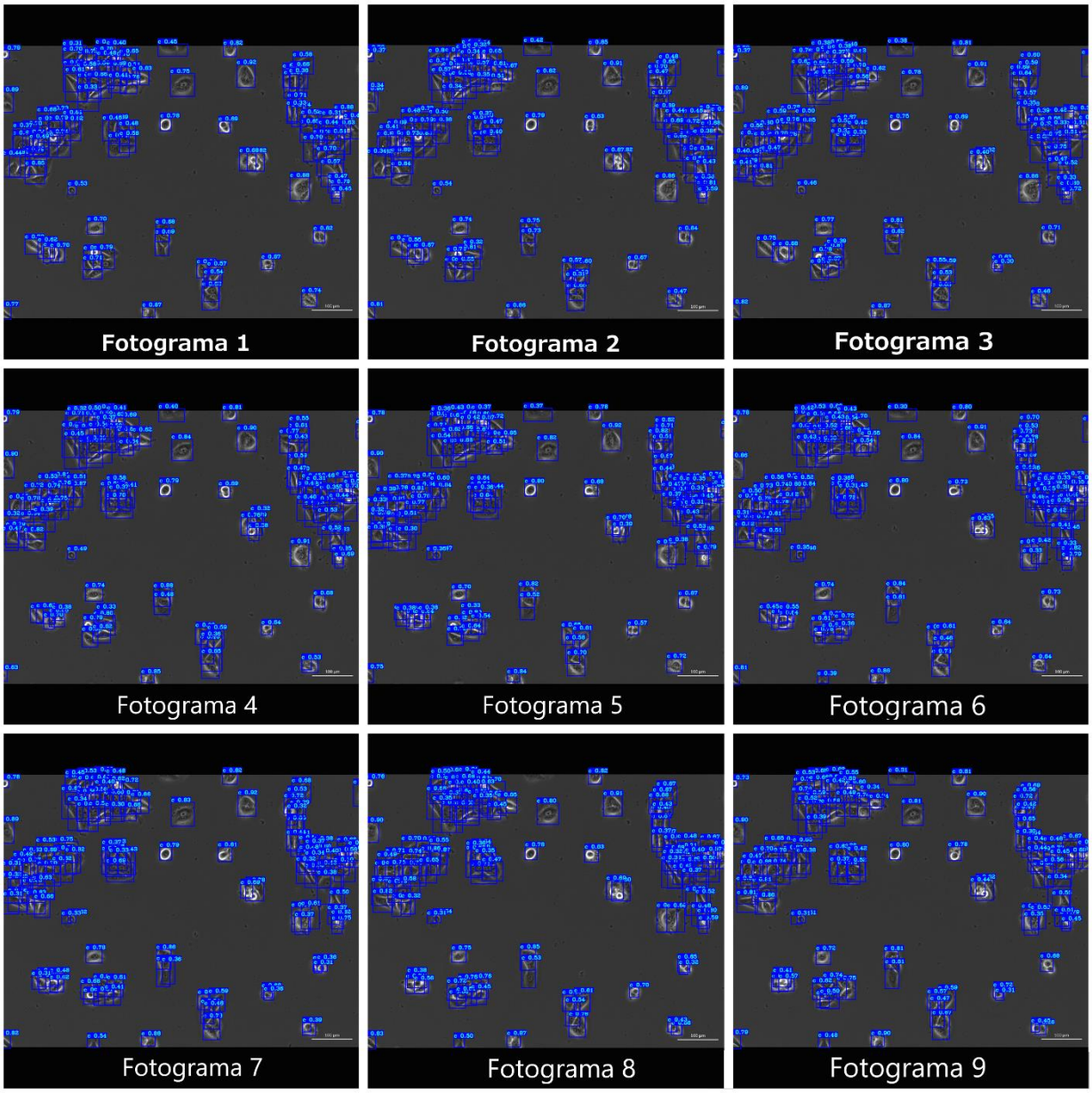

Figura 4.9.- Detección de células en secuencia de fotogramas del video `SCC42B-4.avi` por parte del modelo.

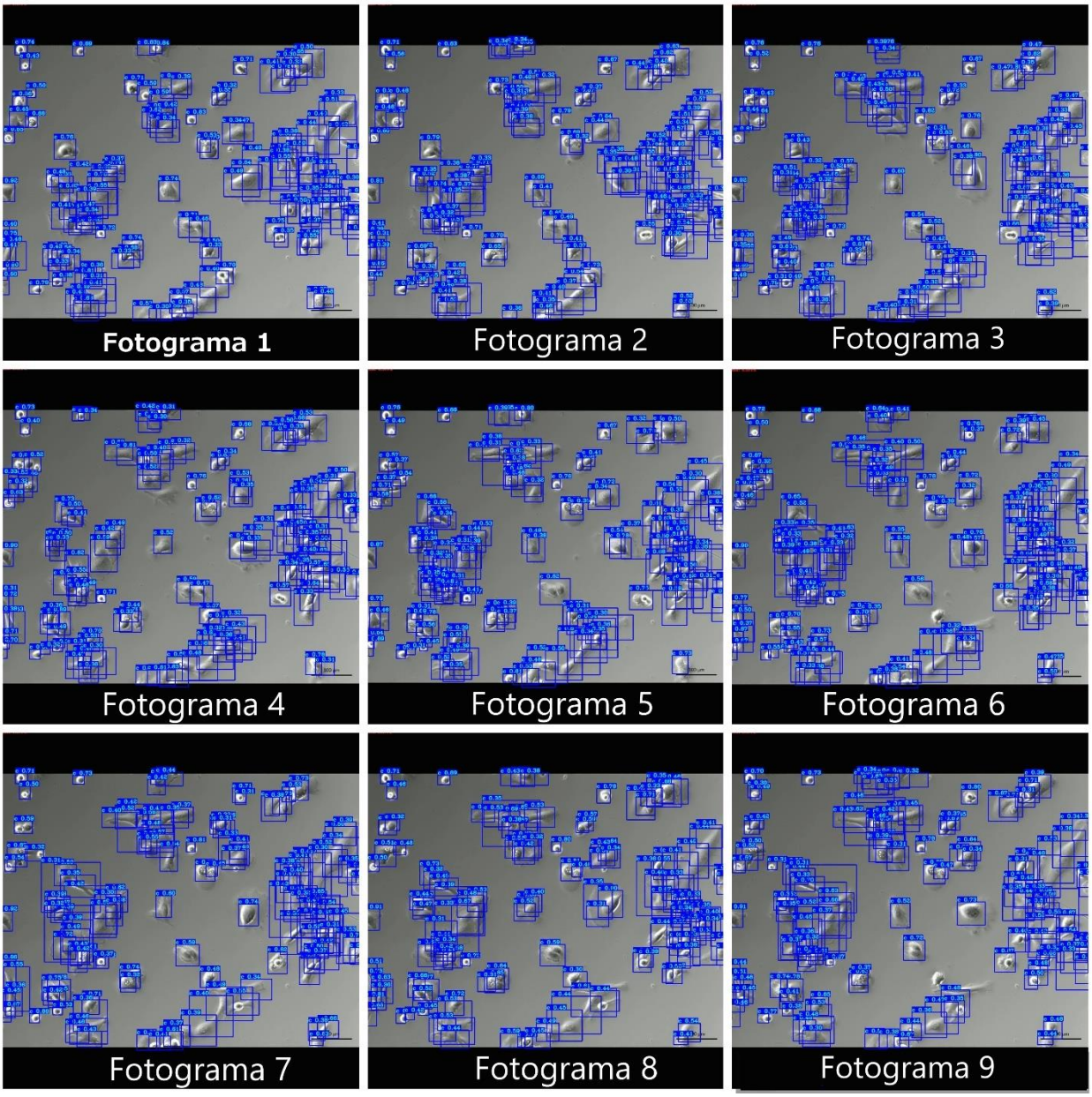

Figura 4.10.- Detección de células en secuencia de fotogramas del video `SCC40.avi` por parte del modelo.

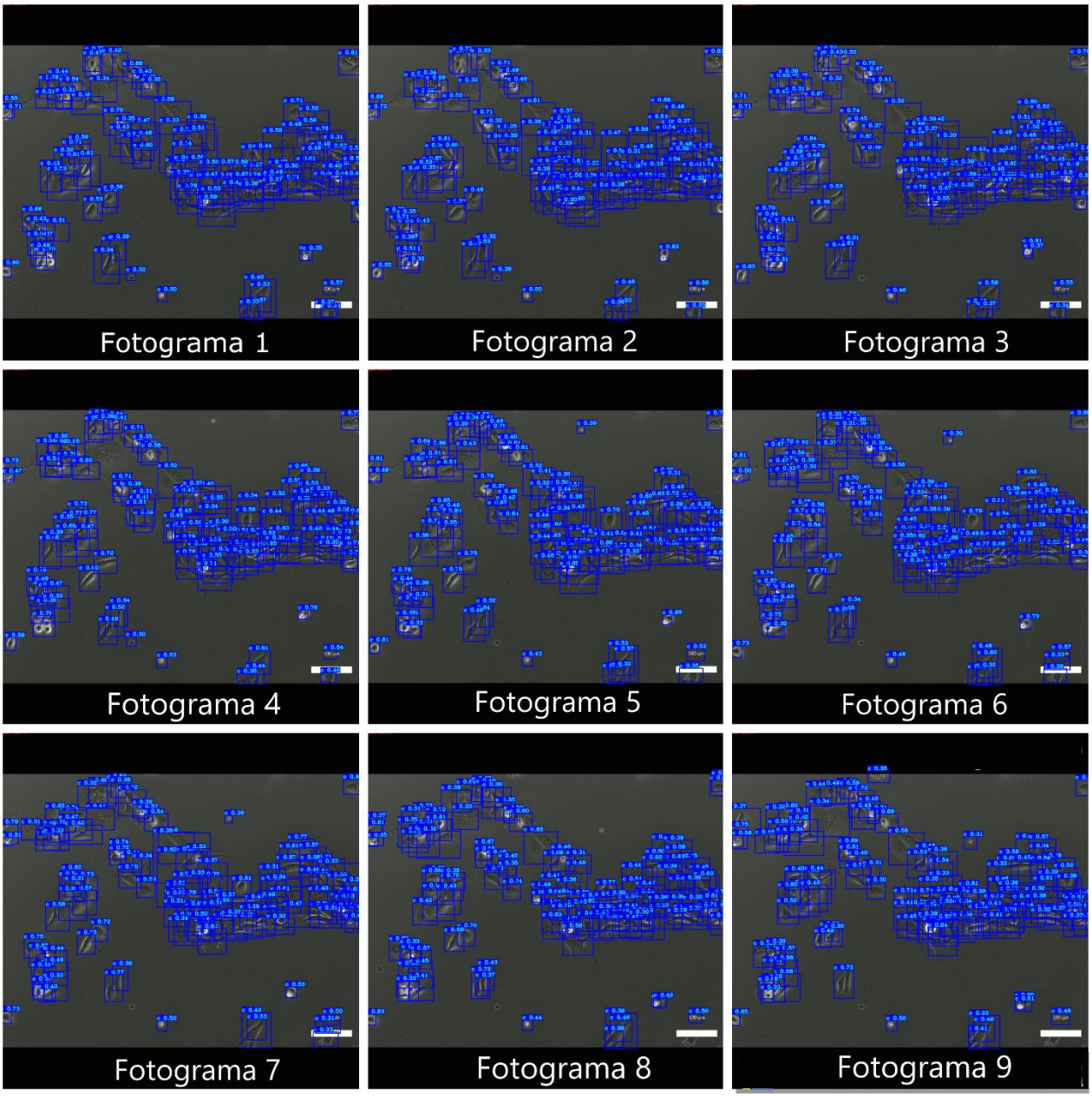

Figura 4.11.- Detección de células en secuencia de fotogramas del video `SCC38.avi` por parte del modelo.

4.4.- BONDADES DEL MODELO

- El modelo ha aprendido los patrones y características de las células y es capaz de localizar las células que aparecen en cada fotograma de los vídeos sin células sin detectar en la gran mayoría de los casos.

- Detecta correctamente las células, tanto cuando están separadas, como cuando están unidas en grupos de hasta 4 células.

4.5.- LIMITACIONES

- El modelo no funciona bien con grupos numerosos de más de cuatro células en contacto detectando mayor número de células de las que realmente hay en unos casos y menor número en otros.
- Está diseñando para únicamente para detectar células y no para clasificar los distintos tipos de líneas celulares o para separar sus partes (núcleo, citosol…).

5. Conclusiones

5.1.- APORTACIONES

- Este trabajo aporta un análisis experimental de la viabilidad de utilizar técnicas de aprendizaje profundo basadas en Python para la identificación de células individuales en vídeos in vivo.
- Se incluye una demostración de la aplicación de estas técnicas con la preparación de un modelo YOLOv3-Tiny en el *framework* TensorFlow.
- Se crea un *dataset* consistente en fotogramas de vídeos tomados de muestras de varios cultivos celulares, en los que se etiquetaron aproximadamente 2200 células mediante la plataforma Roboflow, que posteriormente fueron aumentadas a unas 15400 células mediante técnicas de *data augmentation*.
- Este trabajo, que consiste en el desarrollo de un sistema de detección automática de células en vídeos de cultivos celulares, es un paso necesario dentro del estudio realizado por investigadores del Grupo de Supervisión y Diagnóstico de Procesos Industriales (GSDPI) del área de Ingeniería de Sistemas y Automática (ISA) de la universidad de Oviedo junto al Instituto de Investigación Sanitaria del Principado de Asturias (ISPA).
- Se incluye, en forma de anexo, una guía práctica para poder retomar este proyecto en cualquier momento y seguir trabajando a partir de él.

5.2.- LÍNEAS DE FUTURO TRABAJO

Las líneas de futuro trabajo pasan primero por superar la tasa de acierto en la segmentación de células y principalmente de grupos de células en contacto. Se proponen ahora posibles enfoques para conseguir esto:

- En primer lugar, aumentar el número de imágenes del *dataset* para proporcionar más información de los patrones y características de las células al modelo en el entrenamiento.
- Considerar la opción de realizar una fase previa de entrenamiento del modelo, con un *dataset* más amplio y distinto de células ya etiquetadas, generando unos pesos o

weights con los que iniciar el posterior entrenamiento con el *dataset* de este trabajo. Esta fase, denominada *transfer learning*, permitiría trabajar con un modelo que ya ha aprendido a detectar células y al que únicamente hay que enseñarle a detectar las células específicas de este proyecto. Sería importante que este *dataset* contuviera grupos células visualmente en contacto como en el *dataset* de este proyecto para facilitarle la labor al modelo posteriormente.

- Considerar la opción de utilizar un servidor con una GPU más potente de la utilizada en posteriores entrenamientos para poder pasar del modelo YOLOv3-Tiny, utilizado en este trabajo, a YOLOv4 o YOLOv5, los cuales son modelos más potentes que previsiblemente lograrán mejores resultados.

En paralelo a las mejoras en la precisión, los resultados (outputs) del modelo (posiciones, *bounding boxes* de las células) pueden ser explotados por algoritmos de aprendizaje automático, que: 1) realicen el "tracking", emparejando cada célula con su posición en el fotograma siguiente, de forma que se pueda obtener su trayectoria; 2) analicen las trayectorias resultantes, para extraer descriptores o elaborar modelos del movimiento individual y colectivo.

6. Anexo

6.1.- GUÍA PARA LA EXTRACCIÓN DE FOTOGRAMAS EN VIDEOS

Para poder obtener un video separado en sus fotogramas se adjunta una herramienta en la carpeta `ficheros_adicionales`. Primero colocar el video en la carpeta `Extracción_de_frames`, lanzar el terminal desde dicha carpeta y por último introducir el siguiente comando modificando los apartados en negrita según corresponda (el primer y tercer apartado por el nombre del video y el segundo por la duración en segundos del video o trozo de video del cual queramos extraer los fotogramas):

"`ffmpeg -ss 00:00 -i` **`Scene9SCC42B`**`.avi -t` **`00:22`** `frame_`**`Scene9SCC42B`**`-%03d.jpg`"

6.2.- COMO ETIQUETAR IMÁGENES CON ROBOFLOW

Una vez creada una cuenta en Roboflow, el primer paso es subir el conjunto de imágenes sin etiquetar que formará el *dataset*. Para ello creamos un proyecto (ver figuras 6.1 y 6.2) y seleccionamos y subimos las imágenes a la plataforma (ver figuras 6.3 y 6.4).

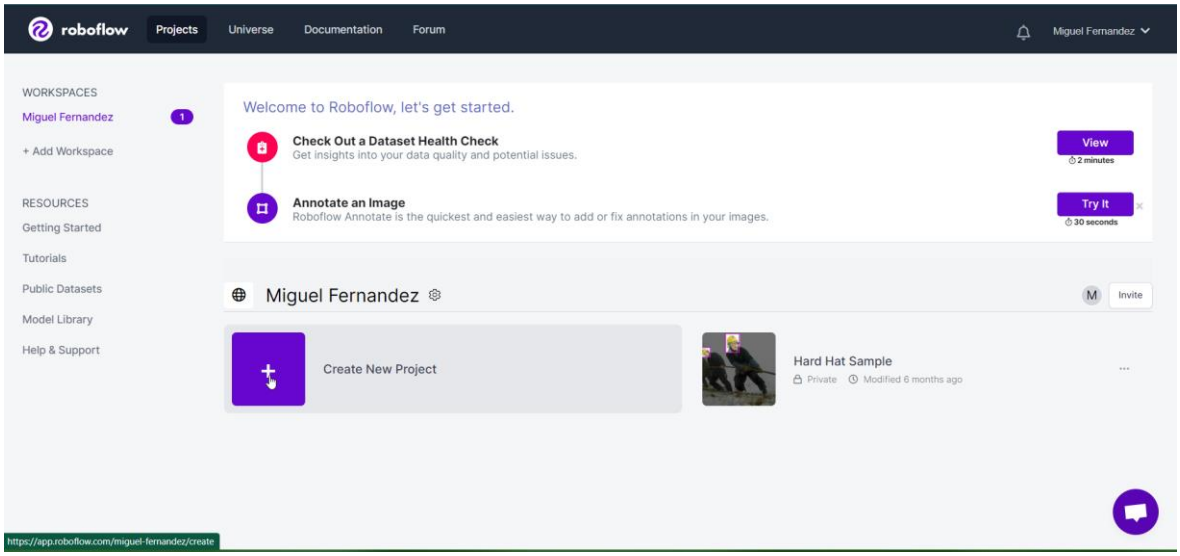

Figura 6.1.- Creación de un nuevo proyecto en Roboflow.

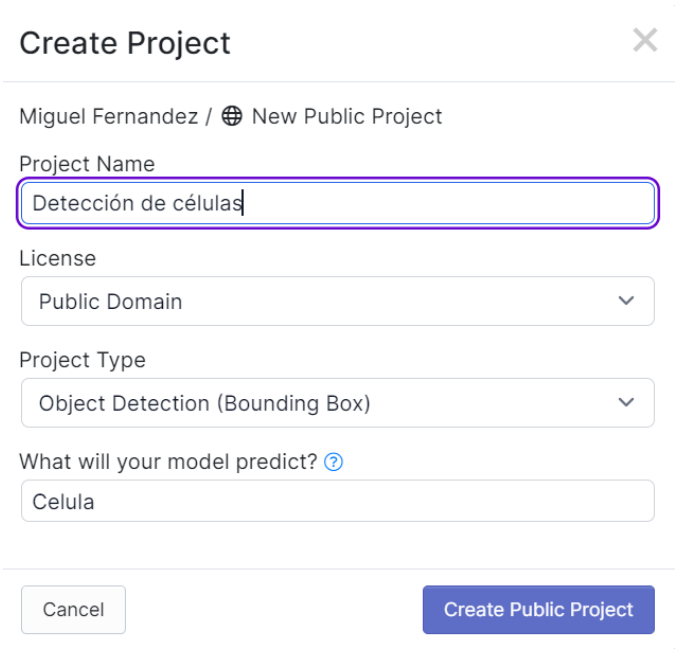

Figura 6.2.- Creación de un nuevo proyecto en Roboflow

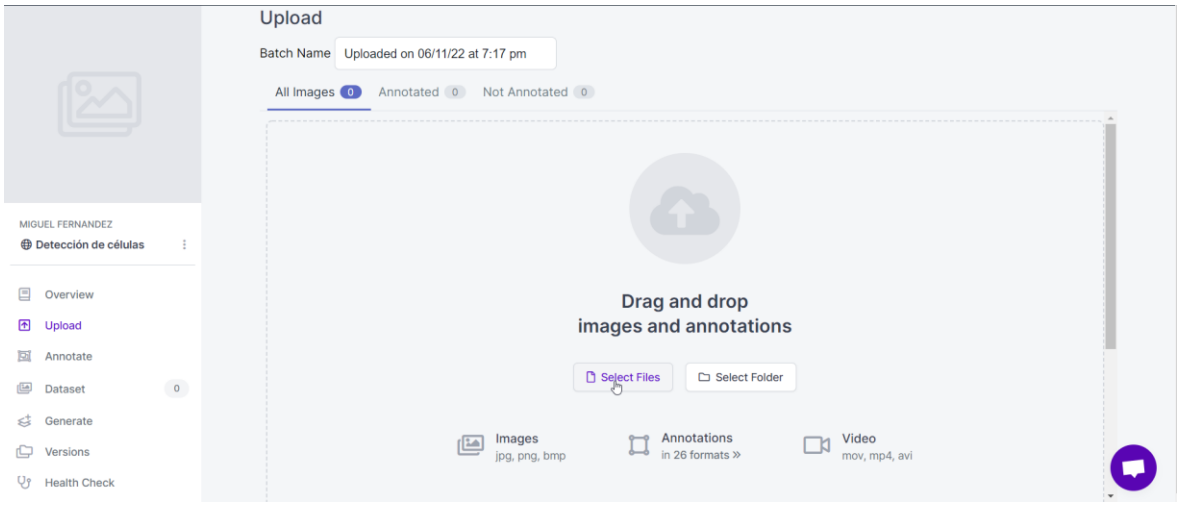

Figura 6.3.- Seleccionar las imágenes que se quieran subir a Roboflow.

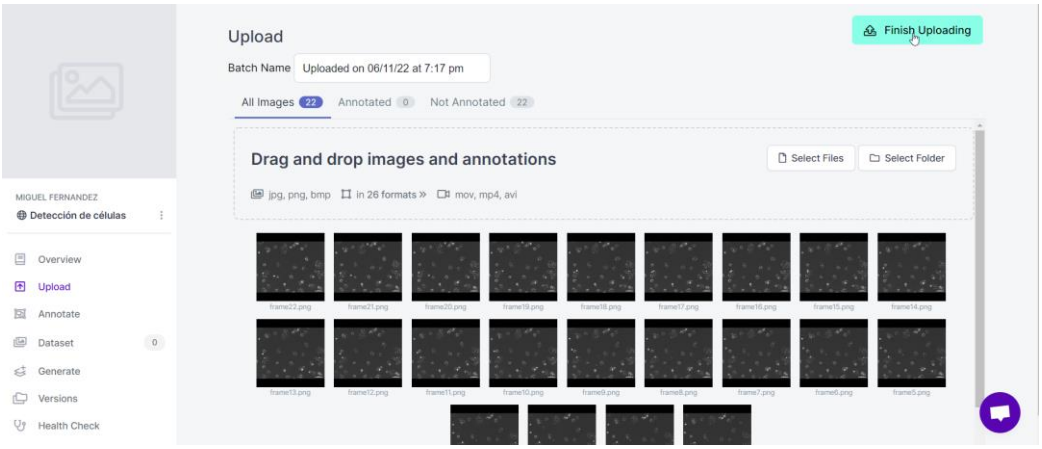

Figura 6.4.- Subir las imágenes seleccionadas a Roboflow.

Para etiquetar los objetos en las imágenes: seleccionar la herramienta de *bounding box* en el panel; segmentar por los bordes; y asignar la clase del objeto (ver Figura 6.5). Etiquetadas todas las imágenes, añadirlas al *dataset* y dividir este entre los conjuntos de imágenes de entrenamiento, validación y testeo (ver figura 6.6 y 6.7) Por último, seleccionar las técnicas de *data augmentation*, configurar los parámetros de estas y generar y exportar el modelo en el formato adecuado. Para el modelo de YOLOv3-Tiny basado en TensorFlow el formado adecuado es YOLOv3 Keras (ver figura 6.8, 6.9 y 6.10)

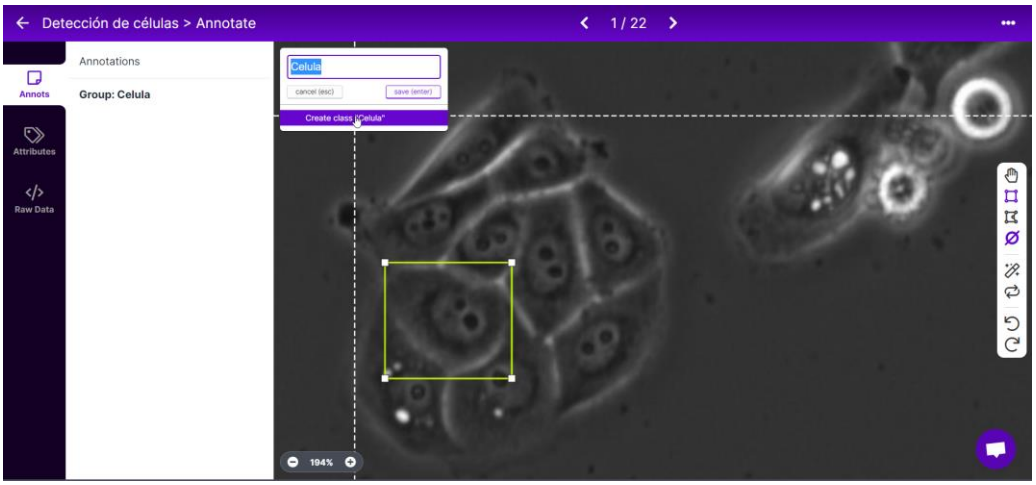

Figura 6.5.- Etiquetar objetos en Roboflow.

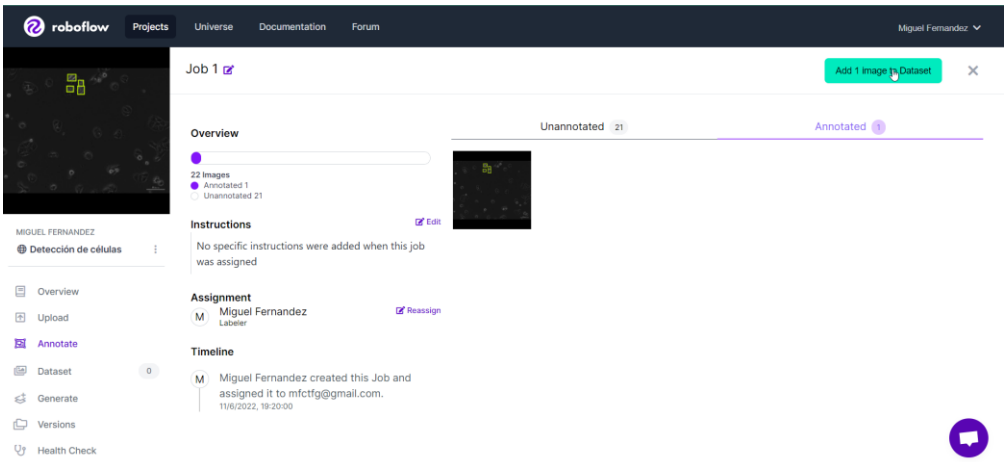

Figura 6.6.- Etiquetadas las imágenes añadirlas al *dataset.*

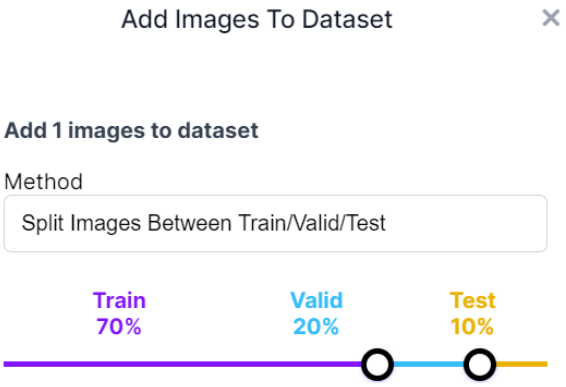

Figura 6.7.- Terminada la fase de etiquetado, dividir el *dataset* entre *trainset*, *validset* y *testset.*

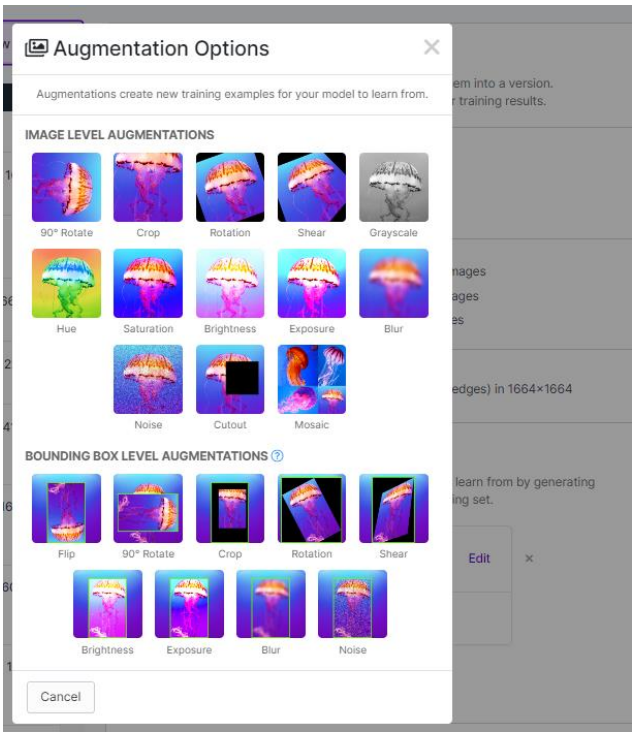

Figura 6.8.- Seleccionar las técnicas de *data augmentation*.

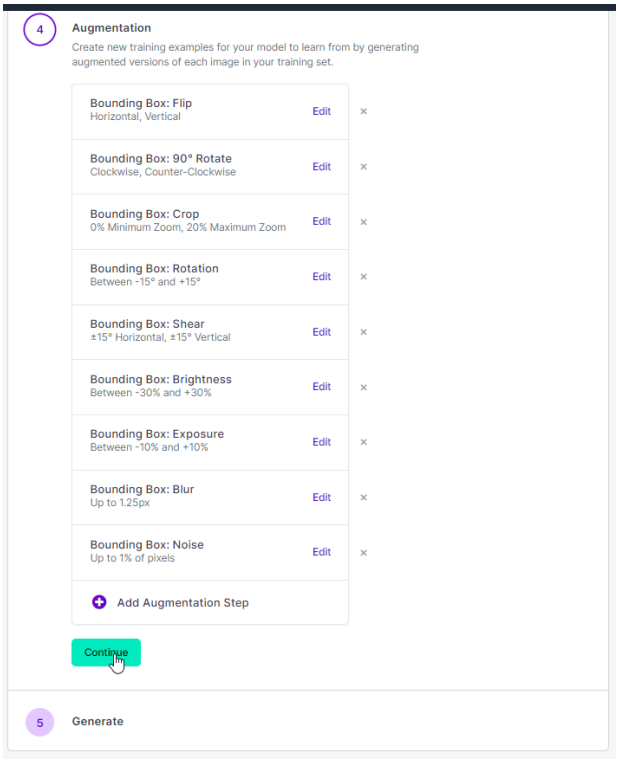

Figura 6.9.- Configurar las técnicas de *data augmentation* seleccionadas.

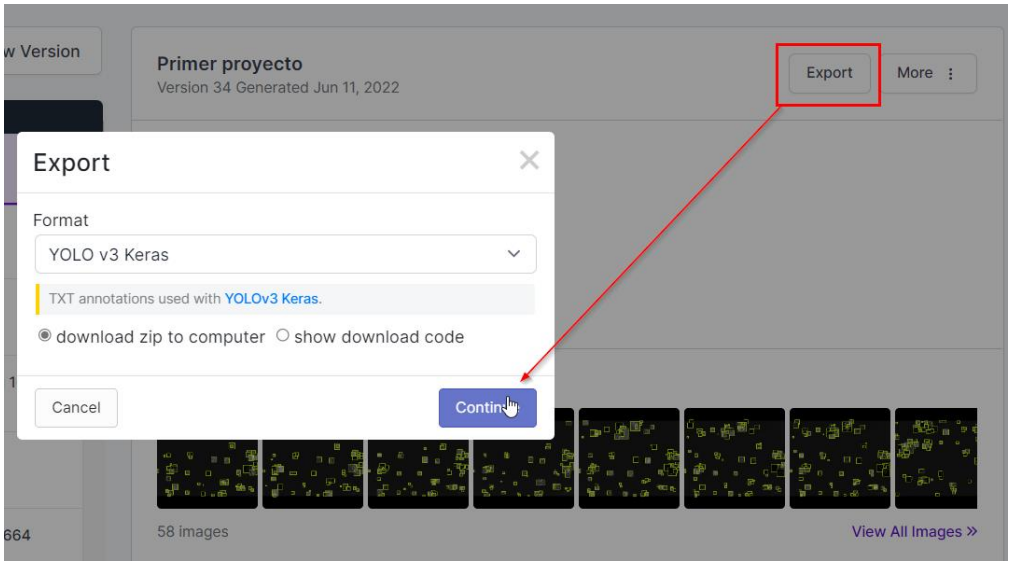

Figura 6.10.- Exportar el *dataset* en el formato adecuado.

6.3.- DESCARGA DEL MODELO Y PREPARACIÓN DEL ENTORNO VIRTUAL

- Para obtener el modelo se debe clonar en el directorio elegido para albergarlo entrando en el siguiente link de la plataforma Github: https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3.
- Crear un nuevo entorno virtual con el comando `python3 -m venv células` (terminal de Windows) y activarlo ejecutando el comando `celulas\Scripts\activate.bat`.
- Preparar el entorno virtual entrando desde el terminal a la carpeta donde se haya descargado el modelo (con el comando `cd` y la ruta a la carpeta) y descargar los requisitos del modelo mediante: `pip install -r requirements.txt`
- Descargar los *weights* pre-entrenados del modelo YOLOv3-Tiny con el siguiente comando: `wget -P model_data` https://pjreddie.com/media/files/yolov3-tiny.weights

6.4.- COMO REALIZAR UNA PRUEBA DE INFERENCIA

Para poder realizar una prueba de inferencia sobre el conjunto de videos e imágenes que componen el *testset* se debe primero añadir los *checkpoints*, el *testset*, la carpeta `data` y los archivos `detectar_celulas.py` y `config.py`, que se encuentran en la carpeta `ficheros_adicionales`, al directorio donde se haya clonado el modelo. Más concretamente añadir los *checkpoitns* a la carpeta `checkpoints`, las carpetas `testset` y `data` y el archivo `detection_celulas.py` a la carpeta principal del modelo y por último el archivo `config.py` a la carpeta `yolov3`.

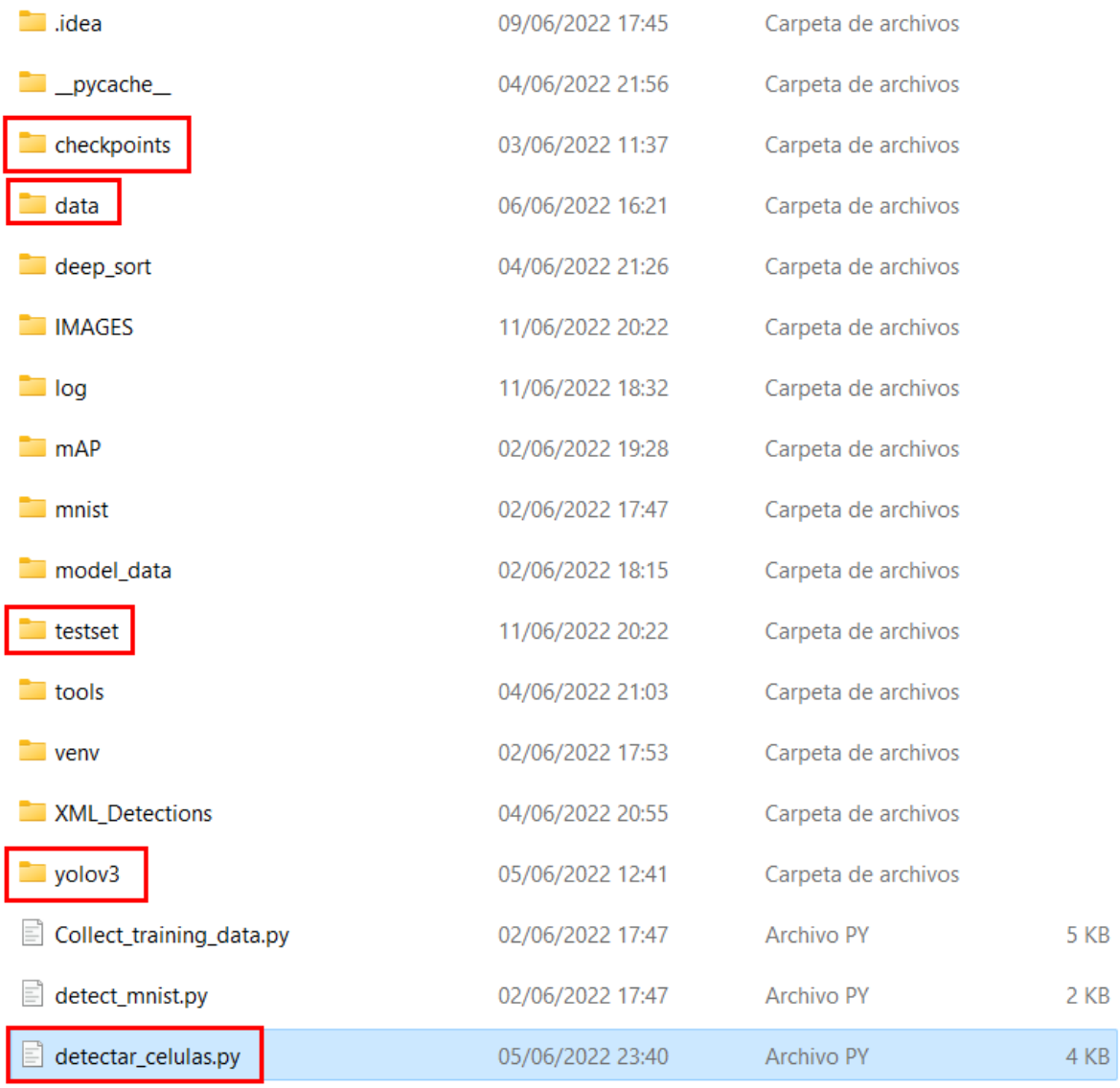

Figura 6.11.- Organización de carpetas dentro del directorio donde se ubica el modelo.

Traspasados los archivos mencionados y habiendo realizados los pasos del apartado anterior ejecutar el archivo `detectar_celulas.py` desde el directorio del modelo con el comando: `python detectar_celular.py`. Los resultados obtenidos se encontrarán en la carpeta `deteccion` dentro de la carpeta `testset`.

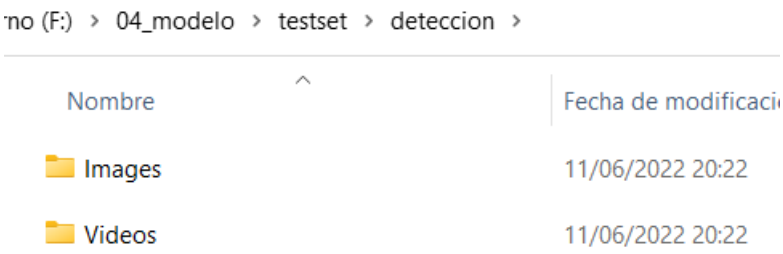

Figura 6.12.- Directorio donde localizan los resultados de la prueba de inferencia.

6.5.- ENTRENAMIENTO EN SERVIDOR

Para entrenar el modelo en el servidor se utilizó una conexión SFTP para la transferencia de archivos y una conexión SSH para la ejecución de comandos y entrenamiento. Los archivos como el *dataset* se enviaron desde el ordenador local al servidor para poder ejecutar el entrenamiento, y una vez completado este, se transfirieron los archivos resultantes como los *logs*, archivos que permiten la posterior evaluación del entrenamiento, o los *checkpoints*, archivos dónde se captura el valor exacto de todos los parámetros usados por el modelo en la época de entrenamiento dónde se ha obtenido el mejor resultado de error de validación (error que comete el modelo al detectar objetos en las imágenes del conjunto de validación cuyos valores son conocidos) (ver figura 6.11).

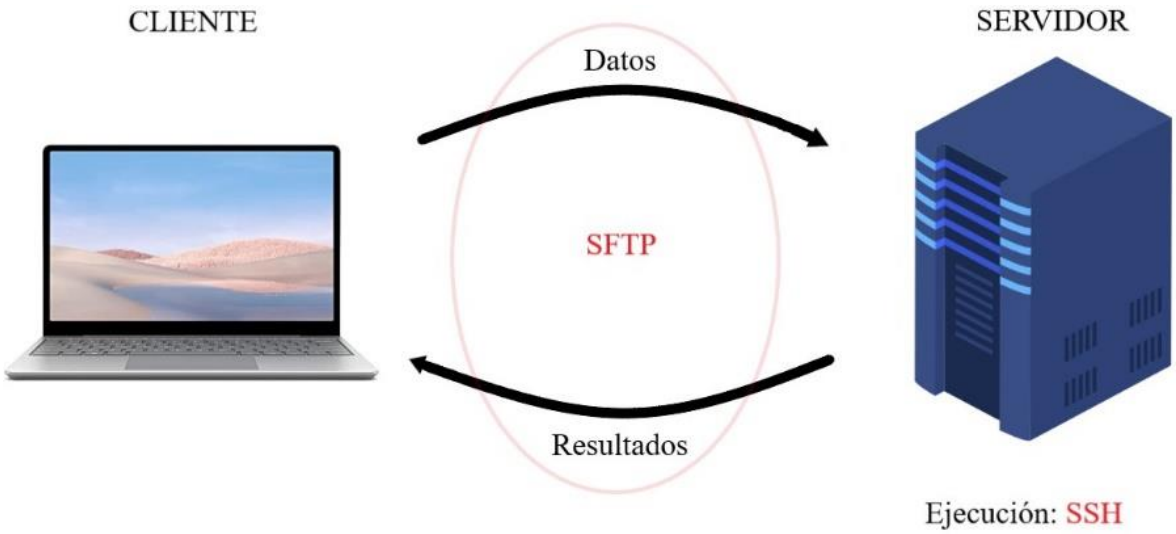

Figura 6.13.- Flujo de trabajo entre cliente y servidor

Para la transferencia de archivos mediante conexión SFTP se utilizó el programa Filezilla y para la ejecución de comandos en el servidor el programa Putty. Para la ejecución del entrenamiento se utilizó el comando `nohup`, pudiendo cerrar de esta manera la sesión en Putty y que no finalice el entrenamiento, y los comandos `&` y `>> log.txt` para realizar la tarea en segundo plano y guardar la salida en el archivo de texto `log.txt` (ver figuras 6.13 y 6.14)

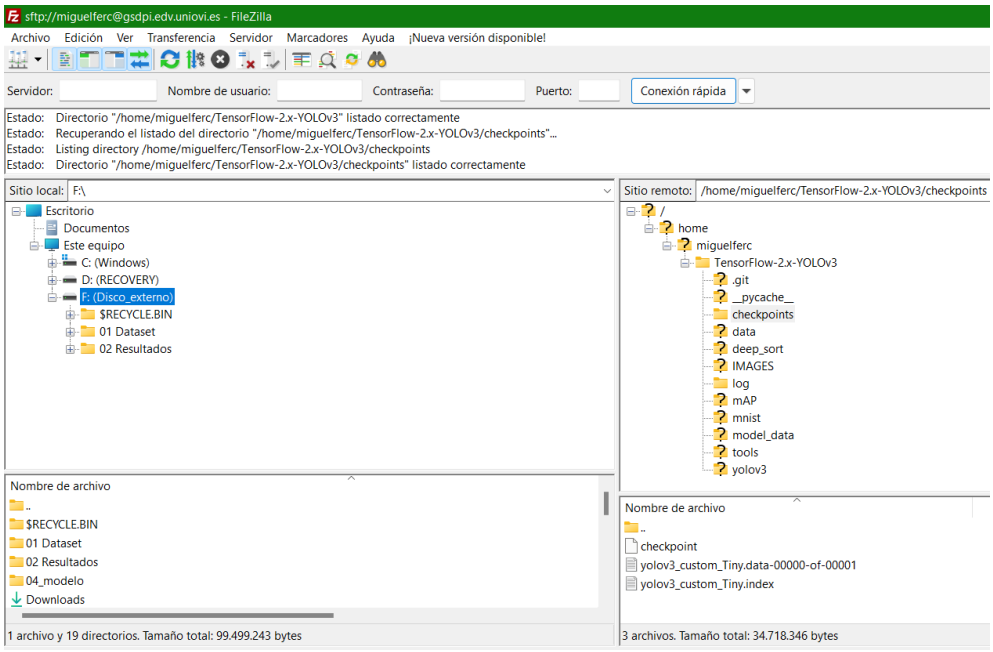

Figura 6.14.- Transferencia de archivos mediante el programa Filezilla.

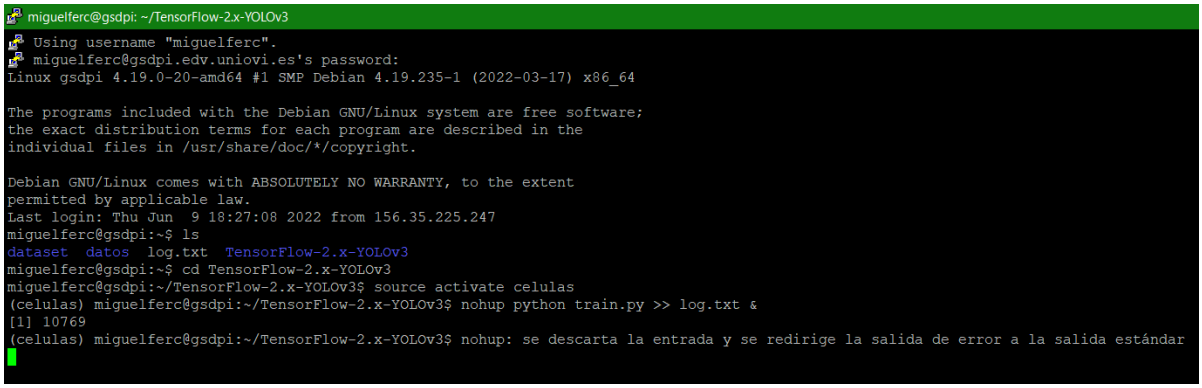

Figura 6.15.- Ejecución del entrenamiento en el servidor mediante el programa Putty.

6.6.- PRESUPUESTO

Para la elaboración del presupuesto se realiza una estimación del coste de este proyecto y se divide en recursos humanos y recursos para el desarrollo. La tabla 6.16 muestra el presupuesto del proyecto para los recursos humanos y la tabla 6.17 muestra el presupuesto de los recursos empleados para llevar a cabo el despliegue del proyecto.

Categoría	Número de personas	Nº de horas	Precio unitario	Importe total
Programador	1	150	30,00 €	4.500,00 €

Figura 6.16.- Presupuesto del proyecto para los recursos humanos.

Descripción	Unidades	Precio unitario	Tiempo de uso neto (meses)	Amotización	Importe amortizado	Importe total
Ordenador personal	1	700,00 €	10	21%	145,83 €	145,83 €
Servidor (PC DELL PRECISION TOWER 3620)	1	2.590,00 €	0,5	1%	26,98 €	26,98 €
Tarjeta gráfica (Gigabyte RTX2080 Ti Turbo Modelo TurboFun)	1	1.115,00 €	0,5	1%	11,61 €	11,61 €
Roboflow	1	0,00 €	1	2%	0,00 €	0,00 €
Pycharm	1	0,00 €	1	2%	0,00 €	0,00 €
Google Colab	1	0,00 €	0,5	1%	0,00 €	0,00 €
Supervisely	1	0,00 €	0,5	1%	0,00 €	0,00 €
					Total	184,43 €

Figura 6.17.- Presupuesto de los recursos empleados para llevar a cabo el despliegue del proyecto.

El coste total del proyecto asciende a cuatro mil seiscientos ochenta y cuatro euros con cuarenta y tres céntimos.

Lista de abreviaturas

RNN	*Recurrent Neural Network*
LSTM	*Long Short-Term Memory*
CNN	*Convolutional Neural Network*
ReLU	*Rectified Linear Unit*
ResNet	*Residual Network*
YOLO	*You Only Look Once*
GSDPI	Grupo de Supervisión y Diagnóstico de Procesos Industriales
ISA	Ingeniería de Sistemas y Automática
ISPA	Instituto de Investigación Sanitaria del Principado de Asturias
GPU	*Graphics Processing Unit*
CPU	*Central Processing Unit*
TPU	*Tensor Processing Unit*
SSH	*Secure Shell*
SFTP	*Secure File Transfer Protocol*

Bibliografía

[1] J. S. K. C. J. &. C. D. J. LaChance, «Learning the rules of collective cell migration using deep attention networks.,» *PLoS computational biology 18(4): e1009293*, 2022.

[2] R. E.-M. S. Mayor, «The front and rear of collective cell migration.,» *Nat Rev Mol Cell Biol*, vol. 17, p. 97–109, 2016.

[3] C. González Garcia, E. Rolando Núñez-Valdez y V. García-Díaz, «A Review of Artificial Intelligence in the Internet of Things,» *MDE Research Group, Department of Computer Science, University of Oviedo, Oviedo (Spain)*, 24 March 2018.

[4] Y. Bengio, A. Courville y P. Vincent, «Representation Learning: A Review and New Perspectives,» 24 Junio 2012.

[5] Y. LeCun, Y. Bengio y G. Hinton, «Deep learning.,» *Nature*, vol. 525, p. 436–444, 2015.

[6] IBM, «Machine learning y ciencia de datos,» [En línea]. Available: https://www.ibm.com/es-es/analytics/machine-learning#:~:text=Deep%20learning%20es%20un%20m%C3%A9todo,patrones%20de%20datos%20no%20estructurados..

[7] «Instituto Nacional del Cáncer de EE.UU: Célula,» [En línea]. Available: https://www.cancer.gov/espanol/publicaciones/diccionarios/diccionario-cancer/def/celula.

[8] « Instituto Nacional del Cáncer de EE.UU: ¿Qué es el cáncer?,» [En línea]. Available: https://www.cancer.gov/espanol/cancer/naturaleza/que-es#:~:text=Las%20c%C3%A9lulas%20cancerosas%20se%20originan,multiplicarse%20o%20que%20deben%20destruirse..

[9] «Ibiantech: El ayer y hoy de las técnicas y medios de cultivo celular,» [En línea]. Available: https://www.ibiantech.com/el-ayer-y-hoy-de-las-tecnicas-y-medios-de-cultivo-celular/.

[10] E. Meijering, «Cell Segmentation: 50 Years Down the Road [Life Sciences],» *IEEE Signal Processing Magazine*, vol. 29, nº 5, pp. 140-145, Sept. 2012.

[11] Z. Tao, D. Zhizhao, W. Jin y L. Guodong, «Research on Distance Transform and Neural Network Lidar Information Sampling Classification-Based Semantic Segmentation of 2D Indoor Room Maps,» *Scientific Figure on ResearchGate*, Febrero 2021.

[12] F.-Y. Chu, «On the origin of shape fluctuations of the cell nucleus,» 2017. [En línea]. Available: http://www.pnas.org/cgi/doi/10.1073/pnas.1702226114.

[13] A. Kan, «Machine learning applications in cell image analysis.,» *Immunology and Cell Biology*, 2017.

[14] Y. Xie, X. Fuyong y X. Shi, «Efficient and robust cell detection: A structured regression approach,» *Medical Image Analysis*, vol. 44, pp. 245-254, 2018.

[15] «IBM, El modelo de redes neuronales,» 17 08 2021. [En línea]. Available: https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=networks-neural-model.

[16] «Towards Data Science,» 5 Enero 2017. [En línea]. Available: https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7.

[17] 27 Mayo 2022. [En línea]. Available: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm#types_of_algorithms_used_in_deep_learning.

[18] J. I. Bagnato, «Aprende machine learning: ¿Cómo funcionan las convolutional neural networks? Visión por ordenador,» 29 Noviembre 2018. [En línea]. Available:

https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/.

[19] «MathWorks: Redes neuronales convolucionales,» [En línea]. Available: https://es.mathworks.com/discovery/convolutional-neural-network-matlab.html.

[20] K. Koos, G. Oláh y T. Balassa, «Automatic deep learning-driven label-free image-guided patch clamp system,» *Nature communications,* 10 Febrero 2021.

[21] A. Tao, J. Barker y S. Sarathy, «DetectNet: Deep Neural Network for Object Detection in DIGITS,» 11 Agosto 2016. [En línea]. Available: https://developer.nvidia.com/blog/detectnet-deep-neural-network-object-detection-digits/.

[22] J. Redmon y A. Fahardi, «YOLOv3: An Incremental Improvement».

[23] J. Redmon, A. Farhardi, S. Divvala y R. Girshick, «You Only Look Once: Unified, Real-Time Object Detection».

[24] S. Sie y R. Girshick, «Aggregated Residual Transformations for Deep Neural Networks,» 2016.

[25] S. Ren, K. He, R. Girshick y J. Sun, «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,» 4 Junio 2015.

[26] J. I. Bagnato, «Aprende Machine Learning: Clasificación con datos desbalanceados,» 16 Mayo 2019. [En línea]. Available: https://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/.

[27] A. Bochkovskiy y W. Chien-Yao, «YOLOv4: Optimal Speed and Accuracy of Object Detection.,» *Institute of Information Science Academia Sinica, Taiwan,* 2020.

[28] N. A Hamilton, R. S Pantelic y K. Hanson, «Fast automated cell phenotype classification,» *BMC Bioinformatics,* 2007.