Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

Detección de instrumentos quirúrgicos en imágenes para cirugía laparoscópica robotizada

Autor:

Sanz Gobernado, Daniel

Tutor(es):

De la Fuente López, Eusebio

Departamento de Ingeniería de sistemas y automática

Valladolid, mayo 2021.

Resumen

El objetivo del actual proyecto es el desarrollar un algoritmo de inteligencia artificial que, a partir del procesamiento de imágenes, sea capaz del seguimiento y detección de instrumental quirúrgico y gasas en operaciones laparoscópicas. Dicho algoritmo será integrado en un robot quirúrgico que le permitirá colaborar con el cirujano con una mayor autonomía. La detección y seguimiento del material quirúrgico evitará adicionalmente que se den situaciones de gasas accidentalmente retenidas en el interior del cuerpo, evitando así este error que puede tener consecuencias fatales para el paciente.

En los algoritmos de seguimiento implementados en este TFG, se ha intentado comparar distintos tipos de redes neuronales convolucionales, buscando unos resultados lo más precisos posibles junto con velocidades de procesamiento altas y/o en tiempo real

Palabras clave

Cirugía laparoscópica; Redes Neuronales Convolucionales; YOLO; MATLAB; Detección de objetos en imágenes; Gasa; Instrumental quirúrgico.

Abstract

The aim of the current proyect is to develop an artificial intelligence algorithm that, from image processing, could be able to detect and automatic monitor gauzes and surgical instruments on laparoscopic surgeries. This algorithm will be integrated into surgical robot which will allow to collaborate with the surgeon with greater autonomy. The detection and automatic monitor of surgical material will additionally avoid situations of accidental retained surgical material inside the patient´s body, avoiding this way this error that could bring fatal consequences for the patient.

On the monitoring algorithms implemented on this Final Proyect, it was tried to compare different types of convolutional neural networks, looking for the most accurate possible results along with high and/or real-time processing speeds.

Keywords

Laparoscopic surgery; Convolutional Neural Networks; YOLO; MATLAB; Object detection in images; Gauze; Surgical instruments.

Agradecimientos

A mi familia y amigos por siempre apoyarme y ayudarme cuando lo necesitaba, a Guillermo Sánchez Brizuela por proporcionar el *dataset* de gasas quirúrgicas y, por último, y no menos importante, a mi tutor Eusebio por toda la ayuda brindada durante estos meses.

Índice general

Índice de tablas

Índice de figuras

1. Introducción

Las operaciones quirúrgicas han tenido una gran evolución a lo largo de la historia, desde un intento de imitar a los animales en la prehistoria, limpiando lingualmente las heridas y parando la hemorragia por compresión, pasando por la cirugía árabe de Abulcasis, el cual ya utilizaba la cauterización para detener sangrados, así como la extirpación de tumores y el tratamiento de luxaciones, fracturas y hernias.

Posteriormente, se produce una revolución en cuanto a la cirugía en el siglo XVI, debido a la introducción de la catedra en este campo en las universidades y de nuevo material quirúrgico, además de contar como revulsivo la gran cantidad de conflictos bélicos desarrollándose en Europa, apareciendo en escena al que después será considerado como el padre de la cirugía moderna, Ambroise Paré, cambiando de forma radical el tratamiento de las heridas y la amputación de miembros. [1]

En los siglos posteriores, se consigue evitar las hemorragias, la infecciones y reducir el dolor con las recientes hemostasia, antisepsia y anestesia, consiguiendo tras esto un rápido desarrollo, destacando el uso de antibióticos, tratamientos posquirúrgicos y los trasplantes.

En la actualidad, se ha conseguido operar mediante cirugías laparoscópicas, donde se reduce drásticamente los postoperatorios gracias a realizar pequeñas incisiones en el paciente, evitando así las grandes incisiones de las operaciones convencionales. Además, está comenzando el uso de la telemedicina, haciendo que no sea necesario que el cirujano se encuentre en el mismo quirófano o incluso en la misma ciudad que el paciente. Por último, la creciente importancia de las cirugías robóticas, siendo un robot el que realiza la operación, aunque por el momento, sea el mismo cirujano el que controla este robot desde un ordenador o consola.

En este proyecto se va a realizar el seguimiento y detección de gasas e instrumental quirúrgico en operaciones laparoscópicas, implementando para esto, redes neuronales artificiales entrenadas mediante *Deep learning* o entrenamiento profundo con el objetivo de implementar estos algoritmos en un robot quirúrgico que asista al cirujano de forma autónoma.

Se elaborará a continuación, una introducción teórica para entender estas últimas cirugías, así como los objetivos de este proyecto. Posteriormente, se mostrará cómo se ha implementado el modelo, con su respectivo código, teniendo para finalizar, una serie de resultados y conclusiones obtenidos de los algoritmos de detección empleados. Presentando, por último, posibles mejoras o líneas de investigación futuras.

1.1. Cirugía laparoscópica

La cirugía laparoscópica es un tipo de cirugía relativamente nueva, la cual es una alternativa mínimamente invasiva a las cirugías abiertas convencionales. Esta cirugía está siendo muy utilizada debido a que tiene una tasa de recuperación más rápida y causa menor daño al paciente, para conseguir esto, se intenta evita total o parcialmente las heridas quirúrgicas en la pared abdominal, además de conseguir una menor pérdida de sangre, una manipulación visceral más cuidadosa y una menor manipulación intestinal durante la operación. [2]

Ilustración 1 - Comparación entre incisiones en una cirugía convencional (izquierda) y una cirugía laparoscópica (derecha) [3]

Para ello, se introduce, por unas pequeñas incisiones, un laparoscopio, que es una lente óptica provista de luz y de una cámara, permitiendo así que el cirujano pueda visualizar el interior del paciente.

En la operación, se introducen los trocares, tubos huecos para realizar una abertura en el cuerpo del paciente por donde posteriormente, se introducirán los instrumentos para manipular, cortar y coser tejidos. A continuación, se introduce el laparoscopio, además de dióxido de carbono por uno de los trocares para levantar la pared abdominal, consiguiendo espacio de trabajo para el cirujano. Por el resto de trocares se introducen los instrumentos de operación. [4]

1.2. Cirugía robótica

La cirugía robótica es considerada por muchos el futuro de la cirugía ya que no solo ha conseguido actualizar la forma de realizar cirugía de forma más precisa, sino que también supone un cambio en la formación de nuevos cirujanos.

Antes de la invención de este tipo de cirugías, los aprendices de cirujanos tenían que adquirir experiencia sobre pacientes reales mediante un método supervisado de prueba y error, gracias a la cirugía robótica, los aprendices pueden aprender mediante simulaciones realizadas con los mismos robots y sistemas de realidad virtual, además de tener modelos blandos recreando los tejidos del cuerpo humano. Esta nueva forma de aprendizaje supone una disminución enorme en el tiempo de formación, ya que, sin estos sistemas, se depende del número de pacientes a operar, consiguiendo además de esta forma que no se comprometa la integridad del paciente. [5]

Para la realización de la operación, el cirujano se coloca delante de la consola maestra donde se encuentran unos binoculares que dotan al cirujano de una visión 3D, además de ofrecerle una imagen magnificada y ampliada de los órganos que opera. Posteriormente, el asistente del cirujano realiza una pequeña incisión al paciente, por donde se introducirán más adelante los instrumentos del robot. Una vez introducidos los instrumentos, el cirujano controla los brazos del robot mediante la consola maestra, recibiendo estos datos a la velocidad de la luz, haciendo posible que el robot reproduzca los movimientos con exactitud en tiempo real. [6] [7]

Ilustración 2 - Cirujano realizando una cirugía robótica desde la consola maestra [8]

La cirugía robótica comprende todo tipo de cirugías en las que se utilicen robots, esto hace que se consiga una serie de ventajas e inconvenientes de la utilización de este tipo de tecnologías.

Ventajas:

- Mayor precisión debido a la ejecución de las acciones ordenadas por el cirujano, ayudándose de un sistema de cómputo. Además, se implementa un filtro de temblor, instrumentos articulados y un control de velocidad de los movimientos, permitiendo que sea más fácil realizar suturas y atar nudos.

- Visión tridimensional a diferencia de la visión bidimensional de la cirugía laparoscópica.
- Se mejora el postoperatorio, consiguiendo que los pacientes regresen a sus actividades cotidianas en un tiempo menor a 7 días
- Permite la realización de operaciones por telemedicina.

Inconvenientes:

- Menor sensación táctil del tejido.
- Movimiento limitado debido a que el cirujano debe controlar todos los brazos robóticos, siendo imposible que ayudantes le ayuden durante la cirugía.
- Tiempos de operación más largos que en laparoscopia u operaciones convencionales, lo que supone un tiempo bajo anestesia mayor para el paciente, conduciendo a un mayor riesgo para este.

1.2.1. AESOP

El robot AESOP o sistema endoscópico automatizado para posicionamiento óptimo, fue el primer robot utilizado en una cirugía laparoscópica, tras ser aprobado por la FDA (Food and Drug Administration) en 1994. El AESOP fue diseñado por la empresa Computer Motion estadounidense, aunque no fue la primera versión, ya que esta empresa fue fundada por una concesión con la NASA para fabricar un brazo robótico para su programa espacial. Posteriormente, este brazo fue modificado, con el objetivo de remplazar al camarógrafo laparoscópico, para sostener un laparoscopio, ya que, este robot no puede distraerse o cansarse como lo podría hacer una persona tras largas horas de cirugía, consiguiendo que la imagen de video no tiemble o se salga del campo de operación.

Ilustración 3 - Robot AESOP [9]

Las primeras versiones del AESOP eran controladas por el cirujano mediante un pedal o control manual, mientras que versiones más recientes permitían un control por voz. Este control de voz es único para cada cirujano, ya que este graba los comandos de voz en una tarjeta de sonido, teniendo que ser remplazada esta tarjeta si se cambia de cirujano durante la operación. Estos comandos de voz guardan tres posiciones distintas del robot, siendo bastante útil para maniobras repetitivas como suturas.

Además, AESOP cuenta con ciertas protecciones para no dañar al paciente. Al principio de la cirugía, el cirujano fija un límite al brazo del robot, consiguiendo que se evite que el brazo lo sobrepase y dañe al paciente. [10]

1.2.2. ZEUS

El sistema robótico ZEUS fue desarrollado por la empresa Computer Motion con el fin de poder realizar teleoperaciones desde lugares remotos como el espacio o zonas de guerra, para esto, se basaron en un sistema maestro/esclavo, utilizando como base su robot AESOP, utilizado, en un principio, en operaciones cardíacas, aunque posteriormente se empezó a utilizar en otro tipo de especialidades como urología, ginecología, cirugía general…

En octubre de 2001, la FDA concedió permiso limitado para realizar operaciones abdominales en los Estados Unidos. En la actualizad, los sistemas robóticos ZEUS y Da Vinci son los más utilizados en cuanto a cirugías cardíacas y abdominales.

Zeus consiste en dos subsistemas robóticos llamados *lado-paciente* y *lado-cirujano.*

Ilustración 4 - Lado-paciente (Izquierda) y lado-cirujano (Derecha) [11]

El lado-paciente consiste en tres brazos robóticos independientes y acopladas a la mesa de operaciones, siendo uno de ellos el sistema robótico, comandado por voz, AESOP, mientras que los otros dos brazos has sido adaptados para sostener el instrumental quirúrgico, convirtiendo las entradas introducidas por el cirujano en movimientos quirúrgicos. Entre el instrumental quirúrgico disponible para conectar a los brazos robóticos se puede encontrar pinzas, tijeras, ganchos… Por otro lado, el lado-cirujano consiste en una consola, la cual es la encargada de transmitir los movimientos realizados por el cirujano sobre los mangos a los brazos robóticos y un monitor donde se proyecta la imagen en tres dimensiones por el brazo camarógrafo, resultando en una cirugía no tan inmersiva como se obtiene mediante el sistema robótico Da Vinci. [10] [12]

1.2.3. Da Vinci

El sistema robótico Da Vinci fue desarrollado por Intuitive Surgical Inc, incorporándose este al mercado en 1999 pero no fue hasta el 4 de junio del 2000 cuando la FDA autorizo su utilización en cirugías abdominales. Esta primera versión contaba con tres brazos robóticos similares al sistema robótico ZEUS, siendo uno de ellos un brazo que sujeta la cámara, mientras que los dos brazos complementarios sostienen los instrumentos. Estos brazos cuentan a mayores con una especie de muñeca, consiguiendo así seis grados de libertad y dos grados de rotación axial. En 2003, aparece la segunda versión, implementando un cuarto brazo robótico además de un mejor control de retracción.

Ilustración 5 - Carro del paciente. Da Vinci (2003) [13]

En 2006, sale al mercado el Da Vinci S consiguiendo una imagen de alta definición y una visualización de múltiples imágenes. En 2009, aparece la versión Da Vinci Si donde se introducen las llamadas nuevas tecnologías o recursos técnicos:

- Capacidad para operar desde una única entrada al paciente, reduciendo las complicaciones postoperatorias.
- Sellador de vasos, permite sellar y cortar vasos sanguíneos de un máximo de 7mm de diámetro.
- Endograpadora, este instrumento no estaba disponible en versiones anteriores teniendo que realizarlo un asistente del cirujano.
- Simulador, permitiendo a los cirujanos practicar cirugías sin necesidad de operar en animales o humanos.
- Red Da Vinci

En 2014, se aprueba la versión más utilizada en todo el mundo. Aunque esta versión no sería la última, si la más importante, contando como mejoras la utilización de un sistema de brazos robóticos suspendidos sobre una plataforma móvil, además de conseguir una imagen real de la zona donde está operando el cirujano debido a la incorporación de doble monitor en la consola, doble procesamiento de imagen, zoom digital de alta definición con posibilidad de 10 aumentos, consiguiendo que el cirujano obtenga movimientos intuitivos y la posibilidad de la alineación del eje visual y motor.

Ilustración 6 - Carro del paciente (Izquierda) y Consola quirúrgica (Derecha). Da Vinci Xi [13]

La consola quirúrgica permite al cirujano controlar el carro del paciente mediante los pedales y los manipuladores a partir de una visión tridimensional y ampliada, además gracias a los manipuladores especiales del sistema Da Vinci se consigue reproducir en el carro del paciente todos los movimientos de la mano, muñeca y dedos del cirujano en tiempo real.

Por último, Da Vinci cuenta con un componente a mayores que el sistema robótico ZEUS, este componente es la torre de visión, siendo la unidad encargada del procesamiento y elaboración de imágenes, además de portar un monitor de 24 pulgadas y todo el instrumental quirúrgico necesario para la operación. [10] [13]

Ilustración 7 - Torre de visión. Da Vinci Xi [13]

1.3. Telemedicina

La telemedicina, o como se está llamando actualmente eSalud, es la prestación de servicios médicos a distancia, aunque comúnmente se piensa que la telemedicina solo comprende la misma operación o cirugía de un paciente, también abarca tanto el diagnostico como el seguimiento de un paciente tras el operatorio, esto es posible gracias al desarrollo de las tecnologías de la información y las comunicaciones. [14]

Una de las primeras intervenciones quirúrgicas telemáticas fue realizada por el doctor Jacques Marescaux en 2001 desde Nueva York, cuando extrajo la vesícula biliar de un paciente de 68 años a más de 14000 km de distancia, operando un brazo robótico situado en un quirófano de Estrasburgo.

La telemedicina permite brindar ayuda a médicos en situaciones de emergencia o desastres, aunque la mayor ventaja es la capacidad de brindar cuidados a domicilio a ancianos o enfermos crónicos, además de empezar a ser posible una hospitalización a domicilio, esto supone un gran avance en tiempos actuales de pandemia mundial.

Según como se trate los datos, se pueden distinguir tres modos de aplicación [15]:

- Modo real-time: permite la comunicación en tiempo real, garantizando la interacción directa entre médico y paciente como en una consulta convencional, haciendo que la toma de decisiones sea inmediata. Además, este modo de aplicación se puede realizar mediante médicos interactuando de forma telemática y un médico presencial que se encarga de tratar las molestias del paciente, un ejemplo de esto se daba ya en 1995 entre la clínica Mayo estadounidense y el Hospital Real de Ammán en Jordania.
- Modo store-and-forward: se centra en almacenar datos médicos de los pacientes para una posterior transmisión para diagnóstico. Estos datos médicos pueden ser imágenes, grabaciones de audio y video o cualquier prueba realizada al paciente como radiografía o resonancias. Es una variante de la consulta clásica en la que, en lugar de estar el médico y el paciente en el mismo lugar y tiempo, se toman las pruebas oportunas y el consultante realizará un diagnostico en el momento que esté disponible, por lo que la desventaja es no tener una respuesta inmediata. Un ejemplo de este modo en nuestro país es el “Preguntas a Expertos”, servicio proporcionado por INFOMED.
- Modo híbrido: es una fusión entre los modos anteriores, en los que se envían los datos previamente al consultor y se realiza un examen en tiempo real al paciente si se considera oportuno.

1.4. Material quirúrgico retenido

El material quirúrgico olvidado o retenido, describe una masa dentro del cuerpo del paciente y la consiguiente respuesta del organismo ante un cuerpo extraño, supone una grave complicación para el postoperatorio del paciente. Aunque los hospitales cuentan con sistemas para evitar este tipo de errores humanos, siempre existe un riesgo de que se produzcan. La recomendación de la AORN (Association of periOperative Registered Nurses) es seguir las siguientes medidas:

- Explorar la cavidad antes del cierre.
- Recuento de todo el material quirúrgico antes y después de la operación, se deben tener en cuenta compresas, gasas e instrumental quirúrgico, siendo obligatorio dejar registrado la ausencia del recuento si se produce una emergencia vital del paciente.
- Los objetos punzantes deben ser contados en todos los procedimientos, además de no permitirse cortar gasas o compresas.
- Todos los procedimientos se deben revisar anualmente, además de estar disponible a todo el personal del quirófano.

La tasa de material quirúrgico retenido en operaciones quirúrgicas oscila entre 1:1000 y 1:5000 operaciones, debido a que algunos pacientes son asintomáticos y nunca llega a detectarse, siendo el 69% de los materiales retenidos gasas y el 31% restante instrumental quirúrgico.

En un estudio presentado en el VII Congreso Europeo de Cirugía [16], donde se estudiaba los síntomas debido a gossypibomas o retención involuntaria de gasas de una serie de 10 pacientes, se obtuvieron los siguientes resultados:

- El 10% de los pacientes son asintomáticos.
- El 30% de los pacientes tenían la sospecha de una lesión ocupativa intraabdominal o dolor abdominal crónico secundario a íleo y/o adherencias.
- El 30% se presentó con síndrome febril e íleo postoperatorio.
- El 10% con síndrome febril en inmediato postoperatorio.
- El 10% con síndrome febril y orificio fistuloso en sitio de herida quirúrgica.
- El 10% con síndrome febril y sospecha de masa intra-abdominal.

Para la eliminación del material retenido es necesario de una nueva cirugía, aunque el paciente sea asintomático, debido a los problemas que pueden causar posteriormente. Esta nueva cirugía suele ser una operación

convencional, aunque también es posible una cirugía laparoscópica, reduciendo así las complicaciones. [16]

Ilustración 8 - Ejemplo de gasa retenida extraída del abdomen [17]

1.5. Objetivos del TFG

En este Proyecto de Fin de Grado se pretende desarrollar un sistema de seguimiento de gasas y pinzas quirúrgicas mediante redes neuronales artificiales con el fin de que un sistema robótico sea capaz de determinar de forma autónoma los movimientos realizados por el cirujano y conseguir de esta forma poder asistirle. Gracias al seguimiento de instrumental quirúrgico se podría conseguir, en un futuro, reconocer en qué fase de la operación se encuentra el paciente y operar en consecuencia de forma autónoma.

El seguimiento del material quirúrgico se realizará a través de las imágenes tomadas del laparoscopio, ubicado en la Escuela de Ingenierías Industriales, que se utilizaran para entrenar una serie de redes neuronales que permitan el seguimiento de gasas o pinzas independientemente. Para realizar esto, se debe localizar el material quirúrgico para incluir esas coordenadas con cada imagen, consiguiendo así un aprendizaje supervisado de cada una de las redes. Finalmente, se procederá a aplicar las redes entrenadas en videos laparoscópicos.

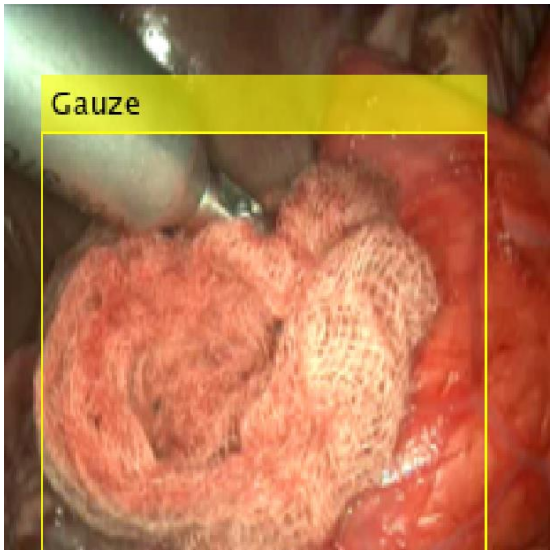


Ilustración 9 - Ejemplo de fotograma del seguimiento de gasas detectado por Resnet-50

Cada una de las redes se ha implementado mediante diferentes algoritmos, con el fin de realizar una comparativa sobre las mejores opciones a la hora de una detección eficaz para una posible futura implementación en el quirófano.

En el siguiente capítulo será una introducción al marco teórico del proyecto, explicando los fundamentos de las redes neuronales artificiales, así como de varios algoritmos y del funcionamiento de las redes entrenadas utilizadas en el mismo.

2. Redes neuronales artificiales

Las redes neuronales parten de la idea de imitar el funcionamiento del cerebro humano o bien de las redes neuronales de los seres humanos, es decir, el funcionamiento de neuronas conectadas que transmiten señales entre sí, consiguiendo unas salidas, inspiradas en el sistema nervioso, a partir de ciertas entradas.

Estas neuronas están formadas por: dendritas, el cuerpo de la célula o soma y el axón, en términos computacionales tendríamos que el axón es el encargado de transmitir la información entre neuronas, la dendrita es la receptora de la red y el soma se encarga de sumar los valores recibidos en las dendritas. Estas neuronas conectadas entre sí formando una red de comunicaciones que pueden resolver problemas muy complejos, es a lo que llamamos red neuronal. La extrapolación de esta conexión de neuronas a las redes neuronales artificiales sería como una serie de neuronas conectadas entre sí que se encuentran organizadas en capas: una capa de entrada, capas intermedias denominadas capas ocultas y una capa de salida. Además, cada una de estas capas está formada por pesos o parámetros, los cuales se van formando y reajustando durante el entrenamiento. [13]

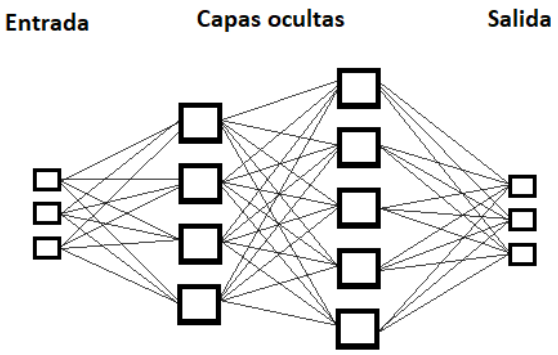


Ilustración 10 - Capas de una red neuronal artificial

Se pueden distinguir tres tipos o paradigmas diferentes de entrenamiento en las redes neuronales:

- **Entrenamiento no supervisado:** no requiere proveer a la red con la respuesta correcta para cada entrada del almacén de entrenamiento. La red se encarga de decidir qué características utilizará para agrupar las entradas en clases según las características elegidas.
- **Entrenamiento supervisado:** se provee a la red de un almacén de datos de entrada junto con sus correspondientes salidas deseadas. La red se encarga de producir las respuestas deseadas, en caso de no producirse, el error es propagado hacia atrás, causando que la red reajuste los

pesos para producir las respuestas deseadas, este proceso se realiza repetidamente durante todo el entrenamiento.

- **Entrenamiento por refuerzo o aprendizaje reforzado:** es una variante del entrenamiento supervisado en el cual a diferencia de esta donde la propia red elimina el error, en el aprendizaje reforzado se intenta maximizar la recompensa, aunque pueda suponer aumentar el error o que los resultados no sean óptimos. La red recibe una recompensa dependiendo de la decisión tomada en la iteración anterior, consiguiendo que realice el mismo comportamiento al seguido si obtiene una recompensa positiva, mientras que, si es negativa, lo penaliza para que actúe de forma distinta ante el mismo comportamiento.

2.1. CNN (Convolutional Neural Network)

Una CNN o red neuronal convolucional es un tipo de red neuronal artificial entrenada mediante aprendizaje supervisado que procesa sus capas para imitar el córtex visual del cerebro humano, para poder de esta forma, identificar características para distinguir objetos. Para poder distinguir objetos, las CNN contienen una serie de capas especializadas con una jerarquía entre sí, siendo las primeras capas más simples, capaces de detectar líneas y curvas, mientras que las capas posteriores se van especializando hasta poder detectar formas más complejas como gasas o instrumental quirúrgico.

Las CNN parten de una imagen de entrada que descomponen en matrices de pixeles. Aunque una imagen convertida a matriz de pixeles suele tener un rango de valores de 0 a 255, este valor se normaliza entre 0 y 1 dividiendo el valor de cada píxel entre 255. [19]

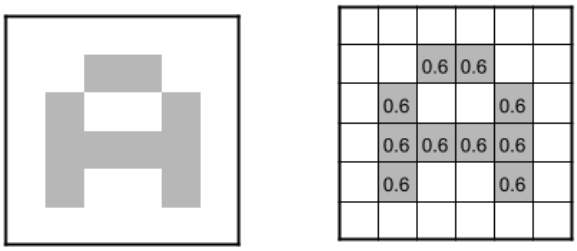


Ilustración 11 - Imagen de entrada convertida en matriz de pixeles

Cabe destacar que normalmente las imágenes están en color, lo que supone que, en lugar de una sola matriz de pixeles, contemos con 3, una para cada canal RGB (Red-Green-Blue) que también deben normalizarse entre 0 y 1. Dependiendo del tamaño de imagen de entrada y si esta está en escala de grises o en color, tendremos una cantidad de neuronas de entrada diferentes, es decir, si tenemos una imagen de entrada de 224x224, como es el caso de AlexNet, GoogLeNet y ResNet-50, tendremos 50176 neuronas si se trata de

imágenes en escala de grises o 150528 neuronas si se trata de imágenes en color.

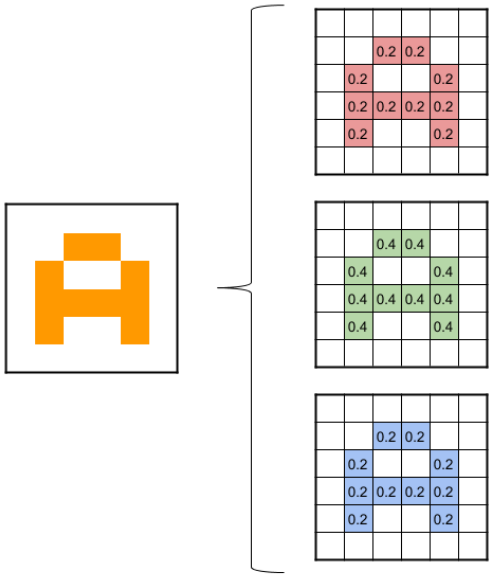


Ilustración 12 - Imagen de entrada RGB convertida en matrices de pixeles

Estos valores de los píxeles son pasados a cada neurona de las capas, donde se realizan distintas operaciones como detectar líneas y curvas en las primeras capas, especializándose hasta poder reconocer formas complejas. Por otro lado, la profundidad de estas redes varía en función del número de capas, siendo mayor cuanto mayor sea el número de estas.

En el presente TFG se utilizarán redes preentrenadas que han demostrado ser eficientes detectando una gran cantidad de objetos diferentes, estas redes son AlexNet, GoogLeNet, ResNet-50 y DarkNet-53. Con el fin de elaborar un sistema de detección de instrumental quirúrgico se modificará el algoritmo de aprendizaje sustituyendo las últimas capas de estas para encontrar cuál de ellas obtienen los mejores resultados.

2.1.1. Arquitectura de las CNN

Una de las grandes diferencias entre las RNA y las CNN es que las neuronas utilizadas en estas últimas se encuentran organizadas en tres dimensiones, la altura y el ancho de la imagen de entrada y, como novedad, la profundidad, siendo el valor de esta profundidad de 1 si las imágenes se encuentran en escala de grises y de 3 si están en escala RGB.

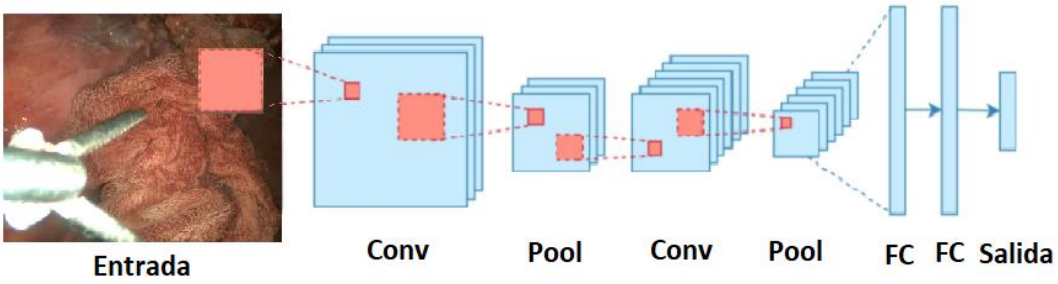


Ilustración 13 - Arquitectura de una red neuronal convolucional

La mayoría de CNN utilizan un algoritmo de aprendizaje llamado *backpropagation.* Este algoritmo es un método de cálculo de gradiente que se basa en calcular las salidas de error, mediante la comparación de la salida deseada con la salida obtenida. Esta salida de error se propaga hacia atrás para corregir los parámetros de los pesos en todas las capas ocultas que contribuyen directamente a la salida.

Al igual que las redes neuronales artificiales, las CNN están compuestas por una secuencia de capas. Las tres capas más importantes son la capa de convolución, "pooling" o agrupación y "fully-connected" o totalmente conectadas.

- Capa de entrada: es la primera capa de la red, transforma las imágenes de entrada como se ha mencionado en el apartado anterior, dependiendo de la red utilizada habrá que redimensionar las imágenes de entrada para que esta dimensión sea acorde a los parámetros de entrada de la red.

- Capa de convolución: son las capas más importantes de una CNN, puesto que se encargan de la mayor carga computacional. Los parámetros de las capas de convolución son una serie de filtros que se encargan de aprender las características de las imágenes de entrada, esto se realiza tomando grupos de pixeles cercanos de la imagen de entrada e ir realizando operaciones con una matriz más pequeña que la matriz de la imagen, esta matriz es llamada kernel o filtros, una vez el kernel ha recorrido toda la imagen, genera una nueva matriz de salida que será la entrada de la siguiente capa de la red.

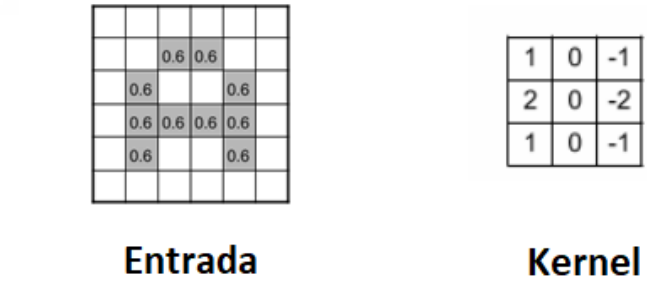


Ilustración 14 - Ejemplo de imagen de entrada y Kernel de una capa de convolución

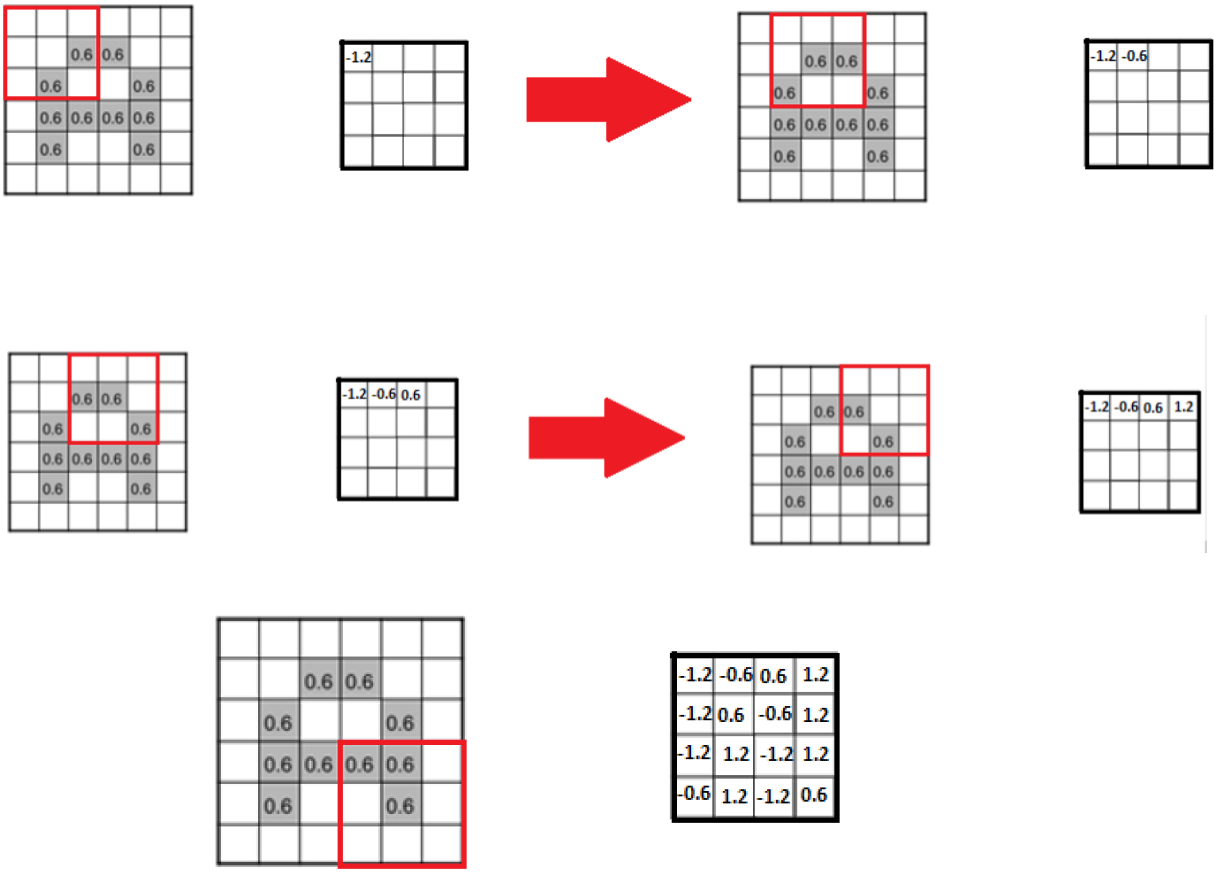
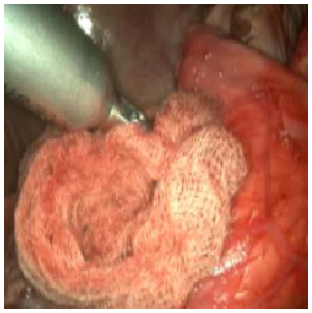
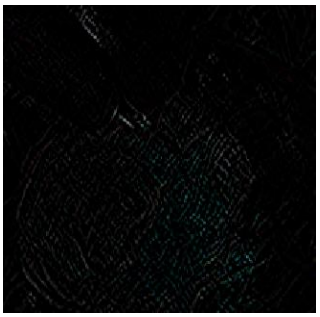
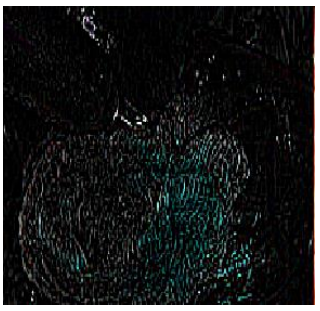
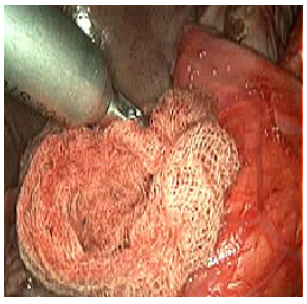
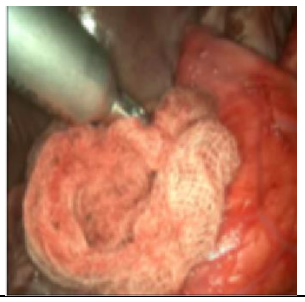


Ilustración 15 - Ejemplo de convolución

Si la imagen es a color el filtro utilizado está compuesto por tres kernels de las mismas dimensiones, posteriormente la salida de cada canal RGB se suma para conseguir una sola salida como si fuera una imagen en escala de grises.

En cada capa de convolución no se aplica un solo kernel sino que se utilizaran un conjunto de ellos, consiguiendo que cada kernel este especializado en obtener ciertas características de la imagen original, por lo que cuantos más filtros especializados tenga la red, será más robusta y fiable a la hora de distinguir un objeto de otro. Los kernels más interesantes son los que permiten extraer características como detección de bordes, líneas horizontales, líneas verticales y desenfocar y enfocar la imagen.

Operación	Kernel	Resultado
Identidad	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Detección de bordes	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Realzado 3x3	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Suavizado	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

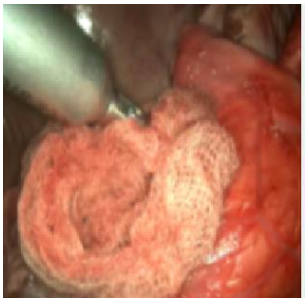
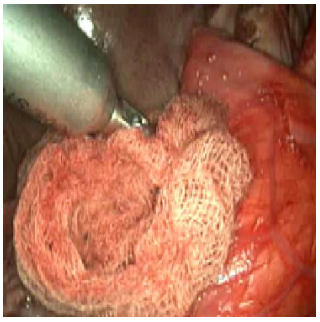
Suavizado gaussiano 3x3	$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$	
Realzado 5x5	$\frac{-1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & -475 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}$	

Tabla 1 - Ejemplos de filtros utilizados por capas de convolución

Los distintos valores de los pesos de los kernels deben ajustarse mediante *backpropagation,* esto supone una gran ventaja debido a que cada kernel es de tamaño reducido teniendo que ajustarse significativamente menos parámetros que en una red neuronal tradicional.

- **Rectified Linear Unit (ReLU) layer:** son las capas encargadas de sobrescribir los valores negativos por 0, manteniendo los valores positivos, esto también se conoce como activación, puesto que solo las características activadas son transmitidas a las siguientes capas. Estas capas son el reemplazo de las unidades *tanh* usadas en RNA convencionales, debido a que se consiguen velocidades de entrenamiento varias veces mayor y la eliminación del *overfitting* o sobreaprendizaje, siendo un fallo producido por la red en la clasificación por no tener estrictamente los mismos valores en el almacén de entrenamiento,
  La función que realizan estas capas puede escribirse de la siguiente forma:

$$f(x) = \max(0, x)$$

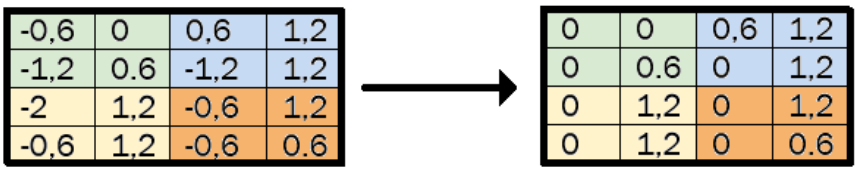


Ilustración 16 - Ejemplo de aplicar una ReLU a una matrix 4x4

- Capa de pooling: las capas de *pooling* o agrupación simplifican la salida llevando a cabo una disminución no lineal de la tasa de muestreo, haciendo que se necesiten menos parámetros de aprendizaje de la red, puesto que prevalecen las características más importantes que detectó cada filtro de la capa de convolución, consiguiendo también una reducción en la carga computacional.
  Aunque hay diferentes tipos de agrupaciones, el más utilizado, y el que ha sido usado en este proyecto, es el "max-Pooling". El max-Pooling se encarga de recorrer las salidas de la capa anterior de izquierda a derecha y de arriba abajo, para reducir dichas salidas a la mitad. Para ello, se preserva el valor más alto de los pixeles vecinos.

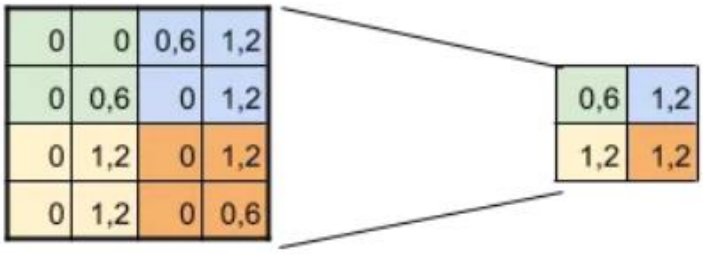


Ilustración 17 - Ejemplo de aplicar max-Pooling a una matriz 4x4

  En la ilustración 17, se muestra el funcionamiento de dicho filtro, donde pasamos de una matriz 4x4 a una 2x2, en la submatriz verde tenemos como valores 0 y 0,6 pasando este último a la siguiente matriz por ser el más alto, se realizó el mismo procedimiento para cada una de las submatrices.

- Fully-connected layer: estas capas totalmente conectadas realizan las mismas funciones que las usadas en redes neuronales artificiales habituales, ubicando y etiquetando la imagen de entrada en su respectiva clase según las activaciones recibidas de las capas anteriores. Estas capas están formadas por tantas neuronas como clases, en este caso al haber solo una clase, ya sea gasas o pinzas quirúrgicas, solo tendrán una neurona, conectada a todos los elementos de la capa anterior. Aunque la capa anterior suele ser una capa de *pooling*, también es necesario aplicar una capa de *ReLU* entre ellas para mejorar el rendimiento de la red. [20]

2.1.2. AlexNet

Es un tipo de CNN, diseñada por Alex Krizhevsky junto con Ilya Sutskever y Geoffrey Hinton, fue la red ganadora del ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) de 2012 consiguiendo un error del 10% menor que el segundo puesto en cuanto a la tasa de error del Top-5, tasa de tasa de no clasificar correctamente el objeto dentro de las 5 mejores predicciones.

La arquitectura de AlexNet consiste en 8 capas, siendo cinco de estas 8 capas convolucionales y tres capas totalmente conectadas, además de añadir las siguientes características [21]:

- Utilización de unidades lineales rectificadas o ReLU (Rectified Linear Units) en lugar de la función tanh, la cual era la función más utilizada en ese momento.
- Capacidad de trabajar con múltiples GPUs.
- Normalización de la respuesta local.
- Capas agrupadas superpuestas, funcionan como una combinación de capas que ayuda a reducir la dimensionalidad de las imágenes, pero con la diferencia de que el filtro que realiza la agrupación se superpone entre sí.

Esta red ha sido entrenada con aproximadamente 1,2 millones de imágenes de entrenamiento, 50000 imágenes de validación y 150000 imágenes de prueba, consiguiendo que sea capaz de clasificar imágenes en 1000 clases diferentes.

2.1.3. GoogLeNet

GoogLeNet, también conocida como Inception V1, es la CNN presentada por Google, junto con la colaboración de varias universidades, ganadora del ILSVRC de 2014, consiguiendo un error del Top-5 de 6,67%.

GoogLeNet está compuesto de 22 capas o 27 contando las capas de pooling, siendo 9 de estas capas, *inception modules* o módulos de comienzo. Estos módulos de comienzo se encargan de realizar convoluciones 1x1, 3x3 y 5x5 junto con max pooling de 3x3 de forma paralela, juntándose las salidas de estos para generar la salida final.

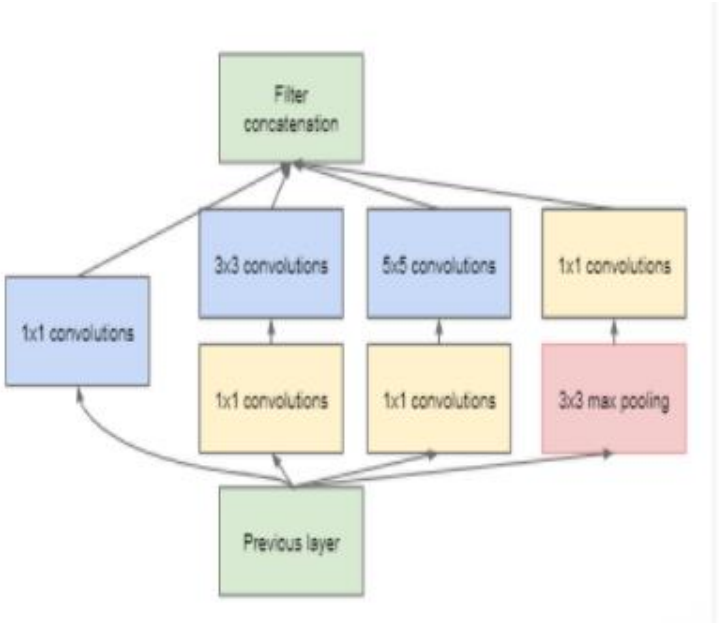


Ilustración 18 - Módulo de comienzo [22]

Estas operaciones tienen la ventaja de permitir arquitecturas más profundas reduciendo el número de parámetros. Por ejemplo, si realizamos una convolución de 5x5 con 48 filtros, como se muestra en la ilustración 19, se obtienen 112,9 millones de operaciones.

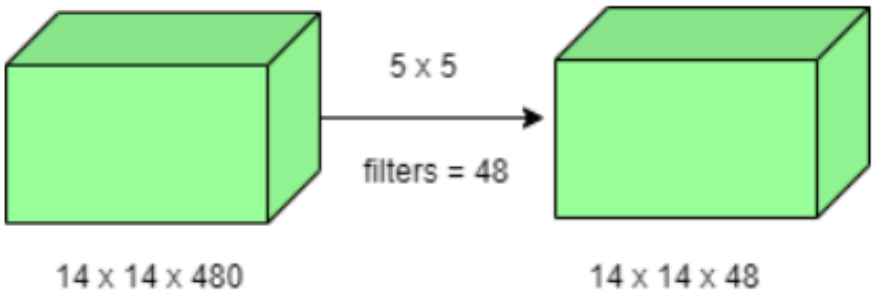


Ilustración 19 - Convolución 5x5 [22]

Por otro lado, si antes de aplicar la misma convolución, se utiliza una convolución 1x1 con 16 filtros, resulta en 1,5 millones de operaciones de la primera convolución y 3,8 millones de la segunda, consiguiendo un total de 5,3 millones de operaciones, un valor mucho menor al obtenido sin la convolución 1x1, lo que reduce significativamente los tiempos de entrenamiento.

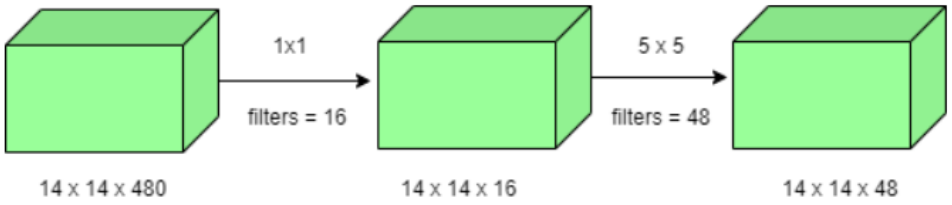


Ilustración 20 - Convolución de 1x1 con la posterior convolución de 5x5 [22]

Como novedad, GoogLeNet sustituye las capas totalmente conectadas al final de su arquitectura, que aumentan la carga computacional, por capas de *global average pooling,* esta capa recibe un mapa de características de 7x7 y lo promedia a una salida de 1x1, aumentando la precisión del top-1 un 0,6% debido a reducir el número de parámetros entrenables a 0.

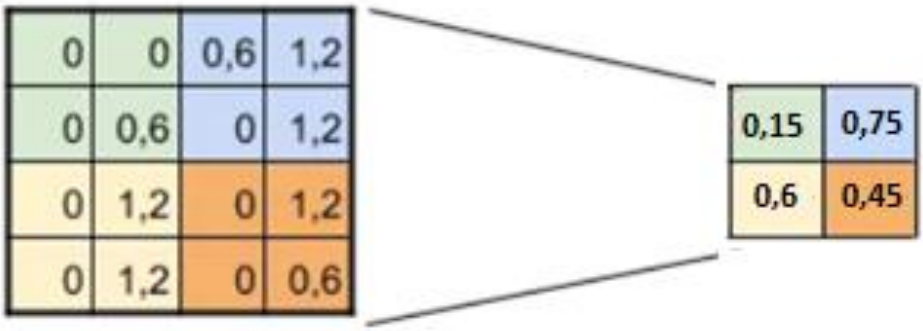


Ilustración 21 - Ejemplo de aplicar global average pooling a una matriz 4x4

Por último, para conseguir que los gradientes se propaguen hacia atrás para reajustarse de una forma efectiva, GoogLeNet implementa unas "redes" auxiliares de clasificación, que solo actúan en el entrenamiento, conectadas a capas intermedias lo que acelera el entrenamiento, reduce el error de gradiente y añade una normalización adicional. [17]

2.1.4. ResNet-50

ResNet o Residual Network fue una CNN introducida por Microsoft, que en el año 2015 ganó la competición ILSVRC (ImageNet Large Scale Visual Recognition Challenge).

Para entender porque fue una gran novedad esta red, debemos saber que a medida que se aumenta el número de capas de una red, se disminuye el nivel de error de esta, hasta cierto punto, ya que, al superar cierto número de capas aparece el llamado error de gradiente, este error consiste en que el gradiente

utilizado para calcular la tasa de error, vaya disminuyendo hasta llegar a cero tras un número de iteraciones, consiguiendo así que la red deje de aprender en el entrenamiento.

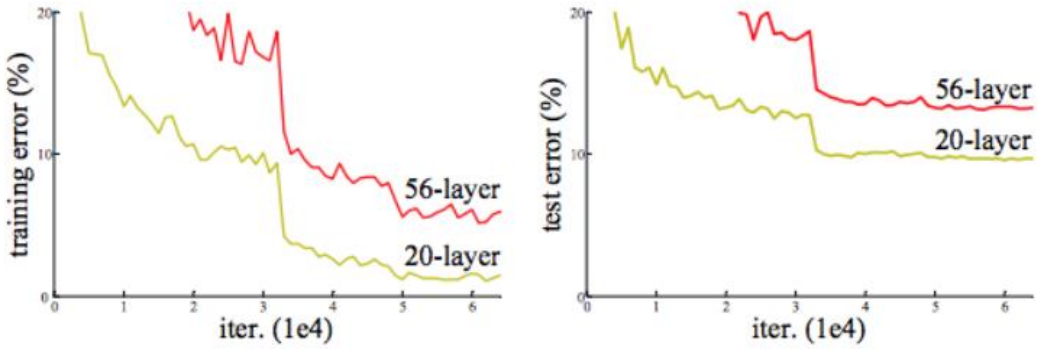


Ilustración 22 - Comparativa entre una red de 56 capas y una de 20. Observándose que la tasa de error tanto en el entrenamiento como en la prueba aumente al aumentar el número de capas [23]

Para evitar este error, se crea ResNet que introduce el concepto de salto de conexiones o *skip connections*, el cual consiste en saltarse el entrenamiento de una o un par de capas, conectando directamente con la salida de estas.

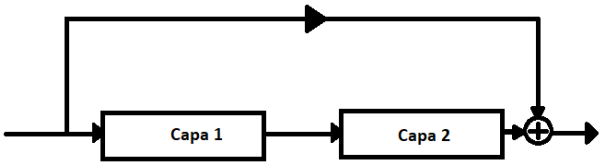


Ilustración 23 - Bloque residual de ResNet

Con este salto de entrenamiento de capas conseguimos que los pesos se reajusten sustituyendo las capas evitadas, consiguiendo así que se utilicen menos capas en el entrenamiento, pudiéndose implementar redes más profundas.

2.1.5. DarkNet-53

DarkNet-53 es una red neuronal convolucional que se utiliza como red base para el algoritmo para la detección de objetos, que introduciremos a continuación, YOLOv3, contiene 53 capas de convolución, las cuales se agrupan en bloques residuales, formados por una capa de convolución de 1x1 seguida de una capa de 3x3 y un salto de conexión que actúa de la misma forma que en ResNet, para ayudar a las activaciones que se propaguen a capas más profundas disminuyendo el error de gradiente.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Ilustración 24 - Arquitectura de DarkNet-53 [24]

En la ilustración 24, se muestran los bloques residuales como rectángulos, teniendo entre ellos capas convolucionales para reducir el número de parámetros que se pasan al siguiente bloque residual.

DarkNet-53 ha sido entrenado con más de un millón de imágenes de la base de datos ImageNet, consiguiendo clasificar objetos en 1000 clases diferentes.

2.1.6. YOLO

El algoritmo YOLO (You Only Look Once) es un sistema de código abierto usado para la detección de objetos en tiempo real propuesto por Redmon et al. [25] en 2016. Este algoritmo está basado en la regresión, es decir, que a partir de dividir la imagen en regiones predice clases y *bounding boxes*, rectángulos imaginarios determinados por las coordenadas superiores izquierda de x e y, además, de por el ancho y altura de este rectángulo, en una sola ejecución del algoritmo, sin necesidad de *back-propagation* como en las redes mencionadas anteriormente. [26]

La primera versión de este algoritmo se inspiró en la red GoogLeNet para clasificación de imágenes, con 24 capas convolucionales seguidas de 2 capas totalmente conectadas. La gran diferencia es la sustitución del módulo de comienzo por una capa de reducción 1x1 y una capa de convolución de 3x3. Posteriormente, se implementó una segunda versión, menos profunda y más

rápida, llamada Fast YOLO o YOLOv2, con 19 capas convolucionales en lugar de 24 y con menos filtros en dichas capas.

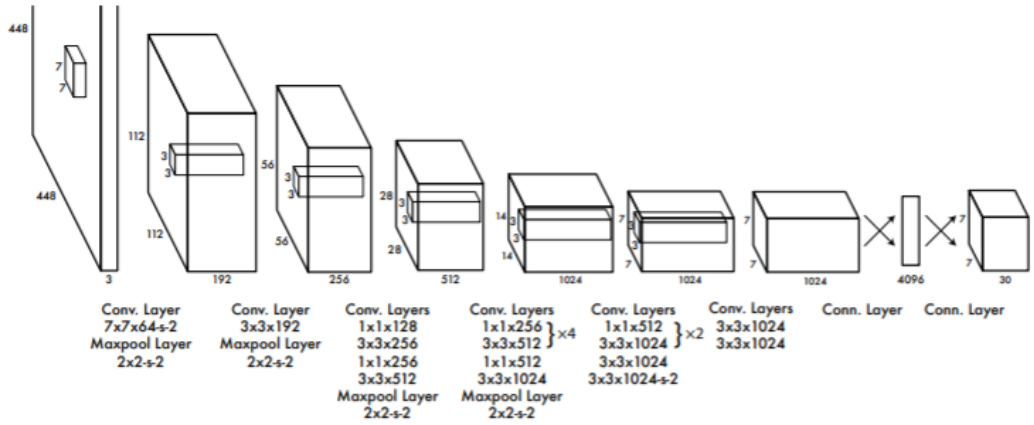


Ilustración 25 - Arquitectura de YOLO [25]

La red divide la imagen en cuadrículas, siendo la cuadrícula donde se encuentre el centro del objeto la que se encarga de detectar dicho objeto. Posteriormente, cada cuadrícula predice la probabilidad de cada clase de estar contenida en la cuadricula mediante el IoU (Intersection over Union), que indica la precisión entre la *bounding box* incluida en el entrenamiento y la predicha por la red mediante la siguiente fórmula:

$$IoU = \frac{Area\ de\ superposición}{Area\ de\ unión}$$

Siendo el área de superposición el área que tienen ambas en común y el área de unión el área total que ocupan ambas *bounding boxes.*

YOLO predice múltiples *bounding boxes* por cada cuadricula, asignando un "predictor" que se encarga de predecir un objeto basado en el mayor IoU perteneciente a la clase dada en el almacén de entrada, lo que lleva a que cada predictor se especialice siendo algunos mejores en predecir ciertos tamaños o distintas clases, lo que conlleva a una mejora en la sensibilidad total. [26]

Posteriormente, Redmon et al. publica la versión 3 de YOLO en 2017. YOLOv3 tiene como red de extracción de características la misma red que su anterior versión, DarkNet, pero con la diferencia de contar con 53 capas convoluciones en lugar de 19. Con el objetivo de mejorar la sensibilidad de YOLO en esta tercera versión se utiliza una predicción en múltiples escalas lo que mejora la detección de objetos pequeños, debido al inconveniente que esto suponía en anteriores versiones.

Lo que convierte a Yolo en un algoritmo más que apropiado es su velocidad detectando objetos, aunque esta mejora en la velocidad es debido a una disminución en la precisión, pero la mejor opción cuando se requiere detección en tiempo real. Esta pérdida de precisión se resuelve con YOLOv3, a costa de una disminución en la velocidad de procesamiento debido a utilizar redes más profundas, aun así, se obtienen mejores resultados que con otros sistemas de detección. [24] [27]

2.2. Conclusiones

En este capítulo se ha realizado una introducción a las redes neuronales artificiales, así como un tipo de estas, las redes neuronales convolucionales.

La inteligencia artificial es una tecnología que cada vez está siendo más utilizada en muy diversos ámbitos, en el presente TFG se ha dispuesto el uso de esta tecnología en el ámbito médico, en concreto, en el de las operaciones laparoscópicas.

Aunque existen diferentes softwares para entrenar e implementar redes neuronales, se ha optado por utilizar MATLAB por su gran variedad de documentación y librerías propias, además de poder realizar su uso de forma gratuito con la licencia de la Universidad de Valladolid.

3. Desarrollo del sistema de detección

Para la detección de instrumental quirúrgico se ha empleado el entorno de programación MATLAB, donde se han elaborado distintos programas para el entrenamiento y su posterior despliegue en videos de cada una de las redes neuronales y algoritmos utilizados. El despliegue de los algoritmos de detección en los videos laparoscópicos sigue el siguiente diagrama:

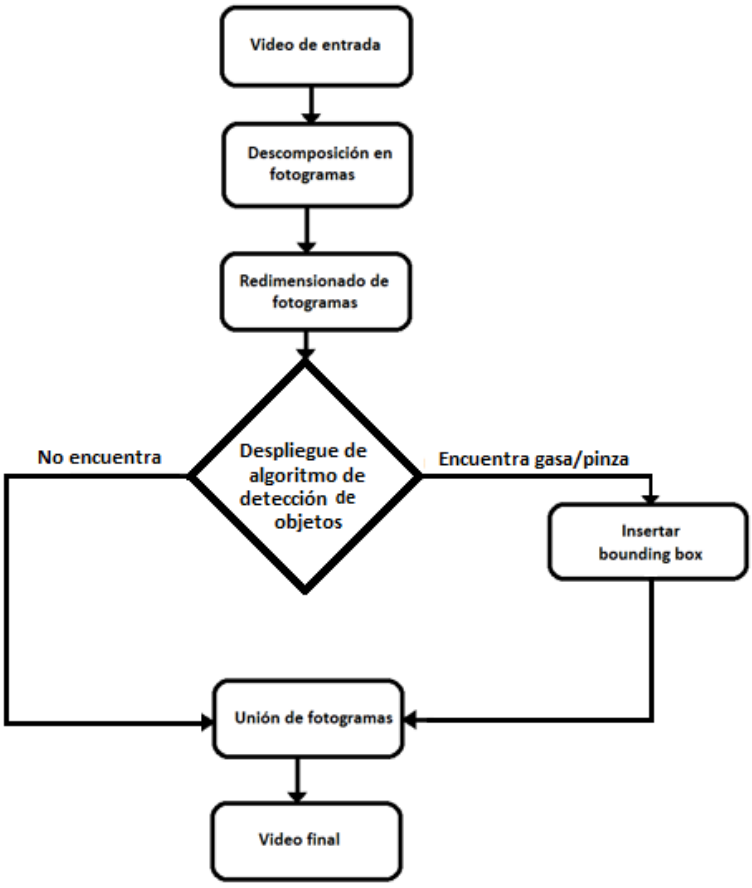


Ilustración 26 - Diagrama del sistema de detección de instrumental quirúrgico

Los algoritmos de detección son capaces de detectar instrumental quirúrgico en imágenes, por lo que el video de entrada se debe dividir en fotogramas para que se pueda implementar el algoritmo. Los videos suelen tener una tasa de fotogramas por segundo de entre 24 y 30, siendo en nuestro caso de 25 fps, es decir, por cada segundo de video se encuentran 25 imágenes distintas. Cada fotograma debe redimensionarse a las dimensiones de entrada de cada CNN utilizado,

Red	Dimensiones de entrada
AlexNet	224x224
GoogLeNet	224x224
ResNet-50	224x224
DarkNet-53	256x256

Tabla 2 - Dimensiones de entrada de las CNN implementadas

Una vez redimensionados los fotogramas, se aplica la red neuronal sobre cada uno de ellos. En cada fotograma, la red otorga una puntuación entre 0 y 1 a los objetos que detecta, si esta puntuación es menor del 0,5 interpreta que no se ha detectado el objeto. Por otro lado, si es mayor, se inserta la *bounding box* en las coordenadas correspondientes.

Tras haber realizado lo anterior sobre todos los fotogramas, se unen las imágenes formando el nuevo video de salida, manteniendo las propiedades que tenía el video de entrada.

3.1. Software

El entrenamiento de cada red neuronal profunda y su posterior implementación en los videos laparoscópicos supone una gran carga computacional, por lo que es necesario utilizar un software con gran capacidad de cómputo numérico. Debido a esto se utilizó el sistema de programación MATLAB, el cual provee un lenguaje de programación propio, además de contar con librerías propias que permiten el uso de visión artificial y entrenamiento profundo. Todas estas librerías tienen una fácil y rápida implementación desde el propio software, además, MATLAB cuenta con una serie de documentos y manuales de forma pública que facilitan también el trabajar con redes neuronales.

Por otro lado, MATLAB es compatible con la tecnología CUDA de las gráficas de Nvidia, que permite el entrenamiento de las distintas redes mediante GPU (Graphics processing unit), lo que reduce la carga de trabajo en la unidad central de procesamiento o CPU, y el computo paralelo, lo que permite preparar los distintos mini lotes de entrenamiento mientras se está entrenando. Esto permite reducir drásticamente los tiempos de entrenamiento y detección de objetos.

3.1.1. MATLAB

MATLAB es la abreviatura de MATrix LABoratory o laboratorio de matrices, es un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio. Entre sus posibilidades podemos encontrar la creación de interfaces de usuario, la representación de datos y funciones, la comunicación con otros lenguajes y

dispositivos hardware, manipulación de matrices y la implementación de algoritmos, siendo estas dos últimas prestaciones las más utilizadas para la elaboración de este trabajo.

Por otro lado, MATLAB permite la implementación de cajas de herramientas o toolboxes para ampliar las anteriores prestaciones. Las toolboxes utilizadas son las siguientes:

- **Deep Learning Toolbox Model for ResNet-50 Network:** implementa el modelo preentrado ResNet-50, entrenado con más de un millón de imágenes y capaz de clasificar imágenes en mil categorías diferentes. [23]
- **Deep Learning Toolbox Model for Alexnet Network:** implementa el modelo preentrenado AlexNet, entrenado con 1,2 millones de imagenes y capaz de clasificar imagenes en mil categorías diferentes. [24]
- **Deep Learning Toolbox Model for GoogLeNet Network:** implementa el modelo preentrado GoogLeNet, entrenado con más de un millón de imágenes y capaz de clasificar imágenes en mil categorías diferentes. [25]
- **Deep Learning ToolboxTM Model for DarkNet-53 Network:** implementa el modelo preentrenado DarkNet-53, entrenado con más de un millón de imágenes y capaz de clasificar imágenes en mil categorías diferentes. [26]
- **Deep Learning Toolbox:** permite implementar y diseñar redes neuronales profundas con algoritmos, además de soportar la transferencia de aprendizaje de modelos previamente entrenados como ResNet-50, red utilizada para la detección de instrumental quirúrgico. [32]
- **Parallel Computing Toolbox:** permite hacer uso de procesadores multinúcleo, GPUs y clusters de ordenadores para resolver problemas con uso intensivo de cálculos y datos, lo que supone una amplia reducción de tiempos de entrenamiento de las redes. [28]
- **Computer Vision Toolbox:** proporciona algoritmos y funciones para diseñar y probar sistemas de procesamiento de video y visión aritificial. Permite realizar detección y seguimiento de objetos.

Además, se utilizado la aplicación Image Labeler, la cual permite marcar las regiones de interés rectangulares (ROI) de las imágenes del dataset, estos ROIs serán los llamados *bounding box* posteriormente.

Por último, cabe destacar que se ha trabajado con la versión MATLAB 2020b y MATLAB 2021a.

3.1.2. CUDA

CUDA o Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo) es una plataforma de computación en paralelo y modelo de programación creado por NVIDIA para operar con unidades de procesamiento gráficas o GPUs, haciendo que se aumente en gran medida las aplicaciones de cómputo, como son el entrenamiento profundo de las redes neuronales utilizadas.

Además, CUDA tiene compatibilidad total con MATLAB, haciendo que se aceleren los procesos de entrenamiento y posterior detección, sin tener que cambiar el código realizado.

Para finalizar, se instalaron los últimos drivers de la tarjeta gráfica, así como, la versión de CUDA a la 11.1.106.

3.2. Hardware

Debido a que los resultados de velocidad de procesamiento en el seguimiento y detección de objetos en redes neuronales profundas dependen significativamente del hardware empleado, me ha parecido altamente importante indicar cual ha sido el hardware utilizado.

El ordenador utilizado en la realización del trabajo es un ordenador portátil ASUS ROG STRIX G531GT, cuyas especificaciones son las siguientes:

- Procesador: Intel Core i7-9750H (6 Núcleos, 12MB Caché, 2,6Ghz hasta 4,5GHz, 64-bit).
- Memoria RAM de 16GB DDR4 2666MHz.
- Tarjeta gráfica NVIDIA GeForce GTX1650 GDDR5 VRAM
- Sistema operativo: Windows 10

Con este dispositivo, se ha llevado a cabo tanto el desarrollo, como las pruebas de detección posteriores.

3.3. Procedimiento

Para la detección de instrumental quirúrgico, se han seguido una serie de pasos a la hora de entrenar las redes neuronales, primero, se ha creado una serie de *datasets*, uno para la detección de gasas y otro para la detección de instrumental quirúrgico.

Una vez, creados estos datasets, se procede a entrenar cada una de las redes neuronales con los algoritmos de YOLOv2 y YOLOv3 para que aprendan a detectar los elementos anteriores. Posteriormente, se obtienen los resultados de estos y se realiza una comparación. Por último, se aplican las redes neuronales entrenadas a los videos laparoscópicos.

3.3.1. Dataset

Para empezar, debemos entender que es un dataset, el cual es un conjunto de datos tabulados, es decir, un conjunto o tabla de datos en el cual cada dato se encuentra etiquetado para el posterior entrenamiento de la red. Dependiendo del tipo de red a entrenar, como por ejemplo una CNN con el algoritmo back-propagation o una CNN con el algoritmo YOLO, el dataset es diferente, puesto que en el primero cada imagen de entrada debe ir seguida de la o las clases en dicha imagen, mientras que, en la segunda, como es el caso de este proyecto, el dataset debe incluir la imagen junto con las coordenadas del objeto que se desea detectar.

En este trabajo, serán necesario dos datasets de imágenes de operaciones laparoscópicas, por un lado, de pinzas quirúrgicas, y por otro, de gasas quirúrgicas.

Se han seguidos dos métodos diferentes para crear cada uno de los datasets. Por un lado, el dataset de pinzas quirúrgicas se ha creado a partir de matlab, introduciendo las imágenes de entrenamiento en la herramienta de MATLAB, *Image Labeler*, en el que se irán marcando mediante rectángulos o *bounding boxes* la posición de cada una de las pinzas, que se encuentren en cada una de las imágenes. Este proceso ha sido largo debido a tener que marcar manualmente cada una de las pinzas de las imágenes.

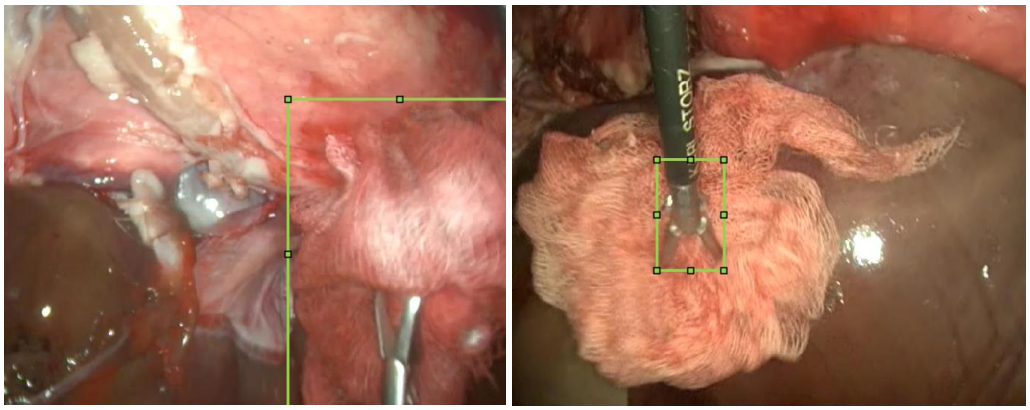


Ilustración 27 - Imágenes etiquetadas de Gasas (izquierda) y pinzas (derecha)

Por otro lado, el dataset de gasas quirúrgicas se ha elaborado mediante la biblioteca OpenCV de Python, utilizada para visión artificial, de una forma automática y menos tediosa, al no tener que marcar cada gasa de forma manual. Para ello, se parte de las imágenes de entrenamiento, sobre las cuales se realiza una segmentación, obteniendo una máscara binarizada de la gasa, esto es, la silueta de la gasa en blanco y negro.

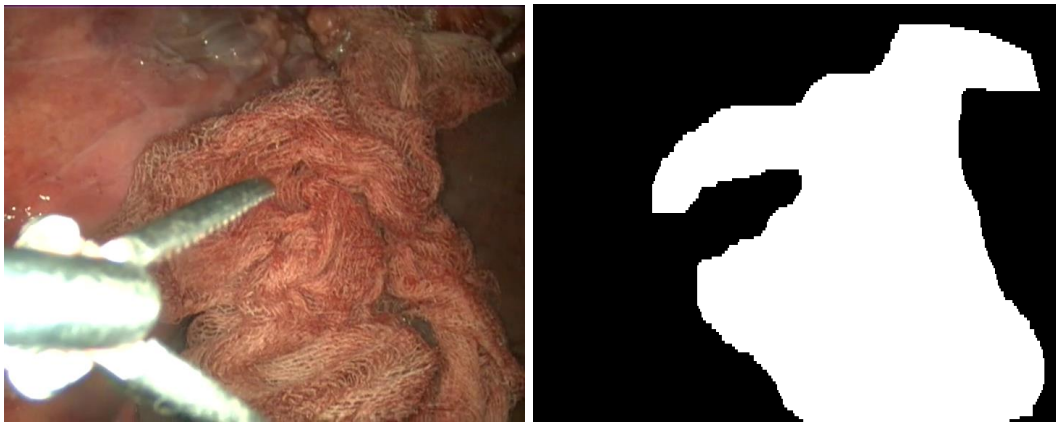


Ilustración 28 - Imagen de entrada (Izquierda) y mascara binarizada (derecha)

Posteriormente, se elabora un fichero con la ubicación de cada imagen en el ordenador, junto con la posición de la gasa en cada una de las imágenes mediante la localización de un recuadro que la contiene plenamente.

```
AnnotationGen/FixedFramesGasa/VID0002.mp410.0.png, 164,28,703,575,0
```

Ilustración 29 - Ejemplo de posición de la gasa de la ilustración 28

Este recuadro debe transformarse para su correcta utilización en MATLAB puesto que este y Python tienen diferencias en su origen de coordenadas, siendo en Python el (0,0) mientras que en MATLAB es el (1,1), y a la hora de almacenar las coordenadas de un rectángulo, ya que, Python guarda las coordenadas de los cuatro vértices del rectángulo y MATLAB trabaja obteniendo la *x* e *y* más cercana al origen, junto con la altura y anchura del rectángulo.

Debido a no disponerse un dataset público de gasas o instrumental quirúrgico, las imágenes se han obtenido de videos tomados en la Universidad de Valladolid, de un modelo laparoscópico hecho con vísceras de animal y utilizando instrumental laparoscópico y gasas quirúrgicas. A partir de estos videos se han obtenido 4014 y 666 imágenes para entrenar las redes para detectar gasas y pinzas, respectivamente.

Por último, cabe destacar que las imágenes tienen unas dimensiones de 720 x 562 por lo que se tendrá que redimensionar estas imágenes a 224 x 224 para AlexNet, ResNet-50 y GoogleNet y a 256 x 256 para DarkNet-53, debido a ser las dimensiones de entrada de cada una de las redes.

3.3.2. Entrenamiento

Con el fin de implementar los algoritmos de detección de instrumental quirúrgico dentro del cuerpo humano mediante *Deep Learning,* se ha utilizado el siguiente diagrama de flujo.

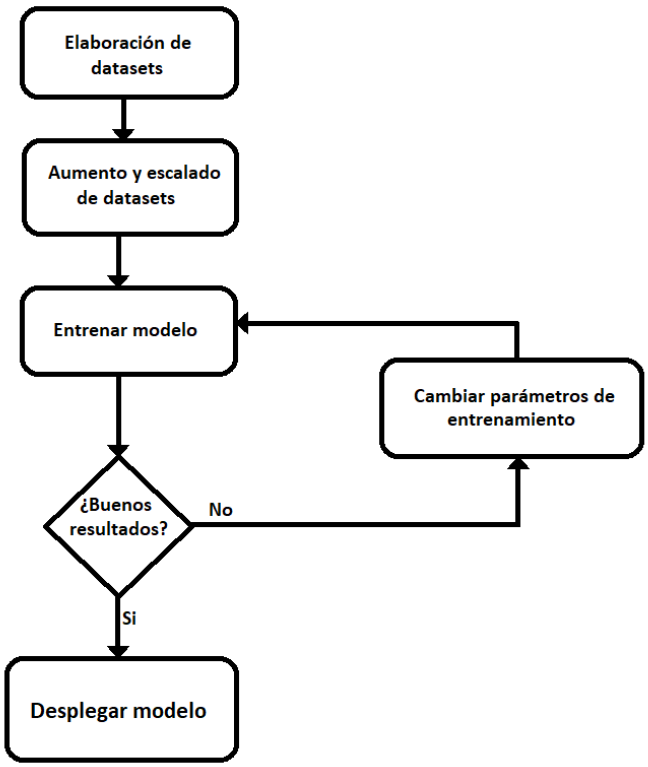


Ilustración 30 - Diagrama del entrenamiento

Para el entrenamiento de las redes, se ha utilizado el método de entrenamiento de *Aprendizaje supervisado*, debido a que se aporta junto a las imágenes de entrada, la posición del instrumental quirúrgico como salida deseada. Cabe destacar, que cuanto más grandes sean los almacenes de entrenamiento, mejores resultados se obtendrán.

Una vez cargados estos almacenes de entrenamiento, se dividen entre entrenamiento y prueba de forma completamente aleatoria, siendo estas últimas las que nos otorgará los resultados de cada una de las redes, con una relación de 90 – 10 %, respectivamente.

Posteriormente, se aumenta el almacén de entrenamiento, con el fin de evitar el *overfitting*, mediante la función augmentData, donde primeramente se altera de forma aleatoria el color de los pixeles de cada imagen entre valores previamente fijados de contraste, brillo, saturación y tono. Tras modificar el color de la imagen, se aplica, también de forma aleatoria, una transformación

de escala o una transformación de reflexión horizontal. Una vez aumentado el almacén, se redimensiona para que las imágenes tengan las mismas dimensiones que la capa de entrada de cada una de las redes.

A continuación, se procede a modificar las redes preentrenadas, debido a que estas redes son utilizadas para clasificación de objetos y no para la detección, por lo que se deben eliminar las últimas capas dejando únicamente la red de extracción de propiedades para el posterior añadido de las capas elaboradas para cada uno de los distintos algoritmos.

Tabla 3. Última capa de extracción de propiedades

Red	Capa de extracción de propiedades
AlexNet	relu5
GoogLeNet	inception_5b-output
ResNet-50	add_16
DarkNet-53	res23

Una vez se ha implementado cada uno de los algoritmos en las distintas redes, estas son entrenadas mediante el almacén de entrenamiento ya procesado. Este entrenamiento se realiza en lotes o épocas, correspondiendo a una pasada del algoritmo sobre todo el conjunto de entrenamiento, cada época tiene un número de iteraciones dependiendo del *mini-batch size*, que es un conjunto del total de entrenamiento para que la red pueda actualizar los pesos de las redes y evaluar la función de pérdida del gradiente.

Al finalizar el entrenamiento, se utiliza el almacén de prueba, el cual, al igual que ocurría con el almacén de entrenamiento, es necesario redimensionar a las dimensiones de entrada de la red, para obtener una serie de resultados con el fin de poder comparar los dos algoritmos utilizados, así como cada una de las redes.

Estos resultados parten de la idea de obtener una coincidencia entre los valores reales, estos son la ubicación del instrumental quirúrgico en cada imagen, y los valores predichos o detectados por la red.

La precisión mide como de preciso son las predicciones, es decir, cual es el porcentaje de las predicciones son correctas.

$$Precisión = \frac{TP}{TP + FP}$$

La sensibilidad o *recall* mide la capacidad del modelo de encontrar todo los positivos, es decir, el porcentaje de los casos positivos correctamente acertados.

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

Además, se tiene una métrica que relaciona la precisión y la sensibilidad, F_1 score, la cual mide la capacidad del modelo de identificar las muestras correctamente con su clase y de no equivocarse con cualquier otra.

$$F_1\ score = \frac{2}{\frac{1}{precisión} + \frac{1}{sensibilidad}}$$

Por último, AP siendo la precisión media, mide el área bajo la curva de la gráfica si representamos como x la sensibilidad y como y la precisión.

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

Siendo p(r) la curva de la gráfica precisión-sensibilidad. Por otro lado, se define mAP como la media de las precisiones medias de cada una de las clases, siendo mAP igual que AP si se tiene una sola clase como es este caso.

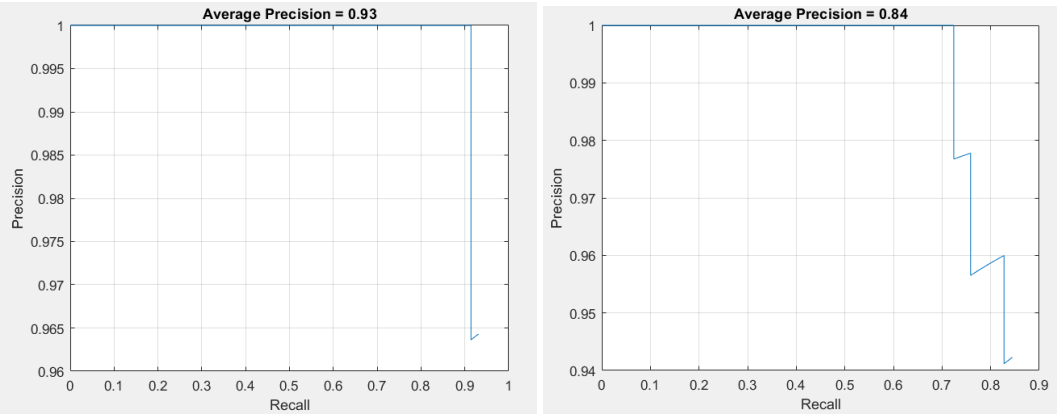


Ilustración 31 - Ejemplo de gráfica precisión - sensibilidad para el seguimiento de pinzas quirúrgicas mediante DarkNet-53 y ResNet-50, respectivamente

3.3.2.1. YOLOv2

Para la implementación del algoritmo YOLOv2 a partir de las CNN preentrenadas se eliminan las capas de clasificación, es decir, las capas a partir de las indicadas en la tabla 3, para ello se utiliza la función de MATLAB yolov2Layers, que además de eliminar las capas de clasificación añaden las capas de detección propias de YOLOv2. Esta función pide como parámetros: el tamaño de imagen de entrada de la red, el número de clases, los *anchor boxes*, la red neuronal preentrenada y la última capa de extracción de propiedades.

Una vez modificada la red neuronal al algoritmo de YOLOv2, se eligen los parámetros de entrenamiento mediante la función trainingOptions que recibe como entrada: el optimizador para el cálculo del mínimo de la función de coste, se utilizará sgdm (descenso de gradiente estocástico con momento) en cada una de las redes neuronales, el ratio inicial de entrenamiento, el número máximo de épocas, el tamaño del mini lote de entrenamiento y además, se ha elegido que se mezcle todo el conjunto de entrenamiento en cada época.

Por último, se entrena la red mediante la función trainYOLOv2ObjectDetector. Esta función trae como entradas: el *dataset* de las imágenes de entrenamiento, la red neuronal modificada y las opciones de entrenamiento.

3.3.2.2. YOLOv3

Para la implementación del algoritmo YOLOv3 a partir de las CNN preentrenadas se han añadido dos cabezas o cabezales de detección a partir de las redes anteriores. La primera cabeza incluye una capa convolucional, una capa *ReLU* y otra capa convolucional adicional. Por otro lado, la segunda cabeza incluye las mismas capas que la primera pero precedidas de una capa de *upsample* y otra capa de *depth concatenation*. Cabe destacar que la segunda cabeza es el doble de grande que la primera con el objetivo de mejorar la detección de elementos más pequeños.

Tabla 4 Capas creadas por cada cabeza de detección

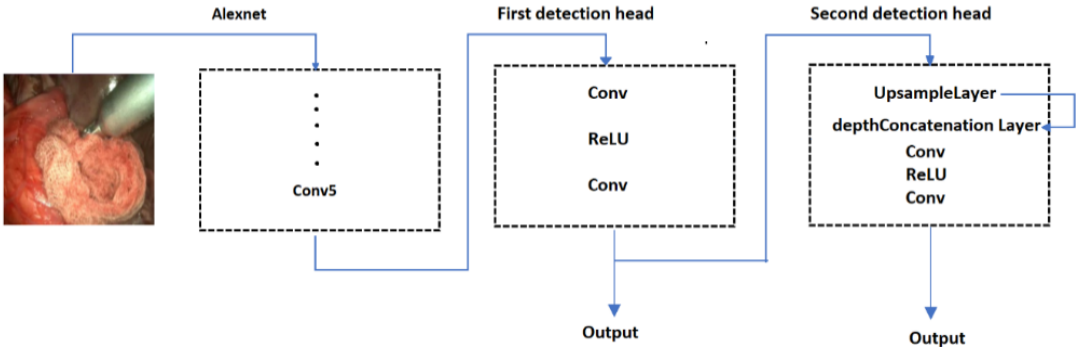
Primera cabeza de detección	Segunda cabeza de detección
Convolutional Layer (conv1Detection1)	Upsample Layer (upsample1Detection2)
ReLU Layer (relu1Detection1)	Depth Concatenation Layer (depthConcat1Detection2)
Convolutional Layer (conv2Detection1)	Convolutional Layer (conv1Detection2)
	ReLU Layer (relu1Detection2)
	Convolutional Layer (conv2Detection2)

La conexión entre estas dos cabezas con cada una de las redes preentrenadas se ha realizado como se muestra en la tabla 2. La última capa de extracción de propiedades se conecta al primer cabezal de detección, mientras que la salida de este cabezal se une con la primera capa del segundo cabezal. La capa de *depth concatenation* se conecta a una capa intermedia de la red con el objetivo de conseguir más información relevante en el segundo cabezal. Por otro lado, en AlexNet las conexiones son algo diferentes debido a los distintos tamaños de las salidas de las capas de extracción de propiedades.

Tabla 5. Conexión entre los cabezales de detección y la red de extracción de propiedades

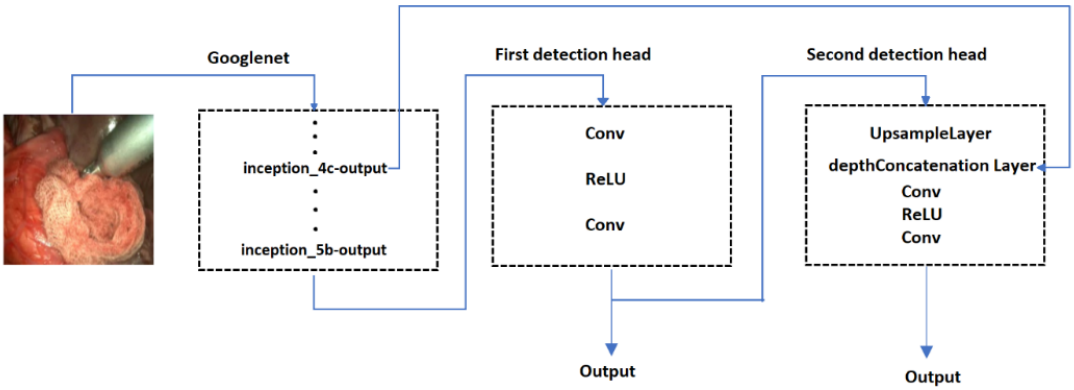
AlexNet

Source Layer	Destination Layer
conv5	conv1Detection1
relu1Detection1	upsample1Detection2
conv4	depthConcat1Detection2/in2



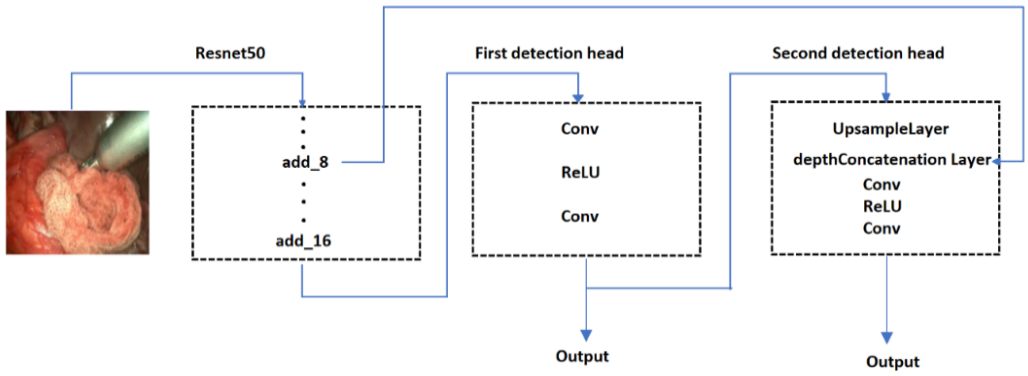
GoogLeNet

Source Layer	Destination Layer
inception_4c-output	depthConcat1Detection2/in2
inception_5b-output	conv1Detection1
relu1Detection1	upsample1Detection2



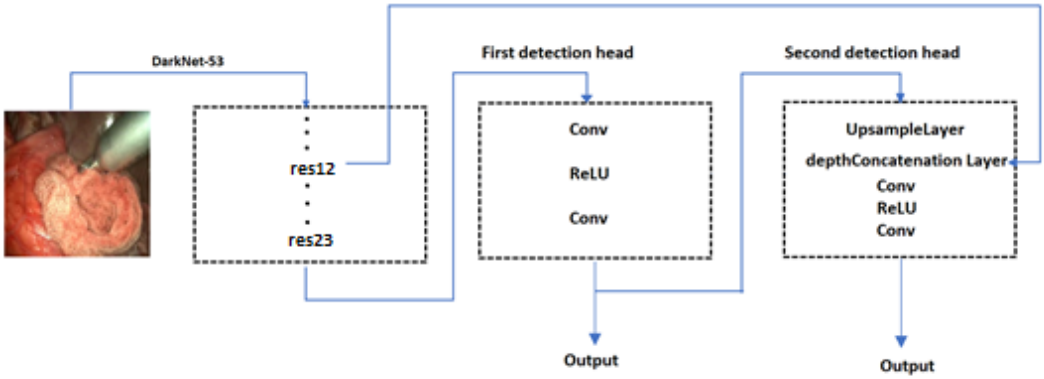
ResNet50

Source Layer	Destination Layer
add_8	depthConcat1Detection2/in2
add_16	conv1Detection1
relu1Detection1	upsample1Detection2



DarkNet-53

Source Layer	Destination Layer
res12	depthConcat1Detection2/in2
res23	conv1Detection1
relu1Detection1	upsample1Detection2



Aunque en un principio se crearon varias funciones para modificar las anteriores redes, cabe destacar que con la versión MATLAB 2021a se incorpora la función yolov3ObjectDetector, la cual realiza el mismo procedimiento que las funciones creadas en MATLAB 2020b por no existir estas mismas funciones que simplifican en gran medida el código. Esta función recibe como entradas: la red, el número de clases y las cajas de anclaje o *anchor boxes* estimadas y, por último, la opción 'DetectionNetworkSource' seguido de la última capa de extracción de propiedades (tabla 2) y de la capa de extracción de propiedades intermedia, en estas capas se añadirán los cabezales de detección.

Tras modificar las redes, se entrenan las mismas mediante un custom loop o bucle customizado con la función minibatchqueue que crea y preprocesa mini lotes, posteriormente se entrena el bucle como se indica en la documentación de MATLAB. [34]

4. Resultados

Se han desarrollado diferentes redes neuronales con el algoritmo YOLO tanto la versión 2 como la versión 3, a fin de realizar un estudio de varios parámetros con el fin de observar cual es la más adecuada para el seguimiento de instrumental quirúrgico. Se han obtenido los conceptos explicados en la sección 3.3.2: precisión, sensibilidad, recall, F_1, mAP, además de los tiempos de procesamiento o fotogramas por segundo, debido a que estos últimos serán necesarios para comprobar una posible implementación del modelo para su utilización en tiempo real.

En la detección de gasas quirúrgicas se debe tener en cuenta que la gasa no es un elemento sólido indeformable, es decir, la gasa varia su forma y tamaño a medida que el cirujano la utiliza para limpiar el interior del paciente, lo que supone también una desventaja a la hora de detectarlas. Por otro lado, en el momento que se introduce la gasa, esta se encuentra limpia o blanca, de forma que cambia su coloración según absorbe sangre, resultando en un amplio abanico de colores que pueden llegar a dificultar su detección sobre el fondo del cuerpo del paciente.

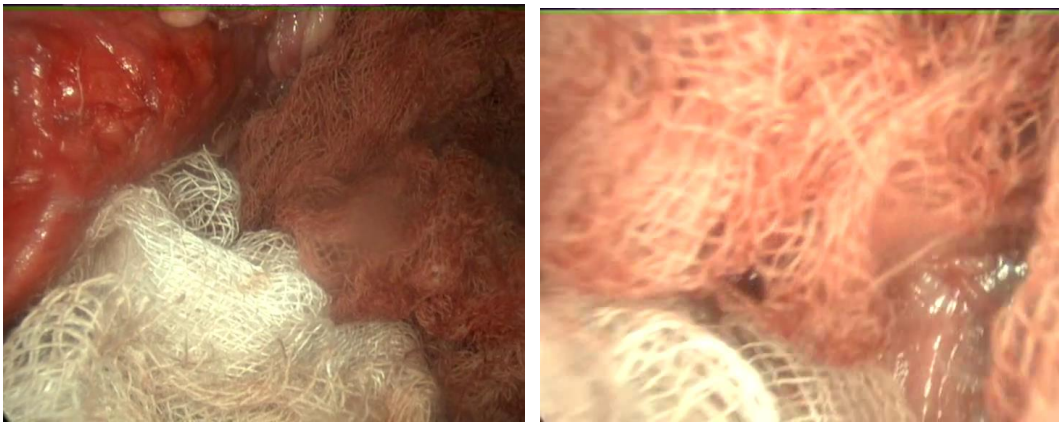


Ilustración 32 - Ejemplo de distinta coloración de las gasas en el interior del paciente

De la misma forma, también en la detección de pinzas quirúrgicas se debe tener en cuenta los distintos tipos o modelos que se pueden encontrar, en el *dataset* de gasas quirúrgicas aparece una pinza que los detectores de las redes no pueden detectar debido a no haber sido etiquetado durante el entrenamiento. Cabe destacar que también las distintas aberturas de las pinzas pueden traer problemas en la detección.

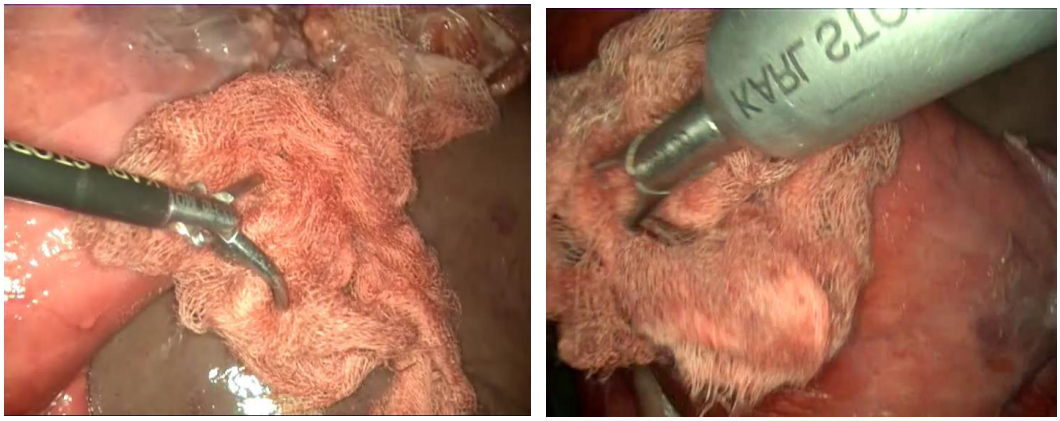


Ilustración 33 - Distintas pinzas quirúrgicas en los datasets

4.1. Seguimiento de gasas quirúrgicas

Primeramente, se ha partido del entrenamiento de las CNN mediante el algoritmo YOLOv2 con el dataset de gasas quirúrgicas, para ello se ha entrenado cada una de las redes con los parámetros de la tabla 6.

	AlexNet	GoogLeNet	ResNet-50	DarkNet-50
Tasa de aprendizaje inicial	0,0001	0,0001	0,0001	0,0001
Número máximo de épocas	15	15	15	15
Tamaño de mini lotes	10	10	8	10

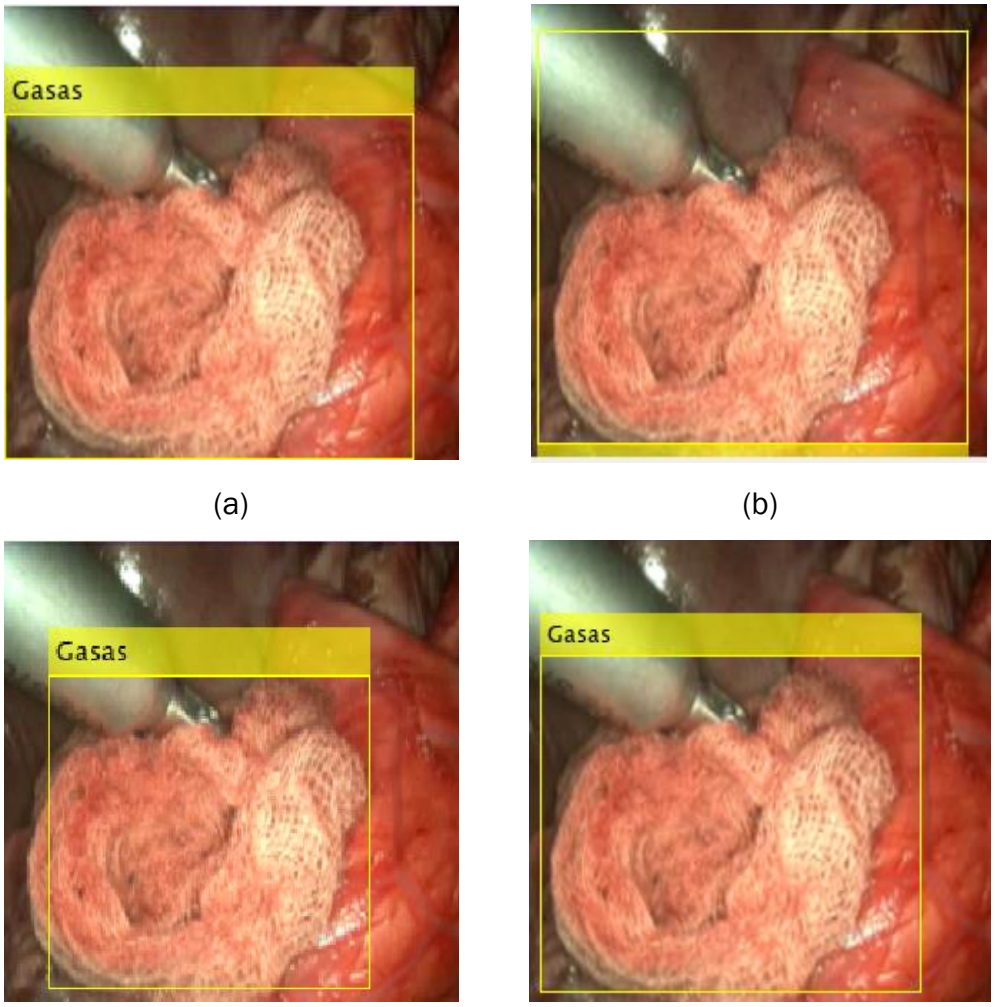
Tabla 6 - Parámetros de entrenamiento para el seguimiento de Gasas. YOLOv2

Aunque en un principio se quería entrenar las redes mediante los mismos parámetros de entrenamiento, en ResNet no fue posible escoger los mismos parámetros debido a no tener espacio suficiente en la GPU como consecuencia de ser una red muy profunda, esto se puede solucionar con un hardware más potente o, como en este caso, disminuyendo el tamaño del mini lote lo que supone un aumento en las iteraciones totales y en el tiempo total de entrenamiento, pero manteniendo los resultados. Estos parámetros se determinan mediante una serie de entrenamiento de prueba y error hasta adquirir buenos resultados.

Network	Precision[%]	Recall[%]	F1 Score[%]	mAP[%]	FPS
AlexNet	66,47	85,46	74,78	80,02	109,65
GoogLeNet	70,60	88,47	78,53	82,99	28,97
ResNet-50	81,01	88,72	84,69	85,01	13,38
DarkNet-53	88,52	92,73	90,57	90,00	22,01

Tabla 7 - Resultados de YOLOv2 para el seguimiento de gasas quirúrgicas

Observando los resultados obtenidos, podemos comprobar que la mejor red para la detección de gasas con el algoritmo YOLOv2 es DarkNet-53 debido a que cuenta con los valores más altos de precisión y sensibilidad, gracias a estos también el valor de mAP es mayor, a la vez de contar con una velocidad de procesamiento de casi tiempo real.



(a) (b)

(c) (d)

Ilustración 34 - Ejemplo del mismo fotograma detectado por AlexNet (a), GoogLeNet (b), ResNet-50 (c) y DarkNet-53 (d). YOLOv2

Observando la ilustración 34, vemos que, aunque ResNet obtiene buenos resultados, no consigue detectar completamente la gasa, además de ser la red con valores más bajos de procesamiento. Por otro lado, AlexNet y GoogLeNet

introducen demasiado fondo en la detección, siendo poco precisos a la hora de la detección. Esta baja precisión se solventa con la alta velocidad de procesamiento de estas dos redes siendo capaces de realizar detecciones en tiempo real.

Posteriormente, se ha implementado el algoritmo YOLOv3 con el mismo *dataset* de entrenamiento. En este algoritmo, se añade un nuevo parámetro de entrenamiento, el periodo de calentamiento. Este nuevo parámetro indica el número de iteraciones para aumentar la tasa de aprendizaje exponencialmente, siguiendo la siguiente formula:

$$tasa\ de\ aprendizaje\ x\ (\frac{iteración}{periodo\ de\ calentamiento})^4$$

El periodo de calentamiento ayuda a estabilizar los gradientes en tasas de aprendizaje altas.

	AlexNet	GoogLeNet	ResNet-50	DarkNet-50
Tasa de aprendizaje inicial	0,001	0,001	0,0005	0,0005
Número máximo de épocas	15	15	15	15
Tamaño de mini lotes	10	10	10	12
Periodo de calentamiento	1000	1000	1500	2000

Tabla 8 - Parámetros de entrenamiento para el seguimiento de Gasas. YOLOv3

De forma análoga al algoritmo YOLOv2, se intentó elegir los mismos parámetros de entrenamiento en todas las redes neuronales, pero, debido a que ResNet y DarkNet son redes mucho más profundas que AlexNet y GoogLeNet, se debe aumentar el periodo de calentamiento y reducir la tasa de aprendizaje inicial para evitar un error en el gradiente durante el entrenamiento. Este error producía que el gradiente tuviera valores negativos o 0, consiguiendo que la red predijera valores no aptos de *bounding boxes* lo que paraba el entrenamiento.

Network	Precision[%]	Recall[%]	F1 Score[%]	mAP[%]	FPS
AlexNet	92.11	43.86	59.42	42.92	28.15
GoogLeNet	93.95	66.17	77.65	65.08	8.95
ResNet-50	95.22	70.43	80.97	69.73	5.48
DarkNet-53	91.53	56.89	70.16	55.53	4.73

Tabla 9 - Resultados de YOLOv3 para el seguimiento de gasas quirúrgicas

Tras observar los resultados, se puede comprobar que las métricas de las redes son significativamente más elevadas, excepto en DarkNet que se reduce drásticamente la sensibilidad. Esta mejora de precisión se consigue a costa de reducir en aproximadamente un tercio la velocidad de procesamiento, resultando en que solo AlexNet es capaz de detectar gasas en tiempo real.

A diferencia del algoritmo YOLOv2, las detecciones de YOLOv3 son más precisas, ajustándose a los bordes de la gasa. AlexNet y GoogLeNet siguen introduciendo demasiado fondo en la detección, mientras que ResNet y DarkNet son más eficientes.

Aunque ResNet y Darknet son bastante precisas a la hora de detectar gasas, la primera es capaz de detectar más gasas con menos falsos positivos que Darknet, a la vez que es ligeramente más rápida.

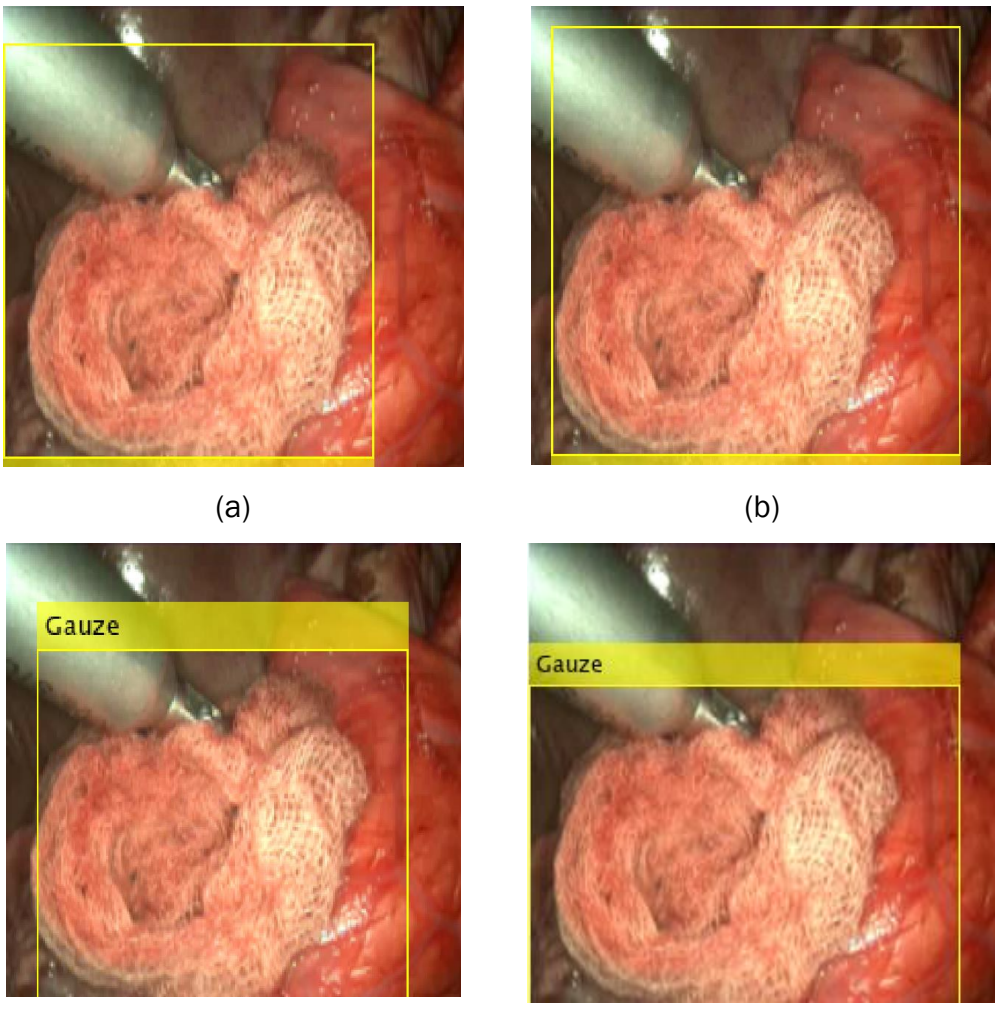


Ilustración 35 - Ejemplo del mismo fotograma detectado por AlexNet (a), GoogLeNet (b), ResNet-50 (c) y DarkNet-53 (d). YOLOv3

Por otro lado, como se mencionaba en la introducción del capítulo, es necesario comprobar también la detección con distintas coloraciones de la gasa que podemos encontrar en una operación real.

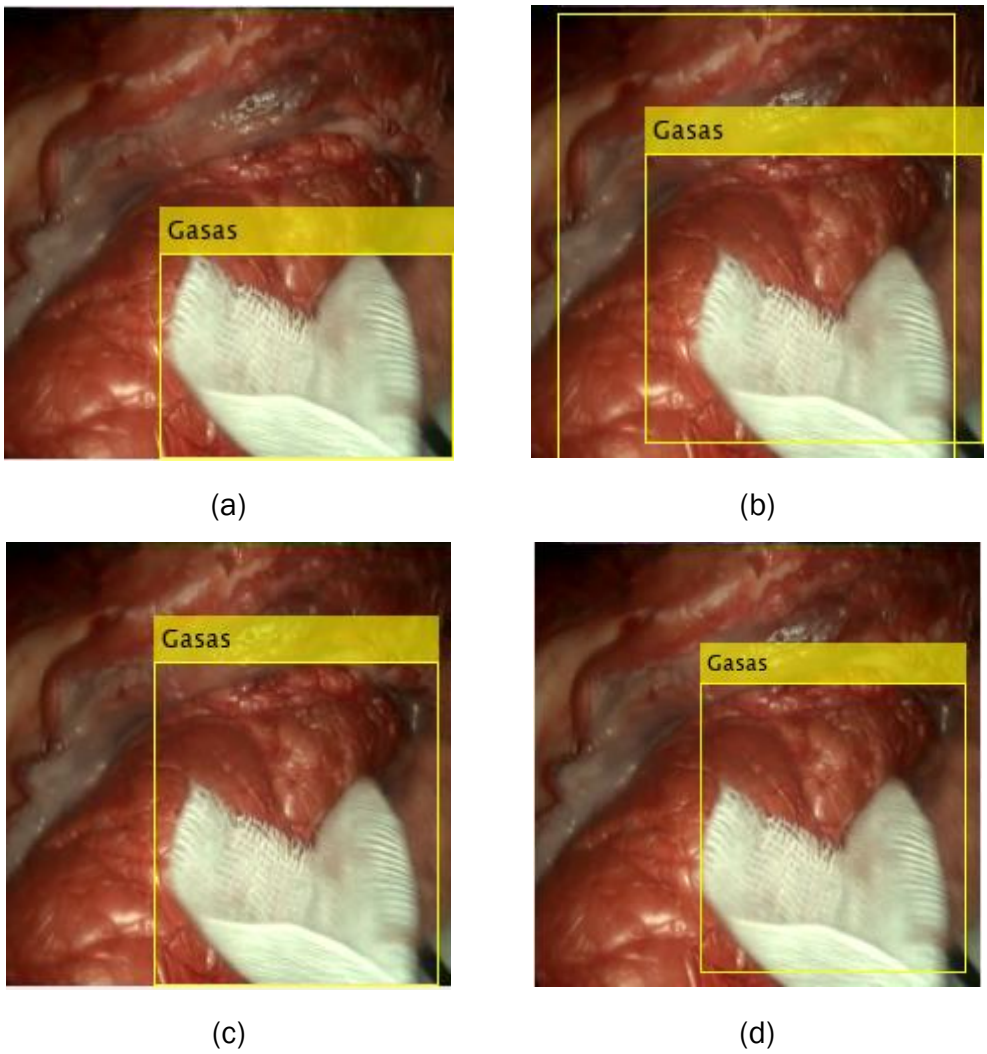
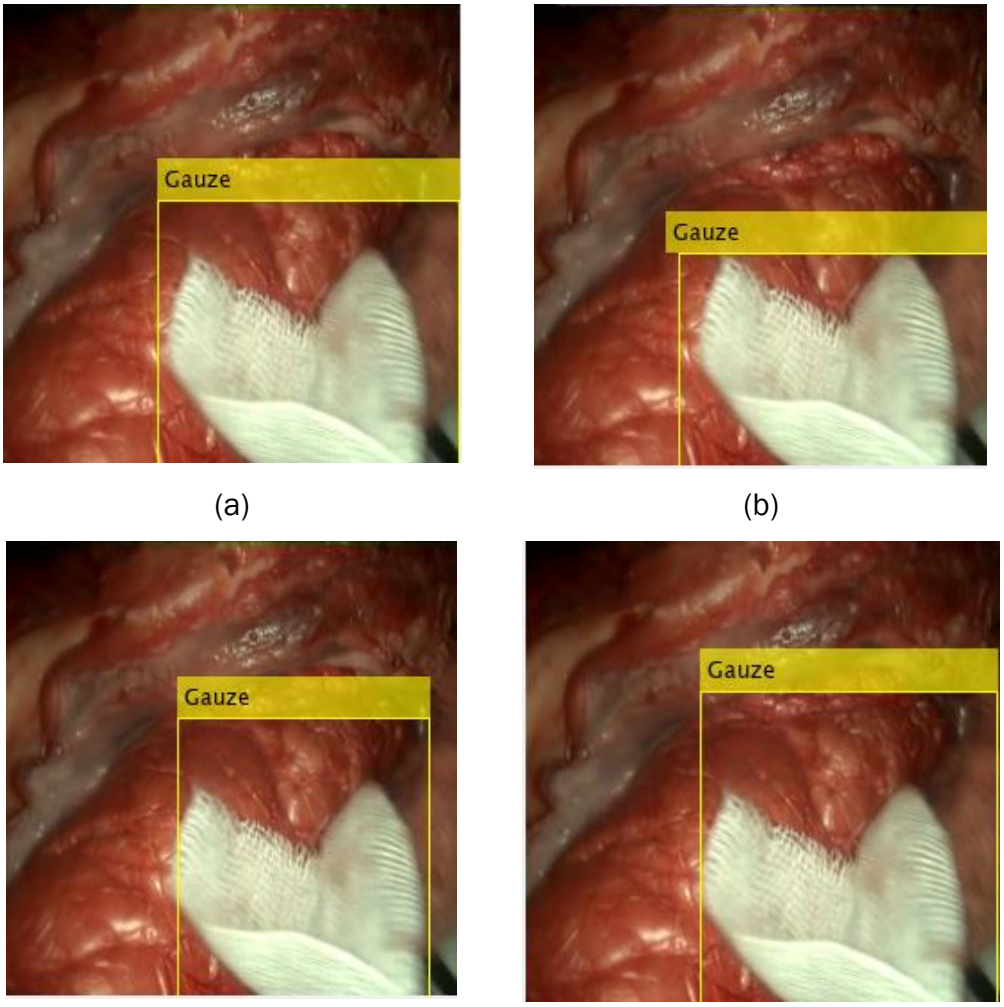


Ilustración 36 - Ejemplo de la detección de una gasa limpia por AlexNet (a), GoogLeNet (b), ResNet-50 (c) y DarkNet-53 (d). YOLOv2

Observando la ilustración 36, GoogLeNet detecta dos gasas diferentes, resultando en una detección errónea. Por otro lado, AlexNet consigue una detección prácticamente perfecta de la gasa, esto puede ser debido al alto contraste entre una gasa limpia el fondo lo que ayuda a una mejor detección a pesar de la baja precisión de esta red. ResNet consigue una detección aceptable, aunque introduzca fondo, mientras que DarkNet pierde una pequeña parte de la gasa.



(a) (b)

(c) (d)

Ilustración 37 - Ejemplo de la detección de una gasa limpia por AlexNet (a), GoogLeNet (b), ResNet-50 (c) y DarkNet-53 (d). YOLOv3

Por otro lado, utilizando el algoritmo YOLOv3, las cuatro redes son capaces de detectar la gasa limpia de manera correcta, siendo en este caso GoogLeNet la que consigue hacerlo de forma más precisa, seguida de ResNet. Aunque AlexNet y DarkNet detectan también la gasa no se ajustan tan bien a los bordes de la gasa introduciendo demasiado fondo en la detección.

Por último, se ha comprobado el funcionamiento de las redes detectando varias gasas simultáneamente, ya que, es bastante común utilizar varias gasas durante la operación que debido a que se introducen en distintos momentos de esta, tienen distinta coloración. Observando los resultados, se comprueba que ninguna de las cuatro redes es capaz de conseguir detectar ambas gasas, solo la limpia. Esto puede ser debido al etiquetado de las imágenes de entrenamiento, ya que, se ha utilizado un método de binarización, se tratan las dos gasas como una solo, por lo que durante el entrenamiento las redes tratan

de obtener características de ambas, lamentablemente, solo absorben características de la gasa que ocupe más espacio en la imagen.

Ilustración 38 - Etiquetado de dos gasas diferentes como una sola

4.2. Seguimiento de pinzas quirúrgicas

De la misma forma a la realizada en el seguimiento de gasas quirúrgicas, se empezó implementando el algoritmo YOLOv2. Como se observa en la tabla 10, los parámetros de entrenamiento de las 4 redes son los mismos, esto es debido a que el dataset de entrenamiento de pinzas es bastante más reducido que el de gasas, por lo que no se produce un error por espacio insuficiente en la GPU.

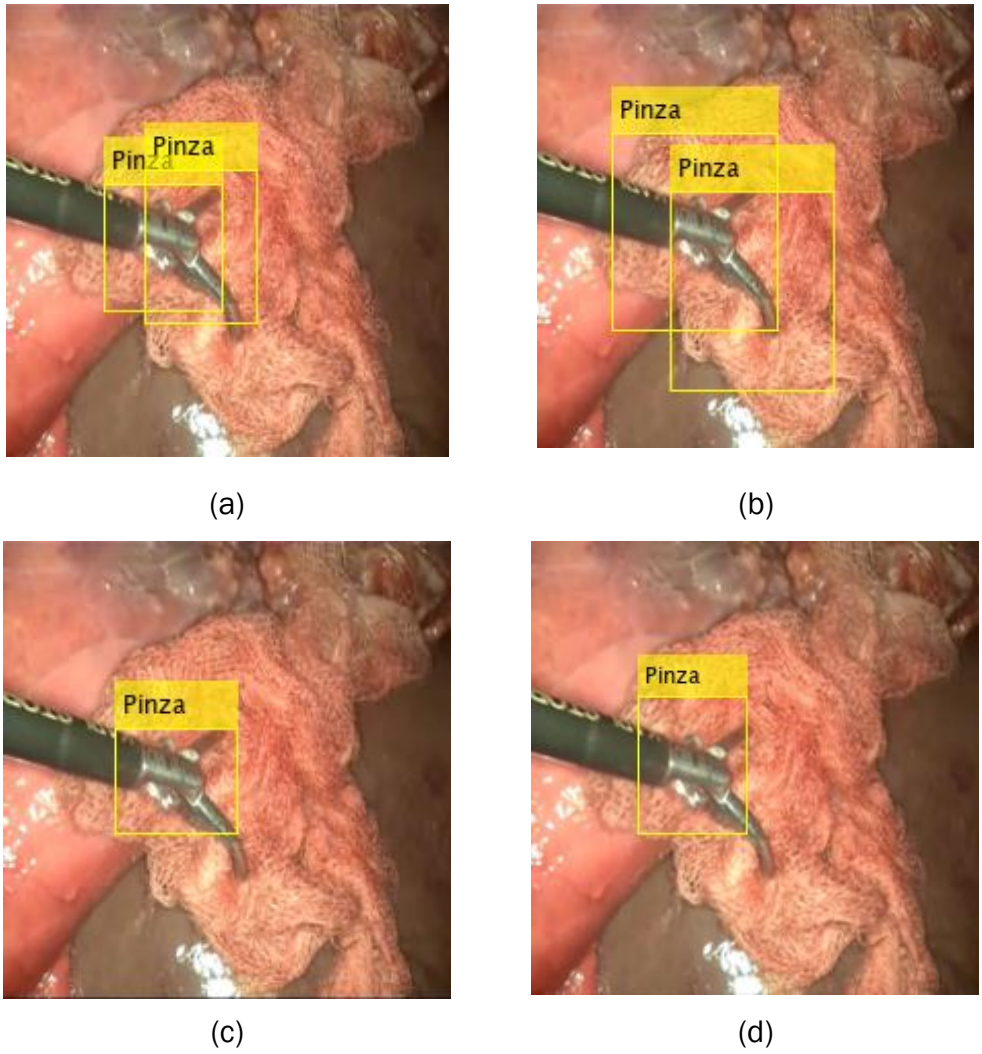
	AlexNet	GoogLeNet	ResNet-50	DarkNet-50
Tasa de aprendizaje inicial	0,0001	0,0001	0,0001	0,0001
Número máximo de épocas	15	15	15	15
Tamaño de mini lotes	10	10	10	10

Tabla 10 - Parámetros de entrenamiento para el seguimiento de pinzas quirúrgicas. YOLOv2

Debido a que el *dataset* de entrenamiento es bastante más reducido que en las gasas se obtienen resultados más pobres en estos sistemas de detección. Por otro lado, ResNet se convierte en la red más precisa, además de ser capaz de detectar pinzas quirúrgicas en tiempo real.

Network	Precision[%]	Recall[%]	F1 Score[%]	mAP[%]	FPS
AlexNet	55,70	75,86	64,24	66,98	110,75
GoogLeNet	32,14	31,09	31,61	21,38	28,56
ResNet-50	73,33	75,86	74,57	73,13	34,82
DarkNet-53	66,67	65,52	66,09	59,01	21,53

Tabla 11 - Resultados de YOLOv2 para el seguimiento de pinzas quirúrgicas



(a) (b)

(c) (d)

Ilustración 39 - Ejemplo del mismo fotograma detectado por AlexNet (a), GoogLeNet (b), ResNet-50 (c) y DarkNet-53 (d). YOLOv2

Como se observa en la ilustración 39, ninguna red es capaz de detectar completamente la pinza, siendo las más acertadas ResNet y DarkNet. Por otro lado, la detección de AlexNet y GoogLeNet no son válidas puesto que detectan dos pinzas, lo que puede suponer un grave error al aplicarse en operaciones reales.

Posteriormente, para implementar el algoritmo YOLOv3 se intentó utilizar los mismos parámetros de entrenamiento que en YOLOv2, pero esto no fue posible debido, de nuevo, al reducido *dataset* de entrenamiento. Como se mencionó en el capítulo 3.3.2, en cada época el algoritmo entrena con todas las imágenes del *dataset*, por lo que como se ve en la tabla 12, el número máximo de épocas pasa de 15 a 40 debido a que YOLOv3 necesita entrenarse con más imágenes debido a ser una red más profundo que su versión anterior.

	AlexNet	GoogLeNet	ResNet-50	DarkNet-50
Tasa de aprendizaje inicial	0,001	0,0005	0,0005	0,0005
Número máximo de épocas	40	40	40	40
Tamaño de mini lotes	10	10	8	10
Periodo de calentamiento	1000	1000	1500	1500

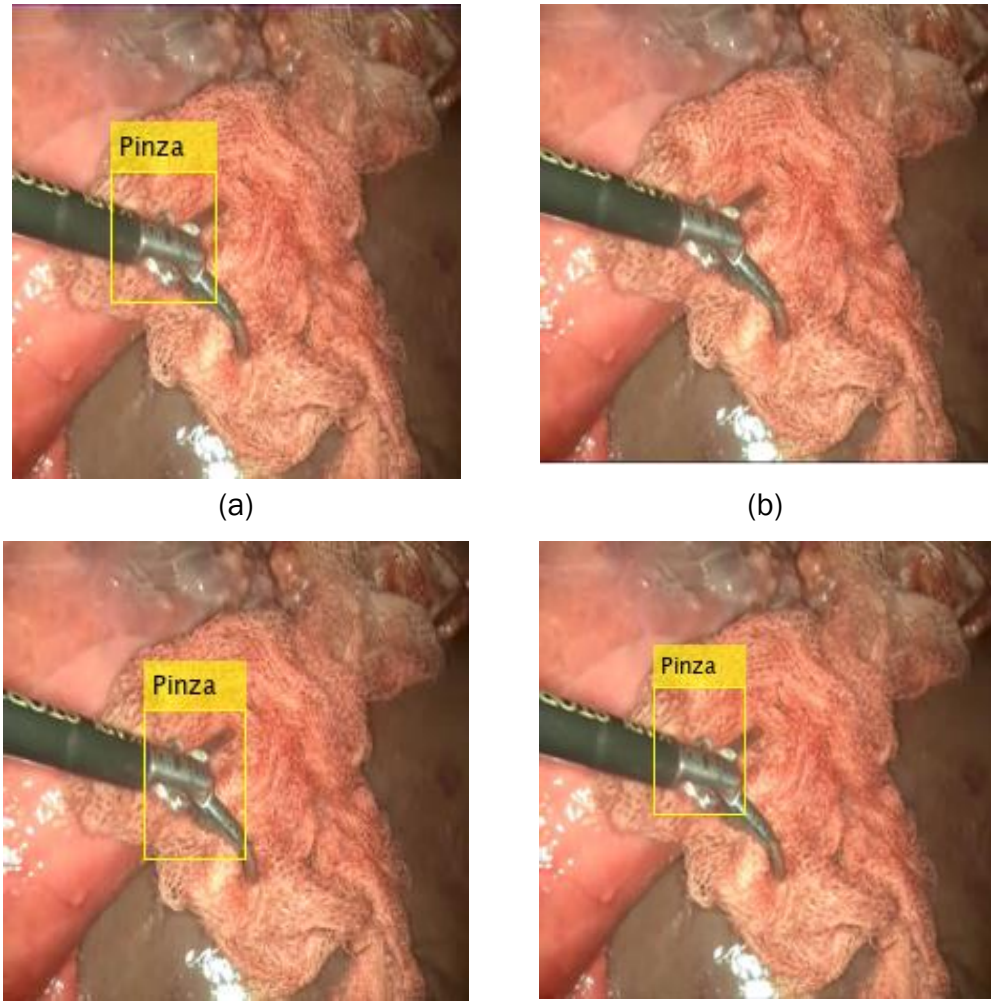
Tabla 12 - Parámetros de entrenamiento para el seguimiento de pinzas. YOLOv3

A la vista de los resultados obtenidos de las métricas de cada una de las redes, podríamos concretar que, aunque GoogLeNet y AlexNet consiguen velocidades de procesamiento superiores a ResNet y DarkNet, sus resultandos son significativamente peores, lo que no las hace óptimas para la detección. Por otro lado, ResNet y DarkNet obtienen precisiones similares en torno al 95%, siendo estos muy buenos resultados.

Network	Precision[%]	Recall[%]	F1 Score[%]	mAP[%]	FPS
AlexNet	66,67	51,72	58,25	38,86	34,43
GoogLeNet	60,00	20,69	30,77	14,77	13,39
ResNet-50	94,23	84,48	89,10	84,02	6,38
DarkNet-53	96,43	93,10	94,74	93,04	6,54

Tabla 13 - Resultados de YOLOv3 para el seguimiento de pinzas quirúrgicas

Observando los resultados de detección del fotograma detectado, ilustración 40, se ve que GoogLeNet no consigue detectar la pinza, DarkNet y AlexNet lo detectan de una forma poco precisa, incluyendo demasiado fondo. Por lo que, aunque DarkNet obtiene los mejores resultados, no es la más adecuada para detectar pinzas debido a que ese error significativo en la detección podría suponer un daño al interior del paciente.



(a) (b)

(c) (d)

Ilustración 40 - Ejemplo del mismo fotograma detectado por AlexNet (a), GoogLeNet (b), ResNet-50 (c) y DarkNet-53 (d). YOLOv3

4.3. Conclusiones

En conclusión, para el seguimiento de gasas quirúrgicas, la red que mejor realiza su función es ResNet detectando de forma más precisa las gasas e introduciendo menos fondo en la detección además de detectar menos falsos positivos, seguido de cerca por la red DarkNet. Por otro lado, debido a que el seguimiento de gasas quirúrgicas tiene la función de detectar cuando una gasa permanece dentro del cuerpo del paciente no es necesario una precisión milimétrica por lo que se puede utilizar tanto YOLOv2 como YOLOv3, aunque si no es necesario el seguimiento en tiempo real debe utilizarse YOLOv3, debido a detectar en mejor medida gasas de menor tamaño.

Por su parte, para el seguimiento de pinzas quirúrgicas, la red más adecuada independientemente del algoritmo utilizado es Resnet-50, debido a conseguir

detectar las pinzas de forma más precisa en comparación con el resto de las redes. A la hora de implementar esta red en el quirófano se debería discutir cuál sería el uso que se daría a estos algoritmos, si lo que se desea es un seguimiento para evitar material quirúrgico retenido, donde no se requiere una precisión milimétrica, sino que encuentre la pinza en todo momento, se podría utilizar el algoritmo YOLOv2 ya que consigue detectar la pinza, además de conseguir un seguimiento en tiempo real. Por otro lado, si la implementación deseada es que un sistema robótico autónomo que asista en la operación al cirujano, se debería implementar el algoritmo YOLOv3 debido a que ese pequeño margen de error en la detección que tiene YOLOv2 puede ocasionar que se sobrepase los límites máximos de trabajo y se dañe el interior del paciente, aunque esta red no consiga realizar el seguimiento en tiempo real, se podría solucionar con sistemas de procesamiento más avanzados.

5. Conclusiones y líneas futuras

Hoy en día, la tecnología brinda una gran ayuda en las cirugías, desde el uso de sistemas robóticos que permiten realizar operaciones laparoscópicas, reduciendo así en gran medida las complicaciones postoperatorias del paciente, hasta el uso de inteligencia artificial para diagnosticar y tratar enfermedades.

En este trabajo de fin de grado se ha partido de cuatro redes neuronales convolucionales distintas. En estas redes neuronales se ha implementado dos algoritmos distintos, YOLOv2 y YOLOv3, para el seguimiento y detección de instrumental quirúrgico con el fin de detectar material quirúrgico. El algoritmo resultante, aunque inicialmente está pensado para el guiado de un robot quirúrgico de forma que este pueda asistir al cirujano durante la operación, puede ser utilizado también para hacer un seguimiento de las gasas introducidas en el cuerpo del paciente y evitar la aprición de gossypibomas debido al riesgo que esto puede suponer para el paciente.

Se han elaborado dos sistemas de detección independientes, uno para el seguimiento de gasas quirúrgicas y otro para el seguimiento de pinzas quirúrgicas.

Para el seguimiento de gasas quirúrgicas la red que mejor elabora su función es ResNet, obteniendo buenos resultados con ambos algoritmos, seguido de cerca por Darknet, aunque este último introduce falsos positivos que no son óptimos para la detección. GoogLeNet y AlexNet obtienen resultados peores que las dos anteriores, cosa que suplen con sus mayores velocidades de procesamiento, lo que puede ser una buena opción si no se dispone de un hardware sofisticado. Por otro lado, se podría utilizar tanto YOLOv2 como YOLOv3 en una implementación en el quirófano, pero sugiero utilizar la tercera versión, debido a que detecta en mejor medida elementos más pequeños, aunque se pierda velocidad de procesamiento. Podemos concluir que se ha logrado el objetivo planteado al principio del proyecto de hacer un seguimiento de las gasas de forma fiable, pudiéndose conseguir una detección en tiempo real mejorando ligeramente el hardware utilizado.

Por otro lado, para el seguimiento de pinzas quirúrgicas se obtuvieron peores resultados que su contraparte de gasas debido al reducido *dataset* de entrenamiento. A diferencia del seguimiento de gasas, AlexNet y GoogLeNet no son adecuadas para su utilización debido a detectar varias veces la misma pinza o no detectarla, lo que puede generar un daño en el interior del paciente si se implementaran en sistemas robóticos. DarkNet consigue detectar la pinza de forma correcta con el algoritmo YOLOv2, pero de forma no totalmente correcta en YOLOv3. Por último, aunque ResNet no obtiene las mejoras

métricas con YOLOv3 si consigue detectar la pinza quirúrgica de manera precisa y conteniéndola de manera casi perfecta, en YOLOv2 al contrario si obtiene los mejores resultados. Para finalizar, podemos concluir que aunque YOLOv2 obtiene buenos resultados en la detección, no es aceptable para su implementación en operaciones debido a su menor precisión, ya que esto puede suponer generar daños en el interior del paciente. Todo lo contrario ocurre con YOLOv3 donde la mejora en la precisión detectando las pinzas quirúrgicas consigue que su utilización sea óptima si se desea implementar en un sistema robótico, debido a reducir las probabilidades de dañar al paciente.

Una vez finalizado el Trabajo Fin de grado se propone como continuación o líneas futuras para continuar el desarrollo del programa de detección de instrumental quirúrgico, se dispone lo siguiente:

- Aumentar el almacén de imágenes de pinzas quirúrgicas: una mejora en el *dataset* supondría una mejora en los resultados, además de que GoogLeNet no detecte tantos falsos positivos, consiguiéndose una posible red óptima para su uso.
- Implementar las nuevas versiones YOLOv4 y YOLOv5: estas nuevas versiones del algoritmo se presentan como más rápidas y precisas que las anteriores.
- Implementar una red neuronal convolucional propia que reduzca el número de capas convolucionales, lo que resulta en una velocidad de procesamiento más alta, pero consiguiendo una especialización en la detección de instrumental quirúrgico únicamente.

6. Estudio económico y temporal del proyecto

Se estima un coste económico del proyecto en 20638 euros, desglosado del siguiente modo:

- Ordenador (Gastos de amortización)..... 375 €
- Material de oficina ..... 50 €
- Conexión de internet (*) ..... 210 €
- Alquiler del lugar del trabajo (**) ..... 3150 €
- Licencia de MATLAB (***) .....467 €
- Mano de obra por el tiempo empleado (****) .....16386 €

- TOTAL .....20638 €

(*) Según la consulta del precio de varias compañías del mercado en Valladolid se estima un precio medio de 30 € por mes.

$$30\frac{€}{mes} * 7\ meses = 210\ €$$

(**) Según la consulta del precio de locales en la zona de Valladolid, se estima un precio medio de 450 € mensuales.

$$450\frac{€}{mes} * 7\ meses = 3150\ €$$

(***) Según la consulta de la página de MathWorks se estima un precio de 800 € para una licencia anual del programa MATLAB.

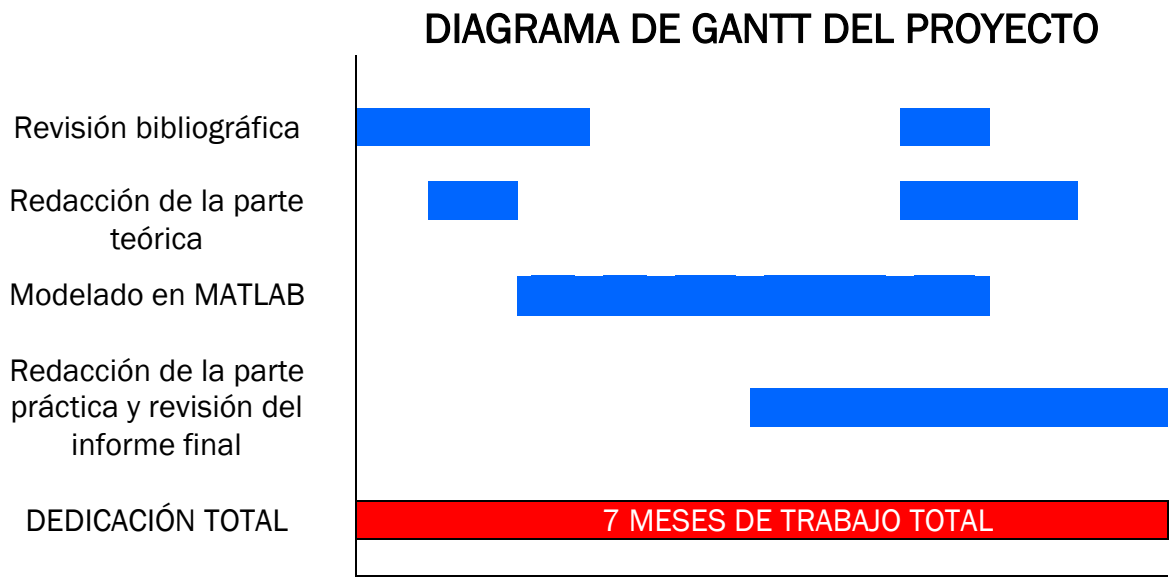
$$800\frac{€}{año} \div 12\ meses = 66{,}667\frac{€}{mes} * 7\ meses = 467\ €$$

(****) Se estima el coste de mano de obra de un ingeniero licenciado entre 14,37 y 14,88 € por hora.

$$7\ meses * \ 20\ \frac{días\ laborables}{mes} * 8\frac{h}{día} = 1120\ horas$$

$$1120\ horas * 14{,}63\frac{€}{hora} = 16386\ €$$

Por último, se muestra en el siguiente diagrama temporal del proyecto o diagrama Gantt, en el que se puede visualizar la proporción de tiempo empleado en el desarrollo de los principales apartados en los que se descompone la realización del presente proyecto.



Línea temporal (Proporciones aproximadas del tiempo empleado)

7. Bibliografía

[1] J. A. Sanz, «Breve historia de la cirugía. Hitos en el desarrollo de la cirugía moderna,» *Revista española de podología,* vol. 23, nº 5, pp. 176-182, 2012.

[2] M. Cuesta, «Cirugía laparoscópica,» *Elsevier,* vol. 68, nº 4, pp. 420-423, 2000.

[3] D. P. M. A. Ramos, «Apendicectomía Laparoscópica Vs Abierta,» 2 Mayo 2019. [En línea]. Available: http://drpabloalcazar.com/apendicectomia-laparoscopica-vs-abierta/. [Último acceso: Septiembre 2020].

[4] D. G. Wechter, «Laparoscopic surgery,» 15 6 2019. [En línea]. Available: https://medlineplus.gov/ency/presentations/100166_4.htm. [Último acceso: Octubre 2020].

[5] «Cirugía robótica: Historia e impacto en la enseñanza,» *Actas Urológicas Españolas,* vol. 35, nº 9, pp. 540-545, 2011.

[6] J. G. P. Fraga, «Actualidad de la cirugía robótica,» *Revista Cubana de Cirugía,* vol. 56, nº 1, pp. 50-61, 2011.

[7] C. M. Ramos, «Robótica y cirugía laparoscópica,» *Elsevier,* vol. 80, nº 4, pp. 189-194, 2004.

[8] «Centro de cirugía robótica,» [En línea]. Available: http://cirugiaroboticaha.com/articulo.php?art=22. [Último acceso: Noviembre 2020].

[9] R. W. Holloway, «Emergence of robotic assisted surgery in gynecologic oncology: American perspective,» *Gynecologic Oncology,* vol. 114, nº 2 Suppl, pp. 24-31, 2009.

[10] A. C. Dupeyral y G. H. Ballantyne, «Sistema quirúrgicos robóticos y telebóticos para cirugía abdominal,» *Revista Gastroenterología Perú,* vol. 23, nº 1, pp. 58-66, 2003.

[11] S. Winata, «Wireless Teleoperation Control Interface of Articulated Forceps for Minimally Invasive Surgery,» Bachelors of Engineering, Nagoya, 2018.

[12] J. Marescaux y F. Rubino, «The ZEUS robotic system: experimental and clinical applications,» *Surgical Clinics,* vol. 6, nº 83, pp. 1305-1315, 2003.

[13] abex, «da Vinci Xi,» [En línea]. Available: https://www.abexsl.es/es/sistema-robotico-da-vinci/da-vinci-xi. [Último acceso: Noviembre 2020].

[14] E. Y. P. Castaño, L. C. Carvajal, J. J. B. García y Y. S. P. Rengifo, «Estado actual de la telemedicina: una revisión de literatura,» *Ingeniare,* nº 20, pp. 105-120, 2016.

[15] L. P. L. Cánovas, L. B. L. Cánovas y A. H. Forcelledo, «Telemedicina, impacto y perspectiva para la sociedad actual,» *Universidad Médica Piñarena,* vol. 14, nº 3, pp. 289-303, 2018.

[16] V. H. R. Lauwers PR, «Intraperitoneal gossypibomas: the need to count sponges,» *World J Surg,* vol. 24, pp. 521-527, 24 Mayo 2000.

[17] I. Lata, «Gossypiboma, a rare cause of acute abdomen: A case report and review of literature,» *International Journal of Critical Illness & Injury Science,* vol. 1, nº 2, pp. 157-160, 2011.

[18] D. Anderson y G. McNeill, «Artificial Neuronal Networks Technology,» Utica, Nueva York, 1992.

[19] MathWorks, «Redes neuronales convolucionales,» [En línea]. Available: https://es.mathworks.com/discovery/convolutional-neural-network-matlab.html. [Último acceso: Octubre 2020].

[20] C. &. L. W. &. J. Y. &. S. P. &. R. S. &. A. D. &. E. D. &. V. V. &. R. A. Szegedy, «Going Deeper with Convolutions,» *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 1-9, 2015.

[21] A. Krizhevsky, I. Sutskever y G. E. Hinton, «ImageNet Classification with Deep Convolutional Neural Networks,» *Association for Computing Machinery,* vol. 60, nº 6, pp. 84-90, 2017.

[22] «Understanding GoogLeNet Model – CNN Architecture,» 2020 Mayo 3. [En línea]. Available: https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/. [Último acceso: Enero 2021].

[23] V. Feng, «An Overview of Resnet and its variants,» towards data science, 15 Julio 2017. [En línea]. Available: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035. [Último acceso: Octubre 2020].

[24] A. Kathuria, "What´s new in YOLOv3," April 2018. [Online]. Available: https://towardsdatascience.com/yolo-you-only-look-once-3dbdbb608ec4. [Accessed Octubre 2020].

[25] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection,» *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 779-788, 2016.

[26] Manishgupta, "YOLO -You Only Look Once," 30 May 2020. [Online]. Available: https://towardsdatascience.com/yolo-you-only-look-once-3dbdbb608ec4. [Accessed Noviembre 2020].

[27] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," Cornell University, Ithaca, 2018.

[28] MathWorks Deep Learning Toolbox Team, «Deep Learning Toolbox Model for ResNet-50 Network,» 10 Marzo 2021. [En línea]. Available: https://es.mathworks.com/matlabcentral/fileexchange/64626-deep-learning-toolbox-model-for-resnet-50-network. [Último acceso: Marzo 2021].

[29] MathWorks Deep Learning Toolbox Team, «Deep Learning Toolbox Model for AlexNet Network,» 10 Marzo 2021. [En línea]. Available: https://es.mathworks.com/matlabcentral/fileexchange/59133-deep-learning-toolbox-model-for-alexnet-network. [Último acceso: Marzo 2021].

[30] MathWorks Deep Learning Toolbox Team, «Deep Learning Toolbox Model for GoogLeNet Network,» 10 Marzo 2021. [En línea]. Available: https://es.mathworks.com/matlabcentral/fileexchange/64456-deep-learning-toolbox-model-for-googlenet-network. [Último acceso: Marzo 2021].

[31] MathWorks Deep Learning Toolbox Team, «Deep Learning ToolboxTM Model for DarkNet-53 Network,» 10 Marzo 2021. [En línea]. Available: https://es.mathworks.com/matlabcentral/fileexchange/74600-deep-learning-toolboxtm-model-for-darknet-53-network. [Último acceso: Marzo 2021].

[32] MathWorks Deep Learning Toolbox Team, «Deep Learning Toolbox,» [En línea]. Available: https://es.mathworks.com/products/deep-learning.html. [Último acceso: Febrero 2021].

[33] «Parallel computing toolbox,» Marzo 2021. [En línea]. Available: https://es.mathworks.com/products/parallel-computing.html. [Último acceso: Marzo 2021].

[34] MathWorks Help Center, «Object Detection Using YOLOv3 Deep Learning,» Marzo 2021. [En línea]. Available: https://es.mathworks.com/help/vision/ug/object-detection-using-yolo-v3-deep-learning.html. [Último acceso: Marzo 2021].

8. Anexos

En este anexo se presentan los códigos para entrenar y su posterior despliegue en video de las cuatro redes neuronales desarrolladas.

Este código se divide en dos partes, una parte para el seguimiento y detección de gasas quirúrgicas, la segunda parte para el seguimiento y detección de instrumental quirúrgico. Cada parte se divide en tres tipos diferentes:

- El primer tipo de código se encarga del entrenamiento de AlexNet, GoogLeNet, ResNet-50 y DarkNet-53 mediante el algoritmo YOLOv2.
- El segundo tipo de código se encarga del entrenamiento de AlexNet, GoogLeNet, ResNet-50 y DarkNet-53 mediante el algoritmo YOLOv3.
- Por ultimo, el tercer tipo se encarga del despliegue de los algoritmos para la detección de instrumental quirúrgico en video.

Los datasets de entrenamiento se cargan de la misma forma en el entrenamiento de las redes, pero de distinta manera dependiendo de si es el dataset de gasas o pinzas quirúrgicas.

8.1. Dataset de gasas quirúrgicas

```
% Reading bbox coordinates and image location from Train.txt
File = fopen('Train.txt');
Format = '%s %f %f %f %f %*[^\n]';
A = textscan(File,Format,'Delimiter',',','EmptyValue',NaN);
fclose(File);

imageFilename = string(A{1,1});
min_x = A{1,2};
min_y = A{1,3};
max_x = A{1,4};
max_y = A{1,5};

Tabla = table(imageFilename,min_x,min_y,max_x,max_y);
Tabla = rmmissing(Tabla);


% Image input size 720x57628
Gauzes = {size(Tabla)};
for i=1:size(Tabla)
    % Adjusting bbox dimension if any exceed the image input size
    imageFilename2(i) = Tabla{i,1};
    min_x(i) = Tabla{i,2};
    min_y(i) = Tabla{i,3};
    max_x(i) = Tabla{i,4};
    max_y(i) = Tabla{i,5};


    if min_x(i) <= 0
        min_x(i) = 1;
    end
```

```
    if min_y(i) <= 0
        min_y(i) = 1;
    end
    if max_x(i) >= 720
        max_x(i) = 720;
    end
    if max_y(i) >= 576
        max_y(i) = 576;
    end
    Gauzes{i,1} = [min_x(i), min_y(i), max_x(i)-min_x(i),
max_y(i)-min_y(i)];

end

imageFilename2 = imageFilename2.';
BoundingBoxes_Gauzes = table(imageFilename2,Gauzes);
BoundingBoxes_Gauzes =
standardizeMissing(BoundingBoxes_Gauzes,NaN);
BoundingBoxes_Gauzes = rmmissing(BoundingBoxes_Gauzes);
```

8.2. Dataset de pinzas quirúrgicas

```
% Cargamos la tabla con las imagenes y bbox
data = load('BoundingBoxes_Herramientas.mat');
imageFilename = data.gTruth.DataSource.Source;
Pinza = data.gTruth.LabelData{:,1};
BoundingBoxes_Herramientas = table(imageFilename,Pinza);
```

8.3. YOLOv2

8.3.1. AlexNet

```
% Cargamos la red alexnet preentrenada
net = alexnet;

rng(0);
shuffledIndices = randperm(height(BoundingBoxes_Herramientas));
idx = floor(0.9 * length(shuffledIndices));
trainingDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(1:idx),:);
testDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(idx+1:end),:);

% Obtenemos los almacenes de entrenamiento y prueba
imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));
imdstest = imageDatastore(testDataTbl{:,'imageFilename'});
bldstest = boxLabelDatastore(testDataTbl(:,2:end));
trainingData = combine(imdsTrain,bldsTrain);
testData = combine(imdstest,bldstest);

Tamano_imagen = [224 224 3];
Num_Clases = 1;
```

```
% Aumentamos el almacen de datos de entrenamiento y lo escalamos a
[224 224]
TrainingData_aumentado = transform(trainingData,@augmentData);
trainingData_aumentado_escalado =
transform(TrainingData_aumentado,@(data)PreprocessDatastore(data,
Tamano_imagen));

% Creamos las anchor boxes para la red Yolo
trainingData_escalado =
transform(trainingData,@(data)PreprocessDatastore(data,
Tamano_imagen));
anchorBoxes = estimateAnchorBoxes(trainingData_escalado,5);

% Fijamos las opciones y las capas de entrenamiento
Opciones = trainingOptions('sgdm','InitialLearnRate',0.0001, ...
      'MaxEpochs',15, 'MiniBatchSize',10,'Shuffle','every-epoch');
Capas = yolov2Layers(Tamano_imagen,Num_Clases, ...
      anchorBoxes,net,'relu5');

% Entrenamos la red YOLO
[Detector_Objetos, informacion] =
trainYOLOv2ObjectDetector(trainingData_aumentado_escalado, ...
      Capas, Opciones);

figure
plot(informacion.TrainingLoss);
grid on
xlabel('Number of Iterations')
ylabel('Training Loss for Each Iteration')

preprocessedTestData =
transform(testData,@(data)PreprocessDatastore(data,Tamano_imagen);
detectionResults = detect(Detector_Objetos, preprocessedTestData);
[ap,recall,precision] =
evaluateDetectionPrecision(detectionResults,preprocessedTestData);

figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f',ap))

save('Detector_Alexnet_Yolov2.mat','Detector_Objetos');
```

8.3.2. GoogLeNet

```
% Cargamos la red GoogleNet preentrenada
net = googlenet;

rng(0);
shuffledIndices = randperm(height(BoundingBoxes_Herramientas));
idx = floor(0.9 * length(shuffledIndices));
trainingDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(1:idx),:);
```

```
testDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(idx+1:end),:);

% Obtenemos los almacenes de entrenamiento y prueba
imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));
imdstest = imageDatastore(testDataTbl{:,'imageFilename'});
bldstest = boxLabelDatastore(testDataTbl(:,2:end));
trainingData = combine(imdsTrain,bldsTrain);
testData = combine(imdstest,bldstest);

Tamano_imagen = [224 224 3];
Num_Clases = 1;

% Aumentamos el almacen de datos de entrenamiento y lo escalamos a
[224 224]
TrainingData_aumentado = transform(trainingData,@augmentData);
trainingData_aumentado_escalado =
transform(TrainingData_aumentado,@(data)PreprocessDatastore(data,
Tamano_imagen));


% Creamos las anchor boxes para la red Yolo
trainingData_escalado =
transform(trainingData,@(data)PreprocessDatastore(data,
Tamano_imagen));
anchorBoxes = estimateAnchorBoxes(trainingData_escalado,5);

% Fijamos las opciones y las capas de entrenamiento
Opciones = trainingOptions('sgdm','InitialLearnRate',0.0001, ...
      'MaxEpochs',15,'MiniBatchSize',10,'Shuffle','every-epoch');
Capas = yolov2Layers(Tamano_imagen,Num_Clases, ...
      anchorBoxes,net,'inception_5b-output');

%Entrenamos la red YOLO
[Detector_Objetos, informacion] =
trainYOLOv2ObjectDetector(trainingData_aumentado_escalado, ...
      Capas, Opciones);

figure
plot(informacion.TrainingLoss);
grid on
xlabel('Number of Iterations')
ylabel('Training Loss for Each Iteration')

preprocessedTestData =
transform(testData,@(data)PreprocessDatastore(data,Tamano_imagen);
detectionResults = detect(Detector_Objetos, preprocessedTestData);
[ap,recall,precision] =
evaluateDetectionPrecision(detectionResults,preprocessedTestData);

figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f',ap))

save('Detector_Googlenet_Yolov2.mat','Detector_Objetos');
```

8.3.3. ResNet-50

```
% Cargamos la red ResNet-50 preentrenada
net = resnet50;

rng(0);
shuffledIndices = randperm(height(BoundingBoxes_Herramientas));
idx = floor(0.9 * length(shuffledIndices));
trainingDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(1:idx),:);
testDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(idx+1:end),:);

% Obtenemos los almacenes de entrenamiento y prueba
imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));
imdstest = imageDatastore(testDataTbl{:,'imageFilename'});
bldstest = boxLabelDatastore(testDataTbl(:,2:end));
trainingData = combine(imdsTrain,bldsTrain);
testData = combine(imdstest,bldstest);

Tamano_imagen = [224 224 3];
Num_Clases = 1;

% Aumentamos el almacen de datos de entrenamiento y lo escalamos a
[224 224]
TrainingData_aumentado = transform(trainingData,@augmentData);
trainingData_aumentado_escalado =
transform(TrainingData_aumentado,@(data)PreprocessDatastore(data,
Tamano_imagen));

% Creamos las anchor boxes para la red Yolo
trainingData_escalado =
transform(trainingData,@(data)PreprocessDatastore(data,
Tamano_imagen));
anchorBoxes = estimateAnchorBoxes(trainingData_escalado,5);

% Fijamos las opciones y las capas de entrenamiento
Opciones = trainingOptions('sgdm','InitialLearnRate',0.0001, ...
      'MaxEpochs',15, 'MiniBatchSize',10,'Shuffle','every-epoch');
Capas = yolov2Layers(Tamano_imagen,Num_Clases,anchorBoxes, ...
      net, 'activation_40_relu');

%Entrenamos la red YOLO
[Detector_Objetos, informacion] =
trainYOLOv2ObjectDetector(trainingData_aumentado_escalado, ...
      Capas, Opciones);

figure
plot(informacion.TrainingLoss);
grid on
xlabel('Number of Iterations')
ylabel('Training Loss for Each Iteration')

preprocessedTestData =
transform(testData,@(data)PreprocessDatastore(data,Tamano_imagen);
detectionResults = detect(Detector_Objetos, preprocessedTestData);
```

```
[ap,recall,precision] =
evaluateDetectionPrecision(detectionResults, preprocessedTestData);

figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f',ap))

save('Detector_Resnet_Yolov2.mat','Detector_Objetos');
```

8.3.4. DarkNet-53

```
%Cargamos la red DarkNet-53 preentrenada
net = darknet53;

rng(0);
shuffledIndices = randperm(height(BoundingBoxes_Herramientas));
idx = floor(0.9 * length(shuffledIndices));
trainingDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(1:idx),:);
testDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(idx+1:end),:);

% Obtenemos los almacenes de entrenamiento y prueba
imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));
imdstest = imageDatastore(testDataTbl{:,'imageFilename'});
bldstest = boxLabelDatastore(testDataTbl(:,2:end));
trainingData = combine(imdsTrain,bldsTrain);
testData = combine(imdstest,bldstest);

Tamano_imagen = [224 224 3];
Num_Clases = 1;

% Aumentamos el almacen de datos de entrenamiento y lo escalamos a
[224 224]
TrainingData_aumentado = transform(trainingData,@augmentData);
trainingData_aumentado_escalado =
transform(TrainingData_aumentado,...
    @(data)PreprocessDatastore(data,Tamano_imagen));

% Creamos las anchor boxes para la red Yolo
trainingData_escalado =
transform(trainingData,@(data)PreprocessDatastore(data,Tamano_imag
en));
anchorBoxes = estimateAnchorBoxes(trainingData_escalado,5);

% Fijamos las opciones y las capas de entrenamiento
Opciones = trainingOptions('sgdm','InitialLearnRate',0.0001, ...
    'MaxEpochs',15,'MiniBatchSize',10, ...
    'Shuffle','every-epoch');
Capas = yolov2Layers(Tamano_imagen,Num_Clases, ...
       anchorBoxes,net,'res23');
```

```
%Entrenamos la red YOLO
[Detector_Objetos, informacion] =
trainYOLOv2ObjectDetector(trainingData_aumentado_escalado,...
    Capas, Opciones);
% figure
plot(informacion.TrainingLoss);
grid on
xlabel('Number of Iterations')
ylabel('Training Loss for Each Iteration')

preprocessedTestData =
transform(testData,@(data)PreprocessDatastore(data,Tamano_imagen));
detectionResults = detect(Detector_Objetos, preprocessedTestData);
[ap,recall,precision] =
evaluateDetectionPrecision(detectionResults,
preprocessedTestData);

figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f',ap))

save('Detector_Darknet_Yolov2.mat','Detector_Objetos');
```

8.4. YOLOv3

Para implementar el algoritmo YOLOv3 se han utilizado dos procedimientos diferentes, uno para la versión de MATLAB 2020b, donde no se contaba con una función para implementar dicho algoritmo y se crearon unas propias. Por otro lado, para la versión 2021a, donde MATLAB añadió la función yolov3ObjectDetector, la cual permite ahorrar código para realizar la misma implementación del algoritmo.

8.4.1. MATLAB 2020b

8.4.1.1. AddFirstDetectionHead

```
function lgraph =
addFirstDetectionHead(lgraph,anchorBoxMasks,numPredictorsPerAnchor)
% The addFirstDetectionHead function adds the first detection head.
numAnchorsScale1 = size(anchorBoxMasks, 2);
% Compute the number of filters for last convolution layer.
numFilters = numAnchorsScale1*numPredictorsPerAnchor;
firstDetectionSubNetwork = [
    convolution2dLayer(3,256,'Padding','same','Name', ...
    'conv1Detection1','WeightsInitializer','he')
    reluLayer('Name','relu1Detection1')
    convolution2dLayer(1,numFilters,'Padding','same', ...
    'Name','conv2Detection1','WeightsInitializer','he')
    ];
```

```
lgraph = addLayers(lgraph,firstDetectionSubNetwork);
end
```

8.4.1.2. AddFirstDetectionHeadAlexnet

```
function lgraph =
addFirstDetectionHeadAlexnet(lgraph,anchorBoxMasks,numPredictorsPer
Anchor)
numAnchorsScale1 = size(anchorBoxMasks, 2);
% Compute the number of filters for last convolution layer.
numFilters = numAnchorsScale1*numPredictorsPerAnchor;
firstDetectionSubNetwork = [
    convolution2dLayer(3,256,'Padding','same','Name', ...
    'conv1Detection1','WeightsInitializer','he')
    reluLayer('Name','relu1Detection1')
    convolution2dLayer(1,numFilters,'Padding','same', ...
    'Name','conv2Detection1','WeightsInitializer','he')
      ];
lgraph = [lgraph;firstDetectionSubNetwork];
end
```

8.4.1.3. AddSecondDetectionHead

```
function lgraph =
addSecondDetectionHead(lgraph,anchorBoxMasks,numPredictorsPerAnchor
)
% The addSecondDetectionHead function adds the second detection
head.

numAnchorsScale2 = size(anchorBoxMasks, 2);
% Compute the number of filters for the last convolution layer.
numFilters = numAnchorsScale2*numPredictorsPerAnchor;

secondDetectionSubNetwork = [
    upsampleLayer(2,'upsample1Detection2')
    depthConcatenationLayer(2, 'Name', 'depthConcat1Detection2');
    convolution2dLayer(3,128,'Padding','same','Name', ...
      'conv1Detection2','WeightsInitializer','he')
    reluLayer('Name','relu1Detection2')
    convolution2dLayer(1,numFilters,'Padding','same','Name', ...
      'conv2Detection2','WeightsInitializer','he')
    ];
lgraph = addLayers(lgraph,secondDetectionSubNetwork);
end
```

8.4.1.4. AddSecondDetectionHeadAlexnet

```
function lgraph =
addSecondDetectionHeadAlexnet(lgraph,anchorBoxMasks,numPredictorsP
erAnchor)
% The addSecondDetectionHead function adds the second detection
head.
numAnchorsScale2 = size(anchorBoxMasks, 2);
% Compute the number of filters for the last convolution layer.
numFilters = numAnchorsScale2*numPredictorsPerAnchor;

secondDetectionSubNetwork = [
    upsampleLayer(2,'upsample1Detection2')
    depthConcatenationLayer(2, 'Name', 'depthConcat1Detection2');
    convolution2dLayer(3,128,'Padding','same','Name', ...
      'conv1Detection2','WeightsInitializer','he')
    reluLayer('Name','relu1Detection2')
    convolution2dLayer(1,numFilters,'Padding','same','Name', ...
      'conv2Detection2','WeightsInitializer','he')
    ];
lgraph = [lgraph;secondDetectionSubNetwork];
end
```

8.4.1.5. AlexnetFeatureExtractor

```
function lgraph = AlexnetFeatureExtractor(net, imageInputSize)
% La función AlexnetFeatureExtractor elimina las capas tras
'relu5' en AlexNet y elimina cualquier data
% normalization en la capa de entrada de la red.

% Convert to layerGraph.
lgraph = net.Layers(2:end-11);

inputLayer =
imageInputLayer(imageInputSize,'Normalization','none', ...
      'Name','data');
lgraph = [inputLayer; lgraph];
end
```

8.4.1.6. GooglenetFeatureExtractor

```
function lgraph = GooglenetFeatureExtractor(net, imageInputSize)
% La función GooglenetFeatureExtractor elimina las capas tras
% 'inception_5b-output' y elimina cualquier
% data normalization en la capa de entrada de la red.

% Convert to layerGraph.
lgraph = layerGraph(net);

lgraph = removeLayers(lgraph, {'pool5-7x7_s1' ...
      'pool5-drop_7x7_s1' 'loss3-classifier' 'prob' 'output'});
```

```
inputLayer =
imageInputLayer(imageInputSize,'Normalization','none', ...
      'Name','data');
lgraph = replaceLayer(lgraph,'data',inputLayer);
end
```

8.4.1.7. Resnet50FeatureExtractor

```
function lgraph = Resnet50FeatureExtractor(net, imageInputSize)
% La función Resnet50FeatureExtractor elimina las capas tras
% 'add_16' y elimina cualquier data normalization
% en la capa de entrada de la red.

% Convert to layerGraph.
lgraph = layerGraph(net);

lgraph = removeLayers(lgraph, {'activation_49_relu' ...
      'avg_pool' 'fc1000' 'fc1000_softmax' ...
      'ClassificationLayer_fc1000'});
inputLayer = imageInputLayer(imageInputSize, ...
      'Normalization','none', 'Name','data');
lgraph = replaceLayer(lgraph,'input_1',inputLayer);
end
```

8.4.1.8. DarknetFeatureExtractor

```
function lgraph = DarknetFeatureExtractor(net, imageInputSize)
% la función DarknetFeatureExtractor elimina las capas desde
% 'res23' en Darknet-53 y cualquier normalización de datos en la
% capa de entrada

lgraph = layerGraph(net);
lgraph = removeLayers(lgraph, {'avg1' 'conv53' ...
      'softmax' 'output'});
inputLayer =
imageInputLayer(imageInputSize,'Normalization','none','Name', ...
      'input');
lgraph = replaceLayer(lgraph,'input',inputLayer);
end
```

8.4.1.9. AlexNet

```
net = alexnet;

% Train parameters
Input_size = [224 224 3];
Num_Clases = 1;
numEpochs = 15;
miniBatchSize = 10;
learningRate = 0.001;
warmupPeriod = 1000;
```

```
l2Regularization = 0.0005;
penaltyThreshold = 0.5;
velocity = [];

rng(0);
shuffledIndices = randperm(height(BoundingBoxes_Gauzes));
idx = floor(0.9 * length(shuffledIndices));
trainingDataTbl = BoundingBoxes_Gauzes(shuffledIndices(1:idx),:);
testDataTbl = BoundingBoxes_Gauzes(shuffledIndices(idx+1:end),:);

% Train and test datastores
imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename2'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));
imdstest = imageDatastore(testDataTbl{:,'imageFilename2'});
bldstest = boxLabelDatastore(testDataTbl(:,2:end));
trainingData = combine(imdsTrain,bldsTrain);
testData = combine(imdstest,bldstest);

validateInputData(trainingData);
validateInputData(testData);

% Augmenting and preprocessing trainingdata
Augmented_TrainingData = transform(trainingData,@augmentData);
Preprocess_TrainingData = transform(Augmented_TrainingData, ...
      @(data)PreprocessDatastore(data,Input_size));


% Creating anchor boxes for YOLO
rng(0);
trainingData_Estimation =
transform(trainingData,@(data)PreprocessDatastore(data,Input_size)
);
[anchorBoxes, meanIoU] =
estimateAnchorBoxes(trainingData_Estimation,6);

area = anchorBoxes(:, 1).*anchorBoxes(:, 2);
[~, idx] = sort(area, 'descend');
anchorBoxes = anchorBoxes(idx, :);
anchorBoxMasks = {[1,2,3],[4,5,6]};

classNames = {'Gauzes'};
numClasses = size(classNames, 2);
numPredictorsPerAnchor = 5 + numClasses;

% Adding detection heads
lgraph = AlexnetFeatureExtractor(net, Input_size);
lgraph = layerGraph(lgraph);

lgraph = addFirstDetectionHead(lgraph, anchorBoxMasks{1},
numPredictorsPerAnchor);
lgraph = addSecondDetectionHead(lgraph, anchorBoxMasks{2},
numPredictorsPerAnchor);

lgraph = connectLayers(lgraph, 'conv5', 'conv1Detection1');
lgraph = connectLayers(lgraph, 'relu1Detection1',
'upsample1Detection2');
lgraph = connectLayers(lgraph, 'upsample1Detection2',
'depthConcat1Detection2/in2');
networkOutputs = ["conv2Detection1","conv2Detection2"];
```

```
mbqTrain = minibatchqueue(Preprocess_TrainingData, ...
      2,"MiniBatchSize", miniBatchSize,"MiniBatchFcn", ...
      @(images, boxes, labels) createBatchData(images, ...
      boxes, labels, classNames), "MiniBatchFormat", ...
      ["SSCB", ""],"DispatchInBackground", true, ...
      "OutputCast", ["", "double"]);

% Convert layer graph to dlnetwork.
net = dlnetwork(lgraph);

% Create subplots for the learning rate and mini-batch loss.
fig = figure;
[lossPlotter, learningRatePlotter] =
configureTrainingProgressPlotter(fig);
iteration = 0;
% Custom training loop.
for epoch = 1:numEpochs
    shuffle(mbqTrain);

    while(hasdata(mbqTrain))
        iteration = iteration + 1;

        [XTrain, YTrain] = next(mbqTrain);



        % Evaluate the model gradients and loss using dlfeval and
        % the modelGradients function.
        [gradients, state, lossInfo] = dlfeval(@modelGradients,...
            net, XTrain, YTrain, anchorBoxes, anchorBoxMasks, ...
            penaltyThreshold, networkOutputs);

        % Apply L2 regularization.
        gradients = dlupdate(@(g,w) g + l2Regularization*w, ...
             gradients, net.Learnables);

        % Determine the current learning rate value.
        currentLR = piecewiseLearningRateWithWarmup(iteration, ...
             epoch, learningRate, warmupPeriod, numEpochs);

        % Update the network learnable parameters using the SGDM
        % optimizer.
        [net, velocity] = sgdmupdate(net, gradients, ...
            velocity, currentLR);

        % Update the state parameters of dlnetwork.
        net.State = state;

        % Display progress.
        displayLossInfo(epoch, iteration, currentLR, lossInfo);

        % Update training plot with new points.
        updatePlots(lossPlotter, learningRatePlotter, ...
            iteration, currentLR, lossInfo.totalLoss);
    end
end
```

```
% Evaluating model
confidenceThreshold = 0.5;
overlapThreshold = 0.5;

% Create the test datastore.
preprocessedTestData = transform(testData,
@(data)PreprocessDatastore(data,Input_size));

% Create a table to hold the bounding boxes, scores, and labels
returned by the detector
numImages = size(testDataTbl, 1);
results = table('Size', [0 3], 'VariableTypes', ...
      {'cell','cell','cell'},'VariableNames', ...
      {'Boxes','Scores','Labels'});

mbqTest = minibatchqueue(preprocessedTestData, 1, ...
      "MiniBatchSize", miniBatchSize, "MiniBatchFormat", "SSCB");

% Run detector on images in the test set and collect results.
while hasdata(mbqTest)
    % Read the datastore and get the image.
    XTest = next(mbqTest);

    % Run the detector.
    [bboxes, scores, labels] = yolov3Detect(net, XTest, ...
            networkOutputs, anchorBoxes, anchorBoxMasks, ...
            confidenceThreshold, overlapThreshold, classNames);

    % Collect the results.
    tbl = table(bboxes, scores, labels, 'VariableNames', ...
            {'Boxes','Scores','Labels'});
    results = [results; tbl];
end

% Evaluate the object detector using Average Precision metric.
[ap, recall, precision] = evaluateDetectionPrecision(results, ...
       preprocessedTestData);

% Plot precision-recall curve.
figure
plot(recall, precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f', ap))

save('Detector_Alexnet.mat','net');
save('Salidas_Red_Alexnet','networkOutputs')
save('AnchorBox_Alexnet','anchorBoxes');
save('AnchorBoxMask_Alexnet','anchorBoxMasks');
```

8.4.1.10. GoogLeNet

```
net = googlenet;

% Train parameters
Input_size = [224 224 3];
Num_Clases = 1;
numEpochs = 15;
miniBatchSize = 10;
learningRate = 0.001;
warmupPeriod = 1000;
l2Regularization = 0.0005;
penaltyThreshold = 0.5;
velocity = [];

rng(0);
shuffledIndices = randperm(height(BoundingBoxes_Gauzes));
idx = floor(0.9 * length(shuffledIndices));
trainingDataTbl = BoundingBoxes_Gauzes(shuffledIndices(1:idx),:);
testDataTbl = BoundingBoxes_Gauzes(shuffledIndices(idx+1:end),:);

% Train and test datastores
imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename2'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));
imdstest = imageDatastore(testDataTbl{:,'imageFilename2'});
bldstest = boxLabelDatastore(testDataTbl(:,2:end));
trainingData = combine(imdsTrain,bldsTrain);
testData = combine(imdstest,bldstest);

validateInputData(trainingData);
validateInputData(testData);

% Augmenting and preprocessing trainingdata
Augmented_TrainingData = transform(trainingData,@augmentData);
Preprocess_TrainingData = transform(Augmented_TrainingData, ...
      @(data)PreprocessDatastore(data,Input_size));

% Creating anchor boxes for YOLO
rng(0);
trainingData_Estimation =
transform(trainingData,@(data)PreprocessDatastore(data,Input_size)
);
[anchorBoxes, meanIoU] =
estimateAnchorBoxes(trainingData_Estimation,6);

area = anchorBoxes(:, 1).*anchorBoxes(:, 2);
[~, idx] = sort(area, 'descend');
anchorBoxes = anchorBoxes(idx, :);
anchorBoxMasks = {[1,2,3],[4,5,6]};

classNames = {'Gauzes'};
numClasses = size(classNames, 2);
numPredictorsPerAnchor = 5 + numClasses;

% Adding detection heads
lgraph = GooglenetFeatureExtractor(net, Tamano_imagen);
```

```
lgraph = addFirstDetectionHead(lgraph, anchorBoxMasks{1}, ...
      numPredictorsPerAnchor);
lgraph = addSecondDetectionHead(lgraph, anchorBoxMasks{2}, ...
      numPredictorsPerAnchor);

lgraph = connectLayers(lgraph, 'inception_5b-output', ...
      'conv1Detection1');
lgraph = connectLayers(lgraph, 'relu1Detection1', ...
      'upsample1Detection2');
lgraph = connectLayers(lgraph, 'inception_4c-output', ...
      'depthConcat1Detection2/in2');
networkOutputs = ["conv2Detection1","conv2Detection2"];

mbqTrain = minibatchqueue(Preprocess_TrainingData, 2, ...
      "MiniBatchSize", miniBatchSize,"MiniBatchFcn", ...
      @(images, boxes, labels) createBatchData(images, boxes, ...
      labels, classNames), "MiniBatchFormat", ["SSCB", ""], ...
      "DispatchInBackground", true,"OutputCast", ["", "double"]);

% Convert layer graph to dlnetwork.
net = dlnetwork(lgraph);

% Create subplots for the learning rate and mini-batch loss.
fig = figure;
[lossPlotter, learningRatePlotter] =
configureTrainingProgressPlotter(fig);

iteration = 0;
% Custom training loop.
for epoch = 1:numEpochs

    reset(mbqTrain);
    shuffle(mbqTrain);

    while(hasdata(mbqTrain))
        iteration = iteration + 1;

        [XTrain, YTrain] = next(mbqTrain);

        % Evaluate the model gradients and loss using dlfeval and
        % the modelGradients function.
        [gradients, state, lossInfo] = dlfeval(@modelGradients,...
            net, XTrain, YTrain, anchorBoxes, anchorBoxMasks, ...
            penaltyThreshold, networkOutputs);

        % Apply L2 regularization.
        gradients = dlupdate(@(g,w) g + l2Regularization*w, ...
            gradients, net.Learnables);

        % Determine the current learning rate value.
        currentLR = piecewiseLearningRateWithWarmup(iteration, ...
             epoch, learningRate, warmupPeriod, numEpochs)


        % Update the network learnable parameters using the SGDM
        % optimizer.
        [net, velocity] = sgdmupdate(net, gradients, ...
             velocity, currentLR);
```

```
        % Update the state parameters of dlnetwork.
        net.State = state;

        % Display progress.
        displayLossInfo(epoch, iteration, currentLR, lossInfo);

        % Update training plot with new points.
        updatePlots(lossPlotter, learningRatePlotter, ...
            iteration, currentLR, lossInfo.totalLoss);
    end
end

% Evaluamos el modelo
confidenceThreshold = 0.5;
overlapThreshold = 0.5;

% Create the test datastore.
preprocessedTestData = transform(testData,
@(data)PreprocessDatastore(data,Input_size));

% Create a table to hold the bounding boxes, scores, and labels
returned by
% the detector.
numImages = size(testDataTbl, 1);
results = table('Size', [0 3], 'VariableTypes', ...
      {'cell','cell','cell'},'VariableNames', ...
      {'Boxes','Scores','Labels'});

mbqTest = minibatchqueue(preprocessedTestData, 1, ...
      "MiniBatchSize", miniBatchSize, "MiniBatchFormat", "SSCB");

% Run detector on images in the test set and collect results.
while hasdata(mbqTest)

    % Read the datastore and get the image.
    XTest = next(mbqTest);

    % Run the detector.
    [bboxes, scores, labels] = yolov3Detect(net, XTest, ...
            networkOutputs, anchorBoxes, anchorBoxMasks, ...
            confidenceThreshold, overlapThreshold, classNames);

    % Collect the results.
    tbl = table(bboxes, scores, labels, 'VariableNames', ...
       {'Boxes','Scores','Labels'});
    results = [results; tbl];
end

% Evaluate the object detector using Average Precision metric.
[ap, recall, precision] = evaluateDetectionPrecision(results, ...
       preprocessedTestData);

% Plot precision-recall curve.
figure
plot(recall, precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f', ap))
```

```
save('Detector_Googlenet.mat','net');
save('Salidas_Red_Googlenet','networkOutputs')
save('AnchorBox_Googlenet','anchorBoxes');
save('AnchorBoxMask_Googlenet','anchorBoxMasks');
```

8.4.1.11. ResNet-50

```
net= resnet50;

% Train parameters
Input_size = [224 224 3];
Num_Clases = 1;
numEpochs = 15;
miniBatchSize = 10;
learningRate = 0.0005;
warmupPeriod = 1500;
l2Regularization = 0.0005;
penaltyThreshold = 0.5;
velocity = [];

rng(0);
shuffledIndices = randperm(height(BoundingBoxes_Gauzes));
idx = floor(0.9 * length(shuffledIndices));
trainingDataTbl = BoundingBoxes_Gauzes(shuffledIndices(1:idx),:);
testDataTbl = BoundingBoxes_Gauzes(shuffledIndices(idx+1:end),:);

% Train and test datastores
imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename2'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));
imdstest = imageDatastore(testDataTbl{:,'imageFilename2'});
bldstest = boxLabelDatastore(testDataTbl(:,2:end));
trainingData = combine(imdsTrain,bldsTrain);
testData = combine(imdstest,bldstest);

validateInputData(trainingData);
validateInputData(testData);

% Augmenting and preprocessing trainingdata
Augmented_TrainingData = transform(trainingData,@augmentData);
Preprocess_TrainingData =
transform(Augmented_TrainingData,@(data)PreprocessDatastore(data,
Input_size));

% Creating anchor boxes for YOLO
rng(0);
trainingData_Estimation =
transform(trainingData,@(data)PreprocessDatastore(data,Input_size)
);
[anchorBoxes, meanIoU] =
estimateAnchorBoxes(trainingData_Estimation,6);

area = anchorBoxes(:, 1).*anchorBoxes(:, 2);
[~, idx] = sort(area, 'descend');
anchorBoxes = anchorBoxes(idx, :);
anchorBoxMasks = {[1,2,3],[4,5,6]};
```

```
classNames = {'Gauzes'};
numClasses = size(classNames, 2);
numPredictorsPerAnchor = 5 + numClasses;

% Adding detection heads
lgraph = Resnet50FeatureExtractor(net, Input_size);

lgraph = addFirstDetectionHead(lgraph, anchorBoxMasks{1}, ...
      numPredictorsPerAnchor);
lgraph = addSecondDetectionHead(lgraph, anchorBoxMasks{2},...
       numPredictorsPerAnchor);

lgraph = connectLayers(lgraph, 'add_16', 'conv1Detection1');
lgraph = connectLayers(lgraph, 'relu1Detection1', ...
      'upsample1Detection2');
lgraph = connectLayers(lgraph, 'add_8', ...
      'depthConcat1Detection2/in2');
networkOutputs = ["conv2Detection1","conv2Detection2"];

mbqTrain = minibatchqueue(Preprocess_TrainingData, ...
      2,"MiniBatchSize", miniBatchSize,"MiniBatchFcn", ...
      @(images, boxes, labels) createBatchData(images, boxes, ...
      labels, classNames), "MiniBatchFormat", ["SSCB", ""], ...
      "DispatchInBackground", true,"OutputCast", ["", "double"]);

% Convert layer graph to dlnetwork.
net = dlnetwork(lgraph);

% Create subplots for the learning rate and mini-batch loss.
fig = figure;
[lossPlotter, learningRatePlotter] =
configureTrainingProgressPlotter(fig);

iteration = 0;
% Custom training loop.
for epoch = 1:numEpochs

    reset(mbqTrain);
    shuffle(mbqTrain);

    while(hasdata(mbqTrain))
        iteration = iteration + 1;

        [XTrain, YTrain] = next(mbqTrain);

        % Evaluate the model gradients and loss using dlfeval and
        % the modelGradients function.
        [gradients, state, lossInfo] = dlfeval(@modelGradients,...
            net, XTrain, YTrain, anchorBoxes, anchorBoxMasks, ...
            penaltyThreshold, networkOutputs);

        % Apply L2 regularization.
        gradients = dlupdate(@(g,w) g + l2Regularization*w, ...
             gradients, net.Learnables);

        % Determine the current learning rate value.
        currentLR = piecewiseLearningRateWithWarmup(iteration, ...
             epoch, learningRate, warmupPeriod, numEpochs);
```

```
        % Update the network learnable parameters using the SGDM
        % optimizer.
        [net, velocity] = sgdmupdate(net, gradients, ...
            velocity, currentLR);

        % Update the state parameters of dlnetwork.
        net.State = state;

        % Display progress.
        displayLossInfo(epoch, iteration, currentLR, lossInfo);

        % Update training plot with new points.
        updatePlots(lossPlotter, learningRatePlotter, ...
             iteration, currentLR, lossInfo.totalLoss);
    end
end

% Evaluating model
confidenceThreshold = 0.5;
overlapThreshold = 0.5;

% Create the test datastore.
preprocessedTestData = transform(testData, ...
      @(data)PreprocessDatastore(data,Input_size));

% Create a table to hold the bounding boxes, scores, and labels
returned by
% the detector.
numImages = size(testDataTbl, 1);
results = table('Size', [0 3], 'VariableTypes', ...
      {'cell','cell','cell'},'VariableNames', ...
      {'Boxes','Scores','Labels'});

mbqTest = minibatchqueue(preprocessedTestData, 1, ...
      "MiniBatchSize", miniBatchSize, "MiniBatchFormat", "SSCB");

% Run detector on images in the test set and collect results.
while hasdata(mbqTest)
    % Read the datastore and get the image.
    XTest = next(mbqTest);

    % Run the detector.
    [bboxes, scores, labels] = yolov3Detect(net, XTest, ...
      networkOutputs, anchorBoxes, anchorBoxMasks, ...
      confidenceThreshold, overlapThreshold, classNames);

    % Collect the results.
    tbl = table(bboxes, scores, labels, 'VariableNames', ...
      {'Boxes','Scores','Labels'});
    results = [results; tbl];
end


% Evaluate the object detector using Average Precision metric.
[ap, recall, precision] = evaluateDetectionPrecision(results, ...
       preprocessedTestData);
```

```
% Plot precision-recall curve.
figure
plot(recall, precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f', ap))

save('Detector_Resnet50.mat','net');
save('Salidas_Red_Resnet50','networkOutputs')
save('AnchorBox_Resnet50','anchorBoxes');
save('AnchorBoxMask_Resnet50','anchorBoxMasks');
```

8.4.1.12. DarkNet-53

```
%Cargamos la red DarkNet-53 preentrenada
net = darknet53;

% Train parameters
Input_size = [256 256 3];
numEpochs = 10;
miniBatchSize = 12;
learningRate = 0.0005;
warmupPeriod = 2000;
l2Regularization = 0.0005;
penaltyThreshold = 0.5;
velocity = [];

rng(0);
shuffledIndices = randperm(height(BoundingBoxes_Gauzes));
idx = floor(0.9 * length(shuffledIndices));
trainingDataTbl = BoundingBoxes_Gauzes(shuffledIndices(1:idx),:);
testDataTbl = BoundingBoxes_Gauzes(shuffledIndices(idx+1:end),:);

% Train and test datastores
imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename2'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));
imdstest = imageDatastore(testDataTbl{:,'imageFilename2'});
bldstest = boxLabelDatastore(testDataTbl(:,2:end));
trainingData = combine(imdsTrain,bldsTrain);
testData = combine(imdstest,bldstest);

validateInputData(trainingData);
validateInputData(testData);

% Creating anchor boxes for YOLO
rng(0);
trainingData_Estimation =
transform(trainingData,@(data)PreprocessDatastore(data,Input_size)
);
anchorBoxes = estimateAnchorBoxes(trainingData_Estimation,6);

area = anchorBoxes(:, 1).*anchorBoxes(:, 2);
[~, idx] = sort(area, 'descend');
anchorBoxes = anchorBoxes(idx, :);
anchorBoxMasks = {[1,2,3],[4,5,6]};
```

```
classNames = {'Gauzes'};
numClasses = size(classNames, 2);
numPredictorsPerAnchor = 5 + numClasses;

% Adding detection heads
net = DarknetFeatureExtractor(net, Input_size);
net = addFirstDetectionHead(net, anchorBoxMasks{1}, ...
      numPredictorsPerAnchor);
net = addSecondDetectionHead(net, anchorBoxMasks{2}, ...
      numPredictorsPerAnchor);
net = connectLayers(net, 'res23', 'conv1Detection1');
net = connectLayers(net, 'relu1Detection1', ...
      'upsample1Detection2');
net = connectLayers(net, 'res12', 'depthConcat1Detection2/in2');
networkOutputs = ["conv2Detection1" "conv2Detection2"];

mbqTrain = minibatchqueue(trainingData_Estimation, 2,...
        "MiniBatchSize", miniBatchSize,...
        "MiniBatchFcn", @(images, boxes, labels)
        createBatchData(images, boxes, labels, classNames), ...
        "MiniBatchFormat", ["SSCB", ""],...
        "DispatchInBackground", true,...
        "OutputCast", ["", "double"]);

% Convert layer graph to dlnetwork.
    net = dlnetwork(net);

    % Create subplots for the learning rate and mini-batch loss.
    fig = figure;
    [lossPlotter,learningRatePlotter]= ...
            configureTrainingProgressPlotter(fig);

    iteration = 0;
    % Custom training loop.
    for epoch = 1:numEpochs

        reset(mbqTrain);
        shuffle(mbqTrain);

        while(hasdata(mbqTrain))
            iteration = iteration + 1;

            [XTrain, YTrain] = next(mbqTrain);

            % Evaluate the model gradients and loss using dlfeval
            % and the modelGradients function.
            [gradients, state, lossInfo] =
                  dlfeval(@modelGradients, net, XTrain, ...
                  YTrain, anchorBoxes, anchorBoxMasks, ...
                  penaltyThreshold, networkOutputs);

            % Apply L2 regularization.
            gradients = dlupdate(@(g,w) g + l2Regularization*w,...
                  gradients, net.Learnables);

            % Determine the current learning rate value.
            currentLR = piecewiseLearningRateWithWarmup(...
                  iteration, epoch, learningRate, ...
```

```
                    warmupPeriod, numEpochs);

                % Update the network learnable parameters using the
                % SGDM optimizer
                [net, velocity] = sgdmupdate(net, gradients, ...
                    velocity, currentLR);

                % Update the state parameters of dlnetwork.
                net.State = state;

                % Display progress.
                displayLossInfo(epoch, iteration, currentLR, ...
                    lossInfo);

                % Update training plot with new points.
                updatePlots(lossPlotter, learningRatePlotter, ...
                    iteration, currentLR, lossInfo.totalLoss);
            end
        end

confidenceThreshold = 0.5;
overlapThreshold = 0.5;

% Create the test datastore.
preprocessedTestData = transform(testData,
@(data)PreprocessDatastore(data,Input_size));

% Create a table to hold the bounding boxes, scores, and labels
% returned by the detector.
numImages = size(testData, 1);
results = table('Size', [0 3], ...
    'VariableTypes', {'cell','cell','cell'}, ...
    'VariableNames', {'Boxes','Scores','Labels'});

mbqTest = minibatchqueue(preprocessedTestData, 1, ...
    "MiniBatchSize", miniBatchSize, ...
    "MiniBatchFormat", "SSCB");

% Run detector on images in the test set and collect results.
while hasdata(mbqTest)
    % Read the datastore and get the image.
    XTest = next(mbqTest);

    % Run the detector.
    [bboxes, scores, labels] = yolov3Detect(net, XTest, ...
      networkOutputs, anchorBoxes, anchorBoxMasks, ...
      confidenceThreshold, overlapThreshold, classNames);

    % Collect the results.
    tbl = table(bboxes, scores, labels, 'VariableNames', ...
      {'Boxes','Scores','Labels'});
    results = [results; tbl];
end


% Evaluate the object detector using Average Precision metric.
[ap, recall, precision] = evaluateDetectionPrecision(results, ...
      preprocessedTestData);
```

```
% Plot precision-recall curve.
figure
plot(recall, precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f', ap))

save('Detector_Darknet.mat','net');
save('Salidas_Red_Darknet','networkOutputs')
save('AnchorBox_Darknet','anchorBoxes');
save('AnchorBoxMask_Darknet','anchorBoxMasks');
```

8.4.2. MATLAB 2021a

8.4.2.1. AlexNet

```
net = alexnet;

% Train parameters
Input_size = [224 224 3];
Num_Clases = 1;
numEpochs = 40;
miniBatchSize = 10;
learningRate = 0.001;
warmupPeriod = 1000;
l2Regularization = 0.0005;
penaltyThreshold = 0.55;
velocity = [];

rng(0);
shuffledIndices = randperm(height(BoundingBoxes_Herramientas));
idx = floor(0.9 * length(shuffledIndices));
trainingDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(1:idx),:);
testDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(idx+1:end),:);


% Obtenemos los almacenes de entrenamiento y prueba
imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));
imdstest = imageDatastore(testDataTbl{:,'imageFilename'});
bldstest = boxLabelDatastore(testDataTbl(:,2:end));
trainingData = combine(imdsTrain,bldsTrain);
testData = combine(imdstest,bldstest);

% Augmenting and preprocessing trainingdata
Augmented_TrainingData = transform(trainingData,@augmentData);
Preprocess_TrainingData =
transform(Augmented_TrainingData,@(data)PreprocessDatastore(data,I
nput_size));

% Creating anchor boxes for YOLO
rng(0);
```

```
trainingData_Estimation =
transform(trainingData,@(data)PreprocessDatastore(data,Input_size)
);
[anchorBoxes, meanIoU] =
estimateAnchorBoxes(trainingData_Estimation,6);

area = anchorBoxes(:, 1).*anchorBoxes(:, 2);
[~, idx] = sort(area, 'descend');
anchorBoxes = anchorBoxes(idx, :);
anchorBoxMasks = {anchorBoxes(1:3,:)
    anchorBoxes(4:6,:)
    };

classNames = {'Pinza'};
numClasses = size(classNames, 2);

lgraph = net.Layers(1:end-11);
net = dlnetwork(lgraph);
net = yolov3ObjectDetector(net,
classNames,anchorBoxMasks,'DetectionNetworkSource',{'conv5','conv4
'});

mbqTrain = minibatchqueue(Preprocess_TrainingData, ...
      2,"MiniBatchSize", miniBatchSize,"MiniBatchFcn", ...
      @(images, boxes, labels) createBatchData(images, boxes, ...
      labels, classNames), "MiniBatchFormat", ["SSCB",""], ...
      "DispatchInBackground", true,"OutputCast", ["", "double"]);

% Create subplots for the learning rate and mini-batch loss.
fig = figure;
[lossPlotter, learningRatePlotter] =
configureTrainingProgressPlotter(fig);

iteration = 0;
% Custom training loop.
for epoch = 1:numEpochs

    reset(mbqTrain);
    shuffle(mbqTrain);

    while(hasdata(mbqTrain))
        iteration = iteration + 1;

        [XTrain, YTrain] = next(mbqTrain);

        % Evaluate the model gradients and loss using dlfeval and
        % the modelGradients function.
        [gradients, state, lossInfo] = dlfeval(@modelGradients,...
            net, XTrain, YTrain, penaltyThreshold);

        % Apply L2 regularization.
        gradients = dlupdate(@(g,w) g + l2Regularization*w, ...
            gradients, net.Learnables);



        % Determine the current learning rate value.
        currentLR = piecewiseLearningRateWithWarmup(iteration, ...
            epoch, learningRate, warmupPeriod, numEpochs);
```

```
        % Update the network learnable parameters using the SGDM
        % optimizer.
        [net.Learnables, velocity] = sgdmupdate(net.Learnables,...
            gradients, velocity, currentLR);

        % Update the state parameters of dlnetwork.
        net.State = state;

        % Display progress.
        displayLossInfo(epoch, iteration, currentLR, lossInfo);



        % Update training plot with new points.
        updatePlots(lossPlotter, learningRatePlotter, ...
            iteration, currentLR, lossInfo.totalLoss);
    end
end

% Create the test datastore.
preprocessedTestData = transform(testData, ...
      @(data)PreprocessDatastore(data,Input_size));

results = detect(net,preprocessedTestData,'MiniBatchSize', ...
      miniBatchSize);

% Evaluate the object detector using Average Precision metric.
[ap, recall, precision] = evaluateDetectionPrecision(results, ...
      preprocessedTestData);

% Plot precision-recall curve.
figure
plot(recall, precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f', ap))

save('Detector_Alexnet.mat','net');
```

8.4.2.2. GoogLeNet

```
net = googlenet;

% Train parameters
Input_size = [224 224 3];
Num_Clases = 1;
numEpochs = 40;
miniBatchSize = 10;
learningRate = 0.0005;
warmupPeriod = 1000;
l2Regularization = 0.0005;
penaltyThreshold = 0.5;
velocity = [];
```

```
rng(0);
shuffledIndices = randperm(height(BoundingBoxes_Herramientas));
idx = floor(0.9 * length(shuffledIndices));
trainingDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(1:idx),:);
testDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(idx+1:end),:);

% Obtenemos los almacenes de entrenamiento y prueba
imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));
imdstest = imageDatastore(testDataTbl{:,'imageFilename'});
bldstest = boxLabelDatastore(testDataTbl(:,2:end));
trainingData = combine(imdsTrain,bldsTrain);
testData = combine(imdstest,bldstest);

% Augmenting and preprocessing trainingdata
Augmented_TrainingData = transform(trainingData,@augmentData);
Preprocess_TrainingData =
transform(Augmented_TrainingData,@(data)PreprocessDatastore(data,I
nput_size));

% Creating anchor boxes for YOLO
rng(0);
trainingData_Estimation =
transform(trainingData,@(data)PreprocessDatastore(data,Input_size)
);
[anchorBoxes, meanIoU] =
estimateAnchorBoxes(trainingData_Estimation,6);

area = anchorBoxes(:, 1).*anchorBoxes(:, 2);
[~, idx] = sort(area, 'descend');
anchorBoxes = anchorBoxes(idx, :);
anchorBoxMasks = {anchorBoxes(1:3,:)
    anchorBoxes(4:6,:)
    };

classNames = {'Pinza'};
numClasses = size(classNames, 2);



net = yolov3ObjectDetector(net, ...
      classNames,anchorBoxMasks,'DetectionNetworkSource', ...
      {'inception_5b-output','inception_4c-output'});

mbqTrain = minibatchqueue(Preprocess_TrainingData, ...
      2,"MiniBatchSize", miniBatchSize,"MiniBatchFcn",...
      @(images, boxes, labels) createBatchData(images, boxes, ...
      labels, classNames), "MiniBatchFormat", ["SSCB", ""], ...
      "DispatchInBackground", true,"OutputCast", ["", "double"]);

% Create subplots for the learning rate and mini-batch loss.
fig = figure;
[lossPlotter, learningRatePlotter] =
configureTrainingProgressPlotter(fig);

iteration = 0;
```

```
% Custom training loop.
for epoch = 1:numEpochs

    reset(mbqTrain);
    shuffle(mbqTrain);

    while(hasdata(mbqTrain))
        iteration = iteration + 1;

        [XTrain, YTrain] = next(mbqTrain);

        % Evaluate the model gradients and loss using dlfeval and
        % the modelGradients function.
        [gradients, state, lossInfo] = dlfeval(@modelGradients,...
            net, XTrain, YTrain, penaltyThreshold);

        % Apply L2 regularization.
        gradients = dlupdate(@(g,w) g + l2Regularization*w, ...
            gradients, net.Learnables);

        % Determine the current learning rate value.
        currentLR = piecewiseLearningRateWithWarmup(iteration, ...
            epoch, learningRate, warmupPeriod, numEpochs);

        % Update the network learnable parameters using the SGDM
        % optimizer.
        [net.Learnables,velocity] = sgdmupdate(net.Learnables, ...
            gradients, velocity, currentLR);

        % Update the state parameters of dlnetwork.
        net.State = state;

        % Display progress.
        displayLossInfo(epoch, iteration, currentLR, lossInfo);

        % Update training plot with new points.
        updatePlots(lossPlotter,learningRatePlotter, iteration,...
            currentLR, lossInfo.totalLoss);
    end
end

% Create the test datastore.
preprocessedTestData = transform(testData, ...
      @(data)PreprocessDatastore(data,Input_size));

results = detect(net,preprocessedTestData,'MiniBatchSize', ...
      miniBatchSize);

% Evaluate the object detector using Average Precision metric.
[ap, recall, precision] = evaluateDetectionPrecision(results, ...
      preprocessedTestData);


% Plot precision-recall curve.
figure
plot(recall, precision)
xlabel('Recall')
ylabel('Precision')
grid on
```

```
title(sprintf('Average Precision = %.2f', ap))

save('Detector_Googlenet.mat','net');
```

8.4.2.3. ResNet-50

```
net = resnet50;

% Train parameters
Input_size = [224 224 3];
Num_Clases = 1;
numEpochs = 40;
miniBatchSize = 8;
learningRate = 0.0005;
warmupPeriod = 1500;
l2Regularization = 0.0005;
penaltyThreshold = 0.5;
velocity = [];

rng(0);
shuffledIndices = randperm(height(BoundingBoxes_Herramientas));
idx = floor(0.9 * length(shuffledIndices));
trainingDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(1:idx),:);
testDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(idx+1:end),:);

% Obtenemos los almacenes de entrenamiento y prueba
imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));
imdstest = imageDatastore(testDataTbl{:,'imageFilename'});
bldstest = boxLabelDatastore(testDataTbl(:,2:end));
trainingData = combine(imdsTrain,bldsTrain);
testData = combine(imdstest,bldstest);

% Augmenting and preprocessing trainingdata
Augmented_TrainingData = transform(trainingData,@augmentData);
Preprocess_TrainingData =
transform(Augmented_TrainingData,@(data)PreprocessDatastore(data,I
nput_size));

% Creating anchor boxes for YOLO
rng(0);
trainingData_Estimation =
transform(trainingData,@(data)PreprocessDatastore(data,Input_size)
);
[anchorBoxes, meanIoU] =
estimateAnchorBoxes(trainingData_Estimation,6);

area = anchorBoxes(:, 1).*anchorBoxes(:, 2);
[~, idx] = sort(area, 'descend');
anchorBoxes = anchorBoxes(idx, :);
anchorBoxMasks = {anchorBoxes(1:3,:)
    anchorBoxes(4:6,:)
    };
```

```
classNames = {'Pinza'};
numClasses = size(classNames, 2);

net = yolov3ObjectDetector(net, ...
      classNames,anchorBoxMasks,'DetectionNetworkSource', ...
      {'add_16','add_8'});

mbqTrain = minibatchqueue(Preprocess_TrainingData, ...
      2,"MiniBatchSize", miniBatchSize,"MiniBatchFcn", ...
      @(images, boxes, labels) createBatchData(images, boxes, ...
      labels, classNames), "MiniBatchFormat", ["SSCB", ""], ...
      "DispatchInBackground", true,"OutputCast", ["", "double"]);

% Create subplots for the learning rate and mini-batch loss.
fig = figure;
[lossPlotter, learningRatePlotter] =
configureTrainingProgressPlotter(fig);

iteration = 0;
% Custom training loop.
for epoch = 1:numEpochs

    reset(mbqTrain);
    shuffle(mbqTrain);

    while(hasdata(mbqTrain))
        iteration = iteration + 1;

        [XTrain, YTrain] = next(mbqTrain);

        % Evaluate the model gradients and loss using dlfeval and
        % the modelGradients function.
        [gradients, state, lossInfo] = dlfeval(@modelGradients,...
            net, XTrain, YTrain, penaltyThreshold);

        % Apply L2 regularization.
        gradients = dlupdate(@(g,w) g + l2Regularization*w, ...
             gradients, net.Learnables);

        % Determine the current learning rate value.
        currentLR = piecewiseLearningRateWithWarmup(iteration, ...
             epoch, learningRate, warmupPeriod, numEpochs);

        % Update the network learnable parameters using the SGDM
        % optimizer.
        [net.Learnables, velocity] = sgdmupdate(net.Learnables,...
            gradients, velocity, currentLR);

        % Update the state parameters of dlnetwork.
        net.State = state;

        % Display progress.
        displayLossInfo(epoch, iteration, currentLR, lossInfo);

        % Update training plot with new points.
        updatePlots(lossPlotter, learningRatePlotter, ...
            iteration, currentLR, lossInfo.totalLoss);
    end
end
```

```
% Create the test datastore.
preprocessedTestData = transform(testData,
@(data)PreprocessDatastore(data,Input_size));

results = detect(net,preprocessedTestData,'MiniBatchSize', ...
       miniBatchSize);

% Evaluate the object detector using Average Precision metric.
[ap, recall, precision] = evaluateDetectionPrecision(results, ...
      preprocessedTestData);

% Plot precision-recall curve.
figure
plot(recall, precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f', ap))
save('Detector_Resnet50.mat','net');
```

8.4.2.4. DarkNet-53

```
net = darknet53;

% Train parameters
Input_size = [256 256 3];
Num_Clases = 1;
numEpochs = 40;
miniBatchSize = 10;
learningRate = 0.0005;
warmupPeriod = 1500;
l2Regularization = 0.0005;
penaltyThreshold = 0.5;
velocity = [];

rng(0);
shuffledIndices = randperm(height(BoundingBoxes_Herramientas));
idx = floor(0.9 * length(shuffledIndices));
trainingDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(1:idx),:);
testDataTbl =
BoundingBoxes_Herramientas(shuffledIndices(idx+1:end),:);

% Obtenemos los almacenes de entrenamiento y prueba
imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));
imdstest = imageDatastore(testDataTbl{:,'imageFilename'});
bldstest = boxLabelDatastore(testDataTbl(:,2:end));
trainingData = combine(imdsTrain,bldsTrain);
testData = combine(imdstest,bldstest);

% Augmenting and preprocessing trainingdata
Augmented_TrainingData = transform(trainingData,@augmentData);
```

```
Preprocess_TrainingData =
transform(Augmented_TrainingData,@(data)PreprocessDatastore(data,I
nput_size));

% Creating anchor boxes for YOLO
rng(0);
trainingData_Estimation =
transform(trainingData,@(data)PreprocessDatastore(data,Input_size)
);
[anchorBoxes, meanIoU] =
estimateAnchorBoxes(trainingData_Estimation,6);

area = anchorBoxes(:, 1).*anchorBoxes(:, 2);
[~, idx] = sort(area, 'descend');
anchorBoxes = anchorBoxes(idx, :);
anchorBoxMasks = {anchorBoxes(1:3,:)
    anchorBoxes(4:6,:)
    };

classNames = {'Pinza'};
numClasses = size(classNames, 2);

% Eliminamos las capas de clasificación, añadimos las propias de
YOLOv3
lgraph = removeLayers(net, {'avg1' 'conv53' 'softmax_' 'output'});
net = dlnetwork(lgraph);
net = yolov3ObjectDetector(net, ...
      classNames,anchorBoxMasks,'DetectionNetworkSource', ...
      {'res23','res12'});

mbqTrain = minibatchqueue(Preprocess_TrainingData, ...
      2,"MiniBatchSize", miniBatchSize,"MiniBatchFcn",...
      @(images, boxes, labels) createBatchData(images, boxes, ...
      labels, classNames), "MiniBatchFormat", ["SSCB", ""], ...
      "DispatchInBackground", true,"OutputCast", ["", "double"]);

% Create subplots for the learning rate and mini-batch loss.
fig = figure;
[lossPlotter, learningRatePlotter] =
configureTrainingProgressPlotter(fig);

iteration = 0;
% Custom training loop.
for epoch = 1:numEpochs

    reset(mbqTrain);
    shuffle(mbqTrain);

    while(hasdata(mbqTrain))
        iteration = iteration + 1;

        [XTrain, YTrain] = next(mbqTrain);

        % Evaluate the model gradients and loss using dlfeval and
        % the modelGradients function.
        [gradients, state, lossInfo] = dlfeval(@modelGradients,...
            net, XTrain, YTrain, penaltyThreshold);
```

```
            % Apply L2 regularization.
            gradients = dlupdate(@(g,w) g + l2Regularization*w, ...
                gradients, net.Learnables);


            % Determine the current learning rate value.
            currentLR = piecewiseLearningRateWithWarmup(iteration, ...
                epoch, learningRate, warmupPeriod, numEpochs);

            % Update the network learnable parameters using the SGDM
            % optimizer.
            [net.Learnables, velocity] = sgdmupdate(net.Learnables,...
                gradients, velocity, currentLR);

            % Update the state parameters of dlnetwork.
            net.State = state;

            % Display progress.
            displayLossInfo(epoch, iteration, currentLR, lossInfo);


            % Update training plot with new points.
            updatePlots(lossPlotter,learningRatePlotter, iteration,...
                currentLR, lossInfo.totalLoss);
        end
    end


    % Create the test datastore.
    preprocessedTestData = transform(testData,
    @(data)PreprocessDatastore(data,Input_size));

    results = detect(net,preprocessedTestData,'MiniBatchSize', ...
          miniBatchSize);

    % Evaluate the object detector using Average Precision metric.
    [ap, recall, precision] = evaluateDetectionPrecision(results, ...
          preprocessedTestData);


    % Plot precision-recall curve.
    figure
    plot(recall, precision)
    xlabel('Recall')
    ylabel('Precision')
    grid on
    title(sprintf('Average Precision = %.2f', ap))
    save('Detector_Darknet53.mat','net');
```

8.5. Detección de video

En este apartado se comentan las funciones utilizadas para implementar las redes neuronales en los videos laparoscópicos, así como la función para calcular los fotogramas por segundo de cada una de las redes.

8.5.1. YOLOv2

```
% Elegimos el video
N = 2;
% Cargamos el video
videoObject = VideoReader(strcat('VID000',num2str(N),'.mp4'));

% Creamos el video de salida
VideoSalida =
VideoWriter(strcat('VideoDetectado_Yolov2_000',num2str(N)),'MPEG-
4'); VideoSalida.FrameRate = videoObject.FrameRate;
open(VideoSalida);

fps = 0;
avgfps = [];
% Extract the frame from the movie structure.
for i=1:videoObject.NumFrames
    % Leemos el frame actual
    thisFrame = read(videoObject, i);
    % Redimensionamos el frame a la entrada de la red YOLO
    thisFrame = imresize(thisFrame,[224 224]);

    tic;

    % Detectamos los objetos del frame
    [bbox, score, label] = detect(net,thisFrame);

    newt = toc;
    fps = .9*fps + .1*(1/newt);
    avgfps = [avgfps; fps];

    % Si encuentra el elemento, inserta la bbox y la etiqueta
    if (~isempty(bbox))
        thisFrame = insertObjectAnnotation(thisFrame, ...
            'rectangle',bbox,label);
    end
    % Guardamos las imagenes detectadas en el video de salida
    writeVideo(VideoSalida,thisFrame);
end

Framerate = CalculoMediaFps(avgfps);
Texto = ['Fps = ',num2str(Framerate)];
disp(Texto);
close(VideoSalida);
```

8.5.2. YOLOv3

8.5.2.1. MATLAB 2020b

```
% Choose video for detection
N = 2;
% Loading video chosen
videoObject = VideoReader(strcat('VID000',num2str(N),'.mp4'));

% Creating output video
OutputVideo =
VideoWriter(strcat('YoloV3_VideoDetectado_000',num2str(N)),'MPEG-
4'); OutputVideo.FrameRate = videoObject.FrameRate;
open(OutputVideo);

fps = 0;
avgfps = [];

% Extract the frame from the movie structure.
for i=1:videoObject.NumFrames
    % Reading frame
    thisFrame = read(videoObject, i);
    thisFrame = imresize(thisFrame,[224 224]);

    tic;

    % Detecting objects on frame
    [bbox, score, label] = detect(net,thisFrame);

    newt = toc;
    fps = .9*fps + .1*(1/newt);
    avgfps = [avgfps; fps];

    % if object found, it inserts the bbox and label
    if (~isempty(score))
        thisFrame = insertObjectAnnotation(thisFrame, ...
            'rectangle',bbox,label);
    end

    % Saving frame on outputVideo
    writeVideo(OutputVideo,thisFrame);

end

Framerate = CalculoMediaFps(avgfps);
Texto = ['Fps = ',num2str(Framerate)];
disp(Texto);

close(OutputVideo);
```

8.5.2.2. MATLAB 2021a

```
% Load rest of detection parameters
confidenceThreshold = 0.5;
overlapThreshold = 0.5;
classNames = {'Gasa'};

% Choose video for detection
N = 3;
% Loading video chosen
videoObject = VideoReader(strcat('VID000',num2str(N),'.mp4'));

% Creating output video
OutputVideo =
VideoWriter(strcat('YoloV3_VideoDetectado_000',num2str(N)),'MPEG-
4'); OutputVideo.FrameRate = videoObject.FrameRate;
open(OutputVideo);

fps = 0;
avgfps = [];

% Extract the frame from the movie structure.
for i=1:videoObject.NumFrames
    % Reading frame
    thisFrame = read(videoObject, i);
    thisFrame = dlarray(im2single(imresize(thisFrame, ...
       [224 224])),'SSCB');

    executionEnvironment = "auto";
    if (executionEnvironment == "auto" ...
      && canUseGPU) || executionEnvironment == "gpu"
        thisFrame = gpuArray(thisFrame);
    end

    tic;

    % Detecting objects on frame
    [bbox, score, label] = yolov3Detect(net,thisFrame, ...
       networkOutputs, anchorBoxes,anchorBoxMasks, ...
       confidenceThreshold, overlapThreshold, classNames);

    newt = toc;
    fps = .9*fps + .1*(1/newt);
    avgfps = [avgfps; fps];

    clear yolov3Detect
    thisFrame = extractdata(thisFrame);
    thisFrame = gather(thisFrame);

    % if object found, it inserts the bbox and label
    if (~isempty(score{1}))
        thisFrame = insertObjectAnnotation(thisFrame, ...
            'rectangle',bbox{1},label{1});
    end

    % Saving frame on outputVideo
    writeVideo(OutputVideo,thisFrame);
```

```
end

Framerate = CalculoMediaFps(avgfps);
Texto = ['Fps = ',num2str(Framerate)];
disp(Texto);

close(OutputVideo);
```

8.5.3. CalculaMediaFps

```
function Framerate = CalculoMediaFps(avgfps)
    suma = 0;
    indices = size(avgfps);
    for i=1:indices(1)
        suma = suma + avgfps(i,1);
    end
    Framerate = suma/indices(1);
end
```