



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Diseño y desarrollo de una aplicación web para el
tratamiento, análisis mediante técnicas de machine
learning y visualización de ovocitos.

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de
Telecomunicación

AUTOR/A: Villena Jiménez, Carlos

Tutor/a: Naranjo Ornedo, Valeriana

Cotutor/a: Pulgarín Ospina, Cristian Camilo

Cotutor/a externo: Paya Bosch, Elena

CURSO ACADÉMICO: 2023/2024

Resumen

La evolución de la pirámide poblacional y su proyección futura prevé un problema social y económico a nivel nacional y global. La baja natalidad está relacionada, en gran medida, con el incremento de la infertilidad en la población debido a diferentes factores sociales, ambientales y económicos que afectan a unos 80 millones de personas en todo el mundo. Aunque las tasas de infertilidad oscilan entre el 5% y el 30%, se calcula que una de cada diez parejas sufre problemas de infertilidad. Para hacer frente a este problema, se desarrollaron las Técnicas de Reproducción Asistida (TRA). Un procedimiento típico comienza con la recuperación de múltiples ovocitos de los ovarios de la paciente y la posterior fecundación in vitro (FIV). En este contexto, la congelación de ovocitos es cada vez más frecuente, lo que conduce a la necesidad de optimizar su evaluación. Muchos de estos ovocitos no son capaces de alcanzar un embrión preimplantacional con probabilidades de dar lugar a un recién nacido vivo. Por lo tanto, el desarrollo de una aplicación que permita la representación y visualización de la cohorte de ovocitos y que integre métodos de análisis para la predicción de la calidad ovocitaria se convierte en una herramienta necesaria para la toma de decisiones en una clínica de reproducción asistida. El objetivo de este TFG es el desarrollo de una aplicación web para la visualización y análisis de los parámetros relacionados con la selección de ovocitos para técnicas de reproducción asistida. La aplicación incluirá visualización de datos y análisis mediante dashboards interactivos, integración de APIs para el manejo de datos externos y modelos básicos de deep learning como un perceptrón multicapa o una red neuronal convolucional para la predicción de la calidad de los ovocitos. Además, se utilizarán tecnologías como Docker para la estrategia de despliegue de la herramienta, asegurando un entorno consistente y escalable. La aplicación desarrollada en este TFG se pretende implantar en la clínica y usarla como parte del flujo diario de trabajo.

Resum

L'evolució de la piràmide de població i la seva projecció futura preveu un problema social i econòmic a nivell nacional i global. La baixa natalitat està relacionada, en gran mesura, amb l'increment de la infertilitat a la població a causa de diferents factors socials, ambientals i econòmics que afecten uns 80 milions de persones a tot el món. Tot i que les taxes d'infertilitat oscil·len entre el 5 i el 30 %, es calcula que una de cada deu parelles pateix problemes d'infertilitat. Per fer front a aquest problema, es van desenvolupar les Tècniques de Reproducció Assistida (TRA). Un procediment típic comença amb la recuperació de múltiples ovòcits dels ovaris de la pacient i la posterior fecundació in vitro (FIV). En aquest context, la congelació d'ovòcits és cada cop més freqüent, cosa que condueix a la necessitat d'optimitzar-ne l'avaluació. Molts d'aquests ovòcits no són capaços d'assolir un embrió preimplantacional amb probabilitats de donar lloc a un nou-nat viu. Per tant, el desenvolupament d'una aplicació que permeti la representació i visualització de la cohort d'ovòcits i que integri mètodes d'anàlisi per a la predicció de la qualitat ovocitària es converteix en una eina necessària per prendre decisions en una clínica de reproducció assistida. L'objectiu d'aquest TFG és el desenvolupament d'una aplicació web per a la visualització i l'anàlisi dels paràmetres relacionats amb la selecció d'ovòcits per a tècniques de reproducció assistida. L'aplicació inclourà la visualització de dades i anàlisi mitjançant dashboards interactius, integració d'APIs per al maneig de dades externes i models bàsics de deep learning com ara un perceptró multicapa o una xarxa neuronal convolucional per a la predicció de la qualitat dels oòcits. A més, es faran servir tecnologies com Docker per a l'estratègia de desplegament de l'eina, assegurant un entorn consistent i escalable. L'aplicació desenvolupada en aquest TFG es pretén implantar a la clínica i fer-la servir com a part del flux diari de treball.

Abstract

The evolution of the population pyramid and its future projection foresees a social and economic problem at a national and global level. Low birth rates are largely related to the increase in infertility in the population due to different social, environmental and economic factors that affect some 80 million people around the world. Although infertility rates range between 5 % and 30 %, it is estimated that one in ten couples suffer from infertility problems. To address this problem, Assisted Reproduction Techniques (ART) were developed. A typical procedure begins with the retrieval of multiple oocytes from the patient's ovaries and subsequent in vitro fertilization (IVF). In this context, oocyte freezing is becoming more frequent, which leads to the need to optimize its evaluation. Many of these oocytes are not capable of reaching a preimplantation embryo likely to give rise to a live newborn. Therefore, the development of an application that allows the representation and visualization of the oocyte cohort and that integrates analysis methods for the prediction of oocyte quality becomes a necessary tool for decision making in an assisted reproduction clinic. . The objective of this TFG is the development of a web application for the visualization and analysis of the parameters related to the selection of oocytes for assisted reproduction techniques. The application will include data visualization and analysis through interactive dashboards, integration of APIs for managing external data and basic deep learning models such as a multilayer perceptron or a convolutional neural network for predicting oocyte quality. In addition, technologies such as Docker will be used for the tool's deployment strategy, ensuring a consistent and scalable environment. The application developed in this TFG is intended to be implemented in the clinic and used as part of the daily work flow.

RESUMEN EJECUTIVO

La memoria del TFG del GTIST debe desarrollar en el texto los siguientes conceptos, debidamente justificados y discutidos, centrados en el ámbito de la IT

CONCEPT (ABET)	CONCEPTO (traducción)	Cumple? (S/N)	Dónde? (páginas)
1. IDENTIFY:	1. IDENTIFICAR:		
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	S	RE1.1:inicio1-
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	S	RE1.2:inicio15- 5, 16-16, 17-17
1.3. Setting of goals	1.3. Establecimiento de objetivos	S	RE1.3:inicio2- 2
2. FORMULATE:	2. FORMULAR:		
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	S	RE2.1:inicio47- 49, 70-72.
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	S	RE2.2:inicio65- 72
3. SOLVE:	3. RESOLVER:		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	S	RE3.1:inicio87- 92
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	S	RE3.2:inicio93- 93

Índice general

I Memoria

1. Introducción	1
1.1. Introducción	1
1.2. Motivación	1
1.3. Objetivos	2
1.4. Proyecto	2
2. Marco teórico	5
2.1. Embriología y ovocito	5
2.1.1. Dismorfismos ovocitarios	10
2.1.1.1. Alteraciones morfológicas citoplasmáticas	11
2.1.1.2. Alteraciones morfológicas extracitoplasmáticas	13
2.1.1.3. Complejo cúmulo-corona radiata-ovocito	16
2.1.2. Protocolo	16
2.2. Deep learning	16
2.2.1. Multi layer perceptron	18
2.2.2. Red neuronal convolucional	19
2.2.2.1. Capa convolucional	20
2.2.2.2. Capa de Pooling	21
2.2.2.3. Capa completamente conectada	22
2.2.2.4. Función de activación	22
2.2.3. Parámetros de las redes neuronales	25
2.3. Servicios web	28
2.3.1. Paradigma de microservicios	29
2.3.2. Docker	30
2.3.3. Lenguajes de programación	32
3. Material	35
3.1. Base de datos	35
3.1.1. Limpieza y manejo de datos	36
3.1.2. Análisis y selección de datos	37
4. Metodología	47
4.1. Análisis de imagen	47
4.2. Deep learning	50
4.2.1. Autocrop	50

4.2.2. Multilayer perceptron	53
4.2.3. Convolutional neuronal network	59
4.3. Desarrollo web	72
4.3.1. Landing page	72
4.3.2. Página web de resultados	79
4.3.3. Inferencia modelo autocrop	80
4.3.4. Inferencia extracción de dimensiones	84
4.3.5. Inferencia modelo de predicción de calidad ovocitaria	85
5. Resultados	87
5.1. Análisis de imagen y modelo de deep learning	87
5.2. Aplicación web	91
6. Conclusiones y propuesta de futuro	93
Bibliografía	95

Índice de figuras

2.1. Inseminación artificial intrauterina.	6
2.2. Desarrollo embrionario desde la fecundación hasta el estadio de blastocisto.	6
2.3. Esquema del tracto reproductor femenino humano y la posición relativa de los embriones preimplantacionales a medida que se desarrollan y se diferencian.	7
2.4. Proceso completo de realización FIV Genetic (FIV + PGT-A).	8
2.5. Morfología característica de un ovocito tras la fase de estimulación ovárica.	9
2.6. Diferenciación entre óvulos perfectos y una serie de óvulos con dismorfismos.	10
2.7. Granulosidad en el centro del ovocito (flecha).	11
2.8. Presencia de AREL (R) y pequeñas vacuolas (V) en el ovocito.	12
2.9. (A) y (B) marcan la presencia de vacuolas en el ovocito (flechas).	12
2.10. Presencia de inclusiones. (A) Ovocito con presencia de cuerpo picnótico (CP) y (B) Ovocito con presencia de cuerpo necrótico (CN) y cuerpos refringentes (I).	13
2.11. (A) y (B) muestran Debris en el espacio perivitelino (flechas).	13
2.12. Anomalías en la zona pelúcida, como elongaciones laterales, color y grosor.	14
2.13. Espacio perivitelino aumentado.	14
2.14. Diferentes tamaños de corpúsculo polar. (A) grande ; (B) normal y (C) pequeño.	15
2.15. Corpúsculo polar doble y Corpúsculo polar fragmentado.	15
2.16. (a) COC del ovocito; (b) Ovocito incluyendo zona pelúcida y espacio perivitelino; y (c) Representación gráfica de las capas de un ovocito.	16
2.17. Jerarquía de la inteligencia artificial (IA).	17
2.18. Arquitectura básica de una red neuronal MLP.	18
2.19. Arquitectura básica de una red neuronal convolucional (CNN).	19
2.20. Arquitectura de una red neuronal Convolucional diseñada en este proyecto.	20
2.21. Representación de la función de la capa convolucional.	21
2.22. Representación de la capa de pooling con la función de MaxPool.	22
2.23. Representación gráfica de la función Sigmoide.	23
2.24. Representación gráfica de la función reLU.	23
2.25. Representación gráfica de la función tangente hiperbólica.	24
2.26. Representación gráfica de la función softmax.	24
2.27. Desglose completo de la función de todas las capas que conforman una CNN.	25
2.28. Gráfico de optimizadores SGD comunes para la creación de modelos de DL.	27
2.29. Diferenciación en diagrama de bloques de una máquina virtual (VM) y un contenedor docker.	30
2.30. Proceso de lanzamiento de aplicaciones con Docker.	32

3.1.	Conjunto de charts que relacionan el rango de puntuaciones activas de cada subcaracterística dentro de cada característica de la BBDD.	41
3.2.	Conjunto de histogramas que relacionan el número de ocurrencias o frecuencia que cada subcaracterística de las características ha tenido en la BBDD.	42
3.3.	Conjunto de métricas obtenidas de la BBDD para cada subcaracterística dentro de cada característica.	43
4.1.	Detección de bordes con algoritmo de Canny.	47
4.2.	Detección de bordes con algoritmo de Sobel.	48
4.3.	Detección de bordes con algoritmo de Laplace.	48
4.4.	Diagrama de bloques del proceso de detección de bordes con los algoritmos Canny, Sobel y Laplacian.	49
4.5.	Diagrama de flujo del código empleado para la función de la obtención de las dimensiones del borde del corpúsculo polar detectado sobre los ovocitos	50
4.6.	Proceso de cropeo de una imagen de ovocito pasada por el modelo de autocrop.	51
4.7.	Diagrama de flujo de la función de la aplicación del modelo de autocrop de las imágenes de los ovocitos.	53
4.8.	Diagrama de flujo del código Python para la carga y preprocesado de imágenes para distinguir entre características de imágenes y etiquetas para su posterior entrenamiento de modelo MLP.	54
4.9.	Diagrama de flujo del código Python para la extracción de características de las imágenes con la librería scikit-learn para modelo MLP.	55
4.10.	Código de python para la obtención de los datos y etiquetas del conjunto de entrenamiento y validación para el modelo de MLP.	55
4.11.	Diagrama de flujo del código de python de la función de entrenamiento del modelo con búsqueda de hiperparámetros para la MLP.	56
4.12.	Diagrama de flujo del código de python para la evaluación del modelo entrenado con mejores hiperparámetros con una MLP. Se utiliza técnicas de clasificación de modelo como la matriz de confusión, reporte de clasificación y la media armónica de la precisión y recall (f1-score).	57
4.13.	Diagrama de flujo del código de python de la implementación de la función para la validación cruzada sobre el modelo de MLP para la clasificación de ovocitos.	58
4.14.	Diagrama de bloques del flujo de generación del modelo de MLP desde carga de datos hasta guardado del modelo.	59
4.15.	Arquitectura general de redes neuronales convolucionales - CNN.	60
4.16.	Diagrama de flujo de la función de python para la carga y preprocesado de los datos para el entrenamiento de los modelos de predicción de calidad de ovocitos.	62
4.17.	Diagrama de flujo de la función de python para la creación del dataloader para el entrenamiento y prueba de los modelos CNN.	63
4.18.	Diagrama de flujo del Dataset personalizado para predicción de calidad ovocitaria.	64
4.19.	Diagrama de bloques del Modelo 1	67
4.20.	Diagrama de bloques del Modelo 2	68

4.21. Diagrama de bloques del Modelo 3	70
4.22. Diagrama de bloques del Modelo 4	72
4.23. Código HTML para generar el navbar de la aplicación web	73
4.24. Código HTML para generar el buscador de archivos y su posterior carga de la aplicación web.	74
4.25. Código python con Flask para generación de carga y guarda de las imágenes seleccionadas. Además se implementa el envío de imágenes al servidor para su posterior cropeo (recorte de imagen).	75
4.26. Enter Caption	76
4.27. Función "send_image" para realizar un APIrest con el servidor sobre el puerto 5001 en el que está situada la aplicación del modelo de autocrop.	77
4.28. Send Image	78
4.29. Código HTML para el desarrollo del botón que enlace con la siguiente página web de forma fluida.	78
4.30. Planteamiento de la interfaz la página web inicial.	79
4.31. Diagrama de flujo del funcionamiento de la página web inicial.	79
4.32. Planteamiento de la interfaz de la página web de resultados.	80
4.33. Diagrama de flujo del funcionamiento de la página web de resultados.	80
4.34. Documento Dockerfile de la aplicación ".app.py".	81
4.35. Documento Dockerfile de la aplicación "autocropapp.py".	82
4.36. Documento Docker-compose para el lanzamiento de los contenedores para la ejecución de la aplicación en conjunto del autocrop.	83
4.37. Proceso de autocrop de las imágenes de los ovocitos desde que se cargan las imágenes en la aplicación, hasta que se devuelven por el modelo.	84
4.38. Proceso de detección de bordes del EPV y extracción de dimensiones de los ovocitos.	85
4.39. Proceso de predicción de calidad de los ovocitos a evaluar.	86
5.1. Imagen de la detección de bordes mediante la aplicación de los algoritmos de Canny, Sobel, Laplace. El último método es una variante del algoritmo de Canny con elipse no explicada.	88
5.2. Primera tupla de resultados de entrenamiento de los modelos CNN.	89
5.3. Segunda tupla de resultados de entrenamiento de los modelos CNN.	89
5.4. Tercera tupla de resultados de entrenamiento de los modelos CNN.	89
5.5. Cuarta tupla de resultados de entrenamiento de los modelos CNN.	90
5.6. Reportes de clasificación de los cuatro modelos entrenados.	90
5.7. Resultado de la primera vista de la aplicación.	91
5.8. Resultado de la segunda vista de la aplicación.	92

Índice de tablas

2.1. Tabla comparativa de las propiedades y usos de las máquinas virtuales con respecto a docker.	31
3.1. Conjunto de características de los dismorfismos ovocitarios.	36
3.2. Formato de cambio para el nombre de las imágenes del directorio.	36
3.3. Conjunto de subcaracterísticas de los Ovocitos con mejores Calidades. . . .	44
3.4. Subcaracterísticas que reflejan una buena calidad.	45

Parte I

Memoria

Capítulo 1

Introducción

1.1. Introducción

Las redes neuronales han demostrado ser una herramienta poderosa en el campo del aprendizaje automático, particularmente en tareas de clasificación de imágenes mediante técnicas como perceptrones multicapa (MLP) o redes neuronales convolucionales (CNN). En el contexto de la investigación en fecundación in vitro (FIV), la evaluación precisa de la calidad de los ovocitos es crucial para predecir el éxito de la implantación y el desarrollo embrionario. La investigación en el Instituto Valenciano de Infertilidad (IVI) [1] de Valencia ha estado a la vanguardia en la automatización mediante el uso de técnicas de aprendizaje profundo fruto de la colaboración con el *Computer Vision and Behaviour Analysis Laboratory* (CVBLab) [2], para automatizar y mejorar la evaluación de la calidad de los ovocitos.

1.2. Motivación

La FIV [3] es una de las técnicas de reproducción asistida más usada y cuya eficacia depende en gran medida a la calidad de los ovocitos, por este motivo, la selección de ovocitos de alta calidad es fundamental para el éxito de la FIV. Actualmente, hay una tendencia creciente hacia el uso de técnicas de inteligencia artificial (IA) como el *deep learning* (DL), ya que esta tecnología puede mejorar significativamente la selección de ovocitos de mayor calidad y, por lo tanto, aumentar las tasas de implantación y embarazo.

Hasta ahora la selección de los ovocitos ha sido manual y sujeta a variabilidad inter e intraobservador. El proceso es laborioso y requiere mucho tiempo debido al método de evaluación de ovocitos, que consiste en la extracción de imágenes de ovocitos con un microscopio antes de criogenizar. Finalmente, se realiza una visualización por parte de los profesionales en embriología del laboratorio para la evaluación de las características morfológicas que puedan tener.

La motivación de este proyecto radica en automatizar la predicción del porcentaje de los ovocitos viables de la cohorte de una mujer, y ofrecerle la posibilidad de realizarle una segunda estimulación ovárica y extracción para cuando los descongelados sean de calidad y

viables para dar lugar a un recién nacido vivo.

1.3. Objetivos

El objetivo de este trabajo es la creación de una aplicación web para la visualización y valoración de la calidad de ovocitos. El enfoque seguido abarca la carga, visualización y edición de factores de calidad, así como la clasificación de la calidad mediante técnicas basadas en algoritmos de deep learning. Para lograr esto se proponen una serie de objetivos específicos:

1. Desarrollar una plataforma web intuitiva y accesible. Donde los embriólogos se podrán conectar a través de conexión a internet para cargar las cohortes de ovocitos para su consecuente análisis. Esta aplicación web contendrá dos vistas con rutas diferentes en las que se separe la parte de carga de imágenes de la de análisis.
2. Implementar un sistema de procesamiento de imágenes. Integrar el modelo de recorte automático (autocrop) aportado por CVBLab dentro de la aplicación para que trabajen en sintonía. También, se requiere de la realización de análisis métrico sobre los ovocitos, con el objetivo de aportar nuevas características o variables que ayuden a mejorar los modelos de predicción.
3. Integrar modelos de deep learning para la clasificación de calidad. Estudiar e implementar técnicas de aprendizaje profundo favorables sobre un set de imágenes y características otorgado por parte de los profesionales del IVI. Posteriormente, estos modelos son implementados como parte de la aplicación web.
4. Gestionar y almacenar datos. Entender, ordenar y gestionar la BBDD dada por el IVI con todas las características morfológicas y calificaciones sobre cada ovocito.
5. Probar y validar el sistema. Testear continuamente la aplicación de principio a fin para ver la progresión en cuanto a funcionalidad hasta su implementación.

1.4. Proyecto

Este trabajo de fin de grado (TFG) se desarrolla en el marco del proyecto OVOVITRIF [4]. Proyecto que nace de la simbiosis entre el grupo de investigación CVBLab, dirigido por la Profesora Valery Naranjo, catedrática del departamento de Comunicaciones de la Universidad Politécnica de Valencia, junto al Instituto Valenciano de Infertilidad (IVI), una de las clínicas de reproducción asistida con más volumen de pacientes del mundo.

El proyecto pretende desarrollar modelos de IA para la identificación de biomarcadores en medio de cultivo de ovocitos, en el líquido folicular y en la morfología estática del ovocito capturada a partir de una imagen que permitan determinar la supervivencia de los ovocitos a la vitrificación y desvitrificación. Actualmente en las clínicas de reproducción asistida no se da información de la calidad de los ovocitos individualmente ni objetiva, sino que está sujeta al criterio profesional de los embriólogos.

La valoración de la calidad ovocitaria al inicio del ciclo aporta información personalizada para cada paciente, predecir las probabilidades de supervivencia a la vitrificación, el ratio de fecundaciones y las probabilidades de llegar al estado de blastocisto. Con los biomarcadores identificados se desarrolla un nuevo algoritmo basado en tecnología de aprendizaje profundo para la evaluación automatizada de los ovocitos previo a su vitrificación, que permita el asesoramiento personalizado de las pacientes.

Capítulo 2

Marco teórico

En este capítulo se desarrolla la base teórica necesaria para la realización del proyecto. En primera instancia los conceptos básicos en embriología necesarios para poder comprender la anatomía y morfología de los ovocitos para su posterior análisis con modelos de inteligencia artificial (IA). En segundo lugar, se comenta de forma clara y exhaustiva la ciencia detrás de la generación de aplicaciones web, conexión entre servidores y plataformas como *Docker* para desplegar aplicaciones rápidamente. Por último, se explica el marco conceptual por el cual se desarrollan los modelos de aprendizaje automático para clasificación de imágenes.

2.1. Embriología y ovocito

La embriología estudia el crecimiento y diferenciación durante las primeras etapas del desarrollo embrionario y nervioso hasta el nacimiento del ser vivo [5].

Sobre el presente proyecto se requiere de la evaluación de ovocitos para la realización de la FIV para su posterior transferencia embrionaria y embarazo. La FIV es una técnica de reproducción asistida (RA) que ayuda a mujeres a lograr un embarazo cuando enfrentan dificultades para concebir de forma natural. Previo a la FIV se hace una revisión exhaustiva de la cantidad y calidad de ovocitos (células sexuales femeninas o gametos) de los ovarios de la mujer. Este proceso conlleva la estimulación ovárica para la generación de ovocitos maduros para su posterior extracción mediante una intervención quirúrgica. Una vez extraídos, estos se evalúan con un microscopio para ver su estado madurativo. Un ovocito es una célula germinal femenina que está en proceso de convertirse en un óvulo maduro. Su función es la formación de un cigoto al fusionar su núcleo con el del gameto masculino (espermatozoide). Una vez que el ovocito es fecundado por el espermatozoide, es introducido en una incubadora simulando el útero femenino mediante la FIV, comienza así el proceso de desarrollo embrionario como se observa en la Figura 2.2.

Existen diferentes técnicas utilizadas para la fecundación del ovocito por parte del gameto masculino, concibiéndose de forma natural (en el útero de la gestante), o no natural, llevándose a cabo la fecundación en el exterior de forma controlada con técnicas como la inyección intracitoplasmática de espermatozoides (ICSI) [6]. La inseminación artificial es una técnica sencilla en la que se recoge el esperma y este se introduce de manera no

natural en el útero de la mujer, como se ve en la Figura 2.1, aunque también se puede hacer intravaginal o intratubárica.

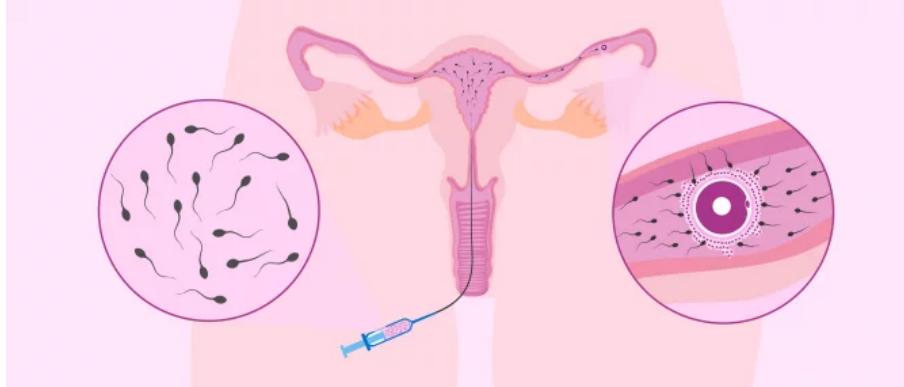


Figura 2.1: Inseminación artificial intrauterina [6].

En cambio la ICSI es una técnica de mayor complejidad utilizada en la FIV, donde la obtención de los óvulos se realiza de igual modo, es decir, mediante una punción folicular, pero la fecundación, no tiene lugar de forma natural, sino que se escoge el espermatozoide a depositar y este se inyecta en el interior del óvulo mediante una microaguja [6]. Posteriormente se introduce en el incubador donde dará comienzo el desarrollo embrionario que tiene comienzo con la fusión del material genético (Figura 2.2).

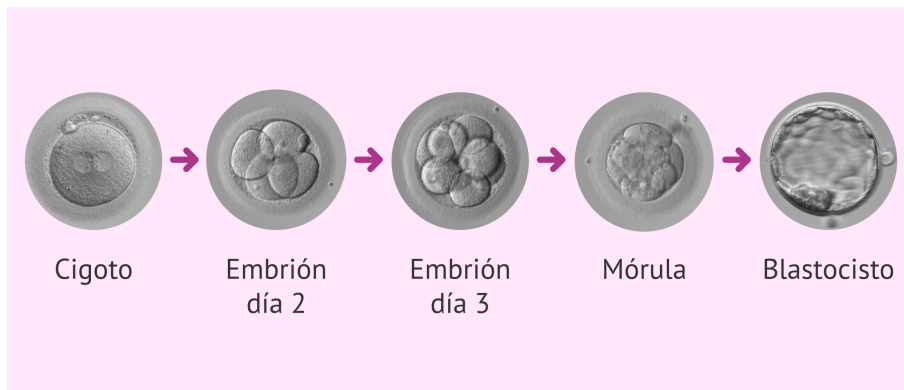


Figura 2.2: Desarrollo embrionario desde la fecundación hasta el estadio de blastocisto [7].

Realizada la inseminación o la ICSI, se formará lo que se conoce como cigoto (primera fase de la Figura 2.2), que es el óvulo ya fecundado. A medida que pasan los días el cigoto se divide en múltiples células, llamadas blastómeros, hasta que se cohesionan convirtiéndose en mórula (fases dos, tres y cuatro de la Figura 2.2), que es el nombre que se le da al cigoto cuando se divide múltiples veces (16 células), formando una estructura cada vez más compacta hasta que se convierte en blastocisto (última fase de la Figura 2.2), siendo una estructura más compleja con una cavidad interna y una masa celular interna (que se convertirá posteriormente en el feto) [8].

En la Figura 2.3 se representa el desarrollo embriológico que tendría un ovocito desde la fecundación hasta la formación del blastocisto de forma natural y posicionándose relativamente la posición de los embriones preimplantados a medida que se desarrollan.

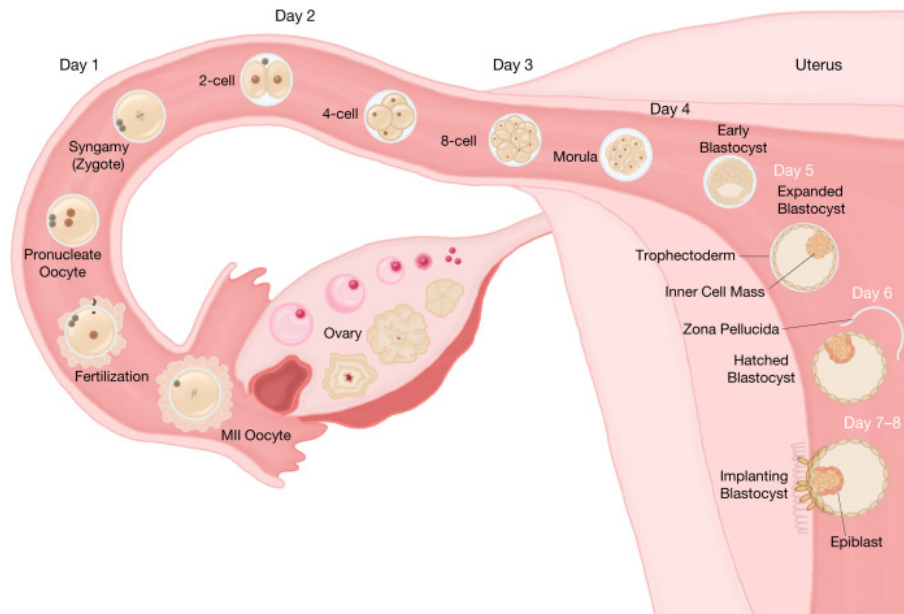


Figura 2.3: Esquema del tracto reproductivo femenino humano y la posición relativa de los embriones preimplantacionales a medida que se desarrollan y se diferencian [8].

En la etapa final del desarrollo embrionario, se clasifican los blastocistos según su calidad, donde se vitrifican (criogenizan) aquellos con mayor calidad y así mantenerlos estables durante un tiempo mientras se realizan análisis y se obtienen los resultados para su transferencia al útero materno. No obstante, no todos los blastocistos se vitrifican, sino que algunos se transfieren en fresco a la paciente, vitrificándose aquellos que puedan servir para ciclos posteriores.

Cuando se han realizado los análisis y la paciente está lista para la transferencia embrionaria, el día programado, los embriólogos proceden a la descongelación de uno de los embriones diagnosticados como "sano". Así, se introduce el mejor embrión en el útero materno por vía vaginal mediante una cánula de transferencia.

Finalmente, tras una espera de alrededor de 11 días, la mujer se someterá a la prueba de embarazo, donde el ginecólogo realiza una prueba de sangre y en 20 días resultando positiva se hace una ecografía de control.

En la Figura 2.4 se observan todos los pasos seguidos desde la asistencia a la clínica de reproducción asistida hasta el alta por embarazo de la paciente. Este proceso dura 86 días como máximo si transcurre de forma seguida, es decir, si la paciente pretende quedarse embarazada en el momento de la asistencia a la cita con el embriólogo, y no para preservar sus óvulos para el futuro.

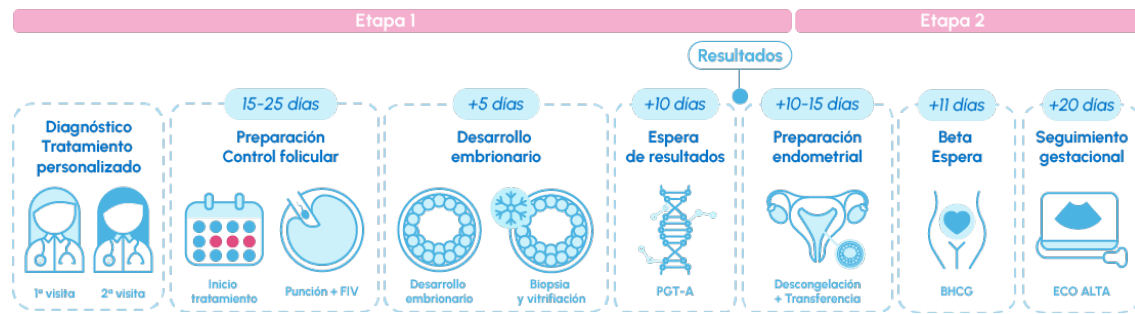


Figura 2.4: Proceso completo de realización FIV Genetic (FIV + PGT-A) [9].

Como objetivo del proyecto, se deben analizar los ovocitos y clasificarlos para predecir cuales llegan a la etapa de blastocisto. Para analizar los ovocitos, se debe tener en cuenta la morfología del mismo y los posibles defectos que puedan tener (dismorfismos ovocitarios).

Un ovocito está formado por cinco elementos principales (véase la Figura 2.5), los cuales en la parte central se encuentran el núcleo rodeado y abordado por el citoplasma, siendo el núcleo la vesícula germinal y el citoplasma un compuesto gelatinoso rodeado de la membrana que lo separa de la zona pelúcida (ZP) [10], llamado espacio perivitelino (EPV), que contiene nutrientes necesarios para el desarrollo embrionario. La ZP es una matriz glicoprotéica extra-celular encargada de múltiples funciones del ovocito y del futuro blastocisto [11]. El corpúsculo polar (CP) es una célula de pequeña, no funcional, localizada en el espacio perivitelino (EPV). Cuando aparece el primer corpúsculo polar, indica de que el óvulo está maduro y, por tanto, es apto para fecundar. En estos casos se conoce como ovocitos en metafase II [12].

Los ovocitos en metafase II (maduros) se consideran morfológicamente normal y, en consecuencia, de buena calidad, aquel que presenta una zona pelúcida proporcionada, bien definida y regular, un espacio perivitelino virtual en el que cabe el corpúsculo polar, un corpúsculo polar único, ligeramente aplanado con forma esférica, una membrana plasmática regular y un citoplasma homogéneo [13] como se puede observar en la Figura 2.5.

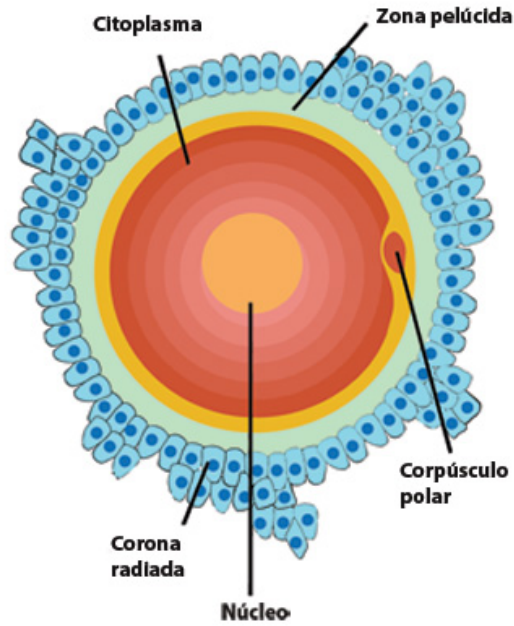


Figura 2.5: Morfología característica de un ovocito tras la fase de estimulación ovárica [13].

Extrapolando esta información sobre los ovocitos del presente proyecto, cuyas imágenes han sido otorgadas por el IVI, un óvulo con morfología perfecta, sería como el que se muestra en la parte superior de la Figura 2.6, en donde se pueden ver perfectamente las características anteriormente mencionadas. Por otra parte, y en la parte inferior de la Figura 2.6 se visualizan una serie de óvulos que presentan diferentes tipos de dismorfismos, por lo que las calidades de ellos ya no serían tan ideales para la fecundación o desarrollo del blastocisto.

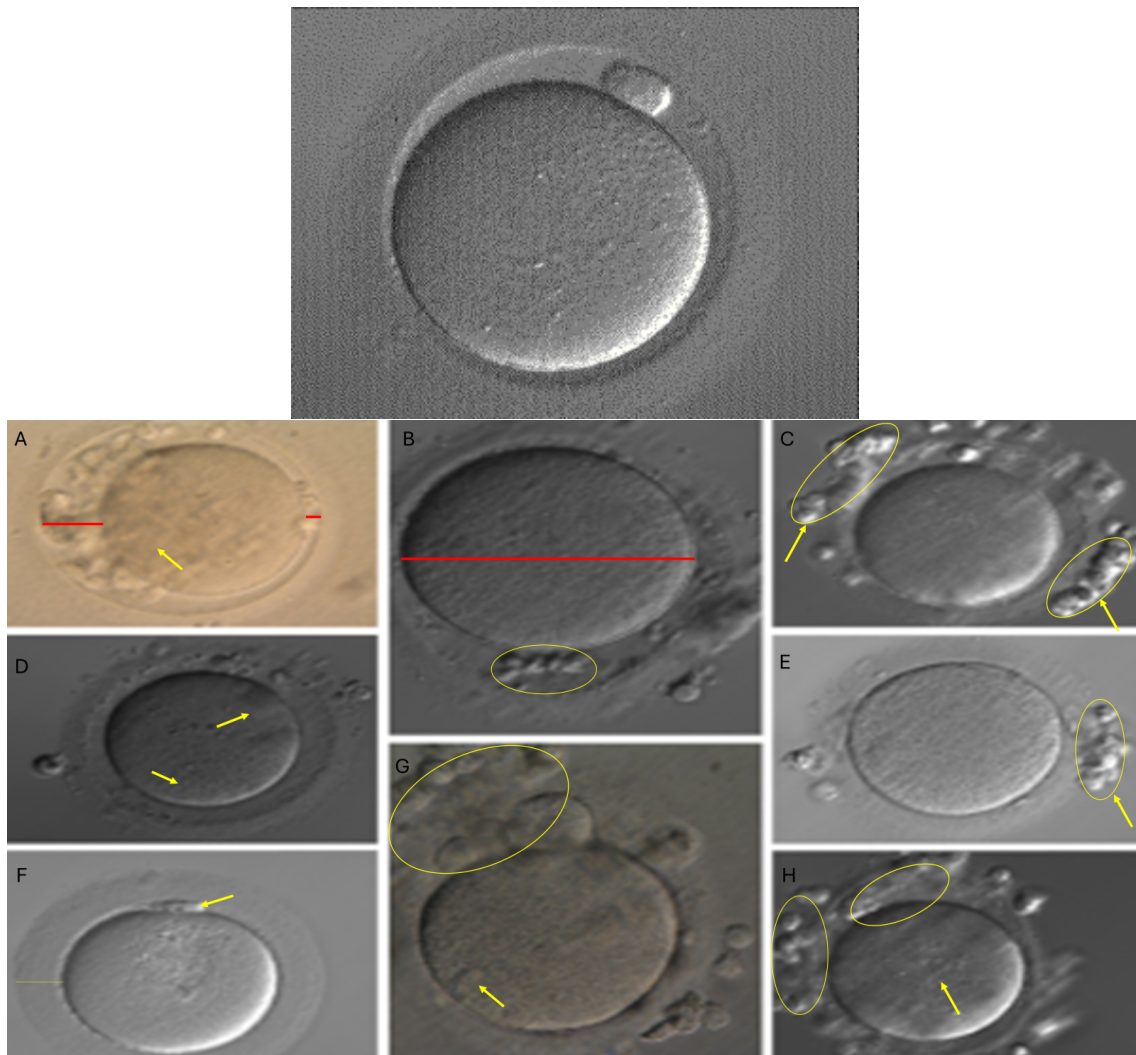


Figura 2.6: Diferenciación entre óvulos perfectos y una serie de óvulos con dismorfismos [14]. (A) muestra en flechas granulosidad lateral y en línea EPV aumentado; (B) muestra ovocito grande y corpúsculo polar fragmentado (flecha); (C) muestra varios corpúsculos polares muy fragmentados; (D) muestra presencia de vacuolas y zona pelúcida oscura; (E) muestra debris en la ZP; (F) muestra un corpúsculo polar muy pequeño en un EPV pequeño y con la línea se muestra un tamaño de ZP grande; (G) muestra mucha debris en la ZP y aparición también de vacuolas (flecha); y en (H) se muestra CP fragmentado, junto a bastante debris, y granulosidad central.

2.1.1. Dismorfismos ovocitarios

Los dismorfismos ovocitarios se describen como características morfológicas anormales presentes en un 60-70% de los ovocitos recogidos en ciclos de estimulación ovárica [15].

Algunas características morfológicas anormales del ovocito, a nivel citoplasmático y extracitoplasmático, pueden influir en los procesos de la reproducción asistida, como son:

la fecundación, la formación y la morfología de los pronúcleos, la división embrionaria, el desarrollo embrionario, la calidad embrionaria, la formación del blastocisto, la implantación, el embarazo bioquímico (embarazo no confirmado por la visualización de un embrión en ultrasonido) y la formación de aneuploidías [16].

2.1.1.1. Alteraciones morfológicas citoplasmáticas

Estas alteraciones ocurren dentro del citoplasma del ovocito, que es el líquido que llena la célula y contiene sus orgánulos.

- Agrupación de orgánulos/granulosidad localizada en el centro del ovocito: se asocia con un bajo potencial de implantación, ver Figura 2.7.



Figura 2.7: Granulosidad en el centro del ovocito (flecha) [16].

- Agregación de retículo endoplasmático liso (AREL): se considera una anomalía severa asociada a un desarrollo embrionario anormal y baja formación de blastocistos, alto porcentaje de gestaciones bioquímicas y complicaciones obstétricas en las gestaciones derivadas de estos embriones, aunque hay autores que no lo consideran un factor determinante para descartar ovocitos. Ver Figura 2.8.



Figura 2.8: Presencia de AREL (R) y pequeñas vacuolas (V) en el ovocito.

- Vacuolas: su presencia escasa y de pequeño tamaño no tiene repercusión conocida, pero las vacuolas grandes se asocian con fracaso de fecundación y menor tasa de llegada a blastocisto. Ver Figura 2.9.

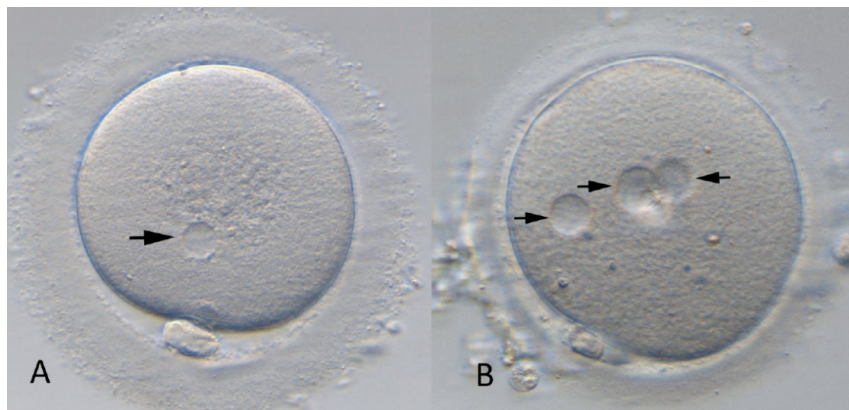


Figura 2.9: (A) y (B) marcan la presencia de vacuolas en el ovocito (flechas).

- Inclusiones citoplasmáticas: se consideran pequeñas áreas de necrosis y pueden ser refringentes (compuestas de lípidos y gránulos) o no refringentes (cuerpos picnóticos). Ver Figura 2.10.

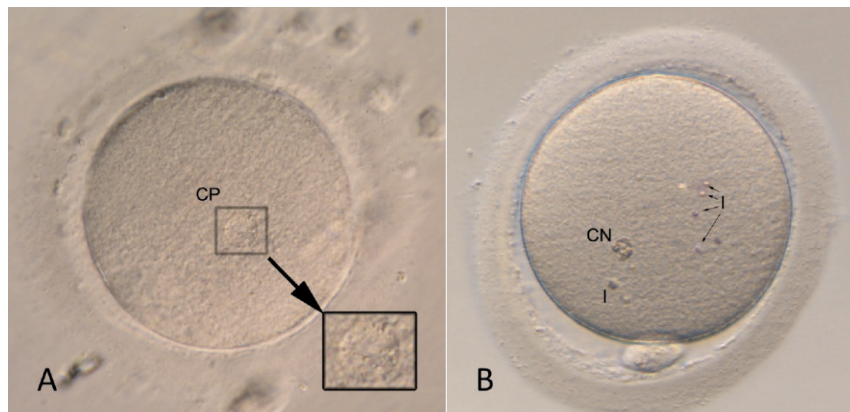


Figura 2.10: Presencia de inclusiones. (A) Ovocito con presencia de cuerpo picnótico (CP) y (B) Ovocito con presencia de cuerpo necrótico (CN) y cuerpos refringentes (I).

2.1.1.2. Alteraciones morfológicas extracitoplasmáticas

Estas alteraciones ocurren fuera del citoplasma, en las estructuras que rodean al ovocito.

- Restos celulares en el espacio perivitelino: se asocia a un exceso de gonadotropinas y a una menor tasa de implantación, aunque no afecta la fecundación, división o calidad embrionaria. Estos restos celulares se denotan como "debrisz" se representan en la Figura 2.11. El término "debris", proviene del francés, con significado de escombro o resto de algo.

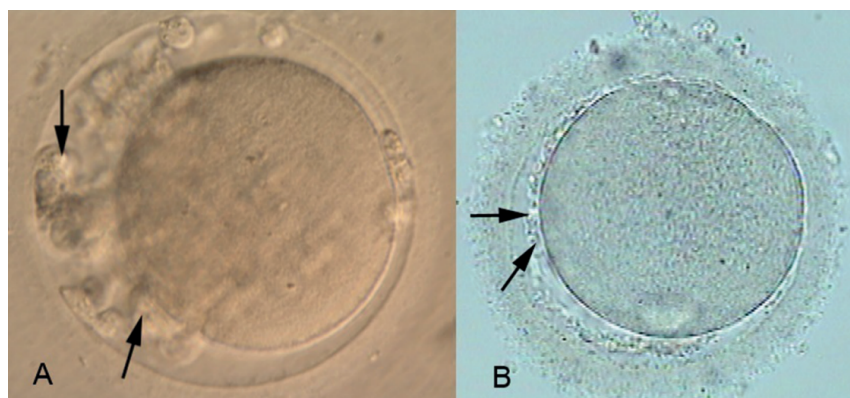


Figura 2.11: (A) y (B) muestran Debris en el espacio perivitelino (flechas).

- Anomalías de la zona pelúcida: la existencia de cambios en el grosor y en la apariencia de la ZP son anomalías normales, pero otras anomalías como contornos no circulares, abultamientos o septos (cortes en la ZP) que pueden detectarse durante la microinyección, indicarían que el ovocito tenga peor desarrollo a blastocisto. Véase la Figura 2.12, donde se exponen tres imágenes en las que se observan tres anomalías

en referencia a la zona pelúcida, como un ovocito con ZP no circular en la Figura 2.12a mostrándose un achatamiento tanto en el corpúsculo polar como en la ZP. En la Figura 2.12b se muestra un ovocito con un ZP que claramente es oscura, y en la Figura 2.12c el tamaño de la ZP casi toca los bordes del recuadro de la imagen, indicándose que es bastante mayor en comparación a las demás, de aproximadamente, unas 20 μm .

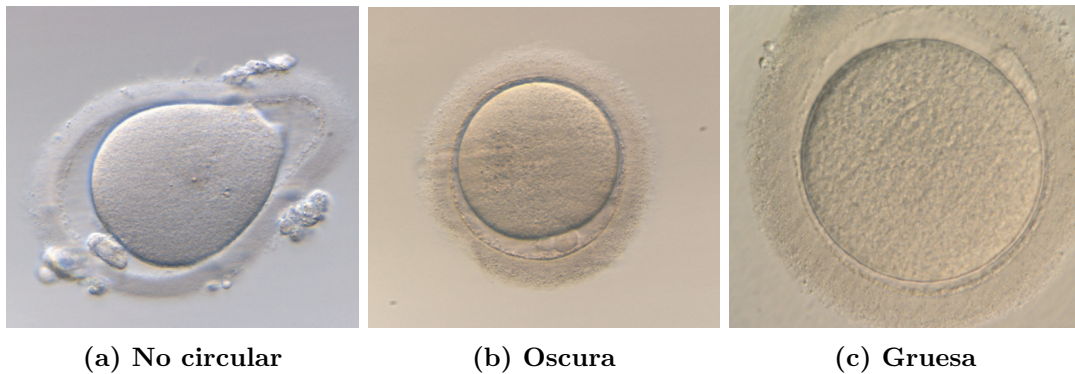


Figura 2.12: Anomalías en la zona pelúcida, como elongaciones laterales, color y grosor.

- Espacio perivitelino aumentado: el aumento del espacio perivitelino denota al ovocito dentro de la zona pelúcida suspendido sobre ella, generándose altitud en la imagen por microscopio como se observa en la Figura 2.13.



Figura 2.13: Espacio perivitelino aumentado.

El ovocito de la figura tiene marcada la zona pelúcida con un desnivel con respecto al espacio perivitelino, que es donde este acaba.

- Alteraciones del primer corpúsculo polar: no hay acuerdo sobre si el tamaño irregular o la fragmentación afectan la calidad embrionaria, el desarrollo a blastocisto, las tasas

de implantación o las aneuploidías. En las Figuras (2.14 y 2.15) se muestran algunos casos en relación a los corpúsculos polares.

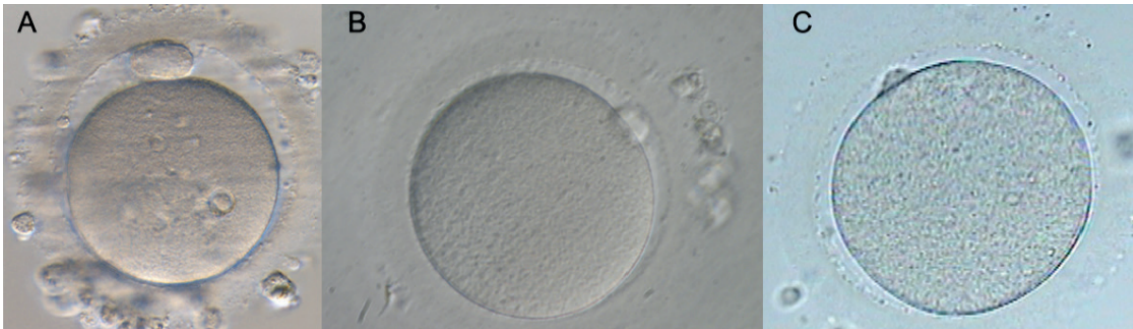


Figura 2.14: Diferentes tamaños de corpúsculo polar. (A) grande ; (B) normal y (C) pequeño.



(a) CP doble



(b) CP fragmentado

Figura 2.15: Corpúsculo polar doble y Corpúsculo polar fragmentado.

2.1.1.3. Complejo cúmulo-corona radiata-ovocito

El cumulus-oocyte complex (COC) hace referencia al estado de madurez del ovocito. Para un ovocito madurado de forma correcta, se necesita de forma coordinada y simultánea la maduración a nivel de núcleo y de citoplasma [17].

El COC se conforma de el "cúmulus", siendo un conjunto de células que rodean al ovocito en el folículo ovárico y después de la ovulación; de la "corona radiata", que son la capa de células que rodea al ovocito tras la ovulación; y el "óvulo", tal como se conoce.

Dependiendo de como sea la relación del COC, el ovocito puede estar en estado de profase I (PI), metafase I (MI), o metafase II (MII), siendo este último el estado de madurez deseado [16].

En la Figura 2.16 se pueden visualizar tres imágenes como representación de ovocitos y en cada una de ellas se señala las partes del mismo. Comenzando por la imagen (a) donde se señala la corona radiata, zona pegada al ovocito con un conjunto denso de células y al cúmulus, como conjunto de células en el folículo ovárico. Las demás imágenes señalan las partes de un ovocito previamente definidas.

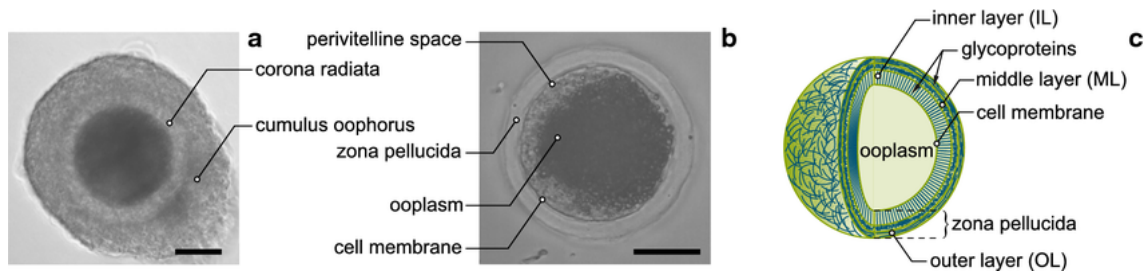


Figura 2.16: (a) COC del ovocito; (b) Ovocito incluyendo zona pelúcida y espacio perivitelino; y (c) Representación gráfica de las capas de un ovocito [18].

2.1.2. Protocolo

Antes se registraba la morfología de los ovocitos para la selección embrionaria, pero esta práctica ha perdido relevancia con la falta de evidencia y la aparición de sistemas de evaluación automática. Actualmente, casi todos los ovocitos se someten a la inyección intracitoplasmática de espermatozoides (ICSI), excepto aquellos con corpúsculo polar gigante en casos de donación o sin diagnóstico genético preimplantacional. A veces, se descartan ovocitos de donante con exceso de debris (restos celulares) en el espacio perivitelino (EPV) que impide distinguir el corpúsculo polar. Aunque no se descarta rutinariamente por morfología, los embriólogos toman nota de dismorfismos destacados.

2.2. Deep learning

Para comprender el significado del término deep learning (DL), se debe ir un paso atrás, comenzando por que es la inteligencia artificial (IA).

La IA, bien conocida estos últimos 10 años atrás, nace para ayudar en procesos cotidianos que los humanos realizan a diario, esos procesos que no tienen por que ser muy complicados, pero siempre hacen perder tiempo que se puede aprovechar en cosas de mayor importancia. Procesos como análisis de patrones en datos de cualquier tipo, resumen de contenidos, análisis de imágenes, predicciones, y muchas más utilidades que llega a tener. Por definición, inteligencia artificial (IA), es la capacidad de una máquina para simular la inteligencia humana en la resolución de problemas [19]. Ramas de la IA son el machine learning (ML) y el deep learning (DL), que mediante la conexión de redes neuronales modelan el comportamiento de un cerebro. Estas redes neuronales son capas interconectadas mediante nodos que extraen características para hacer predicciones gracias a una gran cantidad de datos de entrenamiento. DNNs se diferencian de las MNNs por la cantidad de capas ocultas que se utilizan. Se denota una deep neural network aquella que tiene 3 o más capas ocultas. Además, permiten el aprendizaje no supervisado a parte del supervisado [20].

Aprendizaje no supervisado es aquel que no necesita de supervisión humana, es decir, extrae características de datos no concretos sin que se etiqueten [20]. La mayoría de los modelos usan aprendizaje supervisado, en donde el conjunto de datos está formado por los datos de entrada, como, imágenes y sus correspondientes etiquetas. El no supervisado, gracias al DL está cogiendo mucha fama en el área de la medicina [21].

En medicina, la mayoría de los problemas de salud que se detectan son gracias a imágenes que se realizan sobre los pacientes, para su posterior análisis, ya sea para detectar tumores, grietas, gases o lesiones. Hay todo tipo de técnicas útiles para detectar enfermedades y problemas de salud, pero con el desarrollo de técnicas de clasificación y detección de objetos del aprendizaje profundo, la medicina puede cambiar su rumbo a mejor [22]. Las redes neuronales convolucionales (CNNs), son algoritmos de deep learning, ver Figura 2.17, especializados en el entrenamiento de modelos sobre imágenes. Estos algoritmos se han estado utilizando en ámbitos de la medicina como la radiología, hematología, neurología, oncología, cardiología, oftalmología y en la biología celular [23].

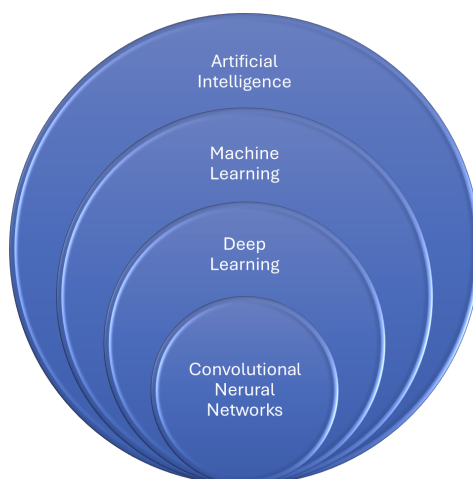


Figura 2.17: Jerarquía de la inteligencia artificial (IA).

Sobre el presente proyecto, se utilizan técnicas de DL para la biología celular, ya que un

ovocito es la célula germinal femenina. En medicina reproductiva, la IA se está utilizando para la selección correcta de los ovocitos en este caso [24].

Se plantean dos algoritmos para realizar la clasificación de la calidad de los ovocitos. Multilayer perceptron (MLP) y las ya comentadas CNNs, ambas son técnicas utilizadas para la clasificación de datos de entrada, pero con unas diferencias clave que hacen a las convolucionales una solución más precisa para nuestro objetivo.

2.2.1. Multi layer perceptron

Un Perceptrón Multicapa (MLP) es un tipo de red neuronal artificial que consta de múltiples capas de neuronas interconectadas. Cada neurona en una capa está conectada a todas las neuronas de la capa siguiente. Las conexiones entre neuronas tienen pesos asociados, que se ajustan durante el proceso de entrenamiento para aprender a realizar una tarea específica [25].

El perceptrón inspirado en el funcionamiento de las redes neuronales biológicas, asigna unos pesos determinados a sus entradas, lleva a cabo una suma ponderada de las mismas, y produce una salida a la cual aplica una función de activación. Ver imagen superior de la Figura 2.18.

Sin embargo, nuestros cerebros están formados por trillones (entre 100 y 1000) de conexiones entre neuronas. Así pues, si queremos conseguir modelos más “inteligentes” necesitamos conectar perceptrones en capas consecutivas capaces de representar distribuciones de datos no lineales. Esta arquitectura se conoce con el nombre de Perceptrón Multicapa. Ver imagen inferior de la Figura 2.18.

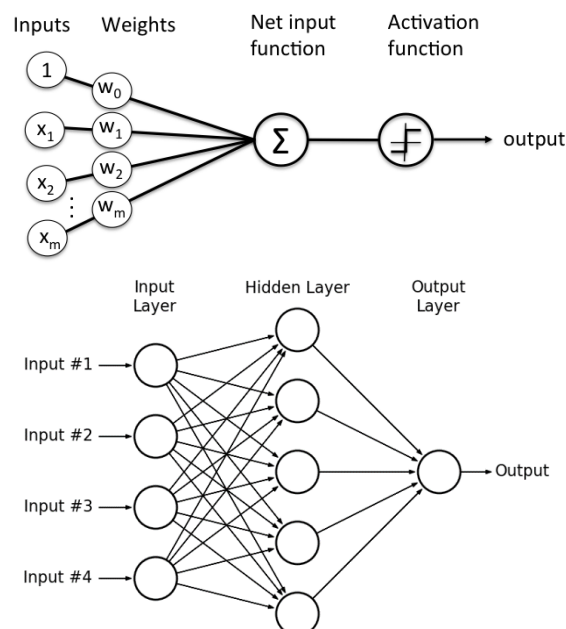


Figura 2.18: Arquitectura básica de una red neuronal MLP.

Una MLP es una muy buena opción para empezar con el proyecto de clasificación de

imágenes, pero tiene el defecto de procesar únicamente datos categóricos [26], por lo que las imágenes como entrada no serían válidas, perdiendo así gran parte de la información. Es por esto que también se pretende la generación de modelos de clasificación mediante algoritmos de CNNs [27].

2.2.2. Red neuronal convolucional

Una Red Neuronal Convolucional (CNN) es una arquitectura de red para Deep Learning diseñada específicamente para procesar datos con una estructura de cuadrícula, como las imágenes [28]. Las CNN se han vuelto muy populares en el campo de la visión por computador debido a su capacidad para aprender características relevantes de las imágenes de forma automática [29].

Las CNNs nacen como herramienta útil en el año 1989 para la clasificación de letras manuscritas y han acaparado toda la atención sobre las arquitecturas multilayer perceptron (MLP) debido a que estas no depende de las imágenes en sí, sino de las características asociadas y el etiquetado de clasificación resultante. Las CNNs conectan la capacidad de las MLP junto con la lectura por capa de las imágenes de entrada.

Una CNN con varias capas convolucionales aprende a partir de la detección de pequeños patrones encontrados en las imágenes y pasado a las neuronas de las siguientes como información útil. En la Figura 2.19 se visualiza una arquitectura de CNN muy básica, donde solo aparece una capa convolucional, lo cual no se entiende como una DNN, pero sigue la estructura necesaria que conforma una CNN. Parte de una serie de imágenes con unas características y se va desplazando por unas capas que extraen dichas características. Una vez llega a la capa final, se interconectan todas las capas mediante la FC y el algoritmo clasifica las salidas.

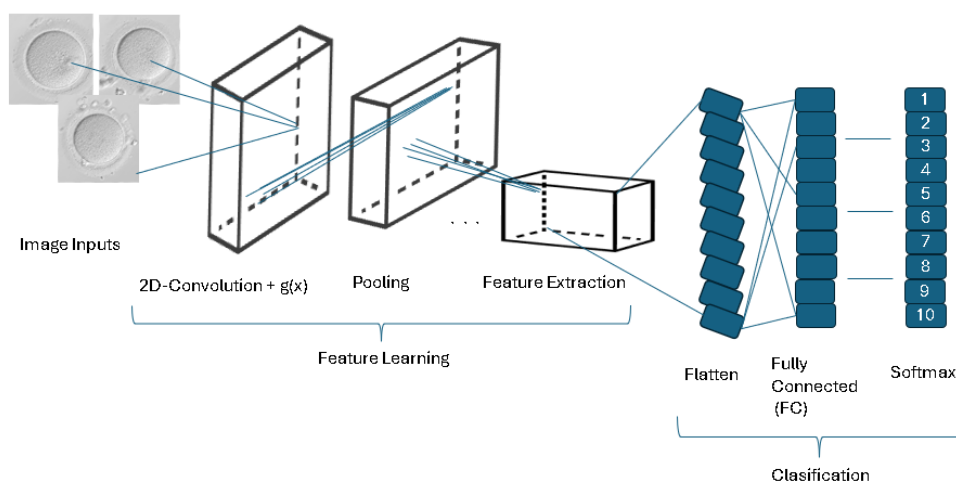


Figura 2.19: Arquitectura básica de una red neuronal convolucional (CNN).

Esta red representa una red neuronal convolucional con una capa convolucional y una capa de pooling. Las imágenes de entrada son en blanco y negro (escala de grises), por lo que los canales de la capa convolucional son de una dimensión (1D), en vez de 3D para casos de imágenes a color en RGB.

Es por eso que en la Figura 2.20 observamos más en detalle en cada recuadro una serie de información. La imagen de entrada es procesada por varias capas convolucionales y de pooling para extraer características y reducir la dimensionalidad. Las características extraídas son aplanadas y combinadas con cualquier característica adicional relevante. Las capas totalmente conectadas procesan estas características para aprender patrones complejos. La capa de salida utiliza la función de activación softmax para proporcionar una clasificación de la calidad del ovocito en una escala del 1 al 10.

Este diseño permite a la red aprender características relevantes de las imágenes de ovocitos y utilizar esas características para clasificar su calidad de forma precisa.

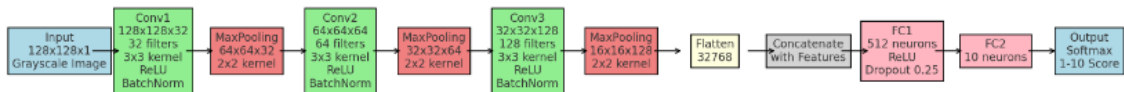


Figura 2.20: Arquitectura de una red neuronal Convolutiva diseñada en este proyecto.

Para comprender en detalle el diagrama de la arquitectura de CNN de la Figura 2.20 se explica la necesidad de cada capa y función usada en el modelo de clasificación.

2.2.2.1. Capa convolucional

Estas capas aplican filtros (también llamados kernels) a la imagen de entrada para extraer características. Los filtros se desplazan por la imagen realizando una operación de convolución en cada posición. El resultado es un mapa de características que resalta las características más importantes de la imagen [30]. Ver Figura 2.21.

Las primeras capas resaltan patrones característicos muy básicos como líneas verticales u horizontales, las siguientes capas con esta información trazan con mejor detalle estas líneas, ya sea tonalidad de gris, incluyendo algunas trazas más complejas como las curvas y así consecutivamente hasta que sepan perfectamente como un ovocito se define visualmente.

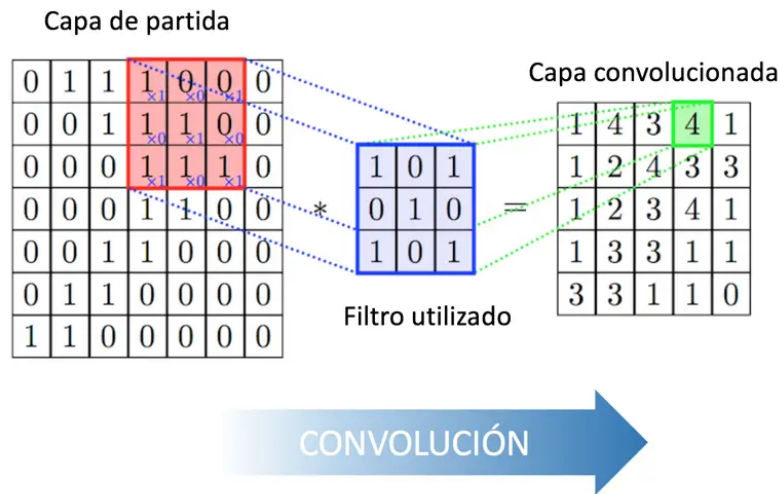


Figura 2.21: Representación de la función de la capa convolucional.

En la Figura 2.21 se aprecia que el tamaño filtro es de 3x3 (3 filas por 3 columnas en la matriz del filtro).

En la capa de convolución mostrada en la Figura 2.21 se observa que de ese trazo se obtiene un 4 como valor de capa. Esto es debido a que se calcula el producto escalar entre el filtro y la porción de la matriz de entrada que cubre, es decir, se realiza un producto escalar de los elementos de la misma posición de ambas matrices. Una vez hecho este paso, cada elemento resultante se suma y se obtiene el valor. En la Ecuación (2.2.2.1) se expresa la lógica matemática para ese elemento resultante.

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \sum_{i=1}^3 \sum_{j=1}^3 a_{ij} \cdot f_{ij} = 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 = 4 \quad (2.2.2.1)$$

2.2.2.2. Capa de Pooling

Estas capas reducen la dimensionalidad de los mapas de características resultantes de las capas convolucionales. El pooling se realiza aplicando una función (como el máximo o el promedio) a una región local del mapa de características. Esto ayuda a reducir el número de parámetros del modelo y a prevenir el sobreajuste [31].

En la Figura 2.22 se puede observar la funcionalidad de esta capa como función de máximo (MaxPool). Si fuese como función de promedio, los resultados serían la suma de cada sección y dividido por el tamaño de dicha sección de la pool.

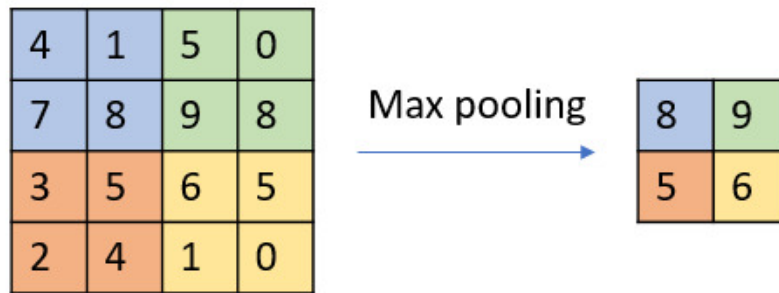


Figura 2.22: Representación de la capa de pooling con la función de MaxPool.

2.2.2.3. Capa completamente conectada

Estas capas se colocan al final de la arquitectura de la CNN y se conectan por completo a todas las neuronas de salida. Después de haber recibido un vector en la entrada, la capa fully connected (FC) aplica sucesivamente una combinación lineal y una función de activación con la finalidad de clasificar la imagen de entrada. Finalmente, en la salida devuelve un vector de tamaño correspondiente al número de clases en el que cada componente representa la probabilidad de que la entrada pertenezca a una clase.

La correcta elección de la función de activación es una parte muy importante de nuestra red, es por eso que comentamos a continuación su relevancia y algunas de las más utilizadas.

2.2.2.4. Función de activación

La elección de la función de activación es una decisión crítica en el diseño de una red neuronal. No existe una función de activación “universal” que sea la mejor para todos los casos, por lo que es importante comprender las propiedades de cada función y seleccionar la más adecuada para el problema específico que se está abordando.

En la documentación de Pytorch sobre las funciones de activación convolucionales no lineales, encontramos alrededor de 40 funciones. Describimos a continuación algunas funciones típicas y algunas funciones que nos podríamos utilizar en nuestro proyecto.

- Sigmoide: común en problemas de clasificación binaria, donde la salida se interpreta como probabilidad. Figura 2.23.

$$\phi(x) = \frac{1}{1 + e^{-x}} \quad (2.1)$$

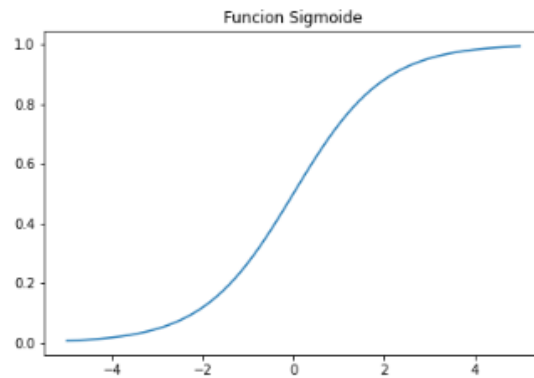


Figura 2.23: Representación gráfica de la función Sigmoide.

- ReLU (Rectified Linear Unit): muy utilizada en capas ocultas de redes neuronales profundas debido a su eficiencia computacional y su capacidad para evitar el problema de desvanecimiento del gradiente. Figura 2.24.

$$\phi(x) = \text{máx}(0, x) \quad (2.2)$$

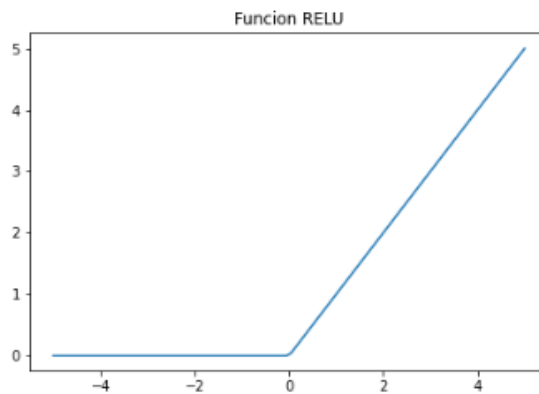


Figura 2.24: Representación gráfica de la función reLU.

- Tanh (Tangente hiperbólica): similar a la sigmoide, pero con un rango de salida de -1 a 1. Figura 2.25.

$$\phi(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \quad (2.3)$$

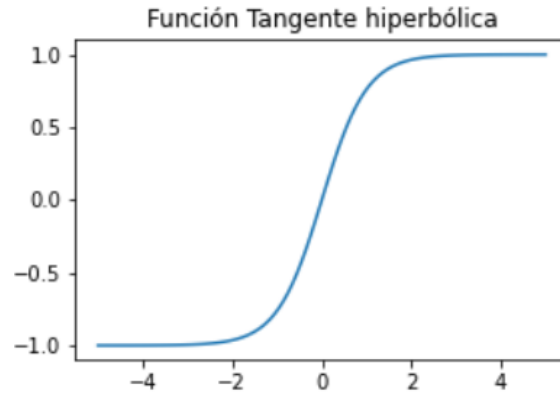


Figura 2.25: Representación gráfica de la función tangente hiperbólica.

- Softmax: utilizada en la capa de salida de problemas de clasificación multiclase para normalizar las salidas en una distribución de probabilidad. Figura 2.26.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad (2.4)$$

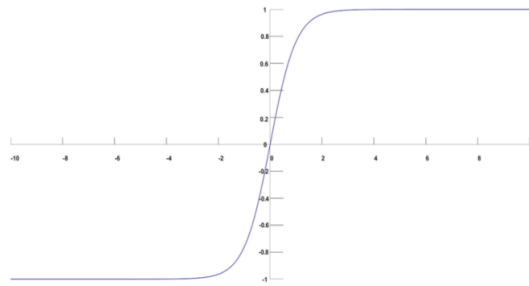


Figura 2.26: Representación gráfica de la función softmax.

- LogSoftmax: realiza la función logaritmo sobre la función softmax(x) de forma separada. Hay una serie de ventajas de usar log softmax sobre softmax, incluidas razones prácticas como rendimiento numérico mejorado y optimización de gradiente. La esencia del uso de log-softmax sobre softmax es el uso de probabilidades logarítmicas sobre probabilidades, lo que tiene buenas interpretaciones teóricas de la información.

$$\text{lsm}(\mathbf{z})_i = \log(\sigma(\mathbf{z})_i) = z_i - \log \left(\sum_{j=1}^K e^{z_j} \right) \quad (2.5)$$

En la Figura 2.27 se observa la secuencia y descripción de las capas de una red neuronal convolucional (CNN), incluyendo el early stopping, que es una herramienta de código que habilita la capacidad de parar el entrenamiento en una época en la que se pueda verificar que no disminuye la validación de la función de pérdidas. Este parámetro se puede poner

con un umbral en específico que el ingeniero decida. Además aparece el término de dropout, que es una técnica de regularización que evita en lo posible el problema del overfitting o sobremuestreo [32], realizando una eliminación de neuronas en esa capa para que nuevas puedan aprender desde cero y así evitar problemas de sobreajuste.

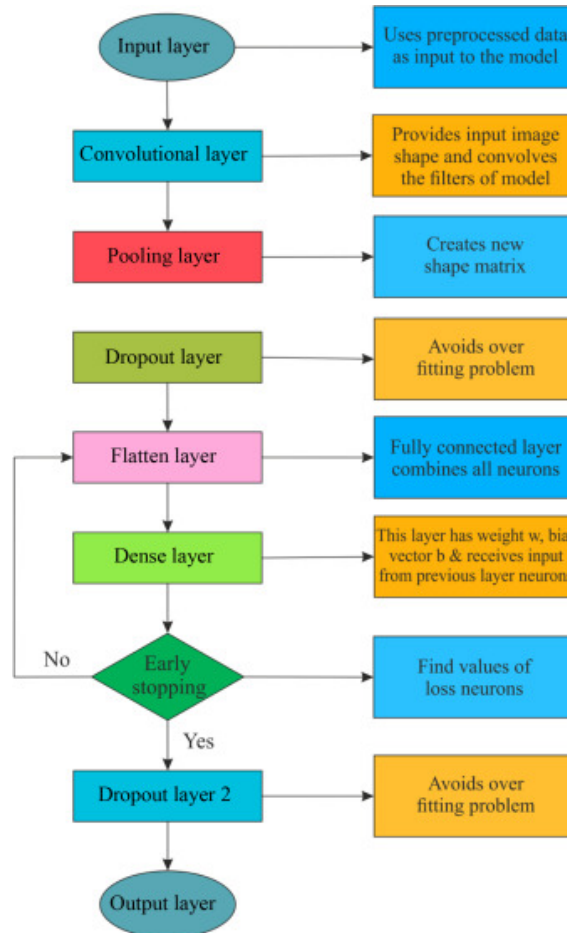


Figura 2.27: Desglose completo de la función de todas las capas que conforman una CNN [31].

2.2.3. Parámetros de las redes neuronales

Desde que se pretende desarrollar modelos de aprendizaje profundo como perceptrones multicapa (MLP) o redes neuronales convolucionales (CNN), se debe plantear cuales son los parámetros que caracterizan estas redes y consiguen una eficiencia destacable en cuanto a tiempo de entrenamiento, validación, fiabilidad, etc.

Estos parámetros pueden ser estáticos o dinámicos, entendiéndose estáticos, como aquellos parámetros que no se adaptan conforme la red se entrena, también conocidos como hiperparámetros y parámetros dinámicos como los que van modificándose y adaptándose al propio entrenamiento de la red, parámetros como, el peso, sesgo, número de neuronas y capas, y funciones de activación.

Por así decirlo, estos últimos parámetros son como la parte visible de las redes neuronales, son aquellos parámetros que podemos visualizar ya que son la estructura base de la red, en cambio, los hiperparámetros actúan internamente y son los encargados de la eficacia de la red, promueven la velocidad y calidad de entrenamiento.

Otros tipos de parámetros o funciones a tener en cuenta, que son muy relevantes e importantes según el tipo de red ha realizar, podrían ser la función de pérdida y el optimizador, siendo métricas que cuantifican la diferencia entre las predicciones de tu modelo y los valores reales y algoritmos que determinan cómo ajustar los pesos y sesgos de la red en función de la señal de error proporcionada por la función de pérdida respectivamente. A fin de cuentas, el objetivo del entrenamiento es minimizar esta pérdida.

Parámetros estáticos:

- Pesos (Weights): son valores numéricos que representan la fuerza de la conexión entre dos neuronas. Determinan cómo la información fluye a través de la red y cuánto influye cada neurona en la siguiente capa.
- Sesgos (Bias): son valores numéricos asociados a cada neurona que actúan como un umbral de activación. Ayudan a controlar la salida de la neurona y a evitar que todas las neuronas se activen al mismo tiempo.
- Número de neuronas y capas: afectan a la capacidad de la red para aprender patrones complejos. Más capas y neuronas generalmente permiten aprender patrones más complejos, pero también aumentan el riesgo de sobreajuste.
- Función de activación: funciones matemáticas (como reLU, sigmoid, tanh, softmax, logSoftmax) que determinan la salida de una neurona. Introducen no linealidad en la red, lo que le permite aprender patrones complejos y se eligen en función del problema y arquitectura de la red.

Hiperparámetros [33]:

- Tasa de aprendizaje (*learning rate*): controla la magnitud de los ajustes realizados en los pesos y sesgos durante cada iteración del entrenamiento. Una tasa de aprendizaje demasiado alta puede llevar a que el modelo no converja, mientras que una tasa demasiado baja puede hacer que el entrenamiento sea muy lento.
- Tamaño del lote (*batch size*): determina cuántos ejemplos de entrenamiento se utilizan para calcular el gradiente en cada iteración. Un tamaño de lote grande puede acelerar el entrenamiento, pero también puede requerir más memoria.
- Número de épocas (*epochs*): indica cuántas veces el modelo verá todo el conjunto de datos de entrenamiento durante el entrenamiento.
- Regularización: técnicas como la deserción (*dropout*) o la regularización L1/L2, que ayudan a prevenir el sobreajuste o *overfitting* [32].

Funciones de pérdida [34]:

- Error cuadrático medio (MSE): común en problemas de regresión.

- Entropía cruzada (CE): común en problemas de clasificación, como en nuestro caso, para clasificar la calidad de los ovocitos en escala del 1 al 10. El algoritmo de CE estima las probabilidades de los eventos en redes estocásticas complejas [34].

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^C \exp(x_{n,c})} \cdot \mathbb{1}_{\{y_n \neq \text{ignore_index}\}} \quad (2.6)$$

- Hinge Loss: utilizada en máquinas de vectores de soporte (SVM). Para clasificación binaria.

Optimizadores [27]:

- Descenso de gradiente estocástico (SGD): simple pero efectivo. Puede ser lento y sensible a la tasa de aprendizaje.
- Momentum sgd: acelera el SGD al acumular el "impulso" de actualizaciones anteriores.
- Accumulated historical gradient (Adagrad): adapta la tasa de aprendizaje para cada parámetro individualmente.
- RMSprop: similar a Adagrad, pero con un decaimiento exponencial para evitar que la tasa de aprendizaje se vuelva demasiado pequeña.
- Adaptive moment estimation (Adam): combina las ventajas de Momentum SGD y RMSprop. Es uno de los optimizadores más populares.

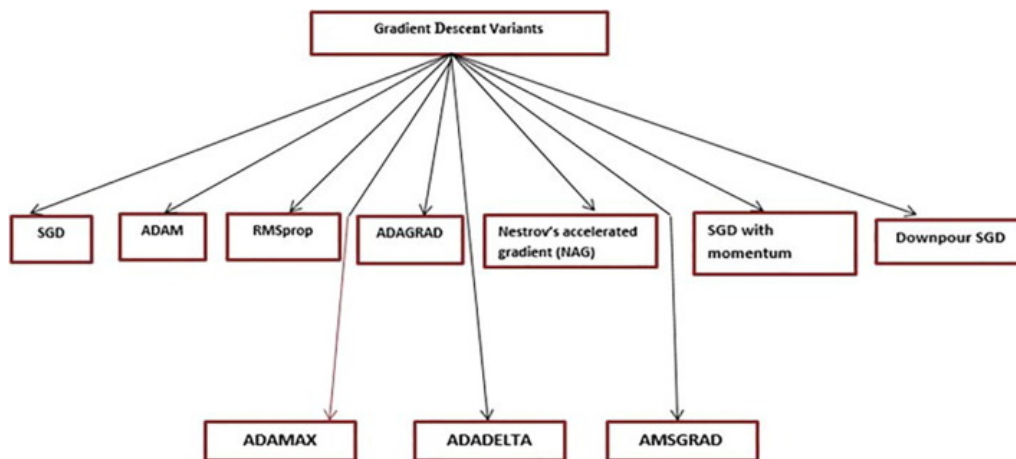


Figura 2.28: Gráfico de optimizadores SGD comunes para la creación de modelos de DL [27].

Otros parámetros que se tienen en cuenta cuando se realizan modelos predictivos mediante técnicas de aprendizaje automático son los parámetros estadísticos calculados de los entrenamientos de la red neuronal. Estos parámetros son útiles para visualizar si un modelo está bien entrenado, si está sobredimensionado y como de fiable es el modelo [35].

- Verdadero positivo (TP): cuando la salida predecida por el modelo es la correcta.

- Verdadero negativo (TN): cuando la salida predecida por el modelo es ciertamente la incorrecta.
- Falso positivo (FP): cuando la salida predecida por el modelo es incorrectamente correcta.
- Falso negativo (FN): cuando la salida predicha por el modelo es incorrectamente incorrecta.

Con la idea de estos parámetros de salida se establecen un conjunto de parámetros para visualizar como el modelo está trabajando y como se ha entrenado [35].

- Sensibilidad o recall: tasa de aciertos o tasa de verdaderos positivos (TPR).

$$TPR = TP * TP + FN \quad (2.7)$$

- Precisión: valores positivos predecidos (PPV).

$$PPV = TP * TP + FP \quad (2.8)$$

- Accuracy: ACC

$$ACC = TP + TN * TP + FP + TN + FN \quad (2.9)$$

- Tasa de falsos positivos (FPR): proporción de la clasificación de negativos como positivos.

$$FPR = \frac{FP}{FP + TN} \quad (2.10)$$

- Área bajo la curva (AUC): determina la capacidad del modelo para distinguir entre clases.

$$AUC = \int_0^1 TPR(FPR) d(FPR) \quad (2.11)$$

2.3. Servicios web

En el ámbito de los servicios web, es fundamental entender las estructuras y tecnologías que permiten la creación, implementación y gestión de aplicaciones distribuidas. Los protocolos web son la columna vertebral de la comunicación entre aplicaciones en una arquitectura orientada a servicios (SOA). A través de protocolos estándar como HTTP, SOAP y REST, los servicios web facilitan la interoperabilidad entre sistemas heterogéneos, permitiendo que las aplicaciones compartan datos y funcionalidades de manera eficiente y segura [36].

Los protocolos estándar utilizados en los servicios web son esenciales para garantizar la comunicación efectiva y segura entre diferentes sistemas y aplicaciones. A continuación, se describen los protocolos más comunes:

HTTP (Hypertext Transfer Protocol): Protocolo que define cómo los mensajes son formateados y transmitidos, y cómo los servidores y navegadores web deben responder a las solicitudes. Es un protocolo sin estado, lo que significa que cada solicitud del cliente al servidor se realiza de forma independiente sin retener información de solicitudes anteriores [37]. HTTP es usado principalmente para la transferencia de documentos HTML, pero también es adaptable para transferir imágenes, vídeos, y datos en formato JSON o XML. HTTP no contempla encriptación y verificación de mensajes, por lo que su versión posterior, llamada HTTPS, añade esta capa de seguridad sobre el protocolo permitiendo encriptar peticiones y respuestas HTTP.

SOAP (Simple Object Access Protocol): SOAP es un protocolo de mensajería basado en XML que permite la comunicación entre aplicaciones a través de diferentes redes [38]. SOAP es conocido por su robustez y extensibilidad, soportando transacciones complejas y seguridad avanzada. Las aplicaciones SOAP pueden operar sobre varios protocolos de transporte, incluyendo HTTP, SMTP y más. SOAP define una estructura de mensajes estándar que permite que las aplicaciones intercambien información de manera consistente y predecible [39].

REST (Representational State Transfer): REST es un estilo de arquitectura para diseñar redes de aplicaciones. En lugar de usar una estructura de mensajería pesada como SOAP, REST se basa en operaciones HTTP estándar como GET, POST, PUT y DELETE [39]. Los servicios RESTful se centran en los recursos y las representaciones de esos recursos. Cada recurso es accesible a través de un URI (Uniform Resource Identifier) y las interacciones con estos recursos se realizan a través de las operaciones HTTP estándar. REST es ampliamente utilizado debido a su simplicidad, escalabilidad y facilidad de integración con sistemas web modernos.

2.3.1. Paradigma de microservicios

El paradigma de microservicios ha emergido como una alternativa moderna a la arquitectura monolítica tradicional. En lugar de construir aplicaciones como una sola unidad cohesiva, el enfoque de microservicios descompone la aplicación en múltiples servicios pequeños, autónomos y modulares, cada uno ejecutando un proceso único y comunicándose a través de interfaces ligeras, típicamente APIs REST [40, 41, 42].

Los beneficios de los microservicios incluyen [43]:

- Escalabilidad: cada microservicio puede escalarse de manera independiente según sus necesidades de carga.
- Resiliencia: la falla de un microservicio no afecta necesariamente a los demás, lo que mejora la robustez de la aplicación.
- Despliegue continuo: permite la implementación continua e independiente de microservicios, facilitando actualizaciones y mejoras.
- Tecnología heterogénea: cada microservicio puede desarrollarse usando diferentes tecnologías y lenguajes de programación que sean más adecuados para su función específica.

2.3.2. Docker

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software. Proporciona una capa adicional de abstracción y automatización de virtualización a nivel de sistema operativo [44]. Esto proporciona una plataforma flexible, portátil y eficiente para desarrollar, desplegar y gestionar aplicaciones que involucran todo tipo de proyectos y componentes web [45]. Su capacidad para encapsular entornos complejos, garantizar la reproducibilidad y escalabilidad convirtiéndolo en una herramienta valiosa para proyectos de investigación de software y aplicaciones del mundo real [46].

Docker tiene dos componentes muy importantes, *Docker images* y *Docker containers*. *Docker image* es un archivo o clase donde se definen las bibliotecas, dependencias y archivos que necesita el contenedor para ejecutarse. En cambio, un contenedor de docker es un entorno de ejecución donde se establecen todas las dependencias y componentes necesarios para ejecutar correctamente el código de la aplicación en sí [47, 44].

Un contenedor docker se asemeja a lo que se conoce como máquina virtual (VM de virtual machine), que es un entorno de sistema que opera únicamente en el host del sistema operativo en el que se usa y cada aplicación su gestor de sistema operativo. Un contenedor docker se puede ejecutar desde cualquier ordenador y en cualquier sistema operativo, ya que trabajara en posterior sobre Linux. En la Tabla 2.1 se establecen unas diferencias claras entre ambas tecnologías. En la Figura 2.29 se observan dos diagrama de bloques, en la parte izquierda se organiza la base de la estructura de una VM, y a la derecha se representa la estructura funcional en bloques de un contenedor docker.

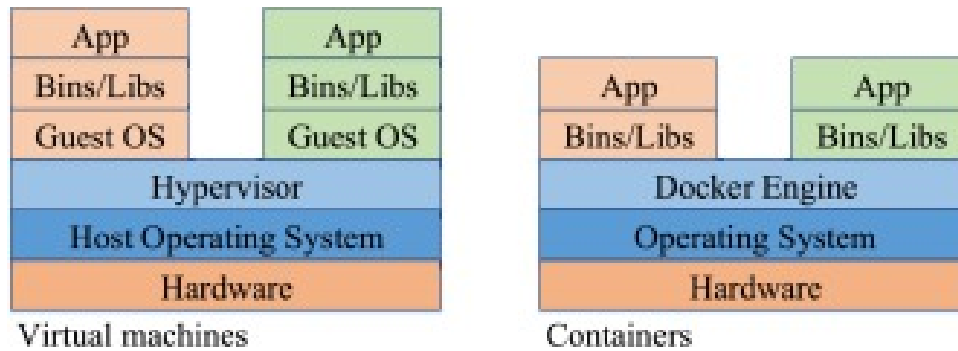


Figura 2.29: Diferenciación en diagrama de bloques de una máquina virtual (VM) y un contenedor docker [44].

Tabla 2.1: Tabla comparativa de las propiedades y usos de las máquinas virtuales con respecto a Docker [46].

	Virtual machine	Container
Definición	Se ejecuta sobre el hipervisor.	Se ejecuta sobre el sistema operativo host.
Aislamiento	Completamente aislado del sistema operativo host y de otras máquinas virtuales.	Aislamiento ligero del sistema operativo host y otros contenedores.
Tamaño	Muy grande.	Muy pequeño.
Tiempo de Arranque	Minutos en ejecutarse.	Segundos para ejecutarse.
Seguridad	Más seguro.	Menos seguro.
OS	Cada aplicación tener un OS diferente.	Cada aplicación comparte el mismo OS.
Memoria	Utiliza más memoria.	Requiere menos memoria.
Entorno	Virtualiza el sistema informático.	Virtualiza el sistema operativo.
Balance de carga	Mueva las máquinas virtuales en ejecución a otro servidor en un clúster de conmutación por error para equilibrar la carga.	Los contenedores no se pueden mover por sí solos. El contenedor utiliza un orquestador para iniciar o detener automáticamente los contenedores.
Portabilidad	Son menos portátiles porque es más difícil moverlas.	Son fáciles de mover.
Despliegue	Una sola máquina virtual mediante el Centro de administración de Windows. Varias máquinas virtuales mediante Virtual Machine Manager (VMM). Ejecutar diferentes	Un solo contenedor con Docker. Usando docker-compose, se pueden lanzar varios contenedores a la vez.
Aplicaciones	aplicaciones utilizando diferentes sistemas operativos.	Ejecutar el máximo de aplicaciones utilizando el mínimo de servidores.
Ejemplos	KVM, VMware, Xen	Docker

El funcionamiento de las aplicaciones web con Docker, va desde la declaración de las dependencias de cada aplicación que se quiera ejecutar, se construye una imagen para cada una de estas aplicaciones, se lanzan los contenedores con un servidor asociado como volumen docker. Un volumen docker es un sistema de almacenamiento independiente de los contenedores activos que es gestionado completamente por Docker, por lo que los datos permanecen aún sin usarse el volumen [48]. Con *docker compose* se orquesta el despliegue de las aplicaciones que implican más de un docker simultáneamente, declarando en un archivo ".yaml" los servicios docker de cada aplicación con un puerto distinto para cada aplicación [49]. En la Figura 2.30 se representa el proceso por el cual se ejecutan las aplicaciones con los contenedores docker. Gracias a que comparten la misma red en la ejecución de las aplicaciones, estas pueden comunicarse entre sí para que los procesos realizados por las aplicaciones puedan compartirse, paralelizando así el flujo de trabajo [50]. El último componente que se fija en la parte central derecha de la Figura 2.30 es la implementación de un archivo *docker-compose.yml* que ejecuta tantos contenedores como se quiera de forma simultánea con una única línea de código Linux.

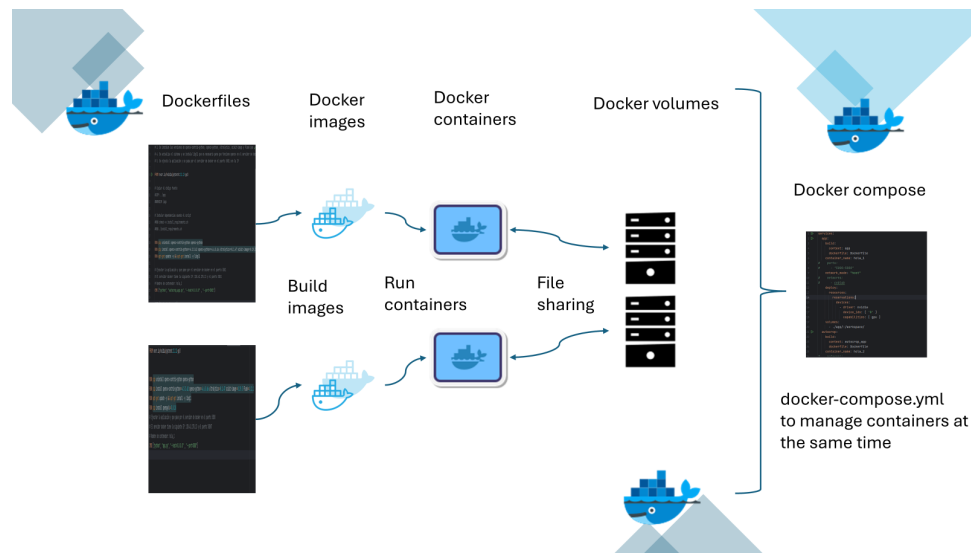


Figura 2.30: Proceso de lanzamiento de aplicaciones con Docker.

2.3.3. Lenguajes de programación

Para el desarrollo web existen infinidad de lenguajes útiles y cada uno de ellos tiene unas especificaciones y utilidades en concreto. Por conveniencia y comodidad con el proyecto a realizar, donde se pretende desarrollar modelos de DL y una aplicación web, se decide utilizar el lenguaje de programación Python [51]. Para desarrollo web, se ha utilizado el framework de python, Flask [52] que es un micro framework open source, sencillo y ampliamente utilizado, el cual utilizamos como lenguaje para el backend de la aplicación. En cuanto al frontend de la aplicación, se han usado lenguajes de marcado de interfaz como HTML y CSS, en conjunto de javascript, siendo utilizado de código lógico, que son los estándares.

Backend:

Python: Lenguaje versátil, popular por su simplicidad y legibilidad. Frameworks como Django y Flask facilitan el desarrollo de aplicaciones web. Utilizado principalmente para DL y *Datascience*.

Flask: Es un microframework de aplicaciones web escrito en Python. Es explícito y simple, de ahí que se le denomine con el término "micro". Permite crear aplicaciones útiles rápidamente y se conecta muy bien con el frontend mediante Python, HTML y Js [53].

Pandas: Pandas es una librería para la gestión de documentos excel sobre el lenguaje de programación Python, lo que permite la carga y control de los datos de un archivo excel [54].

Frontend:

HTML: Es la estructura básica de una página web, define el contenido y su organización,

pero está penalizado por su simplicidad y estructura estática. Debido a esto, toda aplicación web media, necesita de estilo (CSS) y dinamismo (JavaScript) [55].

CSS: Se encarga del estilo visual de la página web, como colores, fuentes, diseño y disposición de los elementos [56].

JavaScript: Proporciona interactividad a la página web, permite crear animaciones, validar formularios, modificar el contenido dinámicamente y comunicarse con el backend [57].

Capítulo 3

Material

En este capítulo se expone brevemente el material necesario sobre el cual se desarrolla el proyecto, como es la base de datos (BBDD).

3.1. Base de datos

Para el desarrollo y entrenamiento de los modelos de clasificación de calidad de ovocitos se necesita de unos datos previos para llevar a cabo el aprendizaje de las características en conjunto de la calidad asociada a los mismos. Los datos, como se comenta en capítulos previos han sido proporcionados por parte del instituto valenciano de infertilidad (IVI), quienes nos han dotado de un set de 1100 imágenes de ovocitos y un documento excel con todas las características anotadas una a una por embriólogos para cada ovocito.

La característica principal de las imágenes es que su formato es jpg, Las imágenes son en blanco y negro (escala de grises), lo cual ahorra tiempo de computación en tareas de preprocesado de imágenes como el cambio de escala (RGB a gray), también, las imágenes en escala de grises tienen un único canal en la matriz en vez de tres canales (como así ocurre en RGB), lo cual ahorra tiempo complicación de computación cuando se entrenan los modelos. Estas imágenes ya están pasadas por el modelo de autocrop (explicado en detalle en las secciones 4.2.1 y 4.3.3 del presente proyecto) previamente (gracias a los compañeros del departamento de Ovovitrif), por lo que ha evitado la necesidad de pasarlas por el modelo de autocrop.

Las características de los ovocitos son las ya mencionadas en la sección de dismorfismos ovocitarios del marco embriológico 2.1.1. En la Tabla 3.1 se sitúan en la primera columna las características o dismorfismos a evaluar. Todos los valores son independientes, la Tabla 3.1 muestra las posibles anotaciones que los embriólogos hacen sobre los dismorfismos anotados en la primera columna.

Tabla 3.1: Conjunto de características de los dismorfismos ovocitarios.

Puntuación	1	2	3	4	5	6	7	8	9	10
Normal	SI (1)	NO (0)								
Ovocito Elongado	SI (1)	NO (0)								
Distribución Granulosidad	AUSENTE	OPUESTA AL CP	CENTRAL	DIFUSA	LATERAL	OTRA				
Granulosidad	AUSENTE	POCO	MUY POCO	MUCHO	BASTANTE	MUCHÍSIMO				
Superficie Vacuolas	<5%	5-10%	10-25%							
Superficie REL										
Número de Cuerpos Refrigrantes	AUSENTE	1-15	>15							
Número de Cuerpos Necróticos	AUSENTE	1-15	>15							
Debris EPV	AUSENTE	MUY POCO	POCO	BASTANTE	MUCHO					
ZP Elongada	SI (1)	NO (0)								
ZP Oscura	SI (1)	NO (0)								
Grosor ZP	NORMAL	GRUESA	FINA							
ZP Septada	SI (1)	NO (0)								
EPV Aumentado	AUSENTE	MUY POCO	POCO	BASTANTE	MUCHO	MUCHÍSIMO				
CP Fragmentado	AUSENTE	MUY POCO	POCO	BASTANTE	MUCHO	MUCHÍSIMO				
Tamaño CP	NORMAL	PEQUEÑO	GRANDE							

3.1.1. Limpieza y manejo de datos

En cuanto al conjunto de imágenes cropeadas de los ovocitos están denotadas con nombres muy variados, es decir, los nombres de las imágenes en el directorio de imágenes no son los mismos que hay en la BBDD asociados a las características. Por lo que se procede a unificar un modelo de nombre de imagen por ambos lados, así cuando se quieran cargar las imágenes del directorio y entrenar los modelos con las etiquetas y características de la base de datos, la búsqueda sea efectiva y correcta.

En cuanto a las imágenes del directorio, el formateo de imágenes se ha conseguido siguiendo un patrón de "_", donde por el número de "_" se eliminan las partes correspondientes para que el nombre sea sencillo de leer y así poder compararlos con los de la BBDD, como se muestra en la Tabla 3.2.

El nombre base u original de las imágenes de la BBDD era diferente al del directorio, pero el proceso en sí ha sido el mismo, solo que aplicando fórmulas matemáticas, en vez de un programa de python que cuente "_" para eliminar así las partes innecesarias. El resultado es el mismo que el de la Tabla 3.2.

Tabla 3.2: Formato de cambio para el nombre de las imágenes del directorio.

Nombres originales	Nombres nuevos
40_24643070_7232151_Egg_01.jpg	24643070_01.jpg
12884206_Egg_02_2023-7-14_12.14.27.jpg	12884206_02.jpg
50_24600711_7232256_01.jpg	24600711_01.jpg
24600711_7232256_01.jpg	24600711_01.jpg
14690460_7233242_Egg_08_2023-11-03_12.22.34.jpg	14690460_08.jpg
24640725_Egg_10_15_06_2023.jpg	24640725_10.jpg
24632480.03_Egg_07_2023-7-05_14.52.36.jpg	24632480.03_07.jpg

3.1.2. Análisis y selección de datos

En la implementación de redes neuronales para clasificación de imágenes es tan importante las características del modelo de entrenamiento y preprocesado de las imágenes como la elección correcta de las características supervisadas en relación a las etiquetas de clasificación.

Tanto para modelos de multi layer perceptrons (MLPs), como para redes neuronales convolucionales (CNNs) se tiene una gran cantidad de datos y características asociadas a las imágenes, es muy importante optimizar el número de variables a tener en cuenta. Es por eso que de 16 características asociadas, nos centraremos en las 5 ó 6 más importantes para optimizar así el entrenamiento de la red neuronal.

Para ello se ha analizado la calidad de los ovocitos con cada de una de las características de forma visual mediante gráficos del tipo histograma y charts en una dash web y se ha obtenido una lista de las 6 combinaciones de características que consiguen una puntuación por encima del '6', es decir, calidad de 7 o superior. Dash [58] es un framework de python como flask, pero especializado en visualizaciones personalizables. Para detectar el impacto de las subcaracterísticas que se implican en una buena calidad de ovocito, se generara una dashboard en la que visualizar gráficas simples para así obtener una perspectiva de los conjuntos de subcaracterísticas asociadas a cada calidad.

Un histograma [59] es una representación gráfica por medio de barras para señalar el número de veces (frecuencia) que aparece una variable en la BBDD. Un ejemplo sencillo sería representar por histograma, el número de ovocitos normales o no normales que hay en su totalidad , es decir, número de "SI(1)" y "NO(0)".

También, en el análisis de datos, es común calcular varias métricas descriptivas para entender la distribución de los datos. A continuación se explica cómo se calcula la media, la desviación estándar y la mediana de un conjunto de datos.

La **media** (o promedio) es una medida de tendencia central que se calcula sumando todos los valores de un conjunto de datos y dividiendo entre el número total de valores. La fórmula para la media de un conjunto de valores x_1, x_2, \dots, x_n es:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (3.1)$$

donde:

- \bar{x} es la media.
- n es el número total de valores.
- x_i representa cada valor individual en el conjunto de datos.

La **desviación estándar** mide la dispersión de los valores con respecto a la media. Se calcula como la raíz cuadrada de la varianza. La fórmula para la desviación estándar σ es:

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2} \quad (3.2)$$

donde:

- σ es la desviación estándar.
- \bar{x} es la media.
- x_i representa cada valor individual en el conjunto de datos.
- n es el número total de valores.

La **mediana** es el valor que divide el conjunto de datos en dos partes iguales. Si el número de observaciones es impar, la mediana es el valor central. Si es par, la mediana es el promedio de los dos valores centrales. La fórmula para la mediana no es una fórmula exacta, sino que depende del ordenamiento de los datos. Si $x_{(1)}, x_{(2)}, \dots, x_{(n)}$ son los valores ordenados, la mediana se define como:

$$\text{Mediana} = \begin{cases} x_{(\frac{n+1}{2})} & \text{si } n \text{ es impar} \\ \frac{x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}}{2} & \text{si } n \text{ es par} \end{cases} \quad (3.3)$$

donde:

- $x_{(i)}$ es el valor en la posición i -ésima después de ordenar los datos.
- n es el número total de valores.

A continuación se explican los resultados obtenidos de la BBDD de cada característica a raíz de el estudio estadístico realizado:

- En el primer histograma de la Figura 3.2, se representa en un 95.81 % la existencia de ovocitos no elongados y el 4.19 % de ovocitos elongados, junto a que el rango de puntuaciones de los no elongados está en el rango [4, 7] como se muestra en la Figura 3.1. Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.3 son de 4.55, 1.18 y 4.55 respectivamente.
- En el segundo histograma y chart de las Figuras 3.2 y 3.1, se representa con un 43.77 % la ocurrencias de granulosidad lateral con una puntuación del [4, 7], en un 27.48 % de granulosidad difusa con puntuación del [4, 6], en un 15.1 % de granulosidad ausente con puntuación del [4, 8], en una 12.28 % de granulosidad central con puntuación del [3, 6], en un 0.91 % de granulosidad opuesta al CP con puntuación del [5, 7], y en un 0.45 % de otro tipo de granulosidad con una puntuación del [3, 5]. Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.3 son de 5.10, 0.79 y 5.07 respectivamente.

- En el tercer histograma y chart de las Figuras 3.2 y 3.1, se representa en un 28.21 % de muy poca granulosidad con una puntuación del [4, 8], en un 24.02 % de poca granulosidad con una puntuación del [4, 7], en un 23.66 % de bastante granulosidad con una puntuación del [4, 6], en un 15.2 % granulosidad ausente con una puntuación del [4, 8], en un 7.55 % de mucha granulosidad con una puntuación del [2, 5], y en un 1.36 % de muchísima granulosidad con una puntuación del [1, 2]. Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.3 son de 4.59, 1.72 y 5.08 respectivamente.
- En el cuarto histograma y chart de las Figuras 3.2 y 3.1, se representa en un 96.82 % la ausencia de vacuolas con una puntuación del [4, 7], en un 2.18 % de 5 % de la superficie en vacuolas con una puntuación del [3, 7], en un 0.73 % de 5-10 % de la superficie en vacuolas con una puntuación del [2, 4], y en un 0.27 % de 10-25 % de la superficie en vacuolas con una puntuación del [3, 6]. Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.3 son de 4.47, 0.84 y 4.58 respectivamente.
- En el quinto histograma y chart de las Figuras 3.2 y 3.1, se representa en un 92.45 % de superficie REL ausente con una puntuación del [4, 7], en un 3.28 % de 5 % de REL con una puntuación del [3, 5], en un 2.64 % de 5-10 % de REL con una puntuación del [2, 3], en un 0.91 % de 25-50 % de REL con una puntuación del [1, 2], en un 0.55 % de 10-25 % de REL con una puntuación del [1, 4], y en un 0.18 % de 50-75 % de REL con una puntuación nula. Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.2 son de 2.92, 1.67 y 2.68 respectivamente.
- En el octavo histograma y chart de las Figuras 3.2 y 3.1, se representa en un 51.77 % de debris ausente con una puntuación del [4, 7], en un 29.66 % de muy poca debris con una puntuación del [4, 7], en un 10.92 % de poca debris con un puntuación del [4, 7], en un 6.37 % de bastante debris con una puntuación del [3, 6], y en un 1.27 % de mucha debris con una puntuación del [1, 5]. Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.3 son de 4.75, 0.91 y 5.03 respectivamente.
- En el noveno histograma y chart de las Figuras 3.2 y 3.1, se representa en un 76.98 % de ZP no elongada con una puntuación del [4, 7], y en un 23.02 % de ZP elongada con una puntuación del [4, 6]. Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.3 son de 5.12, 0.5 y 5.12 respectivamente.
- En el décimo histograma y chart de las Figuras 3.2 y 3.1, se representa en un 90.45 % de ZP clara con una puntuación del [4, 7], y en una 9.55 % de ZP oscura con una puntuación del [4, 6]. Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.2 son de 5.10, 0.37 y 5.10 respectivamente.
- En el decimoprimer histograma y chart de las Figuras 3.2 y 3.1, se representa en un 75.07 % de ZP normal con una puntuación del [4, 7], en un 17.2 % de ZP fina con una puntuación del [4, 7], y en un 7.73 % de ZP gruesa con una puntuación del [3, 7].

Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.3 son de 5.21, 0.18 y 5.28 respectivamente.

- En el decimosegundo histograma y chart de las Figuras 3.2 y 3.1, se representa en un 96.54% de ZP no septada con una puntuación del [4, 7], y en un 3.46% de ZP septada con una puntuación del [2, 6]. Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.3 son de 4.91, 0.62 y 4.91 respectivamente.
- En el decimotercer histograma y chart de las Figuras 3.2 y 3.1, se representa en un 47.32% de EPV ausente con una puntuación del [4, 7], en un 26.3% de muy poco EPV con una puntuación del [4, 7], en un 12.01% de poco EPV con una puntuación del [4, 6], en un 9.55% bastante EPV con una puntuación del [4, 6], en un 3.91% de mucho EPV con una puntuación del [2, 5], y en un 0.91% de muchísimo EPV con una puntuación del [2, 4]. Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.3 son de 4.59, 1.00 y 4.83 respectivamente.
- En el decimocuarto histograma y chart de las Figuras 3.2 y 3.1, se representa en un 83.26% de CP fragmentado ausente con una puntuación del [4, 7], en un 7.64% de muy poco CP fragmentado con una puntuación del [4, 7], en un 5% de poco CP fragmentado con una puntuación del [4, 6], en un 2.27% bastante CP fragmentado con una puntuación del [3, 5], en un 1.46% de mucho CP fragmentado con una puntuación del [2, 4], y en un 0.36% de muchísimo CP fragmentado con una puntuación del [2, 3]. Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.3 son de 4.26, 1.18 y 4.41 respectivamente.
- En el decimoquinto histograma y chart de las Figuras 3.2 y 3.1, se representa en un 94.63% de CP normal con una puntuación del [4, 7], en un 4.09% de CP grande con una puntuación del [4, 7], y en un 1.27% de CP pequeño con una puntuación del [3, 6]. Las métricas de media, desviación típica y mediana total de la característica según se presenta en la Figura 3.3 son de 4.97, 0.65 y 5.33 respectivamente.

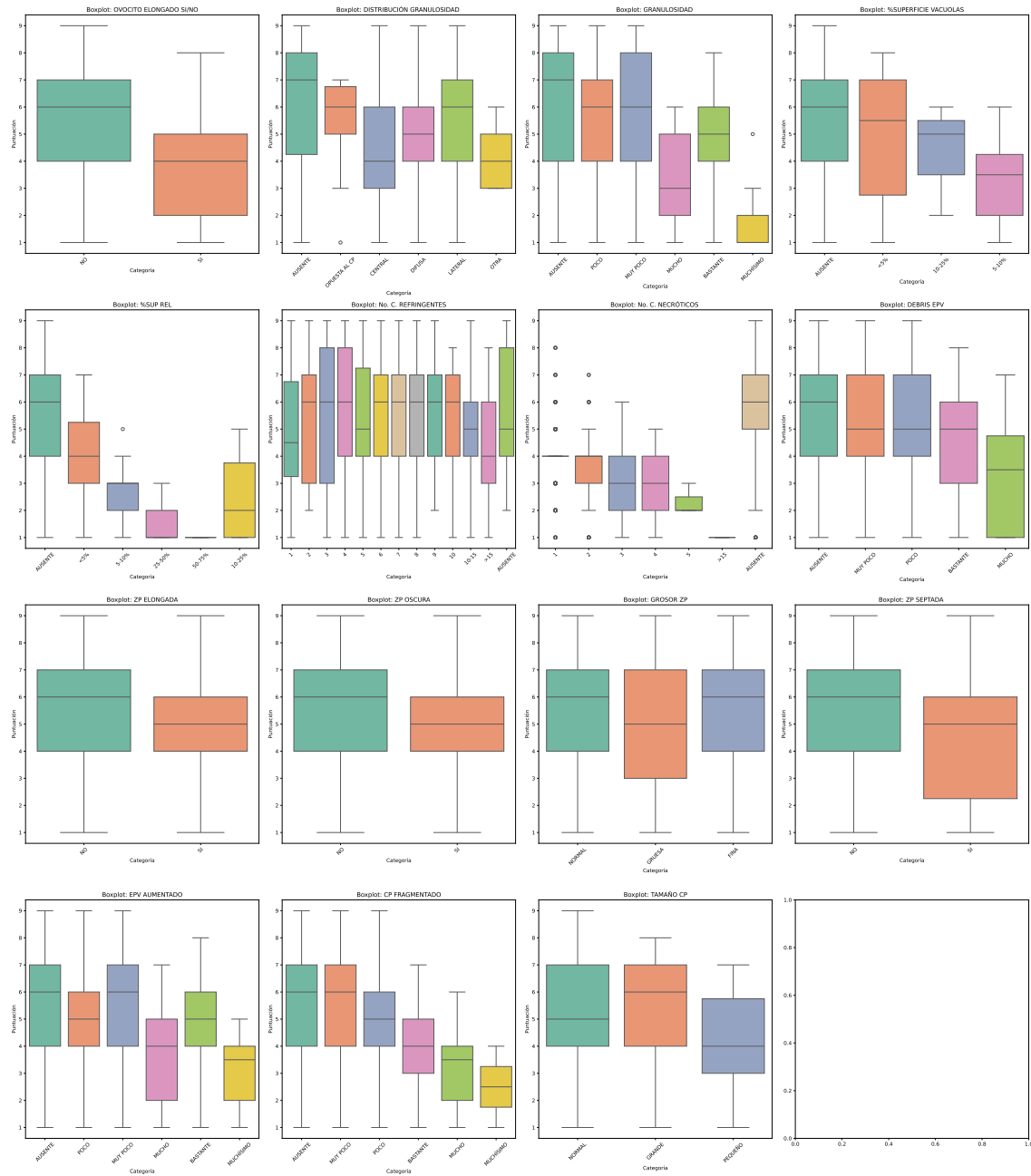
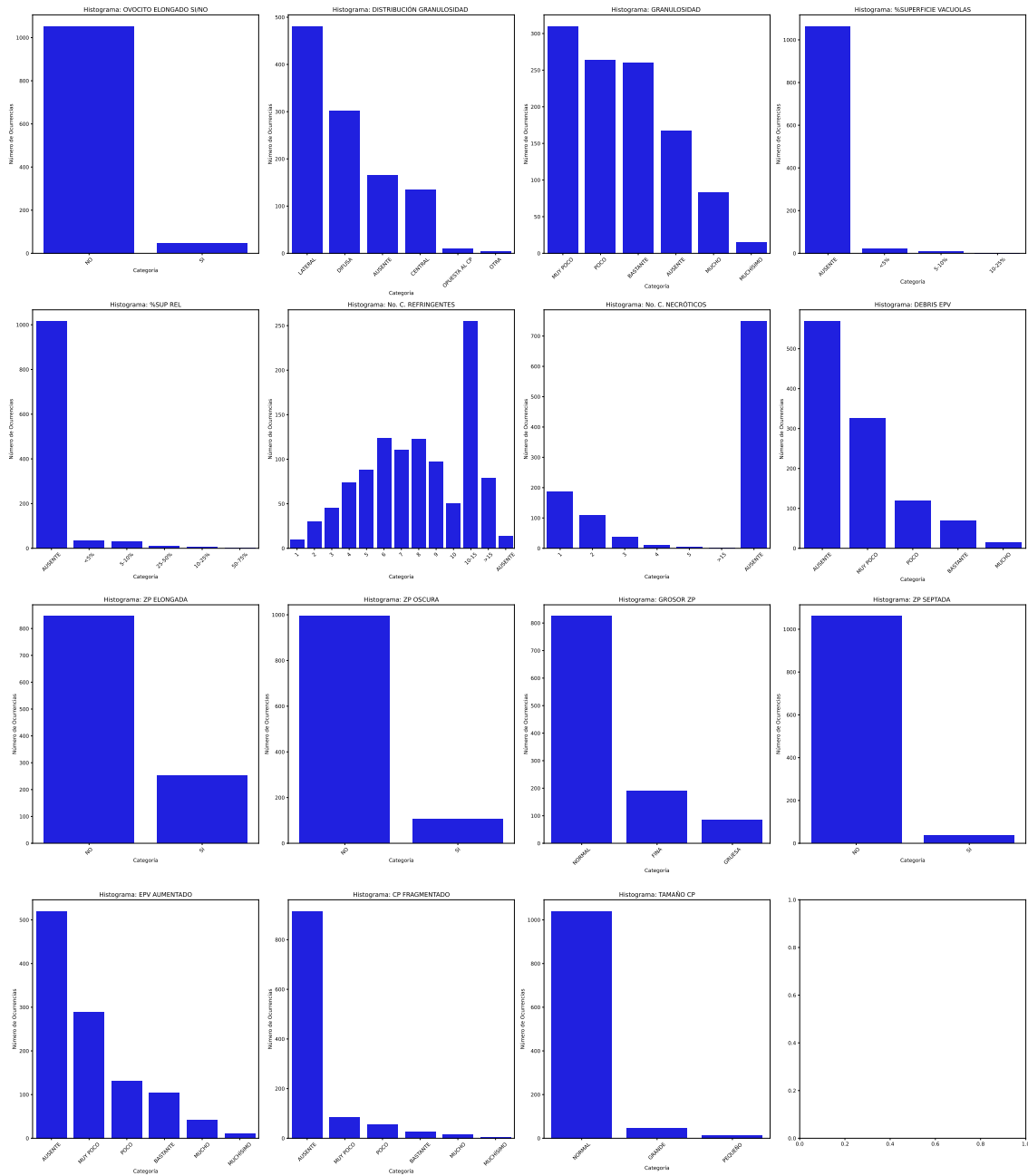


Figura 3.1: Conjunto de charts que relacionan el rango de puntuaciones activas de cada subcaracterística dentro de cada característica de la BBDD.



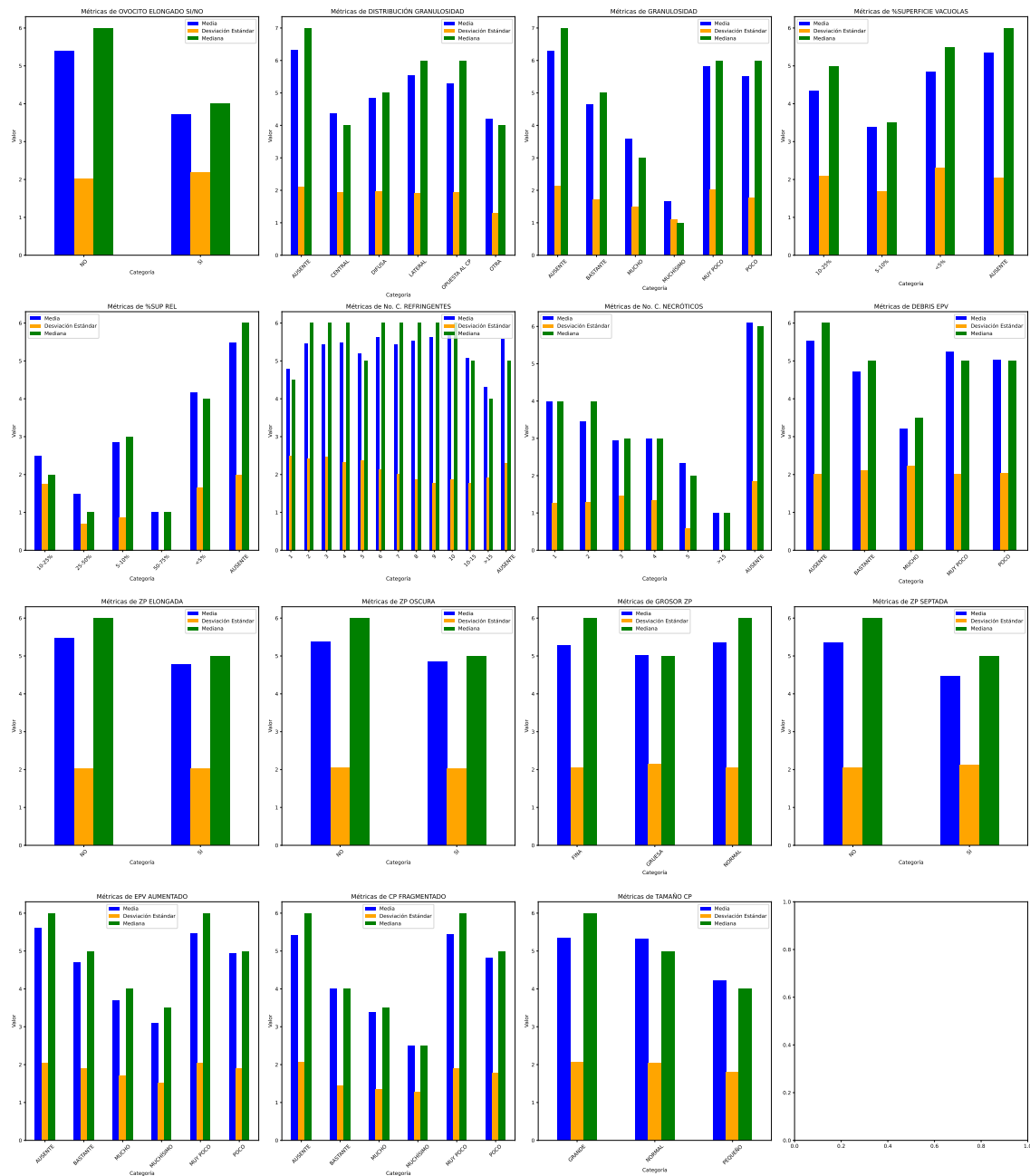


Figura 3.3: Conjunto de métricas obtenidas de la BBDD para cada subcaracterística dentro de cada característica.

Haciendo una evaluación de las combinaciones de subcaracterísticas (ej: ausencia de granulosidad, tamaño de CP grande, 5% de REL, etc) que consiguen una puntuación elevada, de "7.º superior obtenemos la siguiente Tabla 3.3.

Tabla 3.3: Conjunto de subcaracterísticas de los Ovocitos con mejores Calidades.

Características	Subcaracterísticas					
OVOCITO ELONGADO SI/NO	NO	NO	NO	NO	NO	NO
DISTRIBUCIÓN GRANULOSIDAD GRANULOSIDAD	LATERAL	LATERAL	LATERAL	AUSENTE	AUSENTE	AUSENTE
%SUPERFICIE VACUOLAS	MUY POCO	MUY POCO	POCO	AUSENTE	AUSENTE	AUSENTE
%SUP REL	AUSENTE	AUSENTE	AUSENTE	AUSENTE	AUSENTE	AUSENTE
No. C. REFRINGENTES	AUSENTE	AUSENTE	AUSENTE	AUSENTE	AUSENTE	AUSENTE
No. C. NECRÓTICOS	10-15	5	8	8	6	5
DEBRIS EPV	AUSENTE	AUSENTE	AUSENTE	AUSENTE	AUSENTE	AUSENTE
ZP ELONGADA	AUSENTE	AUSENTE	AUSENTE	AUSENTE	AUSENTE	AUSENTE
ZP OSCURA	NO	NO	NO	NO	NO	NO
GROSOR ZP	NO	NO	NO	NO	NO	NO
ZP SEPTADA EPV	NORMAL	NORMAL	NORMAL	NORMAL	NORMAL	NORMAL
AUMENTADO CP	NO	NO	NO	NO	NO	NO
FRAGMENTADO TAMAÑO CP	AUSENTE	AUSENTE	AUSENTE	AUSENTE	AUSENTE	AUSENTE
Frecuencia	AUSENTE	AUSENTE	AUSENTE	AUSENTE	AUSENTE	AUSENTE
	NORMAL	NORMAL	NORMAL	NORMAL	NORMAL	NORMAL
	6	5	5	4	4	4

De la Tabla 3.3 se declaran una serie de patrones que consiguen una calificación en el rango [7, 9]. Estas características mantienen una combinación bastante ajustada, en donde para que el ovocito se vaya a clasificar con buena calidad, ha de cumplir bastantes de las características vistas en la Tabla 3.3 y como se observa en la Tabla 3.4.

Tabla 3.4: Subcaracterísticas que reflejan una buena calidad.

Características	Subcaracterísticas que proporcionan calidad >6
OVOCITO	NO
ELONGADO SI/NO	NO
DISTRIBUCIÓN	LATERAL /
GRANULOSIDAD	AUSENTE
GRANULOSIDAD	MUY POCO /
	AUSENTE
%SUPERFICIE	AUSENTE
VACUOLAS	AUSENTE
%SUP REL	AUSENTE
No. C.	>5
REFRINGENTES	>5
No. C.	AUSENTE
NECRÓTICOS	AUSENTE
DEBRIS	AUSENTE
EPV	AUSENTE
ZP	NO
ELONGADA	NO
ZP	NO
OSCURA	NO
GROSOR ZP	NORMAL
ZP SEPTADA	NO
EPV	AUSENTE
AUMENTADO	AUSENTE
CP	AUSENTE
FRAGMENTADO	AUSENTE
TAMAÑO CP	NORMAL

Capítulo 4

Metodología

En este capítulo 4 se desarrolla la implementación de los modelos predictivos y de la aplicación web en conjunto al servidor y cada sub-aplicación instanciada para su funcionamiento en paralelo con contenedores Docker.

4.1. Análisis de imagen

Se definen como bordes, en términos de procesamiento digital de imágenes, las zonas en las que se produce un fuerte cambio de intensidad. Su detección resulta útil en diversas disciplinas, como la medicina, cuando se plantea la necesidad de distinguir formas o reconocer figuras dentro de una imagen. Hasta hoy se han desarrollado numerosos algoritmos de detección de bordes (Roberts, Prewitt, Sobel, Canny, Laplace, etc.) y es por ello que en este proyecto se han comparado tres técnicas comúnmente utilizadas para este propósito, como lo son el algoritmo de Canny [60], de Sobel [61] y el de Laplace [62].

Canny [60] propone un algoritmo de detección de bordes mediante el uso de la primera derivada, que es usada por que toma el valor de cero en todas las regiones donde no varía la intensidad y tiene un valor constante en toda la transición de intensidad. Por tanto un cambio de intensidad se manifiesta como un cambio brusco en la primera derivada, característica que es usada para detectar un borde, y en la que se basa el algoritmo de Canny. Véase Figura 4.1.

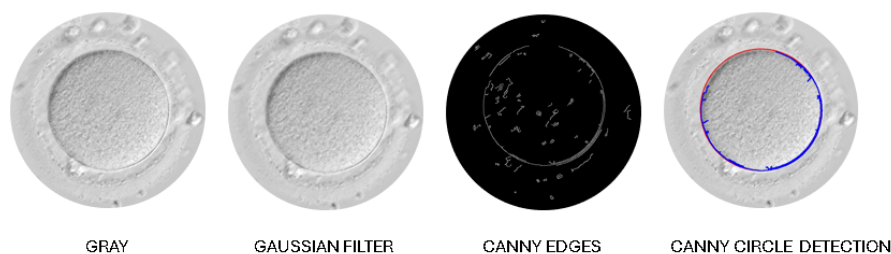


Figura 4.1: Detección de bordes con algoritmo de Canny.

El algoritmo de Sobel [61] es uno de los detectores de bordes más utilizados, y reduce

el ruido a la vez que proporciona una respuesta de distinción y de borde. Véase Figura 4.2.



Figura 4.2: Detección de bordes con algoritmo de Sobel.

Los detectores de bordes laplacianos [62] son diferentes de los detectores de bordes mencionados anteriormente. En esta técnica sólo se utiliza un filtro (también llamado núcleo). La detección de bordes laplaciana ejecuta derivadas de segundo orden en una sola pasada, lo que la hace susceptible al ruido. Antes de utilizar este enfoque, la imagen se suaviza con un suavizado gaussiano para evitar esta susceptibilidad al ruido. Véase Figura 4.3.

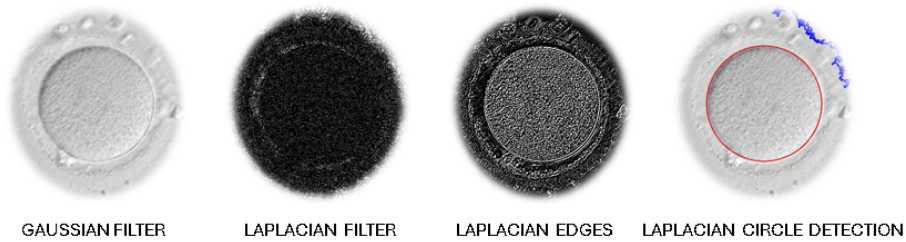


Figura 4.3: Detección de bordes con algoritmo de Laplace.

Las tres técnicas mencionadas anteriormente (algoritmos de Canny, Sobel y Laplace) para detección de bordes siguen la misma estructura de preparación y codificación de algoritmo (véase Figura 4.4). Previa a la búsqueda de los bordes a detectar se realiza la normalización de la imágenes mediante el cambio de tamaño y suavizado de la imagen para así facilitar los procesos siguientes. Seguido, se analizan las imágenes con los algoritmos para detectar los bordes y encontrar aquellos que son adecuados para el propósito de obtención del diámetro y área de los ovocitos (Los parámetros de los algoritmos han sido alterados mediante prueba y error hasta encontrar los adecuados). Finalmente los datos de los círculos que rodean al ovocito son pasados por una función que calcula las dimensiones que se requieren, esta función se muestra posteriormente en la Figura 4.5.

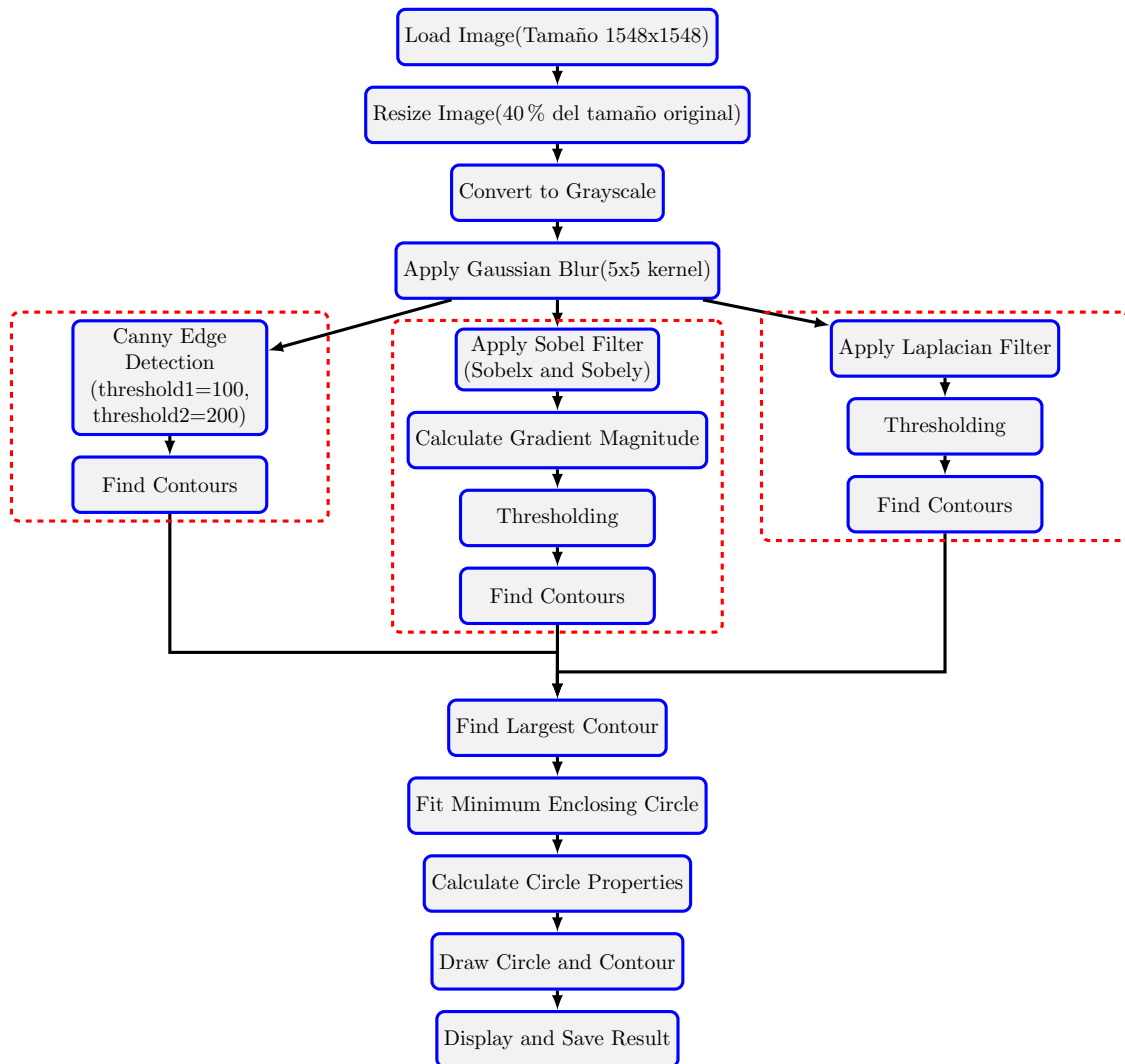


Figura 4.4: Diagrama de bloques del proceso de detección de bordes con los algoritmos Canny, Sobel y Laplacian.

Con los bordes detectados, se puede establecer relaciones con los tamaños de las imágenes para así obtener las dimensiones físicas de los ovocitos, como el diámetro y el área, así como se representa en los últimos bloques de la Figura 4.4. Estos parámetros pueden ser útiles para aplicar técnicas de aprendizaje automático y comparar resultados con otros modelos en los que no se hallen dichas características de los ovocitos.

Con la librería de Python, OpenCV [63] se han detectado mediante las técnicas explicadas los bordes de los ovocitos, pero también esta librería provee métodos para obtener el radio de los círculos dibujados [64] y mediante un radio de una circunferencia se puede obtener su diámetro y su área respectivamente.

La función "minEnclosingCircle(contour)" devuelve el centro del círculo y el radio del círculo dado el contorno detectado previamente. Posteriormente, establecemos la relación ancho-alto de cambio de píxeles para que considere el tamaño de la imagen original y así llamamos a una función creada para el cálculo de las dimensiones del círculo como se

puede observar en la Figura 4.5.

Un ovocito, tiene un diámetro promedio de 150 μm , por lo que los cálculos que hagámos sobre las dimensiones de los ovocitos han de estar expresados en μm . Sabiendo que la salida de la función `minEnclosingCircle(contour)` están expresadas en unidades de píxeles (px), se debe realizar la conversión de px a μm . Como se detalla en el diagrama de flujo de la Figura 4.5, 1 píxel son 264.58 micrómetros y 1 micrómetro son 0.00377957517575 píxeles [65]. Una vez obtenido el radio en micras, se calcula el diámetro y área fácilmente y se devuelve como salida de la función.

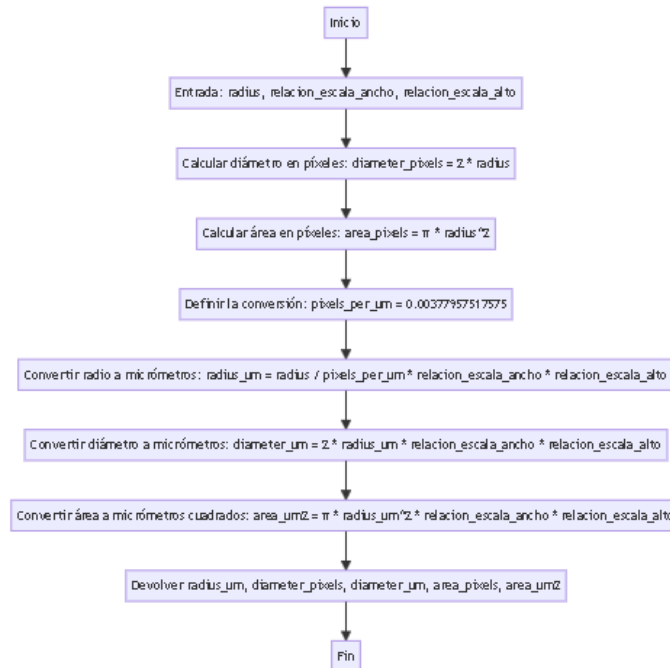


Figura 4.5: Diagrama de flujo del código empleado para la función de la obtención de las dimensiones del borde del corpúsculo polar detectado sobre los ovocitos

4.2. Deep learning

En esta sección del proyecto se muestran los procedimientos y técnicas utilizadas para el desarrollo de los modelos de inteligencia artificial desarrollados para la predicción de la calidad de los ovocitos. Se explica el funcionamiento a nivel de IA del modelo de recorte automático de imagen del ovocito (autocrop) sobre el subapartado 4.2.1, modelo de perceptrón multicapa sobre el subapartado 2.2.1 y de red neuronal convolucional sobre el subapartado 4.2.3.

4.2.1. Autocrop

Como se ha mencionado en capítulos anteriores, el modelo de autocrop proporcionado por el proyecto OVOVITRIF del CVBLab, recorta la imagen original extraída del micros-

copio sobre el área de importancia del ovocito para así eliminar los elementos sobrantes o de nula utilidad para la detección de características que posteriormente se realice para el entrenamiento de los modelos de calidad.

Este modelo se ha desarrollado con las librerías de python, Pytorch [66], OpenCV y Ultralytics [67]. Pytorch es un framework de python para la creación de modelos de DL como reconocimiento y clasificación de imágenes [68] y Ultralytics es una API de python muy potente utilizada para técnicas de visión por ordenador (CV) [69], que en este caso es la encargada de reconocer los bloques importantes de los ovocitos dentro de la imagen. Exactamente reconoce todo aquello que contenga un cambio de color, es por ello que no solo detecte el corpúsculo polar, sino también la zona pelúcida, la corona radiata y más, eliminando así todo lo externo con un color grisáceo.

En la Figura 4.6 se dispone en la parte superior la misma imagen, que es la original de un ovocito cualquiera tomada desde un microscopio en formato jpeg, pero la imagen superior derecha se representa con un rectángulo en rojo la zona detectada del ovocito por el algoritmo para desprestigiar el contenido externo al recuadro. En la zona central inferior queda su imagen cropeada que pasa a ser formato de imagen jpg.

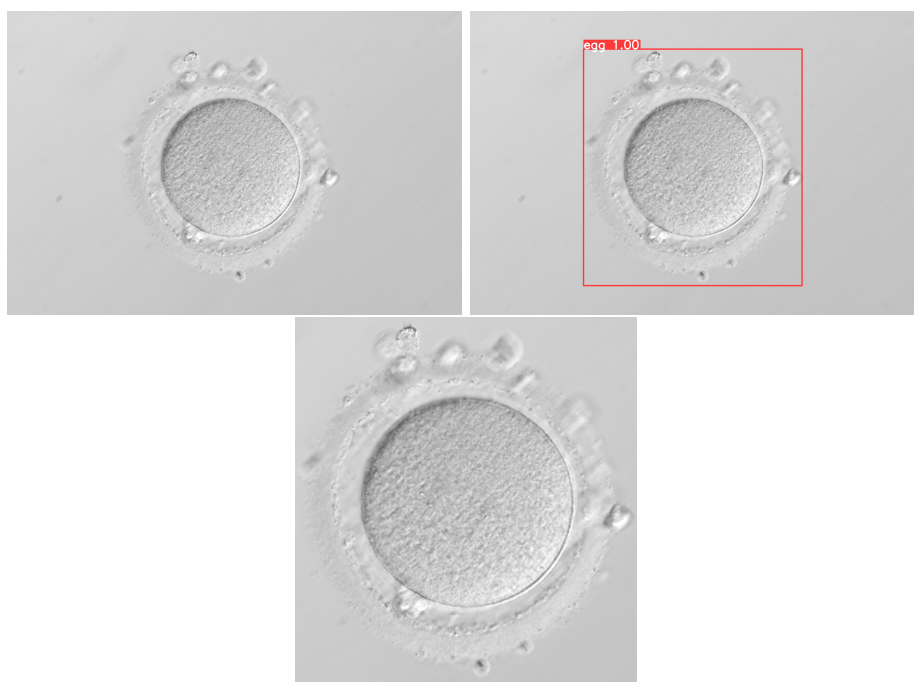


Figura 4.6: Proceso de cropeo de una imagen de ovocito pasada por el modelo de autocrop.

Para el uso del modelo de cropeo, el cual se ejecuta en el servidor debido a la necesidad de potencia hardware, debemos crear microservicios APIrest que atiendan la peticiones de predicción, generando una conexión cliente servidor que explicaremos más adelante en la sección 4.3.

La forma de ejecutar el modelo guardado es como se representa en el diagrama de flujo del código que observamos en la Figura 4.7. Centrándonos en el siguiente comando:

```
comando = f"yolo predict model='model/best.pt'  
↪ source='{app.config['UPLOAD_FOLDER']}' conf=0.85 save_crop=True  
↪ save_txt=True project='pacientes/paciente2' name='crops'"
```

Donde los parámetros de la línea de código están definidos sobre la librería Ultralytics YOLOv8 [70] para realizar predicciones sobre imágenes.

- `yolo predict`: este es el comando principal de YOLOv8 que se utiliza para realizar predicciones sobre imágenes o vídeos.
- `model='model/best.pt'`: especifica la ruta al archivo del modelo YOLOv8 que se utilizará para las predicciones.
- `source='{app.config['UPLOAD_FOLDER']}'`: indica la ubicación de las imágenes sobre las que se realizarán las predicciones.
- `conf=0.85`: establece el umbral de confianza para las predicciones. Solo se mostrarán las detecciones con una confianza superior al 85 %.
- `save_crop=True`: indica que se deben guardar los recortes (crops) de las imágenes que contienen las detecciones.
- `save_txt=True`: Indica que se deben guardar los resultados de las predicciones en archivos de texto.
- `project='pacientes/paciente2'`: especifica la carpeta del proyecto donde se guardarán los resultados.
- `name='crops'`: establece el nombre de la carpeta donde se guardarán los recortes de las imágenes.

En la Figura 4.7 se muestra el diagrama de flujo de la parte principal del código que utiliza la aplicación de autocrop en cuanto recibe las imágenes a procesar, donde lo que devuelve a la aplicación principal es una cadena de caracteres JSON con los datos de cada imagen cropeada.



Figura 4.7: Diagrama de flujo de la función de la aplicación del modelo de autocrop de las imágenes de los ovocitos.

4.2.2. Multilayer perceptron

Como se comentó en el subapartado del marco teórico 2.2.1, una MLP extrae las características de las imágenes para obtener una solución de calidad en predicción mediante el entrenamiento con los datos de calidad proporcionados por los profesionales del IVI. Debido a esto, para realizar un modelo con una MLP es necesario distinguir las características de las imágenes con las etiquetas a clasificar (puntuación). La estructura de carga y disposición de datos se muestran en el diagrama de flujo del código utilizado de la Figura 4.8. Como entrada a la función se le pide el dataframe del documento excel donde se guardan los datos y el conjunto de imágenes utilizado para el entrenamiento y validación, luego se identifica que el set de imágenes de la carpeta coincide con los nombres de las imágenes de la BBDD, seguidamente, se cargan las imágenes en blanco y negro con la librería OpenCV

y se extraen las características de las imágenes con la función "extract_hog_features" (véase diagrama de flujo del código de extracción de características hog de la Figura 4.9), la cual utiliza la función "hog" de la librería de scikit-learn para extracción de características de imágenes.

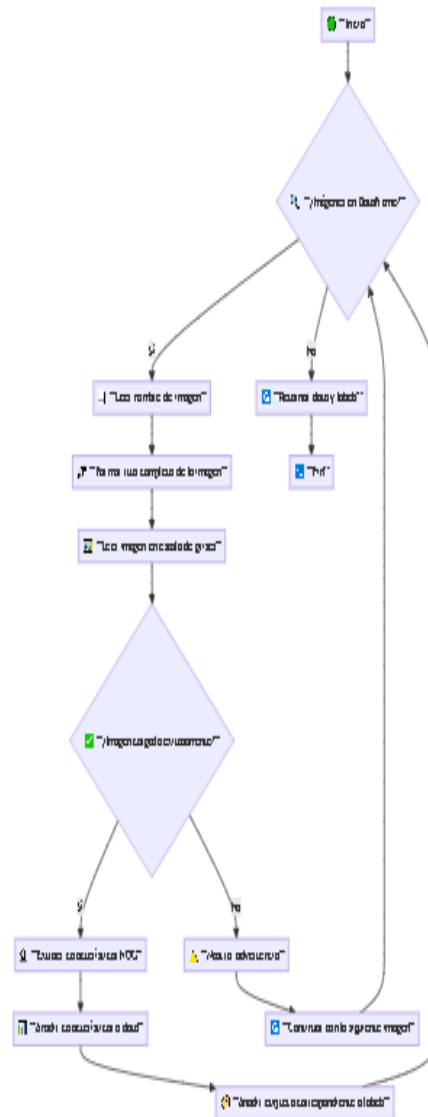


Figura 4.8: Diagrama de flujo del código Python para la carga y preprocesado de imágenes para distinguir entre características de imágenes y etiquetas para su posterior entrenamiento de modelo MLP.

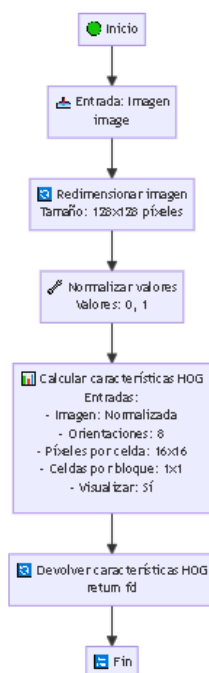


Figura 4.9: Diagrama de flujo del código Python para la extracción de características de las imágenes con la librería scikit-learn para modelo MLP.

Para el entrenamiento del modelo se utiliza un buscador de hiperparámetros de la librería de sklearn como se observa en el código de la Figura 4.11, donde la función "GridSearchCV" selecciona los parámetros de la lista de parámetros (parameter_space) que maximiza la puntuación del score. La función "MLPClassifier" optimiza la función de pérdidas. Previo a la ejecución de la función de entrenamiento del modelo se debe obtener los datos a entrenar y validar, así como las etiquetas del conjunto de entrenamiento y validación mediante la función de la librería de sklearn "train_test_split" como se observa en el código de la Figura 4.10.

```

1 # X_ características de las imágenes, y_ etiquetas de clasificación
2 X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,
  → random_state=42) # Dividir los datos en conjuntos de entrenamiento y prueba
  → (80-20)%
  
```

Figura 4.10: Código de python para la obtención de los datos y etiquetas del conjunto de entrenamiento y validación para el modelo de MLP.

La combinación cruzada de los parámetros en GridSearchCV significa que se prueban todas las combinaciones posibles de los valores de hiperparámetros especificados. Por ejemplo, al definir el espacio de hiperparámetros como hidden_layer_sizes=[(128, 64), (256, 128), (128, 128, 64)], max_iter=[300, 500], alpha=[0.0001, 0.001, 0.01], y learning_rate_init=[0.001, 0.005, 0.01], GridSearchCV generará todas las combinaciones posibles (en este caso, 54 combinaciones). Para cada combinación, el modelo (MLPClassifier) se entrena y evalúa usando validación cruzada con 5 particiones (cv=5). La métrica de rendimiento (f1_weighted)

se calcula para cada partición y se promedia para cada combinación de hiperparámetros. Finalmente, se selecciona la combinación que obtiene la mejor métrica promedio, y el modelo entrenado con esta combinación se considera el mejor modelo. La llamada a 'fit' (función de scikit-learn utilizada al final del código de la Figura 4.11) entrena el modelo para cada combinación de hiperparámetros en el 'parameter_space', utilizando validación cruzada para evaluar el rendimiento de cada combinación.



Figura 4.11: Diagrama de flujo del código de python de la función de entrenamiento del modelo con búsqueda de hiperparámetros para la MLP.

Obtenido el modelo entrenado con el mejor conjunto de hiperparámetros del set propuesto se realiza una evaluación del modelo con la función representada a través del diagrama de flujo de la Figura 4.12. La función consta de la llamada a la función 'predict' de sklearn para predecir con los datos de prueba los resultados y con estos datos crear una matriz de confusión para evaluar el accuracy de la clasificación. También se imprime un

reporte de clasificación de la librería de sklearn al igual que la matriz de confusión, que consiste en mostrar las principales características de la clasificación, como la precisión, el recall y el accuracy. Finalmente, se calcula el factor f1-score [71], que en modelos de multi-clase como en este caso, se calcula como la media de los factores obtenidos para cada caso (1-9), siguiendo la expresión matemática de abajo 4.1. Esta función obtiene un valor que se aproxima a 0 cuando el modelo es malo y a 1 cuando el modelo predice correctamente. Se considera una media armónica de la precisión y el recall.

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (4.1)$$

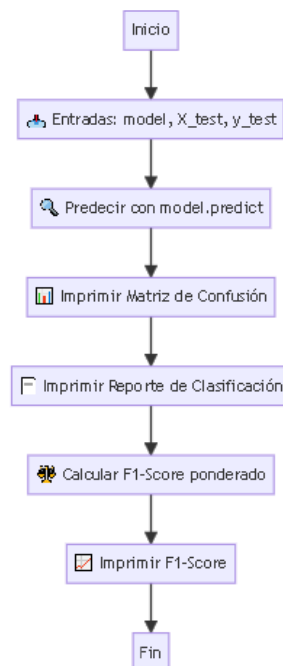


Figura 4.12: Diagrama de flujo del código de python para la evaluación del modelo entrenado con mejores hiperparámetros con una MLP. Se utiliza técnicas de clasificación de modelo como la matriz de confusión, reporte de clasificación y la media armónica de la precisión y recall (f1-score).

Otra función para evaluar el modelo es la de validación cruzada (véase Figura 4.13). La función 'cross_validate_model' realiza una validación cruzada de un modelo de aprendizaje automático, utilizando los datos y las etiquetas proporcionadas. Primero, imprime un mensaje indicando que se está realizando la validación cruzada. Luego, utiliza la función 'cross_val_score' de scikit-learn para llevar a cabo la validación cruzada con 5 particiones (cv=5), evaluando el modelo con la métrica F1 ponderada (scoring='f1_weighted'). 'cross_val_score' divide los datos en 5 pliegues, entrena el modelo con 4 pliegues y lo valida con el pliegue restante, repitiendo este proceso cinco veces para asegurar que cada pliegue se use una vez como conjunto de validación. Los resultados de las métricas F1 ponderadas de cada una de las 5 iteraciones se almacenan en cv_scores. La función luego imprime estos puntajes y calcula el promedio de estos puntajes, proporcionando una estimación

del rendimiento promedio del modelo. Finalmente, se imprime el promedio del F1-Score ponderado, lo cual ofrece una visión general de la capacidad del modelo para generalizar a datos no vistos.

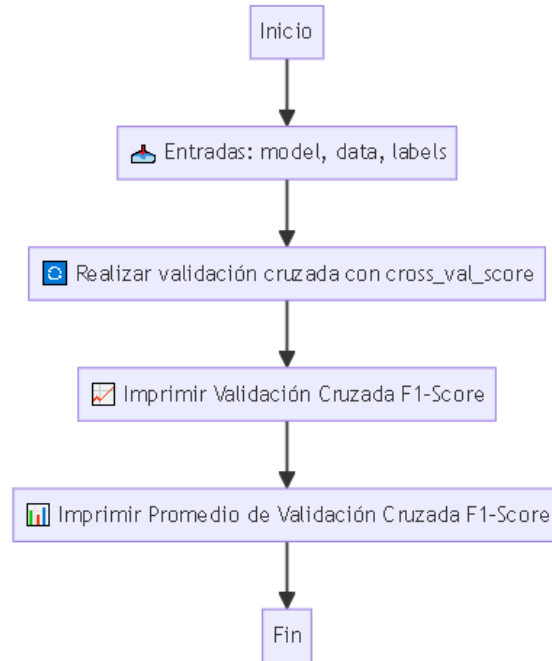


Figura 4.13: Diagrama de flujo del código de python de la implementación de la función para la validación cruzada sobre el modelo de MLP para la clasificación de ovocitos.

Siguiendo este desarrollo de funciones para un modelo de multilayer perceptron (MLP), el flujo de trabajo de la carga de datos de imágenes y etiquetas, entrenamiento y validación del modelo, y guardado del modelo, se plantea en la Figura 4.14. La figura representa la dinámica de izquierda a derecha (sentido de la flecha roja inferior) de la generación del modelo MLP, donde el primer bloque (preprocesado de datos) se encarga de cargar la BBDD donde se encuentran las características morfológicas de los ovocitos y las imágenes de las cuales se extraerán las características "hog" de las mismas, aparte de aquellas características que hayan sido etiquetadas alfa-numéricamente, se pondrán como valores enteros para evitar problemas futuros. El segundo bloque (normalización y división del dataset) representa la normalización de los datos, es decir, se normaliza las características eliminando la media y escalando a la varianza unidad, como así se expresa en la ecuación 4.2, donde la 'u' representa la media y 's' la desviación típica de los datos. Sobre el segundo bloque, se separan los datos y etiquetas de entrenamiento de los de prueba en una relación 80/20. El tercer bloque (entrenamiento del modelo) es una representación de red neuronal MLP que simboliza el entrenamiento del modelo con la búsqueda de hiperparámetros para obtener los mejores resultados dentro de los establecidos. El cuarto bloque (validación completa del modelo) representa las técnicas aplicadas sobre el set de pruebas para la verificación del modelo, como pueden ser la matriz de confusión, reporte de clasificación, validación cruzada y gráfica de la función de pérdidas. Finalmente, se realiza el guardado del modelo.

$$z = \frac{x - \mu}{s} \quad (4.2)$$

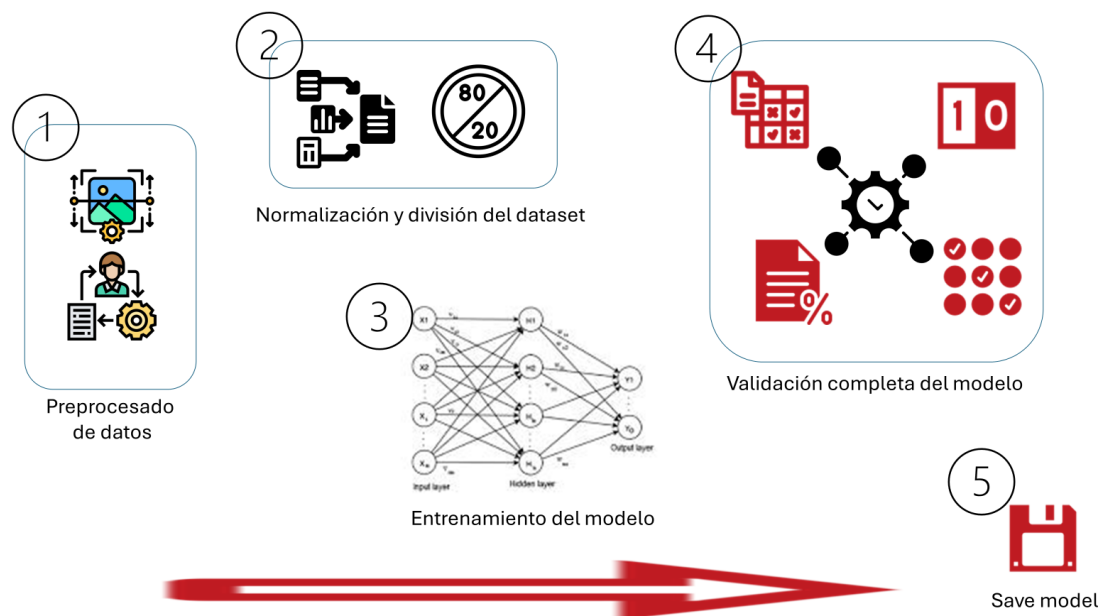


Figura 4.14: Diagrama de bloques del flujo de generación del modelo de MLP desde carga de datos hasta guardado del modelo.

4.2.3. Convolutional neuronal network

Habiéndose comentado en el subapartado 2.2.2 del marco teórico, se desarrolla consistentemente el modelo de predicción de calidad por medio de una red neuronal convolucional. Esta técnica de DL ha llevado a la realización de muchas pruebas para intentar desarrollar un modelo con unas propiedades lo suficientemente fiables como para asegurar que el resultado propuesto por la IA es aceptable. El procedimiento seguido es muy parecido al ya explicado en la sección (4.2.2), por lo que en este caso, la explicación se centra en mayor medida, a la elección de los parámetros de la red neuronal y sus consecuencias.

En la realización de un modelo con una red neuronal convolucional (CNN), se necesita de una preparación previa de los datos que sea consistente en cuanto a las imágenes que se utilizan como en las características de las imágenes anotadas en la BBDD. Como se indica en la EITC [72], la preparación de los datos previo al entrenamiento de la red, conlleva una serie de pasos como, la recopilación, el preprocesamiento, el aumento y la división de los datos.

El primer paso (recopilación de los datos), consiste en obtener un conjunto de imágenes y anotaciones visuales de las imágenes, que sean lo suficientemente grande para cubrir las categorías que se desea clasificar. El segundo paso (preprocesamiento de los datos), consiste en pasar los datos (imágenes y características) en crudo a un formato adecuado para la red. Las imágenes que suelen tener diferente tamaño, o tamaño grande, se realiza

un redimensionamiento para que todas las imágenes de la red tengan el mismo tamaño (usualmente 128x128 ó 224x224), y para las anotaciones, se han de procesar en formato numérico y no categórico para evitar problemas con signos extraños que no detecte el código. El tercer paso (aumento de los datos), se utiliza para conseguir un conjunto de datos mayor, como así indica el nombre, el cual se realiza mediante transformación de rotación, translación, cambios de tono en el color en las imágenes, manteniendo las mismas características para cada imagen. Con esto conseguimos duplicar el tamaño del set de datos para cada transformación. Por último, la división de los datos, consiste en utilizar un porcentaje elevado de los datos para el entrenamiento y el restante para la validación del modelo en sí. También se puede dejar una parte menor de los datos para realizar prueba, es decir, ya habiendo validado el modelo con los algoritmos y técnicas necesarias, se procede a la realización de pruebas, probando imagines del conjunto de pruebas (otro % a separado del de validación) para obtener una calificación del modelo y ver como se comporta en vivo.

En la Figura 4.15 se muestra la arquitectura general de cualquier CNN, la entrada está correspondida por una serie de imágenes o datos, dependiendo de si el propio modelo debe clasificar con la única información extraídas de las imágenes, o la entrada está acompañada de anotaciones propuestas por especialistas sobre el contenido de las imágenes, para que sea más sencillo el proceso de clasificación. Las siguientes capas son las de convolución y pooling, las cuales se pueden iterar varias veces dependiendo del trabajo de extracción de características que se necesite para obtener un buen modelo de predicción. Las capas finales, en las redes CNN suelen ser siempre las FC en las que se consigue conectar toda la información extraída por las capas previas.

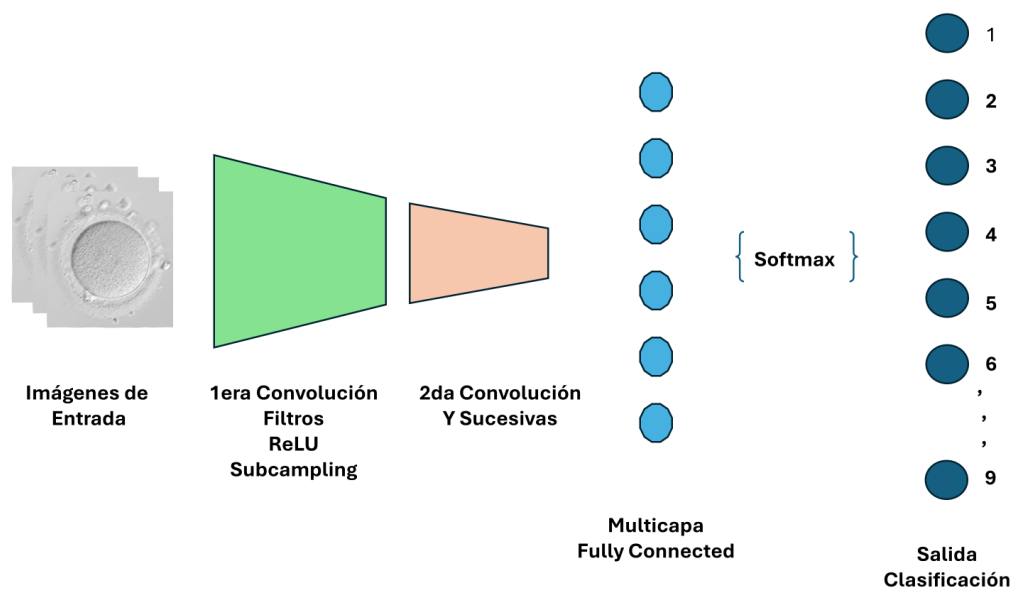


Figura 4.15: Arquitectura general de redes neuronales convolucionales - CNN.

No obstante, hay más allá de las capas y funciones que se representan sobre la Figura 4.15. Se usan funciones de normalización y regularización para evitar o solucionar sobre

ajustes del modelo. Es por ello que se detallan 4 modelos utilizados para entrenar la red y así escoger el que mejor porcentaje de acierto consiga.

La primera etapa de la generación del código para crear el modelo predictivo coincide con la del MLP, que es la carga y preprocesado de los datos a utilizar para entrenar el modelo, como también se muestra en el primer bloque de la Figura 4.14. Para el caso de una CNN, también se deben cargar las imágenes, ya que a diferencia de una MLP, se utilizan para entrenar los modelos porque se extraen características de las propias imágenes, como se explica anteriormente en la sección 2.2.2. Sobre el diagrama de flujo del código para la carga y preprocesado de los datos de la Figura 4.16, se detalla el proceso de carga y preprocesado de los datos de la BBDD.

Las características ovocitarias escogidas para entrenar los modelos son cuatro de todas las posibles vistas en previos capítulos, que son si el ovocito es normal o no, cantidad de granulosidad, si la ZP es elongada o no, y si la ZP es oscura o no respectivamente. Todas estas características son representadas en la Tabla 3.1. Estas características han sido elegidas finalmente, debido a la prueba de varias combinaciones y observando que la fiabilidad de modelo obtenida (aún siendo baja), es mayor que teniendo mayor número de características ovocitarias u otras diferentes. Esto es debido a que tres de las cuatro elegidas solo tienen dos subcaracterísticas, siendo la única con más de dos, la característica de granulosidad.

En el preprocesado de datos (Véase Figura 4.16), es importante asegurarse que los valores de las etiquetas de entrada al modelo no produzcan conflictos, por lo que se convierten valores categóricos a numéricos. El siguiente paso, es estandarizar las características con media nula y varianza unidad, y pasar a variable de tipo tensor los valores de las etiquetas, que son las calificaciones o puntuaciones de los ovocitos de la BBDD por parte de los embriólogos. La última parte de la función representada por el diagrama de flujo de la Figura 4.16, es la de listar las características y etiquetas en el *dataframe* y devolverlo junto a las etiquetas codificadas, que son las correladas con las características.

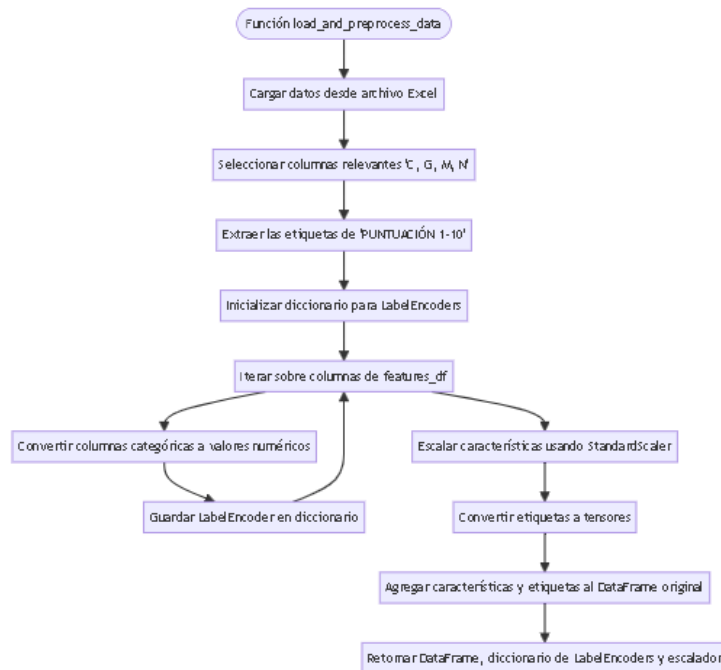


Figura 4.16: Diagrama de flujo de la función de python para la carga y pre-procesado de los datos para el entrenamiento de los modelos de predicción de calidad de ovocitos.

Obtenido el *dataframe* se puede realizar la creación del *dataloader*, que es la carga de los datos para obtener el conjunto de entrenamiento y el conjunto de prueba o validación del modelo. Sobre la Figura 4.17 se representa el diagrama de flujo del código de la función para la creación de este. La función depende de tres entradas, el *dataframe*, el directorio de las imágenes, y las transformaciones de las imágenes que se quieran hacer, que en este caso, se ha utilizado un cambio de tamaño de imagen a (224, 224), un volteo horizontal aleatorio, y un giro aleatorio dentro de un ángulo en grados de [-10, 10].

El primer paso de la función es crear con estas entradas el *dataset* personalizado con la clase *OocyteDataset*, explicado sobre el diagrama de la Figura 4.18. El *dataset* obtenido se filtra para evitar posibles valores "None", y se divide en el conjunto de entrenamiento y prueba en 90/10, es decir, el 90 % de los datos válidos totales son para el entrenamiento del modelo, y el 10 % restante para el conjunto de testeo. Del conjunto de entrenamiento, se calculan los pesos de las muestras a raíz de el número de ocurrencias de las etiquetas de cada característica, y se genera posteriormente los *dataloader* de entrenamiento y prueba.

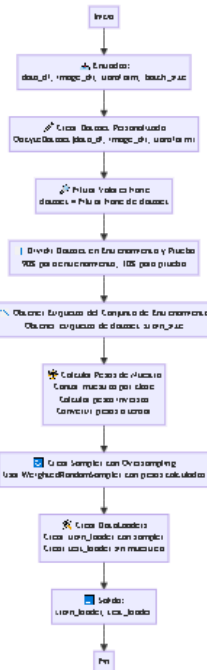


Figura 4.17: Diagrama de flujo de la función de python para la creación del dataloader para el entrenamiento y prueba de los modelos CNN.

La función de la clase del *dataset* personalizado se explica en el diagrama de la Figura 4.18. Principalmente, la clase está diseñada para concluir que en el directorio de las imágenes sean correctas, aplicarles las transformaciones declaradas si es que se declara el transformador y conjuntarlas con las etiquetas y características de la BBDD de forma correcta. Esta clase, devuelve una "tupla" que contiene la imagen, las características (como un tensor) y la etiqueta.



Figura 4.18: Diagrama de flujo del Dataset personalizado para predicción de calidad ovocitaria.

Una vez creados los *dataloaders* se pueden entrenar los modelos que se deseen, ya que los conjuntos de datos de entrenamiento y prueba están definidos y cargados. Como se pretende evaluar varios modelos para guardar el mejor posible dentro de las condiciones existentes, se realiza de forma similar al método de búsqueda de hiperparámetros utilizado para el MLP (explicado en la sección 4.2.2), pero para cuatro modelos con especificaciones diferentes.

El código realiza un proceso de entrenamiento y validación de varios modelos de predicción ajustando los hiperparámetros mediante una búsqueda en cuadrícula y aplicando técnicas para manejar la desproporción de clases en los datos. Primero, se calculan los pesos de clase basados en la distribución de las etiquetas del conjunto de entrenamiento. Esto se logra utilizando la función `torch.bincount()` para contar cuántos ejemplos hay de cada clase, y luego asignando a cada clase un peso inversamente proporcional a su frecuencia:

$$\text{class_weights}[i] = \frac{1}{\text{class_counts}[i]} \quad (4.3)$$

Los pesos se normalizan dividiéndolos por la suma total de los pesos:

$$\text{class_weights} = \frac{\text{class_weights}}{\sum \text{class_weights}} \quad (4.4)$$

Esto asegura que los pesos sumen la unidad, lo cual se verifica con la función `class_weights.sum()`.

Una vez calculados los pesos de clase, se definen cuatro modelos (`Model1`, `Model2`,

Model3, Model4) para entrenar y probar. La búsqueda de hiperparámetros se realiza con la clase `ParameterGrid`, que genera combinaciones de los siguientes valores para los hiperparámetros:

- **Tasa de aprendizaje (lr):** Puede tomar los valores 0,001 y 0,0001.
- **Tamaño del lote (batch_size):** Puede ser 32 o 64.
- **Optimizador (optimizer):** Las opciones son Adam y SGD.
- **Coefficiente de regularización (weight_decay):** Puede ser 0,0 o 0,001.

Para cada combinación de hiperparámetros, se crean optimizadores específicos para cada modelo, utilizando la clase correspondiente (ya sea Adam o SGD) y ajustando la tasa de aprendizaje y el `weight_decay` proporcionado. Además, se definen las funciones de pérdida (`nn.CrossEntropyLoss()`) que incluyen los pesos de clase para penalizar más fuertemente los errores en las clases menos representadas.

El proceso de entrenamiento se realiza sobre los datos, dividiendo el conjunto en lotes, y aplicando oversampling a las clases minoritarias. Los datos se cargan mediante los `DataLoaders` correspondientes, y los modelos se entrenan en función de los hiperparámetros actuales. Los resultados del entrenamiento, incluyendo métricas de evaluación como la precisión, se almacenan para comparar el rendimiento de los modelos y seleccionar el conjunto de hiperparámetros que optimice el rendimiento del modelo.

Al final, se obtiene el modelo más adecuado junto con los mejores hiperparámetros, que son evaluados en función de las etiquetas verdaderas del conjunto de prueba.

A continuación se detallan las especificaciones de los modelos utilizados para el entrenamiento de la red y la obtención del modelo utilizado.

El `Model11`, cuyo diagrama se presenta sobre la Figura 4.19, es una red neuronal convolucional diseñada para procesar imágenes en escala de grises de tamaño 224×224 píxeles. La arquitectura consta de varias capas convolucionales, de normalización por lotes y funciones de activación, que se combinan para extraer características relevantes de las imágenes de entrada. En particular, se utilizan tres capas convolucionales seguidas de capas de normalización por lotes y activación ReLU.

La primera capa convolucional (`Conv1`) aplica 32 filtros de tamaño 3×3 sobre la imagen de entrada de un solo canal, manteniendo el tamaño espacial mediante un `padding` de 1. La salida de esta capa es de tamaño $(32 \times 224 \times 224)$ y pasa por una capa de normalización por lotes (`BatchNorm2D(32)`), seguida de la activación ReLU y una operación de *Max Pooling* que reduce la resolución espacial a $(32 \times 112 \times 112)$.

La segunda capa convolucional (`Conv2`) aplica 64 filtros de 3×3 , manteniendo el tamaño espacial de entrada. Posteriormente, la salida de esta capa de tamaño $(64 \times 112 \times 112)$ también se normaliza mediante `BatchNorm2D(64)` y se pasa por ReLU antes de aplicar nuevamente *Max Pooling*, lo que reduce la resolución a $(64 \times 56 \times 56)$.

La tercera capa convolucional (`Conv3`) funciona de manera similar, aplicando 128 filtros de 3×3 y produciendo una salida de tamaño $(128 \times 56 \times 56)$. Esta salida es normalizada

con `BatchNorm2D(128)` y se activa con `ReLU`, antes de reducir su tamaño espacial a $(128 \times 28 \times 28)$ mediante *Max Pooling*.

A continuación, la salida de la última capa convolucional se aplanar, convirtiéndose en un vector de características de tamaño $128 \times 28 \times 28$, que se concatena con un conjunto adicional de 4 características no visuales. Este vector combinado se pasa a través de una capa lineal (`FC1`) que reduce el número de neuronas a 512. Para prevenir el sobreajuste, se utiliza *Dropout* con una tasa del 25 % después de la primera capa lineal.

La salida de `FC1` pasa nuevamente por una función de activación `ReLU` y se dirige a una segunda capa lineal (`FC2`), que reduce la dimensionalidad a 10 unidades, correspondientes a las clases de salida del modelo. Finalmente, la red aplica la función `Softmax`, que convierte los valores de salida en probabilidades normalizadas entre 0 y 1 para cada clase, permitiendo así la clasificación.

Matemáticamente, las capas convolucionales se pueden expresar como:

$$\text{Conv1} = \text{Conv2D}(1, 32, \text{kernel_size} = 3, \text{stride} = 1, \text{padding} = 1) \quad (4.5)$$

$$\text{Conv2} = \text{Conv2D}(32, 64, \text{kernel_size} = 3, \text{stride} = 1, \text{padding} = 1) \quad (4.6)$$

$$\text{Conv3} = \text{Conv2D}(64, 128, \text{kernel_size} = 3, \text{stride} = 1, \text{padding} = 1) \quad (4.7)$$

Las capas de normalización son:

$$\text{BN}_i = \text{BatchNorm2D}(n_i) \quad (4.8)$$

donde n_i es el número de canales en la capa correspondiente. La salida se aplanará y concatenará con las características adicionales, antes de ser procesada por las capas completamente conectadas.

$$\text{FC1} = \text{Linear}(\text{flattened_size} + 4, 512) \quad (4.9)$$

$$\text{FC2} = \text{Linear}(512, 10) \quad (4.10)$$

Finalmente, la función `Softmax` se utiliza para la clasificación:

$$\text{Softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (4.11)$$

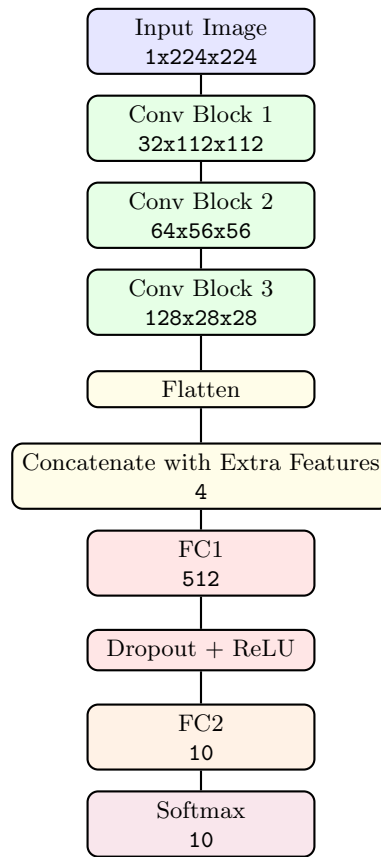


Figura 4.19: Diagrama de bloques del Modelo 1

El modelo Model2, cuyo diagrama se presenta sobre la Figura 4.20, se basa en la arquitectura de ResNet-18, la cual se adapta para trabajar con imágenes de un solo canal y características adicionales. La estructura del modelo se detalla a continuación:

- **Capa Convolutiva Inicial de ResNet-18:**
 - Se modifica la primera capa convolutiva para aceptar imágenes con un solo canal.
 - **Configuración:** `nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)`
- **Capa Final de ResNet:**
 - La capa completamente conectada original se elimina y se reemplaza por una capa de identidad.
 - **Configuración:** `nn.Identity()`
- **Nueva Capa Totalmente Conectada:**
 - Después de extraer características con ResNet, se concatenan características adicionales a la salida.

- **Configuración:** `nn.Linear(num_ftrs + num_extra_features, 10)`, donde `num_ftrs` es el número de características de salida de ResNet y `num_extra_features` es el número de características adicionales.

Proceso de Forward:

- Las imágenes pasan a través de la ResNet modificada.
- Las características adicionales se concatenan con la salida de la red ResNet.
- La salida concatenada se pasa a través de la nueva capa totalmente conectada para obtener la predicción final.

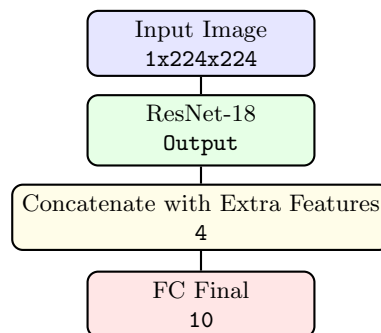


Figura 4.20: Diagrama de bloques del Modelo 2

El modelo `Model3`, cuyo diagrama se presenta sobre la Figura 4.21, es una red neuronal convolucional que combina capas convolucionales con capas completamente conectadas para realizar tareas de clasificación. A continuación se detalla su estructura:

- **Capas Convolucionales:**
 - **Primera Capa Convolucional:**
 - `nn.Conv2d(1, 16, kernel_size=5, padding=2)`
 - Esta capa aplica 16 filtros de tamaño 5x5 a una imagen de entrada en escala de grises, manteniendo el tamaño de la imagen mediante un padding de 2 píxeles.
 - **Normalización por Lotes:**
 - `nn.BatchNorm2d(16)`
 - Normaliza las salidas de la primera capa convolucional para mejorar la estabilidad del entrenamiento.
 - **Función de Activación ELU:**
 - `nn.ELU()`
 - Introduce no linealidad en el modelo utilizando la función Exponential Linear Unit (ELU).
 - **Max Pooling:**

- `nn.MaxPool2d(2, 2)`
 - Reduce las dimensiones espaciales de la imagen, manteniendo solo las características más relevantes con una ventana de 2x2.
 - **Segunda Capa Convolutiva:**
 - `nn.Conv2d(16, 32, kernel_size=5, padding=2)`
 - Aplica 32 filtros de tamaño 5x5 a la salida de la primera capa convolutiva.
 - **Normalización por Lotes:**
 - `nn.BatchNorm2d(32)`
 - Normaliza las salidas de la segunda capa convolutiva.
 - **Función de Activación ELU:**
 - `nn.ELU()`
 - Aplica la función ELU nuevamente para introducir no linealidad.
 - **Max Pooling:**
 - `nn.MaxPool2d(2, 2)`
 - Realiza un pooling adicional para reducir aún más las dimensiones espaciales.
 - **Capas Completamente Conectadas:**
 - **Flatten:**
 - `nn.Flatten()`
 - Aplana la salida de las capas convolutivas para ser alimentada a las capas lineales.
 - **Primera Capa Lineal:**
 - `nn.Linear(flattened_size + 4, 256)`
 - Toma la salida aplanada de las capas convolutivas y la concatena con 4 características adicionales, mapeando el resultado a un espacio de 256 dimensiones.
 - **Función de Activación ELU:**
 - `nn.ELU()`
 - Aplica la función ELU a la salida de la primera capa lineal.
 - **Segunda Capa Lineal:**
 - `nn.Linear(256, 10)`
 - Mapea la salida de 256 dimensiones a 10 dimensiones, correspondiente al número de clases.
 - **Proceso de Forward:**
 - Las imágenes pasan a través de las capas convolutivas.
 - La salida de las capas convolutivas se aplanan y se concatena con las características adicionales.
 - La salida concatenada se pasa a través de las capas completamente conectadas para obtener la predicción final.
-

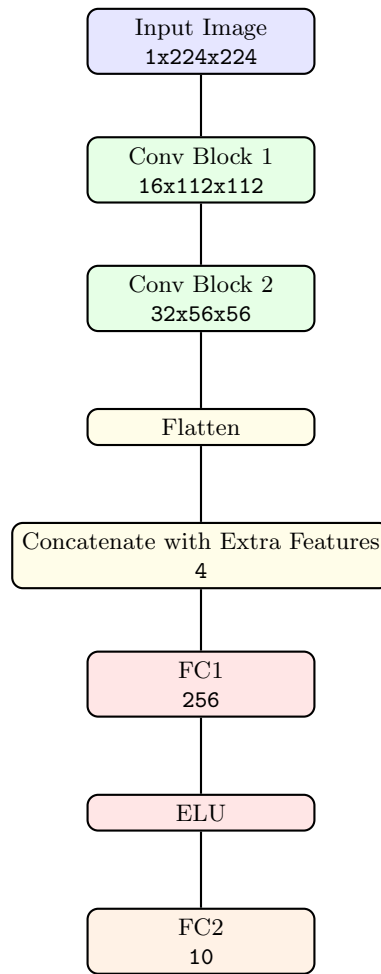


Figura 4.21: Diagrama de bloques del Modelo 3

El modelo Model14, cuyo diagrama se presenta sobre la Figura 4.22, emplea un Vision Transformer (ViT) para la clasificación de imágenes. A continuación se describe en detalle su estructura:

- **Vision Transformer (ViT):**
 - **Configuración del ViT:**
 - **Clase ViTConfig:**
 - ◊ Define los parámetros clave del modelo ViT, como el número de etiquetas (clases), el tamaño de los parches, el número de canales de entrada y el tamaño de la capa oculta.
 - **Número de Canales de Entrada:**
 - ◊ Ajustado a 1 canal para trabajar con imágenes en escala de grises.
 - **Modelo ViT:**
 - **Clase ViTModel:**
 - ◊ Implementa el Vision Transformer utilizando la configuración de ViTConfig.

- ◊ **Proyección de Parches:**
 - ◊ `nn.Conv2d(1, configuration.hidden_size, kernel_size=configuration.patch_size, stride=configuration.patch_size)`
 - ◊ Ajusta la proyección de parches para un canal de entrada.
- **Procesamiento de Características Adicionales:**
 - **Capa Lineal:**
 - `nn.Linear(num_extra_features, configuration.hidden_size)`
 - Procesa las 4 características adicionales y las mapea al tamaño de la capa oculta del ViT.
- **Clasificación Final:**
 - **Capa Lineal de Clasificación:**
 - `nn.Linear(configuration.hidden_size, num_classes)`
 - Realiza la clasificación final basándose en la salida combinada del ViT y las características adicionales.
- **Proceso de Forward:**
 - Las imágenes se procesan a través del Vision Transformer para obtener la representación del primer token (CLS).
 - Las características adicionales se procesan mediante una capa lineal.
 - La representación de la imagen y las características adicionales se combinan mediante una suma.
 - La salida combinada se pasa a través de la capa de clasificación final para obtener la predicción.

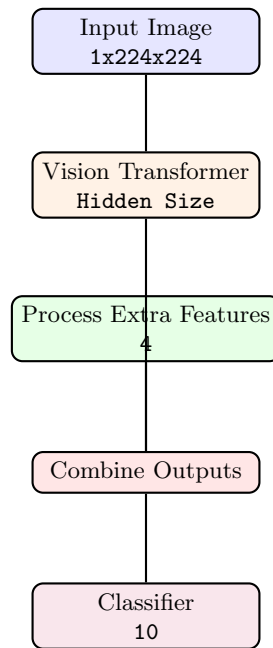


Figura 4.22: Diagrama de bloques del Modelo 4

4.3. Desarrollo web

El desarrollo de la aplicación web ha sido el primer paso para la realización del proyecto, el cual se centra en la programación vía python de rutas url accesibles desde un navegador web como Chrome, en donde se muestra un contenido específico en relación al proyecto. Esta aplicación web pretende establecer dos vistas, una donde se pueda buscar y cargar las imágenes de la cohorte a analizar por los algoritmos, de detección de bordes y modelos predictivos de calidad de ovocitos, y una segunda página donde se observe el contenido y resultado de todas las imágenes cropeadas por el modelo de autocrop, y los datos de las dimensiones de los ovocitos junto a las predicciones resultantes por cada ovocito. Finalmente, se podrá generar un documento en formato pdf sobre estos mismos resultados.

Para el desarrollo del frontend de la aplicación web se utiliza una hoja de estilos denominado bootstrap en la versión 5.3 [73]. Es altamente utilizado para la generación de vistas web debido a la cantidad de ejemplos y utilidades que proporciona este kit, además de la cantidad de información en blogs que puedes encontrar sobre él.

4.3.1. Landing page

Esta primera vista es bastante simple, debido a que únicamente necesitará de una interfaz sencilla en la que aparezcan los logos, tanto de CVBLab, como de IVI, que aparezca un encabezado con el nombre de la aplicación en sí, un botón que enlace con la siguiente página web, en la que se muestre el contenido, y lo más importante, un buscador de archivos en local para que los embriólogos puedan buscar los archivos a cargar.

El planteamiento inicial de la aplicación tiene la apariencia de la Figura 4.30, donde se observa en la parte superior de la web una barra de selección (navbar) con botones de tema oscuro realizada sobre código de etiqueta HTML proporcionado por bootstrap [74] en el template "base.html" para que este navbar permanezca en cada página que se vaya a generar.

El siguiente contenido de la página web se localiza en otro template diferente denominado "home.html", donde se sitúa el buscador de archivos, el gestor de carpetas y el botón de siguiente vista, como se ha detallado en la Figura 4.30. Este código, mostrado en la Figura 4.24 permite mediante bootstrap [75] la búsqueda y carga de archivos en formato "jpeg" múltiple mediante el tipo de archivo de entrada "file". La etiqueta:

```
<form action="/upload" method="post" enctype="multipart/form-data">
```

realiza la subida de archivos múltiples seleccionados sobre la función declarada en el código main de la aplicación con dirección '/upload' representado en la Figura 4.25.

Este template mediante la herramienta de *forms* de bootstrap [75], realiza la búsqueda y carga de archivos en formato "jpeg" múltiple, mediante el tipo de archivo de entrada "file". La etiqueta form, con acción sobre la ruta "/upload" creada en el documento python de la aplicación para la subida de archivos múltiples.histo

```

1      <nav class="navbar navbar-expand-md navbar-dark bg-dark">
2          <div class="container">
3              <a class="navbar-brand" href="/">CVBLab-IVI-Embriology</a>
4              <button class="navbar-toggler" type="button"
5                  → data-toggle="collapse" data-target="#navbarNav"
6                  → aria-controls="navbarNav" aria-expanded="false"
7                  → aria-label="Toggle navigation">
8                  <span class="navbar-toggler-icon"></span>
9                  </button>
10             <div class="collapse navbar-collapse" id="navbarNav">
11                 <ul class="navbar-nav ml-auto">
12                     <li class="nav-item">
13                         <a class="nav-link" href="/">Home</a>
14                     </li>
15                 </ul>
16             </div>
17         </div>
18     </nav>

```

Figura 4.23: Código HTML para generar el navbar de la aplicación web

```
1     <div class="row">
2     <div class="col">
3         <form action="/upload" method="post" enctype="multipart/form-data">
4             <div class="form-group row">
5                 <label for="image_file" class="col-sm-2
6                 ↪ col-form-label">Upload images</label>
7                 <div class="col-sm-10">
8                     <input type="file" class="form-control-file" id="file"
9                     ↪ name="file">
10                </div>
11            </div>
12            <div class="form-group row">
13                <div class="col-sm-10">
14                    <button type="submit" class="btn
15                    ↪ btn-primary">Procesar</button>
16                </div>
17            </div>
18        </form>
19    </div>
20 </div>
```

Figura 4.24: Código HTML para generar el buscador de archivos y su posterior carga de la aplicación web.

```
1 @app.route('/upload', methods=['POST'])
2 def upload():
3     if 'file' not in request.files:
4         flash('No file part')
5         return redirect(request.url)
6     files = request.files.getlist('file')
7     imagenes_crudas = []
8     for file in files:
9         if file and allowed_file(file.filename):
10            filename = secure_filename(file.filename)
11            if not os.path.exists(app.config['UPLOAD_FOLDER']):
12                os.makedirs(app.config['UPLOAD_FOLDER'])
13            ruta = os.path.join(app.config['UPLOAD_FOLDER'], filename)
14            file.save(ruta)
15            imagenes_crudas.append(ruta)
16            # Obtenemos la imagen a enviar al servidor activo para procesarla
17            croppedimgs = send_image(imagenes_crudas)
18            for grupo in croppedimgs:
19                for (name, datos_img_b64) in grupo:
20                    paciente_dir = os.path.join('pacientes', 'paciente2')
21                    os.makedirs(paciente_dir, exist_ok=True)
22                    with open(os.path.join(paciente_dir, name), "wb") as file:
23                        datos_img = base64.b64decode(datos_img_b64.encode())
24                        file.write(datos_img)
25            return redirect('/')
```

Figura 4.25: Código python con Flask para generación de carga y guarda de las imágenes seleccionadas. Además se implementa el envío de imágenes al servidor para su posterior cropeo (recorte de imagen).

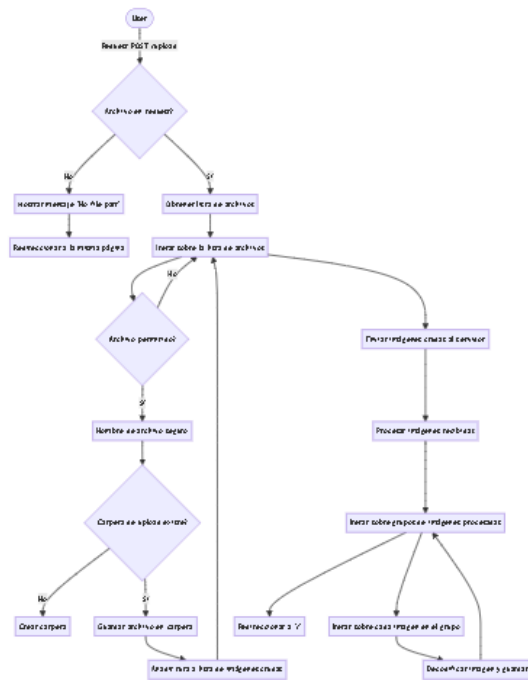


Figura 4.26: Enter Caption

En el código de la Figura 4.25 se desarrolla la ruta de la función de carga de archivos en los directorios pertinentes. Primero se asegura que los archivos a cargar son del formato requerido, luego se cargan los archivos de local a un directorio dedicado para ello, para luego mediante la función creada para enviar las imágenes al puerto 5001 y que se auto-cropeen (véase código de la Figura 4.27). Finalmente, estas imágenes enviadas se reciben como salida de la misma función, guardándose en otro directorio dedicado para ello y su posterior visualización en la página web siguiente.

Una vez cargadas y procesadas las imágenes recortadas, se deben de enviar nuevamente mediante un enlace APIrest para la detección de bordes, cálculo de dimensiones y obtención de la predicción de calidad de los ovocitos, lo cual se explica en los subapartados 4.3.4 y 4.3.5 respectivamente, los cuales para enviarse como imágenes se utiliza la misma función mostrada en el código de la Figura 4.27. El código utilizado para la implementación del botón, que enlaza con la siguiente página web, donde se visualizan los resultados concluyentes del trabajo, se implementa gracias a la documentación de bootstrap sobre botones [76], se ha diseñado un botón con un icono especial para hacer más fluida la interfaz de la página web mediante el código en html representado en la Figura 4.29.

```
1 def send_image(file_list):
2     # Ruta del servidor Docker
3     url = 'http://autocrop:5001' # Cambiar por la dirección del servidor Docker
4     result = []
5     # Ruta de la imagen local:
6     ↪ C:\Users\Carlos\Desktop\Fundamentos_Flask\tienda.io\pacientes\paciente1\test1.jpeg
7     # Envío de la imagen
8     # Abrimos un bucle for para enviar todas las imágenes de un directorio a la
9     ↪ vez (si es necesario)
10    for image in file_list:
11        with open(image, 'rb') as img:
12            files = {'file': img}
13            response = requests.post(url, files=files)
14            # Obtenemos la última parte de la ruta, es decir, el nombre del
15            ↪ archivo
16            image_name = os.path.basename(image)
17            # La respuesta va a ser un array, con tuples. El primer elemento es
18            ↪ el nombre del archivo
19            # y el segundo elemento los bytes de la imagen
20            # Es un array, de parejas, cuyo primer elemento es el nombre del
21            ↪ archivo, y el segundo la imagen en B64
22            respuesta = response.json()
23            result.append(respuesta)
24
25    # Devolvemos la respuesta del servidor
26    return result
```

Figura 4.27: Función "send_image" para realizar un APIrest con el servidor sobre el puerto 5001 en el que está situada la aplicación del modelo de autocrop.

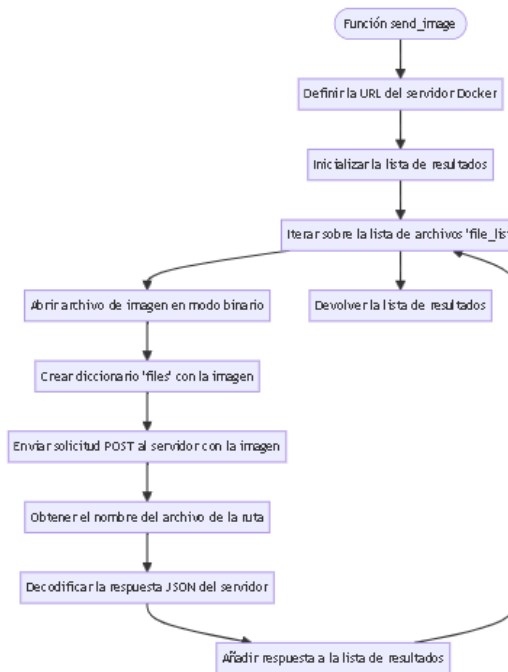


Figura 4.28: Send Image

```

1 <div class="col m-auto">
2   <button type="button" id="nextPage2"
   ↪ onclick="window.location=`${window.location.toString()}/ analisis2`"
   ↪ class="mx-4 border-5" style="background-color: #008CBA; font-weight:
   ↪ bold; float: inline-end">
3     <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16"
   ↪ fill="currentColor" class="bi bi-skip-forward-fill" viewBox="0 0
   ↪ 16 16">
4       <path d="M15.5 3.5a.5.5 0 0 1 .5.5v8a.5.5 0 0 1-1 0V8.753l-6.267
   ↪ 3.636c-.54.313-1.233-.066-1.233-.697v-2.941-6.267 3.636C.693
   ↪ 12.703 0 12.324 0 11.693V4.308c0-.63.693-1.01 1.233-.696L7.5
   ↪ 7.248v-2.94c0-.63.693-1.01 1.233-.696L15 7.248V4a.5.5 0 0 1
   ↪ .5-.5"/>
5     </svg>
6   </button>
7 </div>

```

Figura 4.29: Código HTML para el desarrollo del botón que enlace con la siguiente página web de forma fluida.

En la Figura superior (Figura 4.29), se construye sobre un contenedor un objeto de tipo botón, en el cual, contiene un hipervínculo que redirecciona a a la siguiente página.

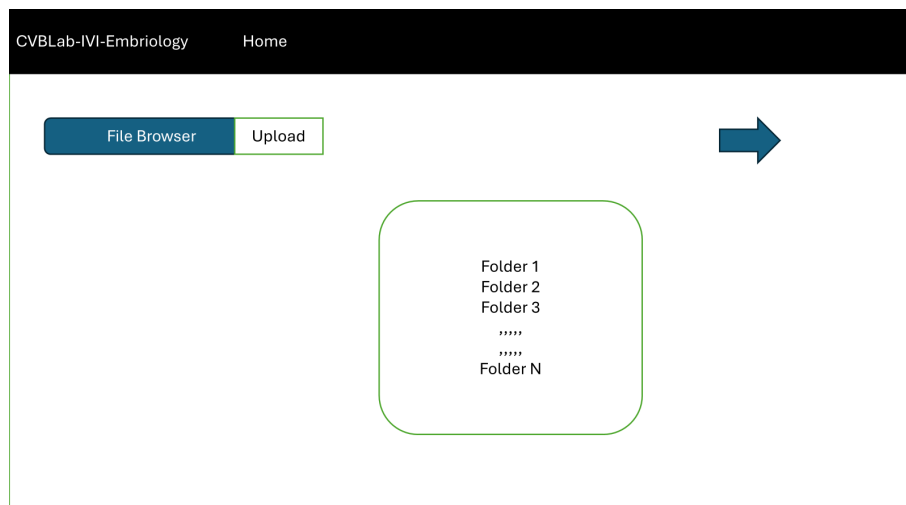


Figura 4.30: Planteamiento de la interfaz la página web inicial.

El flujo de funcionamiento de la página web es el que se representa en el diagrama de bloques de la Figura 4.31. El flujo de trabajo es tal que, accediendo a la dirección de la página web a través de un buscador de internet se visualiza dicha interfaz, donde se accede a un contenido activo de búsqueda de imágenes para realizar el análisis predictivo de calidad de ovocitos. Ya seleccionada la cohorte de ovocitos de la paciente, se cargan las imágenes y se produce los procesos de tratamiento de imágenes mencionados, autocrop (visto en subapartado 4.2.1 y véase Figura 4.6), cálculo de dimensiones (visto en sección 4.1 y véase Figura 4.3.4) y obtención de la calidad de los ovocitos por el modelo entrenado.

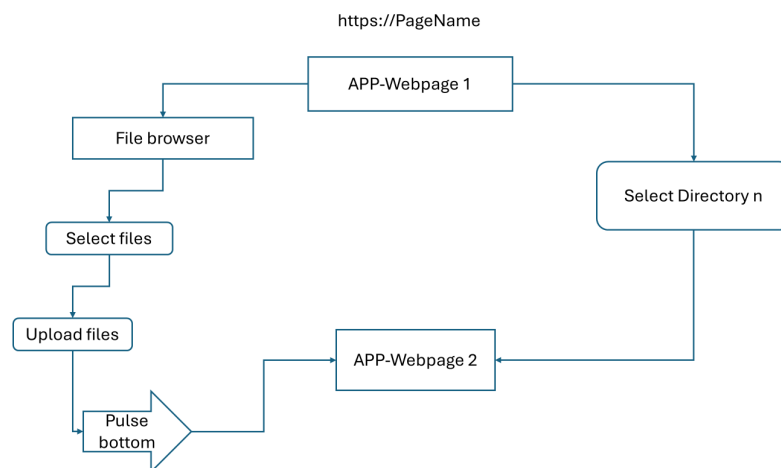


Figura 4.31: Diagrama de flujo del funcionamiento de la página web inicial.

4.3.2. Página web de resultados

Figura 4.32

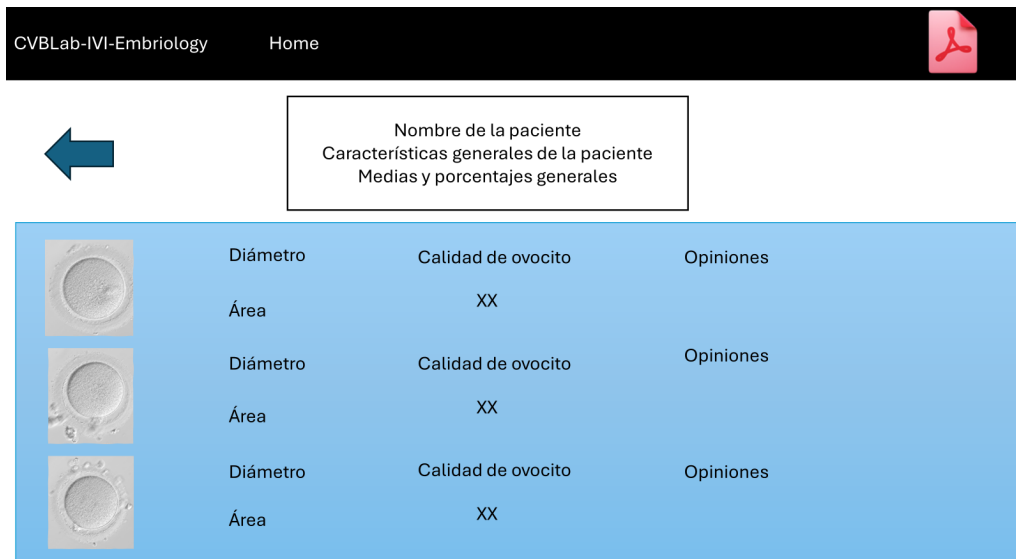


Figura 4.32: Planteamiento de la interfaz de la página web de resultados.

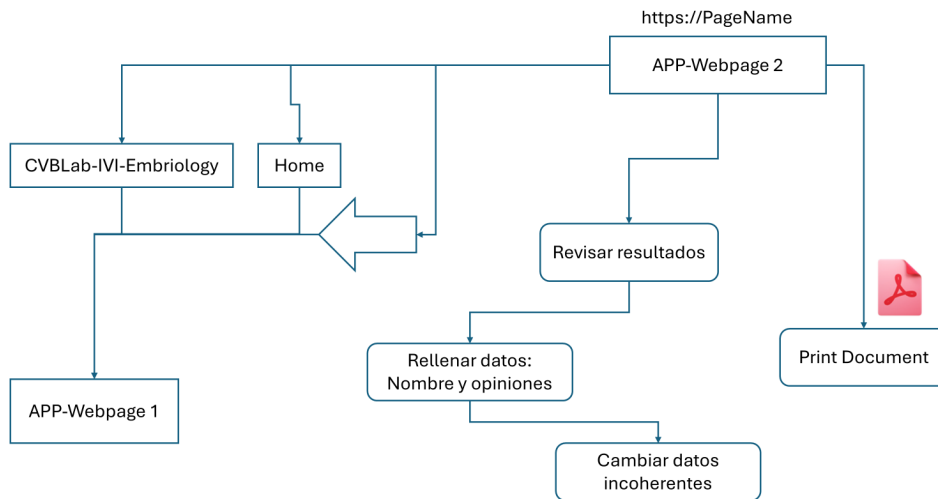


Figura 4.33: Diagrama de flujo del funcionamiento de la página web de resultados.

4.3.3. Inferencia modelo autocrop

El modelo de autocrop se utiliza para realizar un recorte automático a las imágenes de los ovocitos que se quieran analizar por parte de los embriólogos. Este modelo necesita de grandes recursos hardware para ser ejecutado y es por este motivo por el cual que se realiza desde el servidor del laboratorio de CVB Lab. Debido a que se instancia este modelo, se realiza el proyecto por completo para ser ejecutado desde el servidor en cuanto

a procesamiento y análisis de las imágenes, también por comodidad futura de ya tener creadas las imágenes docker correspondientes sobre el servidor.

El primer paso para desarrollar los recortes de imágenes automáticos sobre las imágenes cargadas de la aplicación es estructurar en contenedores docker las aplicaciones, que en este caso son dos, la aplicación general sobre la cual cargamos las imágenes y luego se van a visualizar (app.py), y la aplicación de cropeo automático (autocropapp.py), encargada de obtener las imágenes enviadas, cropearlas por el modelo de ultralytics y devolverlas a la aplicación general como se explica más en detalle sobre la Figura 4.6. Según se explicó en el diagrama de la Figura 2.30, con los dockerfile se crean las imágenes docker y de ahí el contenedor. Cuando los contenedores están creados se desarrolla un documento denominado "docker-compose", como se muestra en la Figura 4.36 para paralelizar el lanzamiento de los contenedores y así las aplicaciones con una única línea de código.

El dockerfile de la aplicación general (app.py) se desarrolla sobre la Figura 4.34 en donde se detalla que se utiliza la imagen de pytorch de nvidia con versión 22.12 y python 3, seguidamente, se desinstalan las versiones de opencv-contrib-python y opencv-python que vienen por defecto en la imagen de pytorch de nvidia y se instalan las versiones que se necesitan para la aplicación. Luego se instalan las versiones de opencv-contrib-python, opencv-python, ultralytics, scikit-image y Flask que se necesitan para la aplicación, así como se actualiza el sistema y se instala libglib que es necesario para que funcione opencv en el servidor de docker. Finalmente se ejecuta la aplicación y se pasa por el servidor de docker en el puerto 5000 con la dirección IP.

En cuanto al archivo de la aplicación de recorte (autocropapp.py) los pasos a seguir son los mismos y mostrados sobre la Figura 4.35, en donde se detalla sobre el código más información relevante, incluyendo la dirección IP del servidor utilizado (158.42.170.13), pero con puerto en el 5001.

```

1 FROM nvcr.io/nvidia/pytorch:22.12-py3
2
3
4 RUN pip uninstall opencv-contrib-python opencv-python
5 RUN pip install opencv-contrib-python==4.5.5.62 opencv-python==4.6.0.66
   → ultralytics==8.1.47 scikit-image==0.19.3 Flask==2.2.2
6 RUN apt-get update -y && apt-get install -y libglib
7 RUN pip install openpyxl==3.0.10
8 # Ejecutar la aplicación y que pase por el servidor de docker en el puerto 5000
9 # El servidor docker tiene la siguiente IP: 158.42.170.13 y el puerto 50007
10 # Nombre de contenedor: hola_1
11 CMD ["python", "app.py", "--host=0.0.0.0" , "--port=5000"]

```

Figura 4.34: Documento Dockerfile de la aplicación .app.py”.

```

1 # Explicación del código del dockerfile:
2 # 1. Se utiliza la imagen de pytorch de nvidia con la versión 22.12 y python 3
   ↪ (FROM nvcr.io/nvidia/pytorch:22.12-py3)
3 # 2. Se desinstalan las versiones de opencv-contrib-python y opencv-python que
   ↪ vienen por defecto en la imagen de pytorch de nvidia y se instalan las
   ↪ versiones que se necesitan para la aplicación
4 # 3. Se instalan las versiones de opencv-contrib-python, opencv-python,
   ↪ ultralytics, scikit-image y Flask que se necesitan para la aplicación
5 # 4. Se actualiza el sistema y se instala libgl1 que es necesario para que
   ↪ funcione opencv en el servidor de docker
6 # 5. Se ejecuta la aplicación y se pasa por el servidor de docker en el puerto
   ↪ 5001 con la IP
7
8 FROM nvcr.io/nvidia/pytorch:22.12-py3
9
10 RUN pip uninstall opencv-contrib-python opencv-python
11 RUN pip install opencv-contrib-python==4.5.5.62 opencv-python==4.6.0.66
   ↪ ultralytics==8.1.47 scikit-image==0.19.3 Flask==2.2.2
12 RUN apt-get update -y && apt-get install -y libgl1
13
14 # Ejecutar la aplicación y que pase por el servidor de docker en el puerto 5001
15 # El servidor docker tiene la siguiente IP: 158.42.170.13 y el puerto 5001
16 # Nombre de contenedor: hola_2
17 CMD ["python", "autocrop_app.py", "--host=0.0.0.0" , "--port=5001"]

```

Figura 4.35: Documento Dockerfile de la aplicación "autocropapp.py".

Un docker compose se utiliza para definir y lanzar aplicaciones multicontenedor, simplificando así el control de las aplicaciones [77]. El término "services" de la Figura 4.36 refiere a los componentes de una aplicación lanzados sobre la misma imagen contenedor. La forma en que los servicios (aplicaciones instanciadas) establecen comunicación entre sí, se declara en la red (networks). Para establecer la gpu a utilizar del servidor, se declara en el término "deploy", para especificar los recursos reservados para el contenedor a lanzar. Con el término "volumes" se declara el directorio de trabajo del servidor donde se ejecutará la aplicación [78].

Para cada aplicación a instanciar se declara con el término "build" la aplicación a ejecutar con su documento Dockerfile pertinente y un nombre de contenedor docker. Cuando el código del compose está preparado, desde la terminal, se puede lanzar las aplicaciones mediante una única línea de código mostrada abajo.

```
docker compose up --build
```

Una vez ejecutado, empiezan a activarse los contenedores y se podrá acceder a la aplicación web permitiendo la gestión de traspaso de imágenes y ejecutándose la aplicación de autocrop para la recepción de las imágenes de ovocitos recién recortadas, como se muestra en el diagrama de la Figura 4.6.

```
1 services:
2   app:
3     build:
4       context: app
5       dockerfile: Dockerfile
6     container_name: hola_1
7     ports:
8       - "5000:5000"
9     networks:
10      - cvblab
11    deploy:
12      resources:
13        reservations:
14          devices:
15            - driver: nvidia
16              device_ids: [ '0' ]
17              capabilities: [ gpu ]
18      volumes:
19        - ./app/:/workspace/
20    autocrop:
21      build:
22        context: autocrop_app
23        dockerfile: Dockerfile
24      container_name: hola_2
25      networks:
26        - cvblab
27      deploy:
28        resources:
29          reservations:
30            devices:
31              - driver: nvidia
32                device_ids: [ '0' ]
33                capabilities: [ gpu ]
34      volumes:
35        - ./autocrop_app/:/workspace/
36    #   ports:
37    #     - "5001:5001"
38
39 networks:
40 cvblab:
```

Figura 4.36: Documento Docker-compose para el lanzamiento de los contenedores para la ejecución de la aplicación en conjunto del autocrop.

El diagrama de flujo de la Figura 4.6 representa los pasos que sigue el programa desde la acción de buscar una imagen en local para subirlos a la aplicación de recorte. Desde el PC personal se accede a la dirección del servidor con puerto 5000 para entrar a la aplicación, donde se puede realizar un búsqueda de imágenes de ovocitos en local y así cropearlas, mandándolas con un post a la aplicación en el puerto 5001 (véase Figura 4.27), donde comienza el proceso de recorte de imagen con el modelo proporcionado para ello (véase Figura 4.7), así se volverá a enviar estas imágenes pero ya recortadas para seguir

con los procesos de la aplicación necesarios.

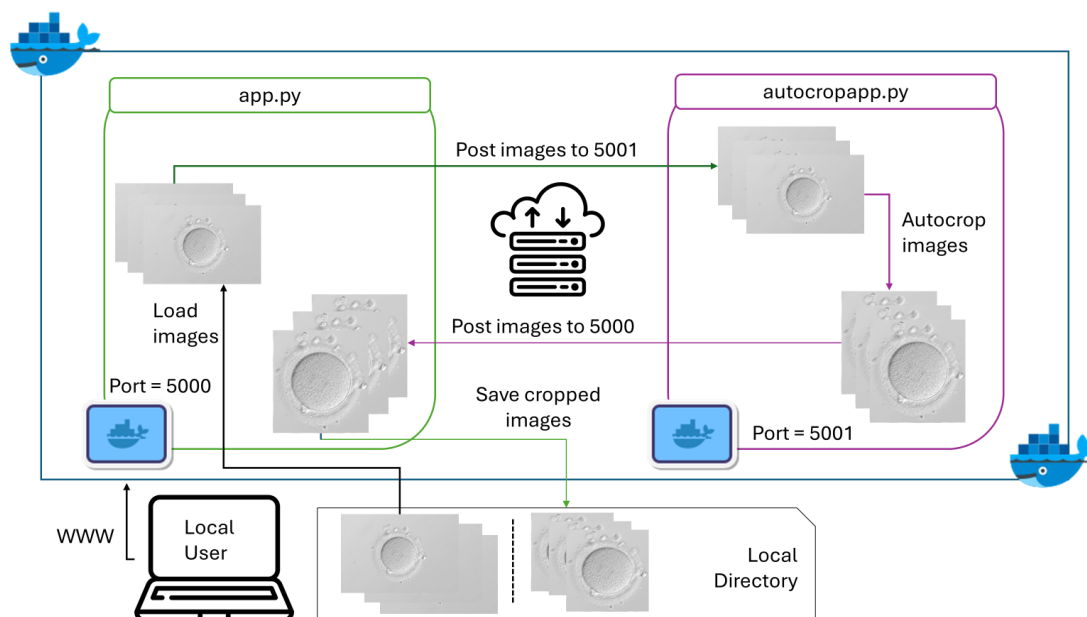


Figura 4.37: Proceso de autocrop de las imágenes de los ovocitos desde que se cargan las imágenes en la aplicación, hasta que se devuelven por el modelo.

4.3.4. Inferencia extracción de dimensiones

La inferencia de los algoritmos de detección de bordes de la zona del espacio perivitelino para el cálculo de las dimensiones de diámetro y área de los ovocitos se realiza semejante al proceso previamente explicado en la sección 4.3.3, donde se ha de declarar los documentos Dockerfile, tanto para la aplicación como para la aplicación de dimensiones, como se detalla en la Figuras 4.34 y 4.35, pero realizándolo para el caso de dimensiones acabada en puerto 5002. En cuanto al documento de docker compose, habrá que añadir un "service" más con el contenido de esta nueva aplicación, es decir, cambiando los términos de "build" y "volumes" (Figura 4.36).

Ya realizada la inferencia, se podrá ejecutar los algoritmos enviando las imágenes recibidas por el modelo de autocrop visto en la Figura 4.37 a la nueva aplicación con el mismo código de envío de imágenes pero con puerto acabado en 5002 y nombre de aplicación también cambiado a "dimensiones" como ruta url de envío. Estos datos serán nuevamente enviados a la aplicación donde se guardarán para su posterior visualización en la página web de resultados. Este proceso está representado sobre la Figura 4.38.

Los algoritmos utilizados para el cálculo de las dimensiones son los explicados en la sección 4.1 del capítulo 4, Canny, Sobel y Laplace, siguiendo el diagrama de bloques de la Figura 4.4. El resultado de las dimensiones seguirá la media de los tres algoritmos para asegurar un resultado lo más correcto posible, debido a posibles errores de alguno de los algoritmos.

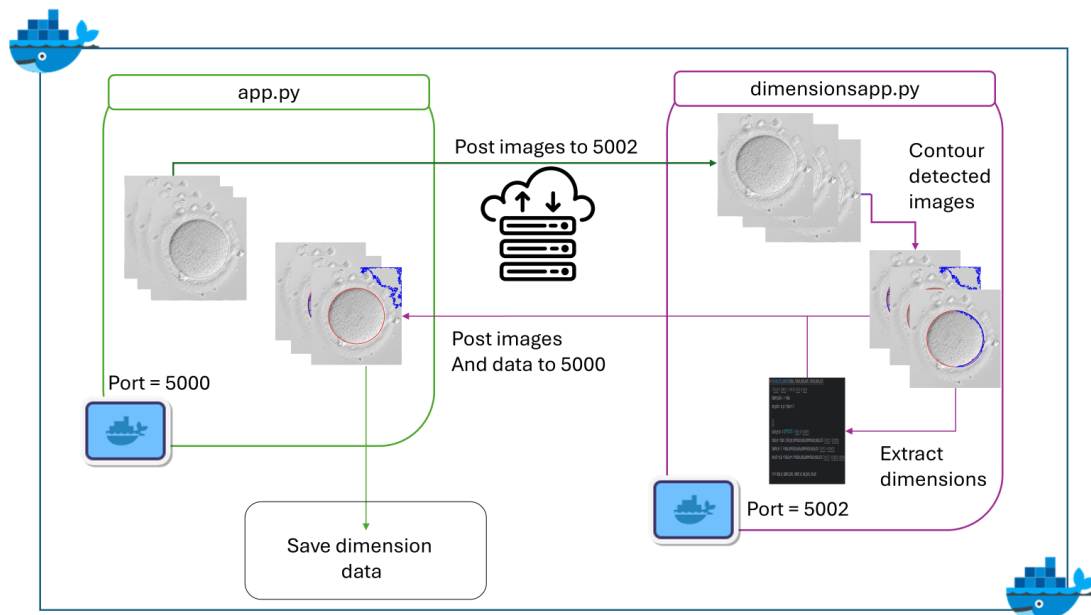


Figura 4.38: Proceso de detección de bordes del EPV y extracción de dimensiones de los ovocitos.

4.3.5. Inferencia modelo de predicción de calidad ovocitaria

La aplicación del modelo de predicción de calidad se infiere del mismo modo que las aplicaciones anteriores (modelo de autocrop sección 4.3.3 y extracción de dimensiones, sección 4.3.4). En primer lugar se genera en el directorio de la aplicación un archivo *dockerfile* en el que se predefinen las librerías que utiliza el modelo en concreto, la ruta de la imagen docker, así como se ve en el caso del modelo de autocrop (véase Figura 4.35). Siguiendo con el proceso de inferencia del modelo, se realiza la declaración de la aplicación como servicio dentro del documento *docker compose* de la aplicación global.

Sobre la Figura 4.39 se explica el proceso de servicios llevados a cabo para lograr la predicción del modelo entrenado sobre la calidad de los ovocitos. Este proceso comienza por la parte final de la etapa de extracción de las dimensiones de los ovocitos, explicada sobre la Figura 4.38 de la sección 4.3.4, donde los datos añadidos a las características de los ovocitos detectadas por el equipo de embriólogos del IVI junto a las imágenes de los mismos ovocitos, son enviados mediante un servicio de APIrest sobre la aplicación donde se encuentra el modelo de predicción de calidad, el cuál otorga una nota del 1 al 10 a cada ovocito enviado. Finalmente, estos resultados serán nuevamente enviados a la interfaz de usuario de la aplicación principal para la representación y visualización de los resultados.

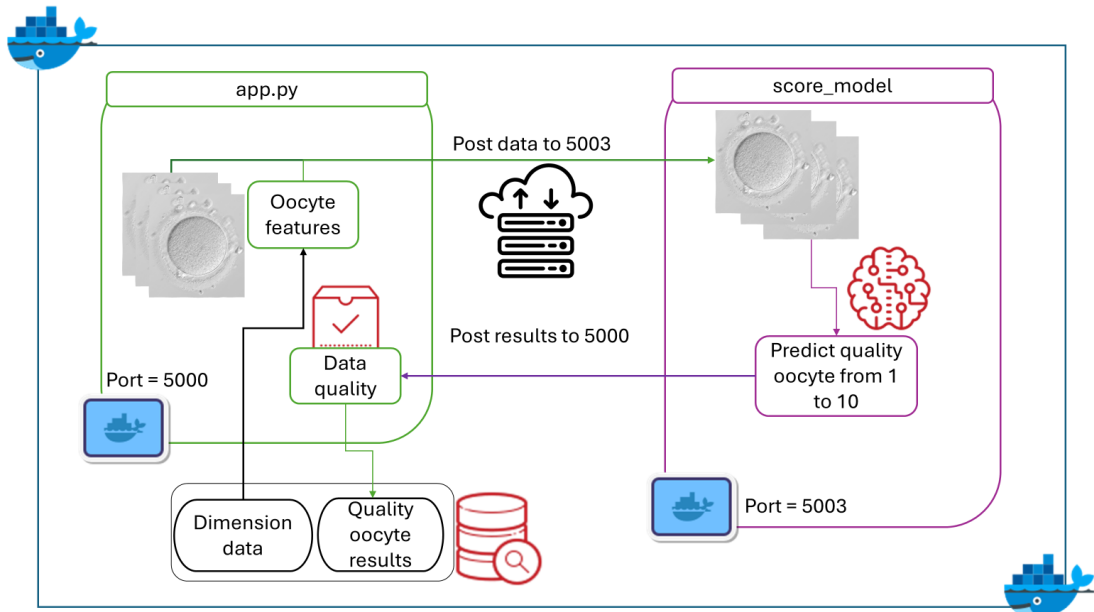


Figura 4.39: Proceso de predicción de calidad de los ovocitos a evaluar.

Capítulo 5

Resultados

5.1. Análisis de imagen y modelo de deep learning

En la presente sección de los resultados del modelo de predicción de calidad y análisis de imagen, se muestran las salidas de la extracción de dimensiones de los ovocitos, y de las técnicas de validación realizadas sobre los modelos convolucionales de los que se han probado su entrenamiento. Se presentan la salida de la función de pérdidas, junto al *accuracy* por época obtenido y la matriz de correlación, que son términos explicados en la sección de deep learning del marco teórico 2.2.

Sobre la Figura 5.1, se presenta las detecciones y dimensiones obtenidas sobre los ovocitos. Los diferentes modelos utilizados otorgan variabilidad en cuanto a detección del borde. Variando las especificaciones de los algoritmos se consiguen que tono de borde se obtiene y la distancia al centro que este detecta. Los algoritmos van en orden de explicación sobre la sección 4.1, en primer lugar, el algoritmo de Canny, siguiendo con el de Sobel y Laplace, y terminando con un método no explicado, pero basándose en el principio del método de Canny, para elipses.

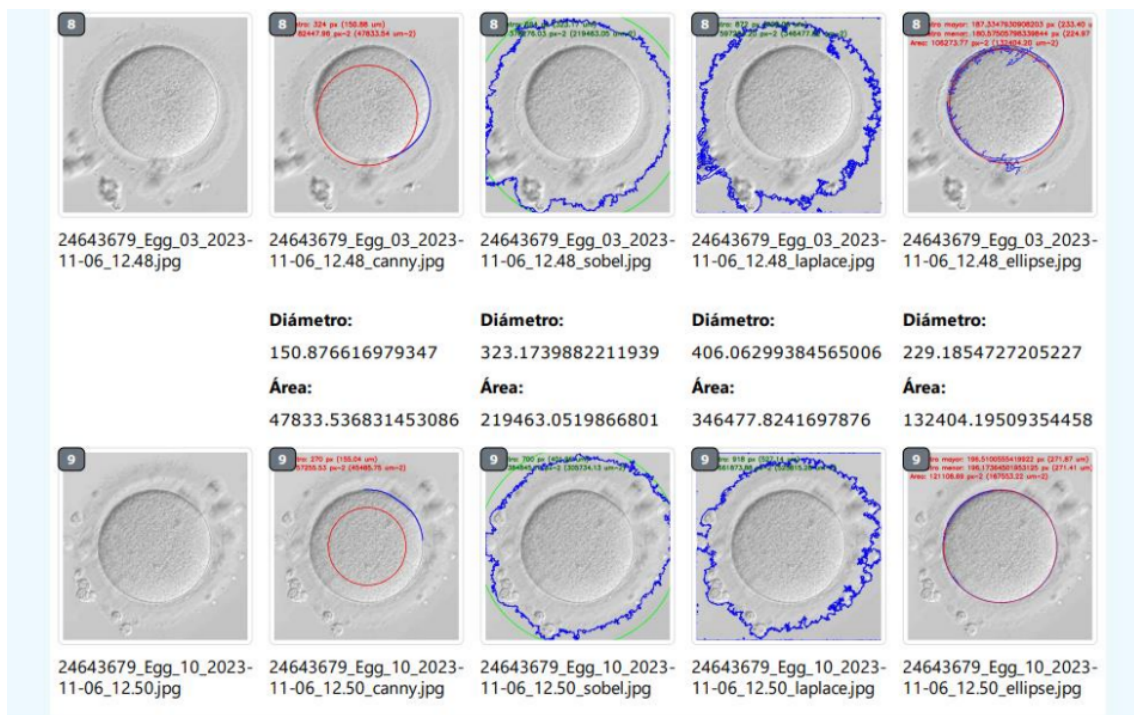


Figura 5.1: Imagen de la detección de bordes mediante la aplicación de los algoritmos de Canny, Sobel, Laplace. El último método es una variante del algoritmo de Canny con elipse no explicada.

Sobre las Figuras 5.2, 5.3, 5.4, 5.5, y 5.6, se presentan y analizan los resultados de cuatro modelos entrenados para la clasificación de imágenes en escala de grises. El **Modelo 1**, con una red convolucional simple, mostró un rendimiento competitivo en términos de precisión, sin embargo, experimentó problemas de sobreajuste debido a su complejidad relativamente baja y la necesidad de ajustar dinámicamente la entrada de la capa completamente conectada. El **Modelo 2**, basado en ResNet-18, ofreció un rendimiento notablemente alto, aprovechando la profundidad de la red y la capacidad de ResNet para aprender características profundas. Sin embargo, la eliminación de la capa completamente conectada original y la adaptación a imágenes en escala de grises introdujeron algunos desafíos en la integración de las características adicionales, lo que a veces afectó la precisión general. El **Modelo 3**, una red convolucional con activaciones ELU y MaxPooling, mostró una capacidad robusta para aprender representaciones de características, pero su rendimiento se vio limitado por la necesidad de calcular dinámicamente el tamaño de entrada para la capa lineal y la dificultad de combinar de manera efectiva las características adicionales. Finalmente, el **Modelo 4**, basado en Vision Transformer (ViT), destacó por su capacidad para capturar representaciones complejas y su habilidad para integrar eficazmente características adicionales, lo que resultó en una precisión superior en comparación con los otros modelos. Aunque el Modelo 4 demandó más recursos computacionales y tiempo de entrenamiento, su capacidad para generalizar y manejar características adicionales de manera efectiva lo hizo el mejor entre los modelos evaluados, proporcionando el mejor rendimiento general en la tarea de clasificación de imágenes.

```

Entrenando con parámetros: {'batch_size': 32, 'lr': 0.001, 'optimizer': 'Adam', 'weight_decay': 0.0}
Epoch [1/200] Model 1 Loss: 1.9406 Val Loss: 1.6416 Model 2 Loss: 1.8494 Val Loss: 0.9218 Model 3 Loss: 10.1995 Val Loss: 1.7871 Model 4 Loss: 2.0594 Val Loss: 0.7863
Epoch [2/200] Model 1 Loss: 1.8474 Val Loss: 1.6416 Model 2 Loss: 1.4928 Val Loss: 0.9317 Model 3 Loss: 3.0543 Val Loss: 4.4541 Model 4 Loss: 1.1796 Val Loss: 2.0617
Epoch [3/200] Model 1 Loss: 1.8301 Val Loss: 1.6416 Model 2 Loss: 1.4096 Val Loss: 0.9187 Model 3 Loss: 2.1848 Val Loss: 2.2024 Model 4 Loss: 1.3560 Val Loss: 1.0997
Epoch [4/200] Model 1 Loss: 1.8396 Val Loss: 1.6416 Model 2 Loss: 1.4720 Val Loss: 0.8070 Model 3 Loss: 1.5872 Val Loss: 1.2741 Model 4 Loss: 1.1271 Val Loss: 0.9092
Epoch [5/200] Model 1 Loss: 1.8725 Val Loss: 1.6416 Model 2 Loss: 1.4397 Val Loss: 0.8561 Model 3 Loss: 1.4442 Val Loss: 2.5688 Model 4 Loss: 1.0982 Val Loss: 0.7109
Epoch [6/200] Model 1 Loss: 1.4263 Val Loss: 0.9198 Model 3 Loss: 1.4390 Val Loss: 0.8421 Model 4 Loss: 1.4126 Val Loss: 0.6876
Epoch [7/200] Model 2 Loss: 1.4227 Val Loss: 0.9015 Model 3 Loss: 1.5707 Val Loss: 0.9459 Model 4 Loss: 1.3999 Val Loss: 0.6644
Epoch [8/200] Model 2 Loss: 1.2741 Val Loss: 0.9770 Model 3 Loss: 1.3761 Val Loss: 0.8884 Model 4 Loss: 1.1448 Val Loss: 0.8946
Epoch [9/200] Model 3 Loss: 1.4516 Val Loss: 1.0736 Model 4 Loss: 0.9640 Val Loss: 0.8703
Epoch [10/200] Model 3 Loss: 1.2949 Val Loss: 1.0634 Model 4 Loss: 1.1922 Val Loss: 0.7180
Epoch [11/200] Model 4 Loss: 1.2927 Val Loss: 0.7287

Finished Training
Entrenando con parámetros: {'batch_size': 32, 'lr': 0.001, 'optimizer': 'Adam', 'weight_decay': 0.001}
Epoch [1/200] Model 1 Loss: 2.2577 Val Loss: 2.3676 Model 2 Loss: 1.4908 Val Loss: 2.4943 Model 3 Loss: 3.3204 Val Loss: 4.9620 Model 4 Loss: 1.2355 Val Loss: 1.6936
Epoch [2/200] Model 1 Loss: 2.4017 Val Loss: 2.3448 Model 2 Loss: 1.3231 Val Loss: 1.9229 Model 3 Loss: 2.0962 Val Loss: 3.5027 Model 4 Loss: 1.4002 Val Loss: 1.7287
Epoch [3/200] Model 1 Loss: 2.4069 Val Loss: 2.3612 Model 2 Loss: 1.1926 Val Loss: 2.0266 Model 3 Loss: 1.5001 Val Loss: 3.2974 Model 4 Loss: 1.2602 Val Loss: 1.6464
Epoch [4/200] Model 1 Loss: 2.4171 Val Loss: 2.3450 Model 2 Loss: 1.2712 Val Loss: 1.9865 Model 3 Loss: 2.2054 Val Loss: 4.1312 Model 4 Loss: 1.2906 Val Loss: 1.5451
Epoch [5/200] Model 1 Loss: 2.4176 Val Loss: 2.3450 Model 2 Loss: 1.1191 Val Loss: 2.1882 Model 3 Loss: 1.7087 Val Loss: 3.0433 Model 4 Loss: 1.0718 Val Loss: 1.8561
Epoch [6/200] Model 1 Loss: 2.4276 Val Loss: 2.3450 Model 2 Loss: 1.2020 Val Loss: 1.9165 Model 3 Loss: 1.6785 Val Loss: 2.7441 Model 4 Loss: 1.1215 Val Loss: 1.6033
Epoch [7/200] Model 2 Loss: 1.1527 Val Loss: 1.9936 Model 3 Loss: 1.2918 Val Loss: 2.2098 Model 4 Loss: 1.0028 Val Loss: 1.7131
Epoch [8/200] Model 2 Loss: 0.9857 Val Loss: 2.9673 Model 3 Loss: 1.0535 Val Loss: 2.2374 Model 4 Loss: 1.0704 Val Loss: 1.6325
Epoch [9/200] Model 2 Loss: 1.4140 Val Loss: 1.8712 Model 3 Loss: 0.8895 Val Loss: 2.2675
Epoch [10/200] Model 2 Loss: 1.3171 Val Loss: 1.7817 Model 3 Loss: 1.2000 Val Loss: 2.2943
Epoch [11/200] Model 2 Loss: 1.4287 Val Loss: 1.6906 Model 3 Loss: 0.9029 Val Loss: 2.1852
Epoch [12/200] Model 2 Loss: 1.3026 Val Loss: 1.7843 Model 3 Loss: 1.0589 Val Loss: 2.4446
Epoch [13/200] Model 2 Loss: 1.2681 Val Loss: 1.7353 Model 3 Loss: 0.9358 Val Loss: 2.7245
Epoch [14/200] Model 2 Loss: 1.1711 Val Loss: 1.7139 Model 3 Loss: 0.7598 Val Loss: 2.6378
Epoch [15/200] Model 2 Loss: 1.1036 Val Loss: 1.7618 Model 3 Loss: 0.8245 Val Loss: 2.5737

```

Figura 5.2: Primera tupla de resultados de entrenamiento de los modelos CNN.

```

Finished Training
Entrenando con parámetros: {'batch_size': 32, 'lr': 0.001, 'optimizer': 'SGD', 'weight_decay': 0.0}
Epoch [1/200] Model 1 Loss: 2.4035 Val Loss: 2.4492 Model 2 Loss: 1.1327 Val Loss: 0.8397 Model 3 Loss: 1.9483 Val Loss: 1.1876 Model 4 Loss: 1.2174 Val Loss: 0.5617
Epoch [2/200] Model 1 Loss: 2.4052 Val Loss: 2.4492 Model 2 Loss: 1.2739 Val Loss: 0.8091 Model 3 Loss: 1.4078 Val Loss: 0.9436 Model 4 Loss: 1.1039 Val Loss: 0.7173
Epoch [3/200] Model 1 Loss: 2.4230 Val Loss: 2.4492 Model 2 Loss: 1.1607 Val Loss: 0.8288 Model 3 Loss: 1.6738 Val Loss: 0.8407 Model 4 Loss: 1.2213 Val Loss: 0.5637
Epoch [4/200] Model 1 Loss: 2.4136 Val Loss: 2.4492 Model 2 Loss: 1.2170 Val Loss: 0.8961 Model 3 Loss: 1.5158 Val Loss: 0.7721 Model 4 Loss: 1.0074 Val Loss: 0.4913
Epoch [5/200] Model 1 Loss: 2.4047 Val Loss: 2.4492 Model 2 Loss: 1.2974 Val Loss: 0.8917 Model 3 Loss: 1.4639 Val Loss: 0.7882 Model 4 Loss: 1.0954 Val Loss: 0.5434
Epoch [6/200] Model 2 Loss: 1.1727 Val Loss: 0.9875 Model 3 Loss: 1.6753 Val Loss: 0.8870 Model 4 Loss: 1.1337 Val Loss: 0.7588
Epoch [7/200] Model 3 Loss: 1.5708 Val Loss: 0.8807 Model 4 Loss: 1.1602 Val Loss: 0.6170
Epoch [8/200] Model 3 Loss: 1.4159 Val Loss: 1.0198 Model 4 Loss: 0.9775 Val Loss: 0.5203
Epoch [9/200] Model 3 Loss: 1.5631 Val Loss: 0.7627
Epoch [10/200] Model 3 Loss: 1.4039 Val Loss: 0.9446
Epoch [11/200] Model 3 Loss: 1.5501 Val Loss: 0.7650
Epoch [12/200] Model 3 Loss: 1.3972 Val Loss: 0.8808
Epoch [13/200] Model 3 Loss: 1.4429 Val Loss: 1.0222

Finished Training
Entrenando con parámetros: {'batch_size': 32, 'lr': 0.001, 'optimizer': 'SGD', 'weight_decay': 0.001}
Epoch [1/200] Model 1 Loss: 2.4268 Val Loss: 2.4069 Model 2 Loss: 1.2196 Val Loss: 1.4258 Model 3 Loss: 1.4195 Val Loss: 1.6888 Model 4 Loss: 1.0323 Val Loss: 1.2270
Epoch [2/200] Model 1 Loss: 2.4138 Val Loss: 2.4069 Model 2 Loss: 1.0212 Val Loss: 1.4417 Model 3 Loss: 1.2591 Val Loss: 1.7000 Model 4 Loss: 1.0411 Val Loss: 1.2951
Epoch [3/200] Model 1 Loss: 2.4199 Val Loss: 2.4069 Model 2 Loss: 1.1229 Val Loss: 1.4511 Model 3 Loss: 1.4139 Val Loss: 1.7134 Model 4 Loss: 0.8761 Val Loss: 1.4374
Epoch [4/200] Model 1 Loss: 2.4169 Val Loss: 2.4069 Model 2 Loss: 1.1380 Val Loss: 1.4584 Model 3 Loss: 1.2803 Val Loss: 1.6512 Model 4 Loss: 0.8844 Val Loss: 1.2836
Epoch [5/200] Model 1 Loss: 2.4130 Val Loss: 2.4069 Model 2 Loss: 1.1622 Val Loss: 1.4462 Model 3 Loss: 1.3527 Val Loss: 1.7759 Model 4 Loss: 1.0163 Val Loss: 1.3099
Epoch [6/200] Model 1 Loss: 2.4050 Val Loss: 2.4069 Model 3 Loss: 1.3078 Val Loss: 1.8566
Epoch [7/200] Model 1 Loss: 2.4198 Val Loss: 2.4069 Model 3 Loss: 1.2438 Val Loss: 1.8898
Epoch [8/200] Model 1 Loss: 2.4312 Val Loss: 2.4069 Model 3 Loss: 1.5710 Val Loss: 1.6795
Epoch [9/200] Model 1 Loss: 2.4151 Val Loss: 2.4069

```

Figura 5.3: Segunda tupla de resultados de entrenamiento de los modelos CNN.

```

Finished Training
Entrenando con parámetros: {'batch_size': 32, 'lr': 0.0001, 'optimizer': 'Adam', 'weight_decay': 0.001}
Epoch [1/200] Model 1 Loss: 1.9004 Val Loss: 1.8506 Model 2 Loss: 1.2453 Val Loss: 1.1235 Model 3 Loss: 1.5096 Val Loss: 1.1778 Model 4 Loss: 1.1641 Val Loss: 1.0180
Epoch [2/200] Model 1 Loss: 1.8784 Val Loss: 1.9202 Model 2 Loss: 1.0643 Val Loss: 1.1281 Model 3 Loss: 1.4043 Val Loss: 1.1513 Model 4 Loss: 1.0683 Val Loss: 1.0189
Epoch [3/200] Model 1 Loss: 1.8339 Val Loss: 1.8505 Model 2 Loss: 1.3437 Val Loss: 1.1242 Model 3 Loss: 1.2424 Val Loss: 1.1750 Model 4 Loss: 1.0469 Val Loss: 1.0204
Epoch [4/200] Model 1 Loss: 1.8456 Val Loss: 1.8506 Model 2 Loss: 0.9759 Val Loss: 1.1267 Model 3 Loss: 1.1118 Val Loss: 1.3070 Model 4 Loss: 1.0503 Val Loss: 1.0174
Epoch [5/200] Model 1 Loss: 1.8506 Val Loss: 1.8505 Model 2 Loss: 1.1469 Val Loss: 1.1260 Model 3 Loss: 1.0209 Val Loss: 1.3822 Model 4 Loss: 1.0382 Val Loss: 1.0221
Epoch [6/200] Model 1 Loss: 1.8765 Val Loss: 1.8507 Model 2 Loss: 1.2593 Val Loss: 1.1112 Model 3 Loss: 0.9850 Val Loss: 1.1198 Model 4 Loss: 1.0623 Val Loss: 1.0279
Epoch [7/200] Model 1 Loss: 1.8434 Val Loss: 1.8510 Model 2 Loss: 1.2067 Val Loss: 1.1110 Model 3 Loss: 0.9144 Val Loss: 1.3778 Model 4 Loss: 0.8931 Val Loss: 1.0144
Epoch [8/200] Model 2 Loss: 1.1055 Val Loss: 1.1024 Model 3 Loss: 0.6711 Val Loss: 1.4357 Model 4 Loss: 1.1921 Val Loss: 1.0616
Epoch [9/200] Model 2 Loss: 1.0611 Val Loss: 1.1100 Model 3 Loss: 0.6913 Val Loss: 1.3566 Model 4 Loss: 1.1042 Val Loss: 1.0410
Epoch [10/200] Model 2 Loss: 1.2617 Val Loss: 1.1096 Model 3 Loss: 0.7507 Val Loss: 2.3769 Model 4 Loss: 1.0456 Val Loss: 1.0471
Epoch [11/200] Model 2 Loss: 1.1735 Val Loss: 1.1136 Model 4 Loss: 0.9817 Val Loss: 1.0470
Epoch [12/200] Model 2 Loss: 1.0906 Val Loss: 1.0994
Epoch [13/200] Model 2 Loss: 1.1922 Val Loss: 1.0995
Epoch [14/200] Model 2 Loss: 1.0757 Val Loss: 1.1051
Epoch [15/200] Model 2 Loss: 1.1905 Val Loss: 1.0822
Epoch [16/200] Model 2 Loss: 1.1552 Val Loss: 1.1039
Epoch [17/200] Model 2 Loss: 1.1216 Val Loss: 1.0959
Epoch [18/200] Model 2 Loss: 1.2163 Val Loss: 1.0853
Epoch [19/200] Model 2 Loss: 1.1978 Val Loss: 1.1454

Finished Training
Entrenando con parámetros: {'batch_size': 32, 'lr': 0.0001, 'optimizer': 'SGD', 'weight_decay': 0.0}
Epoch [1/200] Model 1 Loss: 1.8754 Val Loss: 1.5919 Model 2 Loss: 1.4036 Val Loss: 0.6826 Model 3 Loss: 1.6989 Val Loss: 1.0184 Model 4 Loss: 1.1573 Val Loss: 0.6109
Epoch [2/200] Model 1 Loss: 1.8873 Val Loss: 1.5915 Model 2 Loss: 1.1747 Val Loss: 0.6814 Model 3 Loss: 1.4590 Val Loss: 0.9924 Model 4 Loss: 1.0698 Val Loss: 0.5903
Epoch [3/200] Model 1 Loss: 1.8643 Val Loss: 1.5937 Model 2 Loss: 1.3219 Val Loss: 0.6764 Model 3 Loss: 1.5227 Val Loss: 0.9399 Model 4 Loss: 0.9238 Val Loss: 0.5494
Epoch [4/200] Model 1 Loss: 1.9233 Val Loss: 1.6215 Model 2 Loss: 1.3509 Val Loss: 0.7043 Model 3 Loss: 1.4304 Val Loss: 0.9397 Model 4 Loss: 1.0310 Val Loss: 0.5536
Epoch [5/200] Model 1 Loss: 1.8938 Val Loss: 1.6191 Model 2 Loss: 1.2524 Val Loss: 0.6951 Model 3 Loss: 1.4946 Val Loss: 0.8920 Model 4 Loss: 1.0460 Val Loss: 0.5582
Epoch [6/200] Model 1 Loss: 1.8895 Val Loss: 1.6323 Model 2 Loss: 1.2295 Val Loss: 0.6891 Model 3 Loss: 1.3852 Val Loss: 0.8609 Model 4 Loss: 1.0838 Val Loss: 0.5582
Epoch [7/200] Model 2 Loss: 1.2503 Val Loss: 0.7135 Model 3 Loss: 1.3961 Val Loss: 0.8824 Model 4 Loss: 1.0890 Val Loss: 0.5567
Epoch [8/200] Model 3 Loss: 1.5610 Val Loss: 0.8931
Epoch [9/200] Model 3 Loss: 1.4190 Val Loss: 0.9312
Epoch [10/200] Model 3 Loss: 1.3619 Val Loss: 0.8931

```

Figura 5.4: Tercera tupla de resultados de entrenamiento de los modelos CNN.


```

Finished Training
Entrenando con parámetros: {'batch_size': 64, 'lr': 0.001, 'optimizer': 'SGD', 'weight_decay': 0.001}
Epoch [1/200] Model 1 Loss: 1.7333 Val Loss: 1.7713 Model 2 Loss: 0.7883 Val Loss: 0.9125 Model 3 Loss: 1.6881 Val Loss: 2.5289 Model 4 Loss: 0.6679 Val Loss: 0.8588
Epoch [2/200] Model 1 Loss: 1.7282 Val Loss: 1.7713 Model 2 Loss: 0.7554 Val Loss: 0.9091 Model 3 Loss: 1.1559 Val Loss: 1.5620 Model 4 Loss: 0.7260 Val Loss: 0.8388
Epoch [3/200] Model 1 Loss: 1.7095 Val Loss: 1.7713 Model 2 Loss: 0.7390 Val Loss: 0.9138 Model 3 Loss: 1.6019 Val Loss: 8.6690 Model 4 Loss: 0.7776 Val Loss: 0.8510
Epoch [4/200] Model 1 Loss: 1.7590 Val Loss: 1.7713 Model 2 Loss: 0.8796 Val Loss: 0.9109 Model 3 Loss: 0.9995 Val Loss: 2.1810 Model 4 Loss: 0.8184 Val Loss: 0.8369
Epoch [5/200] Model 1 Loss: 1.7267 Val Loss: 1.7713 Model 2 Loss: 0.8983 Val Loss: 0.9131 Model 3 Loss: 0.9331 Val Loss: 1.5463 Model 4 Loss: 0.7453 Val Loss: 0.8432
Epoch [6/200] Model 2 Loss: 0.8904 Val Loss: 0.9083 Model 3 Loss: 1.1087 Val Loss: 1.6591 Model 4 Loss: 0.8838 Val Loss: 0.8319
Epoch [7/200] Model 2 Loss: 0.7556 Val Loss: 0.9075 Model 3 Loss: 1.3841 Val Loss: 1.2966 Model 4 Loss: 0.7998 Val Loss: 0.8428
Epoch [8/200] Model 2 Loss: 0.7417 Val Loss: 0.9138 Model 3 Loss: 1.2678 Val Loss: 3.4984 Model 4 Loss: 0.8412 Val Loss: 0.8409
Epoch [9/200] Model 2 Loss: 0.6912 Val Loss: 0.9120 Model 3 Loss: 1.1764 Val Loss: 2.8086 Model 4 Loss: 0.7328 Val Loss: 0.8501
Epoch [10/200] Model 2 Loss: 0.7561 Val Loss: 0.9189 Model 3 Loss: 1.1163 Val Loss: 1.4241 Model 4 Loss: 0.8906 Val Loss: 0.8310
Epoch [11/200] Model 2 Loss: 0.8461 Val Loss: 0.9113 Model 3 Loss: 0.8179 Val Loss: 1.3016 Model 4 Loss: 0.8265 Val Loss: 0.8329
Epoch [12/200] Model 3 Loss: 0.8398 Val Loss: 1.2468 Model 4 Loss: 0.7705 Val Loss: 0.8372
Epoch [13/200] Model 3 Loss: 1.0104 Val Loss: 2.1060 Model 4 Loss: 0.7022 Val Loss: 0.8447
Epoch [14/200] Model 3 Loss: 0.9714 Val Loss: 1.3775 Model 4 Loss: 0.8292 Val Loss: 0.8367
Epoch [15/200] Model 3 Loss: 0.9057 Val Loss: 1.2672
Epoch [16/200] Model 3 Loss: 1.1269 Val Loss: 1.5773
Epoch [17/200] Model 3 Loss: 0.7727 Val Loss: 1.2182
Epoch [18/200] Model 3 Loss: 0.7911 Val Loss: 1.7133
Epoch [19/200] Model 3 Loss: 0.9998 Val Loss: 1.2284
Epoch [20/200] Model 3 Loss: 0.8534 Val Loss: 1.2941
Epoch [21/200] Model 3 Loss: 0.8859 Val Loss: 1.2246

Finished Training
Entrenando con parámetros: {'batch_size': 64, 'lr': 0.0001, 'optimizer': 'Adam', 'weight_decay': 0.0}
Epoch [1/200] Model 1 Loss: 1.6933 Val Loss: 1.6807 Model 2 Loss: 0.8008 Val Loss: 0.7335 Model 3 Loss: 1.6918 Val Loss: 1.3382 Model 4 Loss: 0.7796 Val Loss: 0.6749
Epoch [2/200] Model 1 Loss: 1.7488 Val Loss: 1.6807 Model 2 Loss: 0.7512 Val Loss: 0.7302 Model 3 Loss: 1.0253 Val Loss: 1.0412 Model 4 Loss: 0.6421 Val Loss: 0.6709
Epoch [3/200] Model 1 Loss: 1.7773 Val Loss: 1.6807 Model 2 Loss: 0.9200 Val Loss: 0.7307 Model 3 Loss: 0.7123 Val Loss: 0.8862 Model 4 Loss: 0.7672 Val Loss: 0.6746
Epoch [4/200] Model 1 Loss: 1.7051 Val Loss: 1.6807 Model 2 Loss: 0.7347 Val Loss: 0.7331 Model 3 Loss: 0.5995 Val Loss: 0.9087 Model 4 Loss: 0.8164 Val Loss: 0.6823
Epoch [5/200] Model 1 Loss: 1.6897 Val Loss: 1.6807 Model 2 Loss: 0.9067 Val Loss: 0.7342 Model 3 Loss: 0.6164 Val Loss: 0.9722 Model 4 Loss: 1.0173 Val Loss: 0.7044
Epoch [6/200] Model 2 Loss: 0.7146 Val Loss: 0.7321 Model 3 Loss: 0.7229 Val Loss: 1.0528 Model 4 Loss: 0.7544 Val Loss: 0.6874
Epoch [7/200] Model 3 Loss: 0.3962 Val Loss: 1.0250
    
```

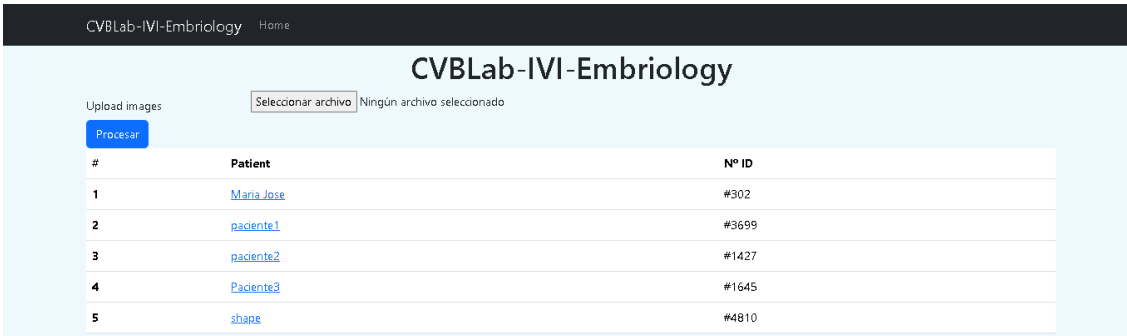
Figura 5.5: Cuarta tupla de resultados de entrenamiento de los modelos CNN.

Evaluación del Modelo 1: Classification Report:					Evaluación del Modelo 2: Classification Report:				
	precision	recall	f1-score	support		precision	recall	f1-score	support
1	0.00	0.00	0.00	5	1	0.00	0.00	0.00	5
2	0.00	0.00	0.00	8	2	0.00	0.00	0.00	8
3	0.00	0.00	0.00	9	3	0.00	0.00	0.00	9
4	0.00	0.00	0.00	27	4	0.68	0.48	0.57	27
5	0.00	0.00	0.00	11	5	0.00	0.00	0.00	11
6	0.00	0.00	0.00	16	6	0.00	0.00	0.00	16
7	0.00	0.00	0.00	12	7	0.00	0.00	0.00	12
8	0.00	0.00	0.00	16	8	0.00	0.00	0.00	16
9	0.05	1.00	0.10	6	9	0.07	1.00	0.12	6
accuracy			0.05	110	accuracy			0.17	110
macro avg	0.01	0.11	0.01	110	macro avg	0.08	0.16	0.08	110
weighted avg	0.00	0.05	0.01	110	weighted avg	0.17	0.17	0.15	110
Proved Accuracy: 10.0%					Proved Accuracy: 25.0%				
Evaluación del Modelo 3: Classification Report:					Evaluación del Modelo 4: Classification Report:				
	precision	recall	f1-score	support		precision	recall	f1-score	support
1	0.00	0.00	0.00	5	1	0.00	0.00	0.00	5
2	0.00	0.00	0.00	8	2	0.50	0.12	0.20	8
3	0.00	0.00	0.00	9	3	0.00	0.00	0.00	9
4	0.33	0.04	0.07	27	4	0.50	0.74	0.60	27
5	0.00	0.00	0.00	11	5	0.00	0.00	0.00	11
6	0.50	0.06	0.11	16	6	0.00	0.00	0.00	16
7	0.00	0.00	0.00	12	7	0.00	0.00	0.00	12
8	0.32	0.38	0.34	16	8	0.25	0.06	0.10	16
9	0.07	1.00	0.13	6	9	0.09	1.00	0.17	6
accuracy			0.13	110	accuracy			0.25	110
macro avg	0.14	0.16	0.07	110	macro avg	0.15	0.21	0.12	110
weighted avg	0.20	0.13	0.09	110	weighted avg	0.20	0.25	0.18	110
Proved Accuracy: 20.0%					Proved Accuracy: 30.0%				

Figura 5.6: Reportes de clasificación de los cuatro modelos entrenados.

5.2. Aplicación web

Sobre el presente apartado se muestran los resultados finales que toma la aplicación en su conjunto, comenzando por la primera vista de la aplicación en la Figura 5.7, que es la vista que habilita la capacidad de cargar las imágenes correspondientes de los ovocitos a analizar, donde en .oculto”se procesarán tanto para el recorte automático de las imágenes (explicado en el apartado 4.2.1) como para la obtención de sus dimensiones mediante los algoritmos mencionados en el apartado 4.1. También se muestra el resultado de la vista final de la aplicación con la Figura 5.8, donde se cargan las imágenes de los ovocitos recortados, que habían sido previamente seleccionados, aparte de las dimensiones respectivas al ovocito, y junto una serie de botones selectores para indicar por parte del equipo de embriólogos los dismorfismos ovocitarios que puedan observar. Cuando se hallan seleccionado los dismorfismos, podrán pulsar un botón que activa el proceso de clasificación del modelo entrenado para ello, el cual otorga un valor del 1 al 10 a cada ovocito de la cohorte.



#	Patient	Nº ID
1	Maria Jose	#302
2	paciente1	#3699
3	paciente2	#1427
4	Paciente3	#1645
5	shape	#4810

Figura 5.7: Resultado de la primera vista de la aplicación.

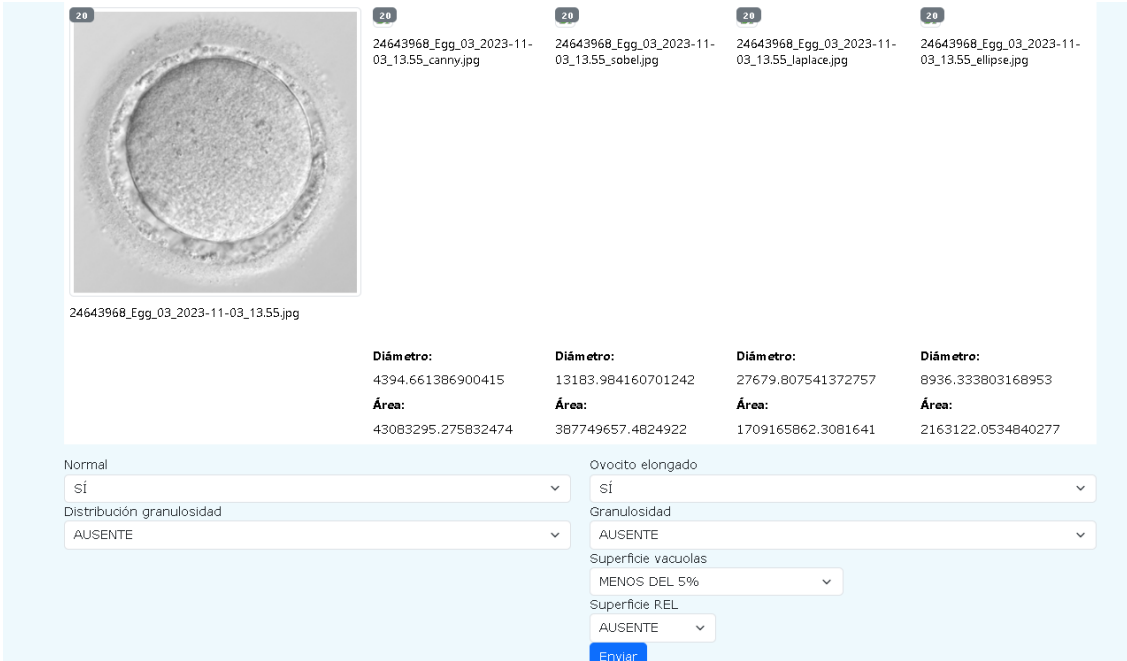


Image	Diámetro:	Área:	Image	Diámetro:	Área:
24643968_Egg_03_2023-11-03_13.55_canny.jpg	4394.661386900415	43083295.275832474	24643968_Egg_03_2023-11-03_13.55_sobel.jpg	13183.984160701242	387749657.4824922
24643968_Egg_03_2023-11-03_13.55_laplace.jpg	27679.807541372757	1709165862.3081641	24643968_Egg_03_2023-11-03_13.55_ellipse.jpg	8936.333803168953	2163122.0534840277

Normal:

Distribución granulosis:

Superficie vacuolas:

Superficie REL:

Figura 5.8: Resultado de la segunda vista de la aplicación.

Capítulo 6

Conclusiones y propuesta de futuro

Este trabajo ha abordado un problema actual y en auge del campo de las telecomunicaciones. La IA habilita la capacidad de mejora sobre la asistencia sanitaria a través del reconocimiento de patrones del ámbito clínico. El tratamiento de imágenes en conjunto de la IA es capaz de visualizar problemas que a simple vista humana no sería posible, a menos que se use instrumentación de vanguardia.

Los campos abarcados sobre el trabajo, son principalmente, el diseño web para la creación de una aplicación web que pueda ser utilizado por un ambiente médico que capacite de visión directa sobre el problema a gestionar (en este caso se trata de las células germinales femeninas), el tratamiento de imágenes, para conseguir obtener nuevos datos de los ovocitos, como son las dimensiones que estos tienen, y finalmente, la IA, para el desarrollo de un modelo predictivo de la calidad de los ovocitos a tratar.

Como se ha observado en el capítulo 5, aún queda mucho trabajo en la búsqueda del modelo perfecto, el cual tenga una fiabilidad aproximada del 90 %, el cual se aleja del obtenido (30 %). Se ha probado con muchos modelos diferentes, parámetros diferentes, y los resultados medios han rondado sobre este porcentaje, lo cual, se asocia a la necesidad de tener una base de datos mucho más extensa la cual permita tener mayor variabilidad de ejemplos de cada tipo de ovocito. Esto permite que el modelo detecte la diferencia entre ovocitos de distinta calidad.

Como propuestas de futuro, se propone la habilitación de mayor cantidad de datos por parte del IVI, incluyendo mayor número de imágenes de ovocitos con sus respectivas características y calificaciones, incluyendo la perspectiva y anotaciones de más embriólogos para que cada conjunto de imagen y características visualizadas, tenga diferentes puntos de vista para que no se asocie un ovocito característico a una calidad en específica, así aumentando la capacidad de aprendizaje de la red. Además, se propone el entrenamiento de una red que no necesite de datos de características ovocitarias como entrada, sino que el propio modelo obtenga estos posibles dismorfismos.

Bibliografía

- [1] IVI. *Instituto Valenciano de Infertilidad (IVI)*. <https://ivi.es/>. Página web oficial del Instituto Valenciano de Infertilidad. 2024.
- [2] CVB Lab. *CVB Lab - Computer Vision and Biometrics Lab*. <https://www.cvblab.webs.upv.es/>. Página web del Laboratorio de Visión por Computador y Biometría de la UPV. 2024.
- [3] IVI. *Fecundación in vitro (FIV)*. <https://ivi.es/tratamientos-reproduccion-asistida/fecundacion-in-vitro/>. Información sobre FIV en la página web del Instituto Valenciano de Infertilidad. 2024.
- [4] CVB Lab - Computer Vision and Biometrics Lab. *OVOVITRIF*. <https://www.cvblab.webs.upv.es/portfolio/ovovitrif-2/>. Consultado el 17 de junio de 2024. 2024.
- [5] Universidad Complutense de Madrid. *Definiciones de la embriología*. Consultado el 2 de julio de 2024. 2013. URL: <https://www.ucm.es/data/cont/docs/465-2013-08-22-A7%20EMBRIOLOGIA.pdf>.
- [6] Reproducción Asistida ORG. *Las técnicas de reproducción asistida*. <https://www.reproduccionasistida.org/las-tecnicas-de-reproduccion-asistida/>. Accessed: 2024-07-17. 2024.
- [7] Reproducción Asistida ORG. *Desarrollo embrionario*. Consultado el 2 de julio de 2024. 2024. URL: <https://www.reproduccionasistida.org/desarrollo-embrionario/>.
- [8] David K. Gardner. «Human Embryo Development and Assessment of Viability». En: *Encyclopedia of Reproduction (Second Edition)*. Ed. por Michael K. Skinner. Second Edition. Oxford: Academic Press, 2018, págs. 176-185. ISBN: 978-0-12-815145-7. DOI: <https://doi.org/10.1016/B978-0-12-801238-3.64857-2>. URL: <https://www.sciencedirect.com/science/article/pii/B9780128012383648572>.
- [9] Instituto Valenciano de Infertilidad (IVI). *Nuevo método FIV con análisis genético*. Consultado el 2 de julio de 2024. 2024. URL: <https://ivi.es/tratamientos-reproduccion-asistida/fiv-genetic/>.
- [10] AISAFIV. *Ovocito secundario: sus partes*. <https://aisafiv.com/es/blog/ovocito-secundario-sus-partes/>. Accessed: 2024-07-17. 2024.
- [11] ASEBIR. *La doble cara de la zona pelúcida*. <https://revista.asebir.com/la-doble-cara-de-la-zona-pelucida/>. Accessed: 2024-07-17. 2024.
- [12] Reproducción Asistida ORG. *Corpúsculo Polar*. <https://www.reproduccionasistida.org/corpusculo-polar/>. Accessed: 2024-07-17. 2024.

-
- [13] Eva Fertility Clinics. *Clasificación de los ovocitos en una FIV/ICSI*. Consultado el 2 de julio de 2024. 2024. URL: <https://www.evafertilityclinics.es/blog/clasificacion-de-los-ovocitos-en-una-fivicsi/>.
- [14] Instituto Valenciano de Infertilidad (IVI). *Óvulos: ¿calidad o cantidad?* Consultado el 2 de julio de 2024. 2024. URL: <https://ivi.es/blog/ovulos-calidad-o-cantidad/>.
- [15] ASEBIR. *Criterios ASEBIR de Valoración Morfológica de Oocitos, Embriones Tempranos y Blastocistos Humanos*. 3ª. Consultado el 2 de julio de 2024. Madrid: ASEBIR, 2015. URL: <https://www.ucm.es/data/cont/docs/465-2013-08-22-A7%20EMBRIOLOGIA.pdf>.
- [16] ASEBIR. «Criterios ASEBIR de Valoración Morfológica de Oocitos, Embriones Tempranos y Blastocistos Humanos». En: 3ª. Consultado el 2 de julio de 2024. Madrid: ASEBIR, 2015. Cap. 1.1. Valoración morfológica del oocito. D+0, págs. 9-20. URL: <https://www.ucm.es/data/cont/docs/465-2013-08-22-A7%20EMBRIOLOGIA.pdf>.
- [17] Victoria Invitro. *Valoración del Ovocito D+0*. Consultado el 3 de julio de 2024. 2024. URL: <https://www.victoriainvitro.com/valoracion-del-ovocito-d-0/>.
- [18] Alberto Stracuzzi et al. «Visco- and poroelastic contributions of the zona pellucida to the mechanical response of oocytes». En: *Biomechanics and Modeling in Mechanobiology* 20 (feb. de 2021). DOI: 10.1007/s10237-020-01414-4.
- [19] IBM. *Artificial Intelligence (AI)*. Consultado el 3 de julio de 2024, sección sobre Aplicaciones de la Inteligencia Artificial. 2024. URL: <https://www.ibm.com/topics/artificial-intelligence>.
- [20] IBM. *Artificial Intelligence (AI)*. Consultado el 3 de julio de 2024, sección sobre Aplicaciones de la Inteligencia Artificial. Deep learning vs. machine learning. 2024. URL: <https://www.ibm.com/topics/artificial-intelligence>.
- [21] A. Esteva, A. Robicquet, B. Ramsundar et al. «A guide to deep learning in health-care». En: *Nature Medicine* 25 (2019). Consultado el 3 de julio de 2024, págs. 24-29. DOI: 10.1038/s41591-018-0316-z. URL: <https://www.nature.com/articles/s41591-018-0316-z#Sec3>.
- [22] Geert Litjens et al. «A survey on deep learning in medical image analysis». En: *Medical Image Analysis* 42 (2017), págs. 60-88. ISSN: 1361-8415. DOI: <https://doi.org/10.1016/j.media.2017.07.005>. URL: <https://www.sciencedirect.com/science/article/pii/S1361841517301135>.
- [23] Tamim Alsuliman, Dania Humaidan y Layth Sliman. «Machine learning and artificial intelligence in the service of medicine: Necessity or potentiality?» En: *Current Research in Translational Medicine* 68.4 (2020), págs. 245-251. ISSN: 2452-3186. DOI: <https://doi.org/10.1016/j.retram.2020.01.002>. URL: <https://www.sciencedirect.com/science/article/pii/S2452318620300192>.
- [24] Jullin Fjeldstad et al. «An artificial intelligence tool predicts blastocyst development from static images of fresh mature oocytes». En: *Reproductive BioMedicine Online* 48.6 (2024), pág. 103842. ISSN: 1472-6483. DOI: <https://doi.org/10.1016/j.rbmo.2024.103842>. URL: <https://www.sciencedirect.com/science/article/pii/S1472648324000312>.
-

- [25] ScienceDirect. *Multilayer Perceptrons*. <https://bit.ly/3zKkasf>. Consultado el 3 de julio de 2024.
- [26] Tim Menzies et al. «Chapter 24 - Using Goals in Model-Based Reasoning». En: *Sharing Data and Models in Software Engineering*. Ed. por Tim Menzies et al. Boston: Morgan Kaufmann, 2015, págs. 321-353. ISBN: 978-0-12-417295-1. DOI: <https://doi.org/10.1016/B978-0-12-417295-1.00024-2>. URL: <https://www.sciencedirect.com/science/article/pii/B9780124172951000242>.
- [27] Gousia Habib y Shaima Qureshi. «Optimization and acceleration of convolutional neural networks: A survey». En: *Journal of King Saud University - Computer and Information Sciences* 34.7 (2022), págs. 4244-4268. ISSN: 1319-1578. DOI: <https://doi.org/10.1016/j.jksuci.2020.10.004>. URL: <https://www.sciencedirect.com/science/article/pii/S1319157820304845>.
- [28] Santosh Kumar Singh, Arun Kumar Tiwari y H.K. Paliwal. «A state-of-the-art review on the utilization of machine learning in nanofluids, solar energy generation, and the prognosis of solar power». En: *Engineering Analysis with Boundary Elements* 155 (2023), págs. 62-86. ISSN: 0955-7997. DOI: <https://doi.org/10.1016/j.enganabound.2023.06.003>. URL: <https://www.sciencedirect.com/science/article/pii/S0955799723003119>.
- [29] Kalidas Yeturu. «Chapter 3 - Machine learning algorithms, applications, and practices in data science». En: *Principles and Methods for Data Science*. Ed. por Arni S.R. Srinivasa Rao y C.R. Rao. Vol. 43. Handbook of Statistics. Elsevier, 2020, págs. 81-206. DOI: <https://doi.org/10.1016/bs.host.2020.01.002>. URL: <https://www.sciencedirect.com/science/article/pii/S0169716120300225>.
- [30] Qingchen Zhang et al. «A survey on deep learning for big data». En: *Information Fusion* 42 (2018), págs. 146-157. ISSN: 1566-2535. DOI: <https://doi.org/10.1016/j.inffus.2017.10.006>. URL: <https://www.sciencedirect.com/science/article/pii/S1566253517305328>.
- [31] D. Devi, S. Sophia y S.R. Boselin Prabhu. «Chapter 4 - Deep learning-based cognitive state prediction analysis using brain wave signal». En: *Cognitive Computing for Human-Robot Interaction*. Ed. por Mamta Mittal, Rajiv Ratn Shah y Sudipta Roy. Cognitive Data Science in Sustainable Computing. Academic Press, 2021, págs. 69-84. ISBN: 978-0-323-85769-7. DOI: <https://doi.org/10.1016/B978-0-323-85769-7.00017-3>. URL: <https://www.sciencedirect.com/science/article/pii/B9780323857697000173>.
- [32] Evgeny A. Smirnov, Denis M. Timoshenko y Serge N. Andrianov. «Comparison of Regularization Methods for ImageNet Classification with Deep Convolutional Neural Networks». En: *AASRI Procedia* 6 (2014). 2nd AASRI Conference on Computational Intelligence and Bioinformatics, págs. 89-94. ISSN: 2212-6716. DOI: <https://doi.org/10.1016/j.aasri.2014.05.013>. URL: <https://www.sciencedirect.com/science/article/pii/S2212671614000146>.
- [33] Edwin Malahina et al. «Teachable Machine: Optimization of Herbal Plant Image Classification Based on Epoch Value, Batch Size and Learning Rate». En: *Journal of Applied Data Sciences* 5.2 (2024), págs. 532-545. DOI: 10.47738/jads.v5i2.206. URL: <https://bright-journal.org/Journal/index.php/JADS/article/view/206>.

-
- [34] Yingjie Tian et al. «Recent advances on loss functions in deep learning for computer vision». En: *Neurocomputing* 497 (2022), págs. 129-158. ISSN: 0925-2312. DOI: <https://doi.org/10.1016/j.neucom.2022.04.127>. URL: <https://www.sciencedirect.com/science/article/pii/S0925231222005239>.
- [35] Jainy Sachdeva et al. «Resolving Autism Spectrum Disorder (ASD) through Brain Topologies using fMRI Dataset with Multi-Layer Perceptron (MLP)». En: *Psychiatry Research: Neuroimaging* (2024), pág. 111858. ISSN: 0925-4927. DOI: <https://doi.org/10.1016/j.psychresns.2024.111858>. URL: <https://www.sciencedirect.com/science/article/pii/S0925492724000817>.
- [36] Roy Thomas Fielding. «Architectural Styles and the Design of Network-based Software Architectures». Tesis doct. University of California, Irvine, 2000.
- [37] Cloudflare. *¿Qué es el Protocolo de Transferencia de Hipertexto (HTTP)?* <https://www.cloudflare.com/es-es/learning/ddos/glossary/hypertext-transfer-protocol-http/>. Consultado: 15-07-2024. 2024.
- [38] W3Schools. *SOAP Tutorial*. https://www.w3schools.com/xml/xml_soap.asp. Consultado: 15-07-2024. 2024.
- [39] Red Hat. *¿Cuál es la diferencia entre SOAP y REST?* <https://www.redhat.com/es/topics/integration/whats-the-difference-between-soap-rest>. Consultado: 15-07-2024. 2024.
- [40] Cesare Pautasso, Olaf Zimmermann y Frank Leymann. «RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision». En: *Proceedings of the 17th International Conference on World Wide Web*. ACM. 2008, págs. 805-814.
- [41] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Inc., 2015.
- [42] James Lewis y Martin Fowler. «Microservices: a definition of this new architectural term». En: *Martin Fowler's Blog* (2014). URL: <https://martinfowler.com/articles/microservices.html>.
- [43] Atlassian. *Advantages of microservices and disadvantages to know*. Accessed: 2024-07-15. 2024. URL: <https://www.atlassian.com/microservices/cloud-computing/advantages-of-microservices>.
- [44] Dijiang Huang y Huijun Wu. «Chapter 2 - Virtualization». En: *Mobile Cloud Computing*. Ed. por Dijiang Huang y Huijun Wu. Morgan Kaufmann, 2018, págs. 31-64. ISBN: 978-0-12-809641-3. DOI: <https://doi.org/10.1016/B978-0-12-809641-3.00003-X>. URL: <https://www.sciencedirect.com/science/article/pii/B978012809641300003X>.
- [45] Javad Dogani, Reza Namvar y Farshad Khunjush. «Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey». En: *Computer Communications* 209 (2023), págs. 120-150. ISSN: 0140-3664. DOI: <https://doi.org/10.1016/j.comcom.2023.06.010>. URL: <https://www.sciencedirect.com/science/article/pii/S0140366423002086>.
-

- [46] Karamjeet Kaur, Veenu Mangat y Krishan Kumar. «A review on Virtualized Infrastructure Managers with management and orchestration features in NFV architecture». En: *Computer Networks* 217 (2022), pág. 109281. ISSN: 1389-1286. DOI: <https://doi.org/10.1016/j.comnet.2022.109281>. URL: <https://www.sciencedirect.com/science/article/pii/S1389128622003395>.
- [47] AWS. *The Difference Between Docker Images and Containers*. Accessed: 2024-07-15. 2024. URL: <https://aws.amazon.com/es/compare/the-difference-between-docker-images-and-containers/#:~:text=Docker%20images%20are%20read%20only,for%20an%20application%20to%20run..>
- [48] Docker Documentation. *Use volumes*. <https://docs.docker.com/engine/storage/volumes/>. Accessed: 2024-08-19.
- [49] Sonali Jain. *Docker Components: Understanding How Web Applications Work with Docker*. Accessed: 2024-07-15. 2020. URL: <https://medium.com/@sonalijain0605/docker-components-41e847893d8a>.
- [50] Adictos al Trabajo. *Despliegue de aplicaciones con Docker Compose*. <https://adictosaltrabajo.com/2022/12/19/despliegue-de-aplicaciones-con-docker-compose/>. Accessed: 2024-08-19.
- [51] Python Software Foundation. *Applications for Python*. Accessed: 2024-07-15. 2024. URL: <https://www.python.org/about/apps/>.
- [52] Flask Documentation. *Welcome to Flask — Flask Documentation (3.0.x)*. Accessed: 2024-07-15. 2024. URL: <https://flask.palletsprojects.com/en/3.0.x/>.
- [53] PythonBasics.org. *What is Flask Python?* Accessed: 2024-07-15. 2024. URL: <https://pythonbasics.org/what-is-flask-python/>.
- [54] Pandas Development Team. *Pandas: Python Data Analysis Library*. Accessed: 2024-07-15. 2024. URL: <https://pandas.pydata.org/>.
- [55] MDN Web Docs. *HTML: Lenguaje de Marcado de Hipertexto*. Accessed: 2024-07-15. 2024. URL: <https://developer.mozilla.org/es/docs/Web/HTML>.
- [56] MDN Web Docs. *CSS: Hojas de Estilo en Cascada*. Accessed: 2024-07-15. 2024. URL: <https://developer.mozilla.org/es/docs/Web/CSS>.
- [57] MDN Web Docs. *JavaScript: Lenguaje de Programación*. Accessed: 2024-07-15. 2024. URL: <https://developer.mozilla.org/es/docs/Web/JavaScript>.
- [58] OpenWebinars. *¿Qué es Dash?* <https://openwebinars.net/blog/que-es-dash/>. Consultado el 3 de julio de 2024.
- [59] Wikipedia. *Histograma*. <https://es.wikipedia.org/wiki/Histograma>. Consultado el 3 de julio de 2024.
- [60] Lijun Ding y Ardeshir Goshtasby. «On the Canny edge detector». En: *Pattern Recognition* 34.3 (2001), págs. 721-725. ISSN: 0031-3203. DOI: [https://doi.org/10.1016/S0031-3203\(00\)00023-6](https://doi.org/10.1016/S0031-3203(00)00023-6). URL: <https://www.sciencedirect.com/science/article/pii/S0031320300000236>.

- [61] X. Jane Jiang y Paul J. Scott. «Chapter 11 - Characterization of free-form structured surfaces». En: *Advanced Metrology*. Ed. por X. Jane Jiang y Paul J. Scott. Academic Press, 2020, págs. 281-317. ISBN: 978-0-12-821815-0. DOI: <https://doi.org/10.1016/B978-0-12-821815-0.00011-3>. URL: <https://www.sciencedirect.com/science/article/pii/B9780128218150000113>.
- [62] Rice University OwlNet. *Laplacian Edge Detection*. <https://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/laplacian.html>. Accessed: 2024-08-19.
- [63] OpenCV. *OpenCV: Open Source Computer Vision Library*. Accessed: 2024-07-15. 2024. URL: <https://opencv.org/>.
- [64] OpenCV. *Contours : Getting Started - Contour Features*. Accessed: 2024-07-15. 2024. URL: https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html.
- [65] TranslatorsCafe. *Convertir Micrómetros a Píxeles*. Último acceso: 15 de julio de 2024. 2024. URL: <https://www.translatorscafe.com/unit-converter/es-ES/length/13-110/micrometer-pixel/>.
- [66] PyTorch. *PyTorch*. Último acceso: 15 de julio de 2024. 2024. URL: <https://pytorch.org/>.
- [67] Ultralytics. *Ultralytics*. Último acceso: 15 de julio de 2024. 2024. URL: <https://www.ultralytics.com/es>.
- [68] NVIDIA. *PyTorch*. Último acceso: 15 de julio de 2024. 2024. URL: <https://www.nvidia.com/en-us/glossary/pytorch/#:~:text=PyTorch%20is%20a%20fully%20featured,developers%20to%20learn%20and%20use>.
- [69] Ultralytics. *Ultralytics Explorer API*. <https://learnopencv.com/ultralytics-explorer-api/>. Último acceso: 15 de julio de 2024. 2024.
- [70] Ultralytics. *Predict Mode - Image and Video Formats*. <https://docs.ultralytics.com/modes/predict/#image-and-video-formats>. Accessed: 2024-07-17. 2024.
- [71] scikit-learn developers. *sklearn.metrics.f1_score*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html. Accessed: 2024-07-17.
- [72] EITCA Academy. *How Do We Prepare the Data for Training a CNN Model?* <https://es.eitca.org/artificial-intelligence/eitc-ai-dlptfk-deep-learning-with-python-tensorflow-and-keras/convolutional-neural-networks-cnn/introduction-to-convolutional-neural-networks-cnn/examination-review-introduction-to-convolutional-neural-networks-cnn/how-do-we-prepare-the-data-for-training-a-cnn-model/>. Accessed: 2024-07-17.
- [73] Bootstrap. *Introduction to Bootstrap*. <https://getbootstrap.com/docs/5.3/getting-started/introduction/>. Accessed: 2024-07-17. 2024.
- [74] Bootstrap. *Navbar - How it works*. <https://getbootstrap.com/docs/5.3/components/navbar/#how-it-works>. Accessed: 2024-07-17. 2024.
- [75] Bootstrap. *Forms overview*. Accessed: 2024-07-17. 2024. URL: <https://getbootstrap.com/docs/5.3/forms/overview/#overview>.
- [76] Bootstrap. *Buttons: Base class*. Accessed: 2024-07-17. 2024. URL: <https://getbootstrap.com/docs/5.3/components/buttons/#base-class>.

- [77] Docker. *Docker Compose Documentation*. Accessed: 2024-07-17. 2024. URL: <https://docs.docker.com/compose/>.
- [78] Docker. *Docker Compose Application Model Documentation*. Accessed: 2024-07-17. 2024. URL: <https://docs.docker.com/compose/compose-application-model/>.