TRABAJO FIN DE GRADO

UCAM
UNIVERSIDAD CATÓLICA
DE MURCIA

ESCUELA UNIVERSITARIA POLITÉCNICA

Ingeniería Informática

SEGMENTACIÓN DE IMÁGENES DE CÉLULAS CERVICOVAGINALES CON APRENDIZAJE PROFUNDO

Oscar David Romero Humber

Directores:

D. Juan Morales García

Dr. *Andrés Bueno Crespo*

Murcia, mayo de 2023

TRABAJO FIN DE GRADO

UCAM
UNIVERSIDAD CATÓLICA
DE MURCIA

ESCUELA UNIVERSITARIA POLITÉCNICA

Ingeniería Informática

SEGMENTACIÓN DE IMÁGENES DE CÉLULAS CERVICOVAGINALES CON APRENDIZAJE PROFUNDO

Oscar David Romero Humber

Directores:

D. Juan Morales García

Dr. *Andrés Bueno Crespo*

Murcia, mayo de 2023

Agradecimientos

A mi familia, por apoyarme en todo momento durante mi vida.

A mis tutores, por introducirme en el mundo de la investigación y la inteligencia artificial.

A mis compañeros de clase, por compartir risas, sudor y lágrimas durante estos cuatro años.

Listado de Abreviatura

IoU, Intersección partida de unión (*Intersection over Union*)

IA, Inteligencia artificial.

FCN, Redes completamente convolucionales (*fully convolutional networks*)

CNN, Redes neuronales convolucionales (*convolutional neural networks*)

ÍNDICE

1. INTRODUCCIÓN **24**
2. JUSTIFICACIÓN **27**
3. MARCO TEÓRICO **28**
3.1. Inteligencia Artificial 28
3.2. Aprendizaje automático y aprendizaje profundo 29
3.3. Redes neuronales artificiales 30
3.4. Redes neuronales convolucionales 32
3.5. Segmentación 37
4. OBJETIVOS **40**
4.1. Objetivo General 40
4.2. Objetivos Específicos 40
5. METODOLOGÍA **41**
5.1. Estimación de recursos y planificación 42
5.2. Valoración de dedicación y coste económico 44
5.3. Tecnologías y herramientas utilizadas en el proyecto 46
5.3.1. Infraestructura de software 46
5.3.2. Infraestructura de hardware 46
6. DESARROLLO DEL PROYECTO **48**
6.1. Sprint 1 49
6.1.1. Segmentación no supervisada con K-Means 49
6.1.2. Generación de Dataset de máscaras con VIA 51
6.1.3. Retrospectiva del sprint 52
6.2. Sprint 2 52
6.2.1. Generación de Dataset de máscaras con Apeer 53
6.2.2. Buscando un modelo de segmentación 54
6.2.3. Retrospectiva del sprint 56
6.3. Sprint 3 56
6.3.1. Pruebas con U-Net 57
6.3.2. Retrospectiva del sprint 61
6.4. Sprint 4 61
6.4.1. Cambio de métricas y más pruebas con U-Net 62
6.4.2. Prueba solo con imágenes de cada clase 64
6.4.3. Retrospectiva del sprint 69
6.5. Sprint 5 70
6.5.1. Comparaciones de U-Net utilizando modelos pre-entrenados 70
6.5.2. Generación del dataset completo 75
6.5.3. Retrospectiva del sprint 76
6.6. Sprint 6 77
6.6.1. Pruebas con dataset por clase y comparación global 77
6.6.2. Retrospectiva del sprint 83
6.7. Sprint 7 83
6.7.1. Pruebas de diferentes modelos de segmentación 83

6.7.2. Retrospectiva del sprint 88
6.8. Sprint 8 88
6.8.1. Clasificación de imágenes con máscaras de segmentación 89
6.8.2. Concatenación de modelos de clasificación para imágenes y máscaras 92
6.8.3. Retrospectiva del sprint 94
6.9. Sprint 9 95
6.9.1. Mejora del modelo de imágenes y máscaras 95
6.9.2. Retrospectiva del sprint 100
6.10. Sprint 10 100
6.10.1. Balanceo de clases 101
6.10.2. Prueba de GAP 102
6.10.3. Concatenación de modelos de segmentación y clasificación para imágenes y máscaras 106
6.10.4. Retrospectiva del sprint 110
7. RESULTADOS 111
7.1. Segmentación semántica 111
7.2. Clasificación de imágenes 114
8. CONCLUSIONES 118
8.1. Objetivos alcanzados 118
8.2. Conclusiones del proyecto 119
9. REFLEXIÓN Y VALORACIÓN PERSONAL 121
10. REFERENCIAS 122
11. ANEXOS 127
11.1. Anexo 1. Explicación de ficheros anexos 127
11.2. Anexo 2. Importar ficheros .ipynb a Google Colab 131

ÍNDICE DE ELEMENTOS GRÁFICOS

ILUSTRACIONES

Ilustración 1.1: Ejemplos procedentes del dataset utilizado: (a) benigna (b) ascus (c) bajogrado (d) altogrado. Fuente: Servicio de Anatomía Patológica del Complejo Hospitalario de Cartagena Santa Lucía-Santa María del Rosell (Cartagena, Murcia). 24

Ilustración 1.2: Ejemplo de célula benigna (a) y máscara segmentada correspondiente (b). Fuente: Elaboración propia 25

Ilustración 3.1: Representación de áreas dentro de IA. Fuente: Elaboración propia 28

Ilustración 3.2: Funciones de activación, (izquierda) función escalón, (derecha) función Sigmoid Fuente: Han et al. (2018)..... 30

Ilustración 3.3: Conexiones y pesos entre neuronas de cada capa en una red neuronal artificial. Fuente: Han et al. (2018) 31

Ilustración 3.4:Gráfica de una función de coste. Fuente: Han et al. (2018) 32

Ilustración 3.6: Representación visual de los primeros cálculos de una capa convolucional. Fuente: Taye. (2023) 34

Ilustración 3.7: Representación visual de diferentes tipos de pooling y sus mapas resultantes. Fuente: Taye. (2023) 36

Ilustración 3.8: Funciones de activación. Fuente: Taye. (2023)..... 36

Ilustración 3.9: Arquitectura simple de CNN para clasificación con 5 capas. Fuente: O'Shea & Nash. (2015)..... 37

Ilustración 3.10: Máscaras de segmentación para segmentación semántica (izquierda) y segmentación de instancias (derecha). Fuente: Sharma (2019). 38

Ilustración 3.11: Estructura típica de una CNN. Fuente: Sharma (2019). 39

Ilustración 3.12: Estructura de una FCN, con capas de pooling en verde y capas de unpooling en naranja. Fuente: Sharma (2019). 39

Ilustración 5.1: Estructura de una FCN, con capas de pooling en verde y capas de unpooling en naranja. Fuente: Cohn, M (2004). 42

Tabla 5.1: Organización de sprints. Fuente: Elaboración propia. 43

Ilustración 5.2: Gráfico de puntos de historia por sprint. Fuente: Elaboración propia. 44

Ilustración 5.3: Gráfico de estimación de horas por sprint. Fuente:

Elaboración propia. 45

Ilustración 5.4: Gráfico de estimación de coste y coste acumulado por sprint. Fuente: Elaboración propia. 45

Ilustración 6.1: Distribución del dataset original. Fuente: Elaboración propia. 48

Tabla 6.1: Organización de sprint 1. Fuente: Elaboración propia. 49

Ilustración 6.2: Células originales (izquierda) y segmentadas (derecha) con K-Means, K=3 (arriba) y K=4 (abajo). Fuente: Elaboración propia. 50

Ilustración 6.3: Células segmentadas entre citoplasma y núcleo. Fuente: Elaboración propia. 51

Ilustración 6.4: Células segmentadas entre citoplasma y núcleo, manualmente (izquierda) y predicción (derecha). Fuente: Elaboración propia. 52

Tabla 6.2: Organización de sprint 2. Fuente: Elaboración propia. 53

Ilustración 6.5: Células originales (izquierda) y segmentadas (derecha). Benigna (arriba) y altogrado (abajo) Fuente: Elaboración propia. 53

Ilustración 6.6: Arquitectura de una FCN. Fuente: Jonathan Long et al. 55

Ilustración 6.7: Arquitectura de U-Net. Fuente: Olaf Ronnenberg et al. . 55

Tabla 6.3: Organización de sprint 3. Fuente: Elaboración propia. 56

Ilustración 6.8: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia. 58

Ilustración 6.9: Gráficas de entrenamiento. (Izquierda) Pérdida. (Derecha) Tasa de aciertos. Fuente: Elaboración propia. 59

Ilustración 6.10: Resultados al segmentar las imágenes del conjunto de prueba. Fuente: Elaboración propia. 59

Ilustración 6.11: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia. 60

Ilustración 6.12: Gráficas de entrenamiento. (Izquierda) Pérdida. (Derecha) Tasa de aciertos. Fuente: Elaboración propia. 60

Ilustración 6.13: Resultados al segmentar las imágenes del conjunto de prueba. Fuente: Elaboración propia. 61

Tabla 6.4: Organización de sprint 4. Fuente: Elaboración propia. 62

Ilustración 6.14: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia. 63

Ilustración 6.15: Métricas de segmentación. Fuente: Elaboración propia. 63

Ilustración 6.16: Distribución de las máscaras segmentadas automáticamente. Fuente: Elaboración propia. 65

Ilustración 6.17: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia. 65

Ilustración 6.18: Métricas de segmentación. Fuente: Elaboración propia. 66

Ilustración 6.19: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia. 67

Ilustración 6.20: Métricas de segmentación. Fuente: Elaboración propia. 67

Ilustración 6.21: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia. 68

Ilustración 6.22: Métricas de segmentación. Fuente: Elaboración propia. 68

Ilustración 6.23: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia. 69

Ilustración 6.24: Métricas de segmentación. Fuente: Elaboración propia. 69

Ilustración 6.25: Gráfico de barras de comparación de resultados de IoU entre modelos entrenados. Fuente: Elaboración propia. 69

Tabla 6.5: Organización de sprint 5. Fuente: Elaboración propia. 70

Ilustración 6.26: IoU durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia. 72

Ilustración 6.27: Pérdida total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia. 73

Ilustración 6.28: Métrica de IoU (arriba). Resultados de segmentación (abajo). (1) Predicción con Resnet34. (2) Predicción con InceptionV3. (3) Predicción con VGG16. (4) Predicción con U-Net. Fuente: Elaboración propia. 73

Ilustración 6.29: Resultados de segmentación. (1) Predicción con Resnet34. (2) Predicción con InceptionV3. (3) Predicción con VGG16. (4) Predicción con U-Net. Fuente: Elaboración propia. 74

Ilustración 6.30: Máscara y célula benigna. Fuente: Elaboración propia. 75

Ilustración 6.31: Máscara y célula bajogrado. Fuente: Elaboración propia. 75

Ilustración 6.32: Máscara y célula ascus Fuente: Elaboración propia. .. 76

Ilustración 6.33: Máscara y célula altogrado. Fuente: Elaboración propia. 76

Tabla 6.6: Organización de sprint 6. Fuente: Elaboración propia. 77

Ilustración 6.34: IoU durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia. 78

Ilustración 6.35: Pérdida total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia. 78

Ilustración 6.36: IoU durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia. 79

Ilustración 6.37: Pérdida total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia. 79

Ilustración 6.38: Resultados de segmentación. Células bajogrado (arriba) y altogrado (abajo) Fuente: Elaboración propia. 81

Ilustración 6.39: Resultados de segmentación. Células altogrado (arriba) y benigna (abajo) Fuente: Elaboración propia. 82

Ilustración 6.40: Métricas con el dataset de prueba manual. Fuente: Elaboración propia. 82

Ilustración 6.41: Métricas con el dataset de entrenamiento manual. Fuente: Elaboración propia. 82

Tabla 6.7: Organización de sprint 7. Fuente: Elaboración propia. 83

Ilustración 6.42: Modelos de Keras U-Net Collection. Fuente: Sha, Y. 2021 84

Ilustración 6.43: Métricas con el dataset de prueba manual. Fuente: Elaboración propia. 85

Ilustración 6.44: Métricas con el dataset de entrenamiento manual. Fuente: Elaboración propia. 85

Ilustración 6.45: Resultados de segmentación. Células bajogrado (arriba) y altogrado (abajo) Fuente: Elaboración propia. 86

Ilustración 6.46: Resultados de segmentación. Células ascus(arriba) y benigna (abajo) Fuente: Elaboración propia. 87

Tabla 6.8: Organización de sprint 8. Fuente: Elaboración propia. 88

Ilustración 6.47: Pérdida total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia. 89

Ilustración 6.48: Tasa de aciertos total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo). Fuente: Elaboración propia. 90

Ilustración 6.49: Matriz de confusión de entrenamiento con imágenes. Fuente: Elaboración propia. 90

Ilustración 6.50: Matriz de confusión de entrenamiento con máscaras. Fuente: Elaboración propia. 91

Ilustración 6.51: Modelo compuesto para clasificación. Fuente: Elaboración propia 92

Ilustración 6.52: Pérdida total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia. 93

Ilustración 6.53: Tasa de aciertos total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia. 93

Ilustración 6.54: Matriz de confusión de entrenamiento con imágenes y máscaras. Fuente: Elaboración propia. 94

Tabla 6.9: Organización de sprint 9. Fuente: Elaboración propia. 95

Ilustración 6.55: Matriz de confusión de entrenamiento con imágenes y máscaras. Fuente: Elaboración propia. 96

Ilustración 6.57: Matriz de confusión de entrenamiento con imágenes y máscaras. Fuente: Elaboración propia. 98

Ilustración 6.58: Matriz de confusión de entrenamiento con imágenes y máscaras. Fuente: Elaboración propia. 99

Ilustración 6.59: Tasas de acierto por prueba en sprint 9. Fuente: Elaboración propia 99

Tabla 6.10: Organización de sprint 10. Fuente: Elaboración propia. 101

Ilustración 6.61: Pérdida total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia. 102

Ilustración 6.62: Tasa de aciertos total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia. 103

Ilustración 6.63: Matriz de confusión de entrenamiento con imágenes y

máscaras, clasificador tradicional. Fuente: Elaboración propia. 103

Ilustración 6.64: Matriz de confusión de entrenamiento con imágenes y máscaras, clasificador GAP. Fuente: Elaboración propia. 104

Ilustración 6.65: Matriz de confusión 2 clases de entrenamiento con imágenes y máscaras, clasificador tradicional. Fuente: Elaboración propia. . 104

Ilustración 6.66: Matriz de confusión 2 clases de entrenamiento con imágenes y máscaras, clasificador GAP. Fuente: Elaboración propia. 105

Ilustración 6.67: Tasas de acierto por prueba en sprint 10. Fuente: Elaboración propia 105

Ilustración 6.68: Modelo compuesto para segmentación y clasificación. Fuente: Elaboración propia 106

Ilustración 6.69: Matriz de confusión de entrenamiento con imágenes y máscaras, clasificador tradicional. Fuente: Elaboración propia. 107

Ilustración 6.70: Matriz de confusión de entrenamiento con imágenes y máscaras, clasificador GAP. Fuente: Elaboración propia. 108

Ilustración 6.71: Matriz de confusión para cuatro clases de entrenamiento con imágenes y máscaras, clasificador GAP. Fuente: Elaboración propia. 109

Ilustración 6.72: Matriz de confusión de dos clases de entrenamiento con imágenes y máscaras, clasificador GAP. Fuente: Elaboración propia. 110

Ilustración 7.1: Métricas globales con el dataset manual. Fuente: Elaboración propia. 112

Ilustración 7.2: Precisión por clase con el dataset de entrenamiento manual. Fuente: Elaboración propia. 112

Ilustración 7.3: Cobertura por clase con el dataset de entrenamiento manual. Fuente: Elaboración propia. 113

Ilustración 7.4: F-1 Score por clase con el dataset de entrenamiento manual. Fuente: Elaboración propia. 113

Ilustración 7.5: Métricas globales para cuatro clases . Fuente: Elaboración propia. 115

Ilustración 7.6: Tasa de acierto por clase para cuatro clases . Fuente: Elaboración propia. 115

Ilustración 7.7: Precisión por clase para cuatro clases . Fuente: Elaboración propia. 115

Ilustración 7.8: Métricas globales para dos clases . Fuente: Elaboración propia. 116

Ilustración 7.9: Tasa de acierto por clase para dos clases . Fuente:

Elaboración propia. 116

Ilustración 7.10: Precisión por clase para dos clases . Fuente: Elaboración propia. 117

Ilustración 11.1: Captura de Google Colab pantalla inicial . Fuente: Elaboración propia. 131

Ilustración 11.2: Captura de Google Colab cambio de entorno de ejecución . Fuente: Elaboración propia. 132

Ilustración 11.3: Captura de Google Colab cambio de entorno de ejecución . Fuente: Elaboración propia. 132

TABLAS

Tabla 5.1: Organización de sprints. Fuente: Elaboración propia. 43
Tabla 6.1: Organización de sprint 1. Fuente: Elaboración propia. 49
Tabla 6.2: Organización de sprint 2. Fuente: Elaboración propia. 53
Tabla 6.3: Organización de sprint 3. Fuente: Elaboración propia. 56
Tabla 6.4: Organización de sprint 4. Fuente: Elaboración propia. 62
Tabla 6.5: Organización de sprint 5. Fuente: Elaboración propia. 70
Tabla 6.6: Organización de sprint 6. Fuente: Elaboración propia. 77
Tabla 6.7: Organización de sprint 7. Fuente: Elaboración propia. 83
Tabla 6.8: Organización de sprint 8. Fuente: Elaboración propia. 88
Tabla 6.9: Organización de sprint 9. Fuente: Elaboración propia. 95
Tabla 6.10: Organización de sprint 10. Fuente: Elaboración propia. 101

RESUMEN

Este estudio explora el uso de técnicas de aprendizaje profundo para la segmentación y clasificación de imágenes de células cervicales. Las imágenes provienen de citologías de cuatro clases de anormalidad según el sistema Bethesda 2014. Los métodos utilizados son redes completamente convolucionales para la segmentación semántica, que etiquetan cada píxel como núcleo, citoplasmas o fondo, y redes neuronales convolucionales para la clasificación de imágenes, que asignan cada imagen de célula a una de las cuatro clases. Los resultados muestran que las máscaras de segmentación mejoran el rendimiento de la clasificación. Los mejores modelos son U-Net para segmentación y una concatenación de dos redes neuronales convolucionales para clasificación. El estudio concluye que la inteligencia artificial puede ayudar a patólogos en el diagnóstico de cáncer cervical al proporcionar una segmentación y clasificación precisas y eficientes de las imágenes de células.

Palabras claves:

Inteligencia Artificial, Aprendizaje Profundo, Visión Artificial, Redes Neuronales Convolucionales, Redes Completamente Convolucionales, Segmentación de Imágenes, Segmentación Semántica, Clasificación de Imágenes

ABSTRACT

This study explores the use of deep learning techniques for the segmentation and classification of cervical cell images. The images come from cytologies of four classes of abnormality according to the Bethesda 2014 system. The methods used are fully convolutional networks (FCN) for semantic segmentation, which labels each pixel as nucleus, cytoplasm or background, and convolutional neural networks (CNN) for image classification, which assigns each cell image to one of the four classes. The results show that the segmentation masks improve the classification performance. The best models are U-Net for segmentation and a concatenation of two CNNs for classification. The study concludes that artificial intelligence can assist pathologists in diagnosing cervical cancer by providing accurate and efficient segmentation and classification of cell images.

Keywords:

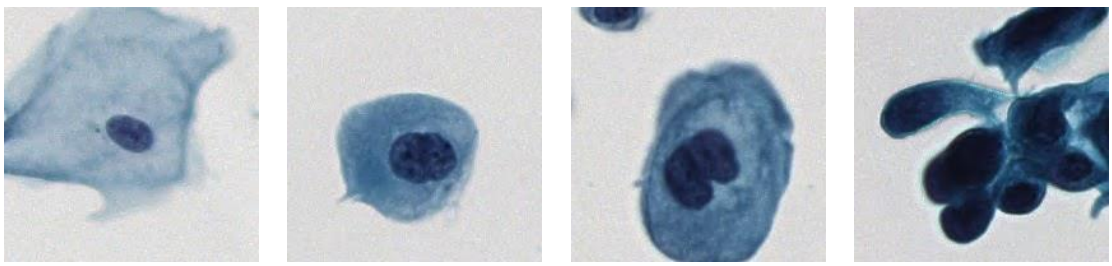
Artificial Intelligence, Deep Learning, Computer Vision, Convolutional Neural Networks, Fully Convolutional Networks, Image Segmentation, Semantic Segmentation, Image Classification

1. INTRODUCCIÓN

La meta de este Trabajo de Fin de Grado, es aportar a la detección de cáncer de cérvix a través de Inteligencia Artificial (IA), concretamente utilizando redes neuronales profundas en los ámbitos de la segmentación de imágenes y en la clasificación de imágenes, para células provenientes de citologías.

Las imágenes médicas por utilizar proceden del Servicio de Anatomía Patológica del Complejo Hospitalario Universitario Santa Lucía-Santa María del Rosell (Cartagena, Murcia). Provienen de la segmentación automática de citologías para obtener células individuales, clasificadas en incremental grado de anormalidad, en 4 clases, siguiendo el sistema Bethesda 2014 (Nayar & Wilbur, 2015):

- Células normales o benignas: Células sin ningún tipo de anormalidad. (Ilustración 1.1 a)
- ASC-US o células ascus: Células con anomalías de significado no determinado. (Ilustración 1.1 b)
- L-SIL o células bajogrado: Células con anomalías leves, que indican una infección concurrente por VPH. (Ilustración 1.1 c)
- H-SIL o células altogrado: Células con anomalías muy marcadas, comúnmente asociadas a la infección persistente de VPH. (Ilustración 1.1 d)



(a) (b) (c) (d)

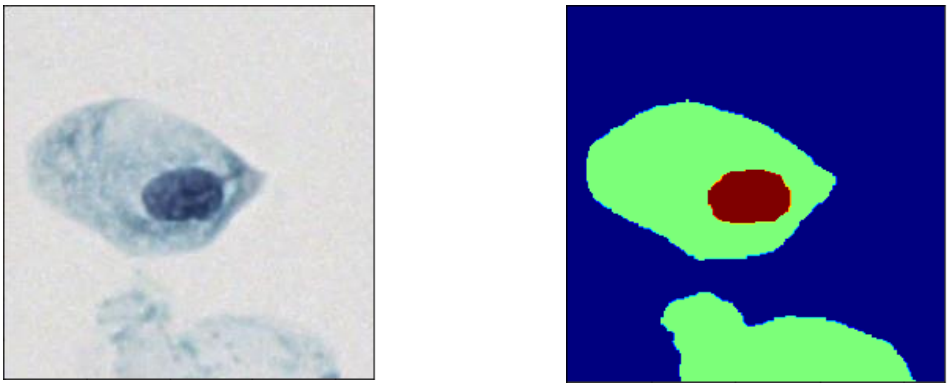
Ilustración 1.1: Ejemplos procedentes del dataset utilizado: (a) benigna (b) ascus (c) bajogrado (d) altogrado. Fuente: Servicio de Anatomía Patológica del Complejo Hospitalario de Cartagena Santa Lucía-Santa María del Rosell (Cartagena, Murcia).

Dado que algunas de las anomalías de las células consisten en el tamaño del núcleo con respecto al citoplasma, la utilización de redes neuronales para

segmentación semántica posee el potencial de asistir a otras redes neuronales para la clasificación de imágenes.

La segmentación semántica se basa en indicar en cada píxel de la imagen de entrada qué es, en este caso se indicará si cada píxel es parte del núcleo, citoplasma o fondo de la imagen, lo cual define la problemática a segmentación semántica multiclase. Posterior a la identificación, el modelo de segmentación genera una máscara segmentada correspondiente a la imagen de entrada. (Taye, 2023)

En la Ilustración 1.2 se puede observar una imagen de entrada de una célula benigna y su correspondiente máscara segmentada, en rojo el núcleo, en verde el citoplasma y en azul el fondo de la imagen.



(a) (b)

Ilustración 1.2: Ejemplo de célula benigna (a) y máscara segmentada correspondiente (b). Fuente: Elaboración propia

La capacidad de identificar el tamaño del núcleo y del citoplasma puede ayudar al diagnóstico realizado por patólogos profesionales. Esto se debe a que permite calcular la proporción de núcleo con citoplasma que posee cada célula.

La segmentación por su cuenta no es suficiente para clasificar las imágenes en las cuatro categorías previamente mencionadas, para esto es necesaria una red neuronal para la clasificación de imágenes.

La red neuronal para la clasificación de imágenes consiste en recibir una imagen de entrada y generar una predicción entre las clases definidas, en este caso la salida sería un número entero entre 0 y 3 correspondiente a las clases de la Ilustración 1.1.

Las técnicas de segmentación y de clasificación de imágenes están implementadas en redes neuronales profundas, las que requieren un dataset amplio para el entrenamiento, así aprenden las características necesarias.

2. JUSTIFICACIÓN

Los progresos en los algoritmos de *computer vision* o visión artificial tienen un amplio espectro de aplicaciones en diversos campos, incluyendo la imagenología biomédica. El uso de las redes neuronales convolucionales para la detección y segmentación tiene implicaciones significativas en el ámbito médico, donde las tareas que suelen ser realizadas de manera laboriosa por investigadores pueden ser sustituidas por sistemas automatizados.

La segmentación precisa requiere conocimientos especializados y las imágenes pueden contener hasta decenas de miles de núcleos a ser etiquetados manualmente por un experto, un trabajo muy costoso en tiempo.

Esto representa una aplicación idónea para la visión artificial, ya que los algoritmos pueden ser entrenados para equiparar la precisión de los expertos, pero etiquetando múltiples imágenes en una fracción del tiempo.

Tradicionalmente, la segmentación de núcleos se ha realizado con métodos clásicos de visión artificial, como el algoritmo de contornos activos, y con algoritmos de aprendizaje automático, como el clasificador *Random Forest*. Sin embargo, Caicedo et al. (2019) demuestran que los métodos de aprendizaje profundo realizan menos errores de segmentación comparados a los métodos clásicos, y con la creciente cantidad de modelos de segmentación y librerías de código abierto, las redes neuronales se presentan como una alternativa viable para su uso diario en laboratorios.

A pesar de las ventajas, el entrenamiento e implementación de estos modelos de aprendizaje profundo para alcanzar la precisión de un experto requiere conocimientos profundos de la problemática y del área de segmentación por aprendizaje profundo, además de un dataset amplio, de calidad y representativo para aportar resultados verdaderamente útiles en un entorno real.

3. MARCO TEÓRICO

Este TFG abarca dos áreas amplias de la IA, la segmentación de imágenes utilizando técnicas de aprendizaje profundo y la clasificación de imágenes con redes neuronales convolucionales, en este apartado se explican los conceptos más importantes de estos campos.

3.1. Inteligencia Artificial

La inteligencia artificial es un campo que busca aprovechar la capacidad de procesamiento de un ordenador para simular la habilidad de procesamiento de datos de un ser humano. IBM (n.d.) define la inteligencia artificial como “un campo que combina la ciencia informática y los conjuntos de datos robustos para permitir la resolución de problemas.”

El campo de la IA es amplio, engloba también los campos del aprendizaje automático y el aprendizaje profundo, como es representado en la Ilustración 3.1.

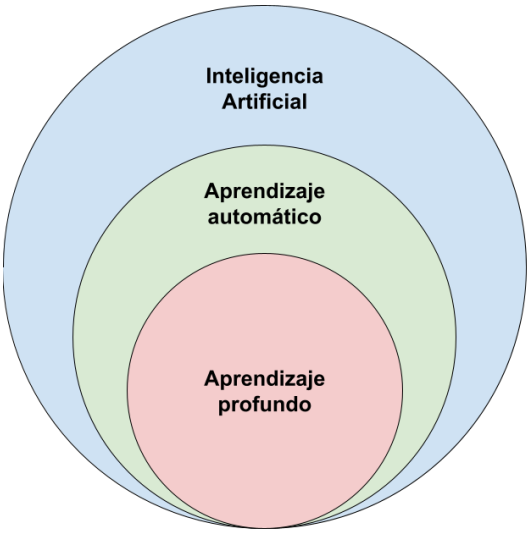


Ilustración 3.1: Representación de áreas dentro de IA. Fuente: Elaboración propia

La IA posee un campo de aplicación prácticamente ilimitado, mientras que existan datos digitalizados sobre los cuales trabajar, lo cual la hace una herramienta interdisciplinar indispensable en el mundo moderno.

Entre los ejemplos de IA no relacionados con el aprendizaje automático o el aprendizaje profundo están los sistemas expertos, los cuales utilizan una base de conocimientos y un motor de inferencia para imitar el razonamiento humano en un campo específico (Schreiber, 2000).

3.2. Aprendizaje automático y aprendizaje profundo

De la misma manera que es más fácil entender la diferencia entre un gato y un perro utilizando ejemplos, en vez de formular una definición precisa que lo explique, la rama de la IA, de *Machine Learning* o aprendizaje automático, busca aprender automáticamente de manera iterativa relaciones significativas y patrones provenientes de ejemplos y observaciones, en vez de programar este conocimiento en un algoritmo (Janiesch et al., 2021).

El aprendizaje automático se puede diferenciar en tres tipos, según IBM (n.d.):

- Aprendizaje supervisado: Aprendizaje en el que se proporciona a un algoritmo un conjunto de datos etiquetados para entrenar, el algoritmo utiliza estos datos para aprender a predecir la etiqueta de nuevos datos no vistos. Ejemplos comunes incluyen la clasificación y la regresión.
- Aprendizaje no supervisado: Aprendizaje en el que no se proporcionan etiquetas a los datos de entrenamiento, en su lugar, el algoritmo busca patrones y relaciones en los datos por sí mismo. Ejemplos comunes incluyen la clusterización y la reducción de dimensionalidad.
- Aprendizaje por refuerzo: Aprendizaje en el que un agente aprende a tomar decisiones en un entorno mediante la interacción con él y recibiendo recompensas o castigos por sus acciones, el objetivo del agente es aprender una política que maximice la recompensa total a largo plazo.

El *Deep learning* o aprendizaje profundo es un subcampo del aprendizaje automático que utiliza redes neuronales artificiales para modelar problemas complejos (Janiesch et al., 2021). Para muchas aplicaciones, los modelos de aprendizaje profundo superan a los modelos de aprendizaje automático no profundos y a los enfoques tradicionales de análisis de datos.

3.3. Redes neuronales artificiales

Para comprender las redes neuronales artificiales, es necesario comprender la unidad más básica que conforma la red, una neurona artificial. Han et al. (2018) definen una neurona artificial desde nuestro conocimiento de una neurona biológica, estas reciben datos de entrada, realizan cierto procesamiento y generan una salida. Sin embargo, la salida no es constante; la salida es generada cuando la entrada excede cierto umbral. La función que recibe una señal de entrada y genera una señal de salida después de un valor umbral es conocida como función de activación. Como se puede observar en la Ilustración 3.2, el valor de entrada define el valor de salida según la función utilizada.

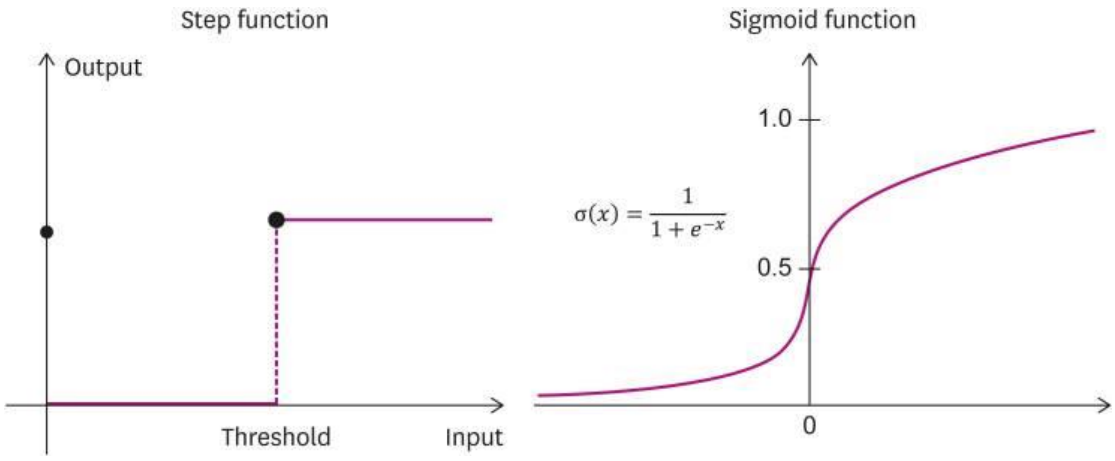


Ilustración 3.2: Funciones de activación, (izquierda) función escalón, (derecha) función Sigmoide Fuente: Han et al. (2018)

En la realidad, una red neuronal artificial utiliza otras funciones además de la función de activación, y las principales funciones de activación utilizadas son funciones logísticas, generando una serie de valores de salida según el valor de entrada.

Actualmente, las redes neuronales artificiales utilizan métodos de modificación con pesos en el proceso de aprendizaje. Al modificar estos pesos, la salida generada por red puede ser diferente a pesar de recibir los mismos datos de entrada. (Han et al., 2018)

Las arquitecturas de redes neuronales artificiales consisten en el enlace de varias neuronas artificiales por capas, definiendo 3 grupos diferentes según su orden, la capa de entrada, la capa escondida y la capa de salida. La capa de entrada debe tener la estructura adecuada para aceptar los datos de entrada, igual que la capa de salida tiene que tener la estructura adecuada para generar la salida en el formato necesario. La capa escondida conecta la salida de la capa de entrada con la entrada de la capa de salida, procesando los datos en el camino, muchas veces es conocida como “caja negra” porque somos incapaces de interpretar cómo se genera esa salida. (Rashid, 2003)

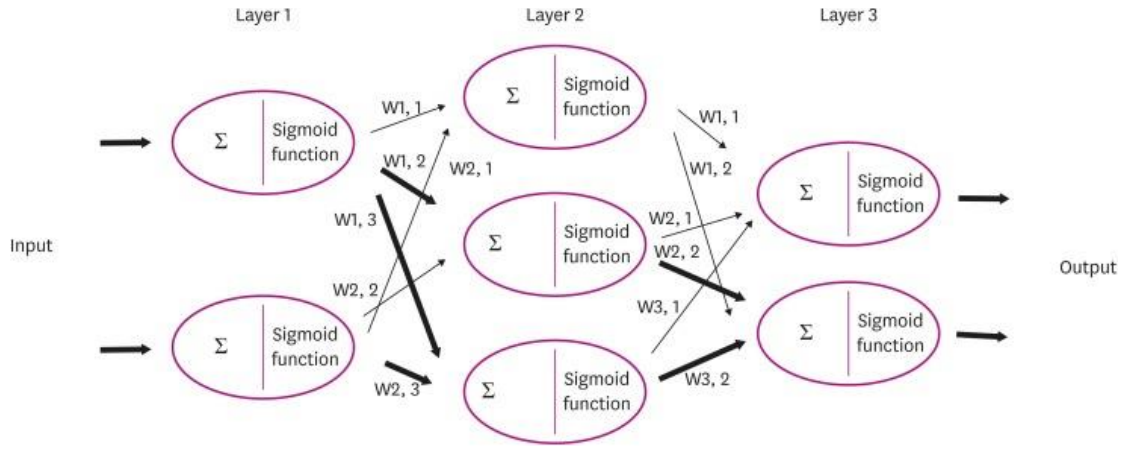


Ilustración 3.3: Conexiones y pesos entre neuronas de cada capa en una red neuronal artificial. Fuente: Han et al. (2018)

El aprendizaje de las redes neuronales artificiales consiste en actualizar variables para incrementar la tasa de aciertos de los valores de salida durante el proceso de aprendizaje. En la Ilustración 3.3 se puede observar una representación de una red neuronal artificial entrenada, los pesos *W1,2, W1,3, W2,3* de la capa de entrada y los pesos *W2,2* y *W3,2* de la capa escondida, representados por flechas en negrita, serían pesos con mayor magnitud que los representados por flechas normales, causando que sus salidas sean más relevantes al momento de generar la salida global. Un peso igual a 0 causa que la señal no sea generada y, por consecuencia, que la red sea influenciada. (Han et al., 2018)

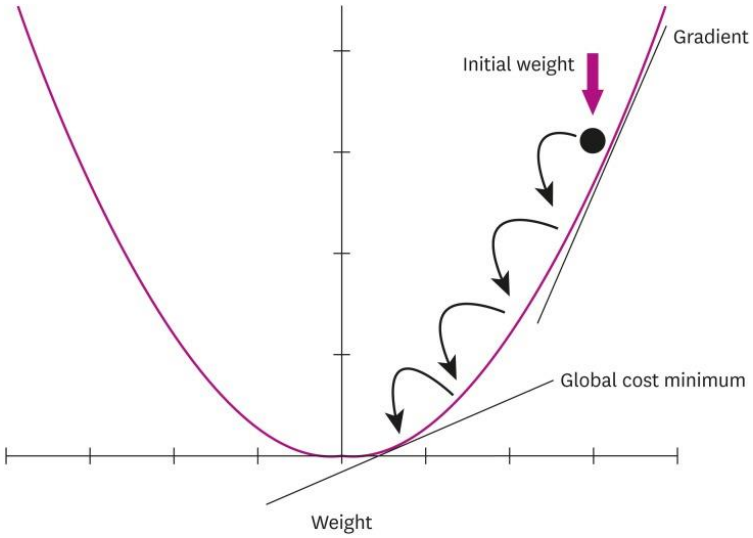


Ilustración 3.4: Gráfica de una función de coste. Fuente: Han et al. (2018)

La actualización de los pesos es determinada por el error entre la salida predicha y la salida correcta. Este error es dividido entre los pesos y luego es *back propagated* o propagado hacia atrás, pero el cálculo matemático de los nuevos pesos es complejo en estructuras jerárquicas (Han et al., 2018). Como alternativa, se utiliza el método del descenso del gradiente para alcanzar la respuesta correcta, como se puede observar en la Ilustración 3.4, esta técnica consiste en buscar el punto menor donde el coste es minimizado en la función de coste (definida como la diferencia entre el valor predicho y la respuesta obtenida). En otras palabras, los valores de las variables de aprendizaje son determinados por la derivada de la función de coste (Orr et al., 2012).

3.4. Redes neuronales convolucionales

Las redes neuronales convolucionales (*convolutional neural networks*, CNN) son uno de los principales tipos de redes neuronales utilizadas para el reconocimiento, segmentación y clasificación de imágenes. Entre los múltiples usos de las CNN se puede encontrar el reconocimiento de objetos, el procesamiento de imágenes, la visión artificial y el reconocimiento facial. (Taye, 2023).

Las CNN son entrenadas utilizando imágenes, así que aprovechan la capacidad del aprendizaje profundo de extraer automáticamente patrones de los

datos, en este caso, características de las imágenes, evitando la necesidad de crear características manualmente.

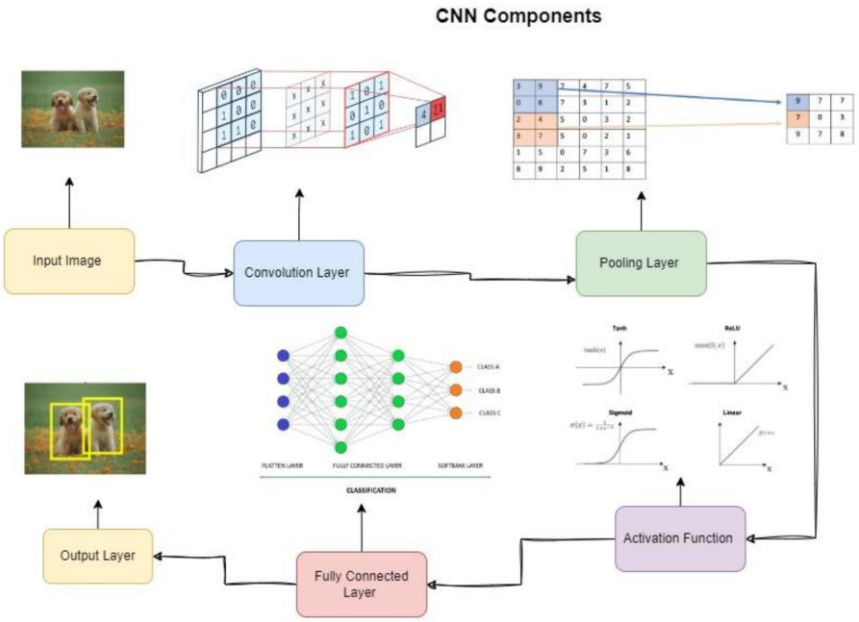


Ilustración 3.5: Componentes de una CNN. Fuente: Taye. (2023)

Según Taye (2023), una CNN está típicamente compuesta por cuatro tipos de capas:

- Convolucionales
- Pooling
- Función de activación
- Densamente conectadas

Las imágenes de entradas están compuestas por píxeles, organizados en forma de matriz, aunque según la representación de la imagen puede variar, generalmente con valores de 0 hasta 255, utilizando 3 matrices o “canales” diferentes para representar el brillo en rojo, verde y azul. Para un humano, el extraer información de una imagen no es una dificultad, pero para un ordenador la imagen solo es un conjunto de números organizados de cierta manera.

Para extraer información de esta estructura de dos dimensiones se utilizan dos tipos de capas, las capas convolucionales y las capas de *pooling*, estas capas realizan operaciones sobre los datos de una manera bidimensional,

extrayendo y consolidando información de los píxeles y sus relaciones entre ellos.

En las capas convolucionales se utilizan pequeños núcleos o *kernels* con capacidad de aprendizaje, implementaciones de una neurona artificial en un espacio bidimensional. Como se puede observar en la Ilustración 3.6, estos kernels recorren la imagen de entrada, actuando como un filtro, generando un mapa de activación bidimensional (O'Shea & Nash, 2015).

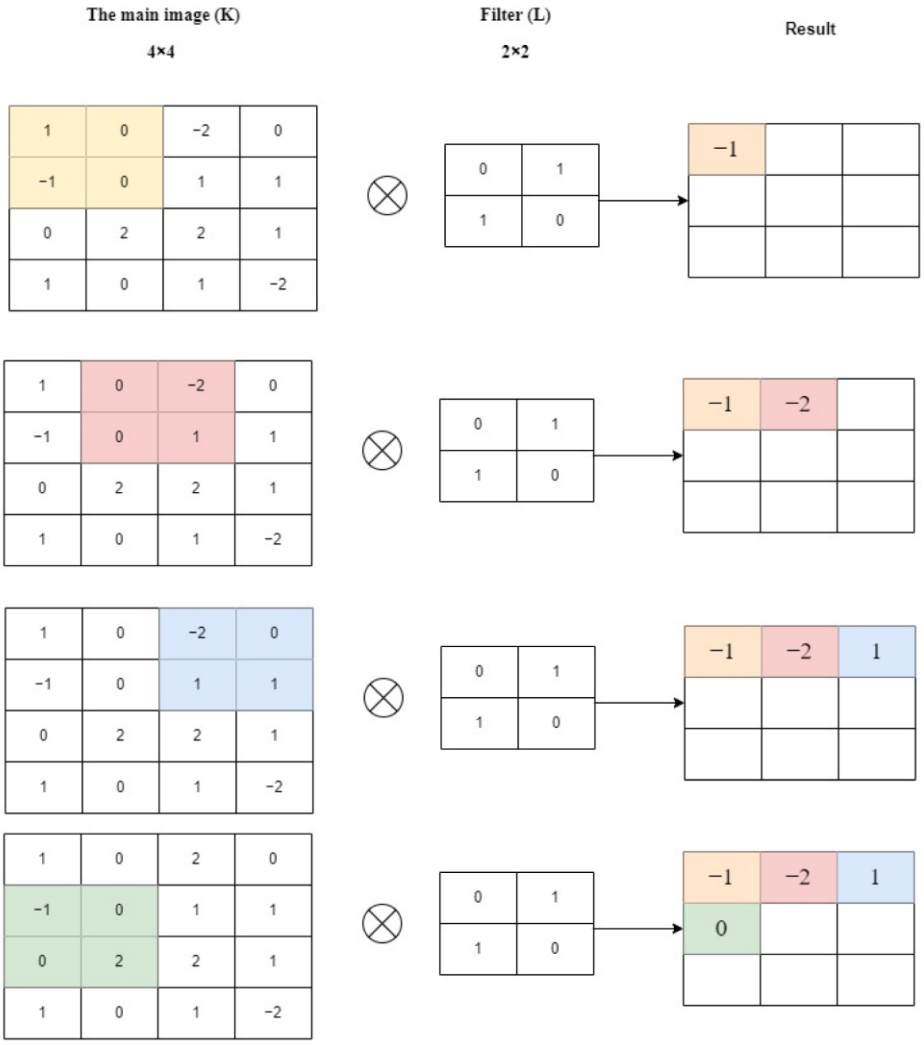


Ilustración 3.6: Representación visual de los primeros cálculos de una capa convolucional. Fuente: Taye. (2023)

El kernel realiza el producto escalar de la imagen por el filtro para formular el mapa de activación, permitiendo a la red aprender características, según el nivel de profundidad de la capa convolucional, la complejidad de estas

características. Las capas más cercanas a la imagen de entrada extraen las características más sencillas como líneas rectas o curvas.

Finalmente, cada mapa de activación es apilado en un volumen y se pasa a la siguiente capa del modelo. Para optimizar la complejidad de las CNN es necesario optimizar las salidas de las capas convolucionales, esto se puede realizar con ciertos hiperparámetros; el paso o *stride* y el relleno de ceros o *zero-padding*.

El stride se refiere al desplazamiento que realiza el kernel después de realizar una convolución, la Ilustración 3.6 utiliza un stride de 1. Al utilizar un stride mayor se reduce el tamaño del mapa de activación resultante (O'Shea & Nash, 2015).

El zero-padding consiste en aumentar el tamaño de la imagen añadiendo ceros alrededor de los datos, esto es útil porque las capas convolucionales pueden perder información en los bordes ya que solamente los “ven” una vez. Añadiendo un borde completo de ceros permite al kernel inspeccionar mejor los bordes sin añadir información innecesaria. Este proceso aumenta el tamaño del mapa de activación resultante (Taye, 2023).

La capa de pooling es utilizada para reducir el tamaño de los mapas de características mientras retienen los datos más relevantes. Un filtro utiliza una operación de pooling (máximo, mínimo, promedio, etc.) sobre los datos de entrada, desplazando el filtro hasta generar un mapa de características condensado. Comúnmente la función de máximo es la más utilizada (Taye, 2023).

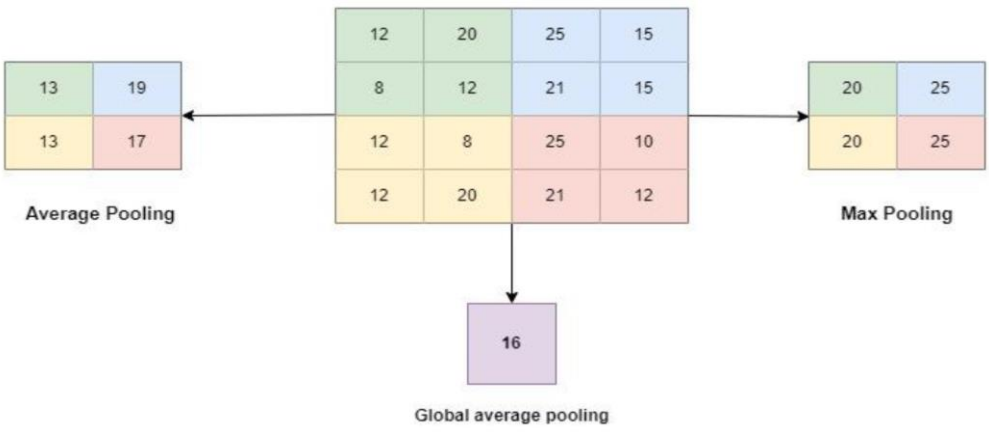


Ilustración 3.7: Representación visual de diferentes tipos de pooling y sus mapas resultantes. Fuente: Taye. (2023)

En la Ilustración 3.7 se pueden observar los diferentes mapas de características generados por diferentes tipos de pooling, en este caso el filtro tiene una tamaño de 2x2 y utiliza un stride de 2. Es posible realizar pooling con stride de 1 pero el propósito generalmente es reducir el tamaño de la salida así que no es comúnmente utilizado.

Posterior a una capa de convolución hay una capa de función de activación o capa no lineal. La no linealidad permite que la salida generada sea modificada o eliminada, con el propósito de mapear la entrada a la salida (Taye, 2023). Las capas de función de activación también se utilizan en la salida final del modelo, para dar el formato deseado, por ejemplo, un 0 o 1 en clasificación binaria.

Existen varios tipos de funciones de activación, estas dependen de la finalidad que tengan, en la Ilustración 3.8 se pueden observar algunas de las funciones de activación más comúnmente utilizadas. La función más comúnmente utilizada es la ReLu.

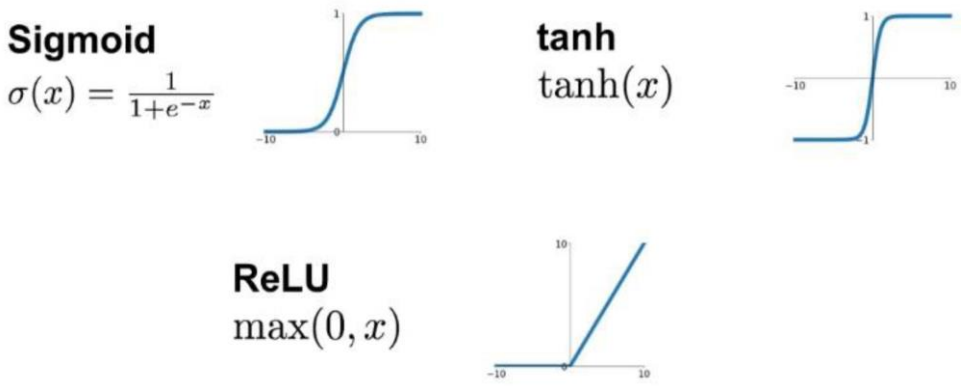


Ilustración 3.8: Funciones de activación. Fuente: Taye. (2023)

Las capas densamente conectadas contienen neuronas que están directamente conectadas con las dos capas adyacentes, similar a como están conectadas las redes neuronales tradicionales del aprendizaje profundo (O'Shea

& Nash, 2015). En las CNN se utilizan como clasificadores al recibir la información final de la extracción de características.

En la Ilustración 3.9, se puede observar una arquitectura sencilla que tendría una CNN para la clasificación de imágenes. Esta arquitectura es básica, para mejorar la calidad de las predicciones es posible incrementar la cantidad de capas convolucionales y de pooling, esto permite aprender características de mayor complejidad.

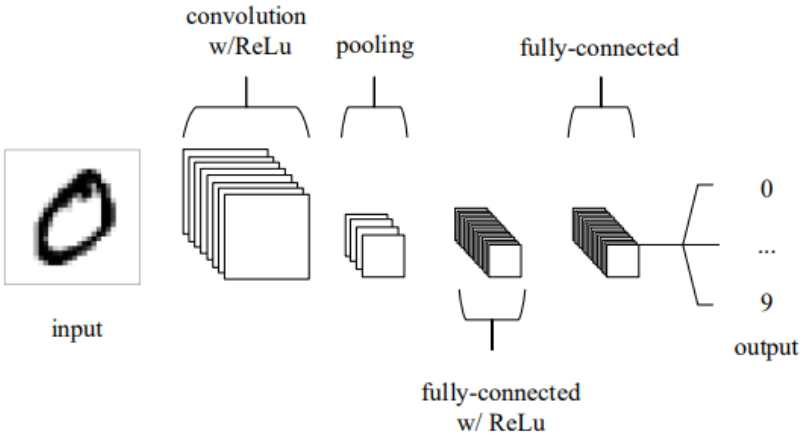


Ilustración 3.9: Arquitectura simple de CNN para clasificación con 5 capas. Fuente: O'Shea & Nash. (2015)

Hasta ahora se ha hablado de las CNN principalmente para la clasificación de imágenes, pero existen múltiples aplicaciones como previamente mencionadas, las que requieren modificaciones a la arquitectura sencilla mostrada en la Ilustración 3.10. Por ejemplo, para la segmentación de imágenes, se reemplazan las capas densamente conectadas por más capas convolucionales para generar una salida bidimensional, estas redes son llamadas redes completamente convolucionales (*fully convolutional networks*, FCN) (Taye, 2023).

3.5. Segmentación

El proceso de segmentación, como indica el nombre, consiste en segmentar una imagen en varias partes. Cada píxel de la imagen es asignado a

una clase en este proceso. Las dos principales categorías de segmentación son segmentación semántica y segmentación de instancias (Taye, 2023).

En la segmentación de instancias, se buscan varias instancias de cada clase en la imagen, agrupando un conjunto de píxeles a esa instancia en particular, a diferencia de la segmentación semántica donde los píxeles solo son asignados una clase. En la Ilustración 3.10, se puede observar la diferencia de resultados entre los tipos de segmentación.

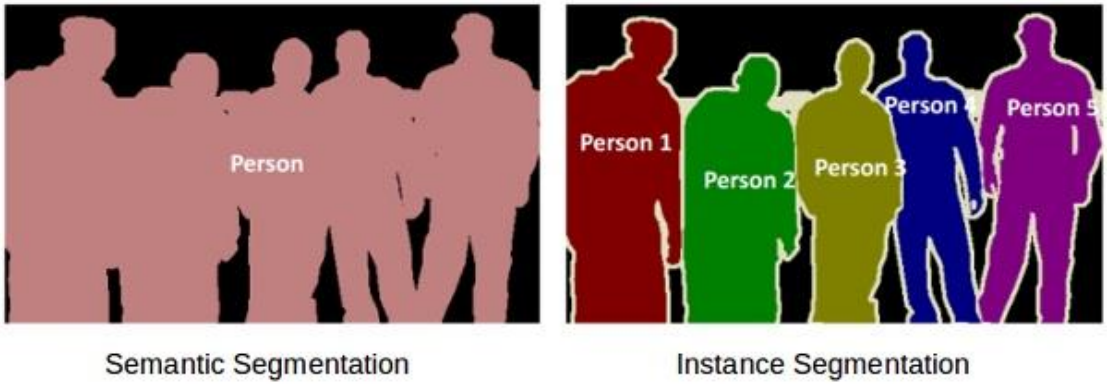


Ilustración 3.10: Máscaras de segmentación para segmentación semántica (izquierda) y segmentación de instancias (derecha). Fuente: Sharma (2019).

Entre las múltiples aplicaciones que tiene la segmentación de imágenes podemos encontrar la identificación de cáncer, sistemas de control de tránsito vehicular, coches autónomos, identificación de objetos en imágenes satelitales, etc. (Sharma, 2019).

Las técnicas de segmentación de imágenes han avanzado con los avances en aprendizaje profundo, principalmente a través de las FCN, dado que los avances en CNN para la clasificación de imágenes son extrapolables a la segmentación. Al reemplazar las capas densamente conectadas con capas convolucionales y de *unpooling*, una FCN convierte arquitecturas populares para la clasificación de imágenes en modelos que generan máscaras de segmentación en vez de puntuaciones de clasificación (Taye, 2023).

En las Ilustraciones 3.11 y 3.12 se puede observar cómo sería el resultado de convertir una CNN en una FCN, al añadir la misma cantidad de capas de

unpooling como capas de pooling, es posible retornar el tamaño de los mapas de características al tamaño original de la imagen (MathWorks, n.d.).

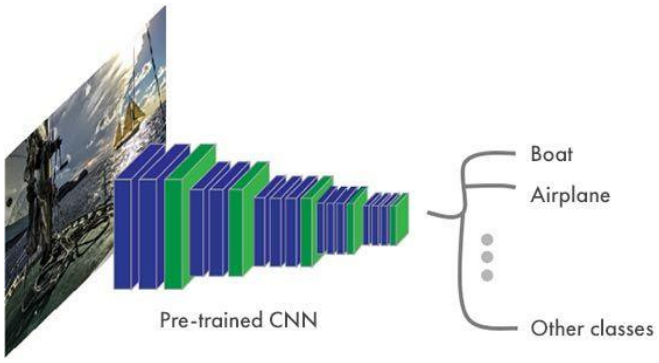


Ilustración 3.11: Estructura típica de una CNN. Fuente: Sharma (2019).

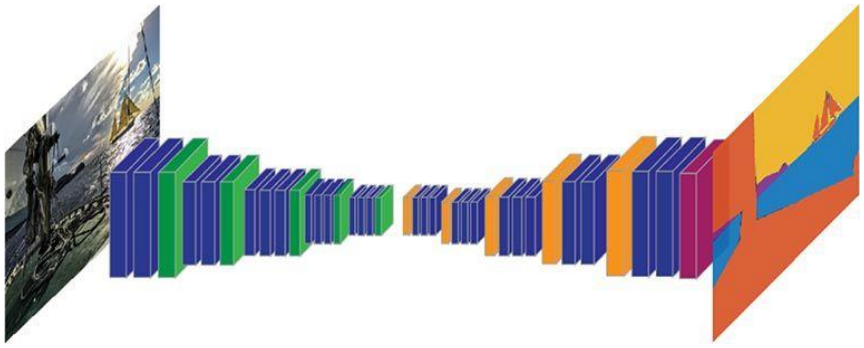


Ilustración 3.12: Estructura de una FCN, con capas de pooling en verde y capas de unpooling en naranja. Fuente: Sharma (2019).

Este proceso de conversión es muy común en el campo de segmentación, ya que permite aprovechar las múltiples arquitecturas avanzadas de clasificación de imágenes para una problemática diferente. Partiendo de estas arquitecturas, han surgido una serie de modelos que mejoran los resultados, por ejemplo Faster R-CNN o U-Net y sus sucesores, exploradas más profundamente en el desarrollo del proyecto.

4. OBJETIVOS

Partiendo del hecho que existen estudios previos sobre el uso de redes convolucionales para la clasificación de imágenes de células cervicovaginales, este estudio busca aplicar otra área de la inteligencia artificial a la problemática.

4.1. Objetivo General

- Clasificación de imágenes médicas mediante máscaras de segmentación para células procedentes de citologías cervicovaginales utilizando redes neuronales convolutivas.

4.2. Objetivos Específicos

- OE1: Entrenamiento de una red neuronal de segmentación semántica para etiquetar el núcleo y el citoplasma de imágenes médicas de células cancerígenas.
- OE2: Creación de un dataset de máscaras segmentadas por núcleo y citoplasma correspondiente al dataset de imágenes original.
- OE3: Estudio de la relación entre la precisión del modelo de segmentación semántica y la calidad y cantidad del dataset de entrenamiento
- OE4: Comparación de la precisión y el gasto de recursos de diferentes modelos de segmentación semántica.
- OE5: Entrenamiento de una red neuronal de clasificación de imágenes y máscaras de células.
- OE6: Estudio de los hiperparámetros necesarios para optimizar los modelos de inteligencia artificial

5. METODOLOGÍA

Antes de elegir la metodología a utilizar, es necesario explorar las opciones para encontrar la más adecuada al proyecto, la que permita optimizar lo más posible el proceso de desarrollo. Comúnmente se dividen en las metodologías clásicas o tradicionales y las metodologías ágiles.

Las metodologías clásicas o tradicionales son las que siguen un proceso lineal y secuencial. Como indican Velásquez Restrepo et al. (2019), “Exige que se presente gran atención a la planificación total de todo el trabajo a realizar y, una vez que está todo detallado, se inicia el desarrollo del producto.” En esta categoría podemos encontrar metodologías como Cascada, Modelo en V y Modelo en espiral.

Por otro lado, las metodologías ágiles, son aquellas que se basan en principios de rapidez, colaboración e iteratividad. Navarro Cadavid et al. (2013) definen los proyectos que utilizan metodologías así:

“Los proyectos ágiles se subdividen en proyectos más pequeños mediante una lista ordenada de características. Cada proyecto es tratado de manera independiente y desarrolla un subconjunto de características durante un periodo de tiempo corto, entre dos y seis semanas. La comunicación con el cliente es constante al punto de requerir un representante de él durante el desarrollo.”

Considerando la naturaleza de la investigación a realizar, se ha optado por una metodología ágil, ya que no existe una planificación detallada a realizar, los requerimientos son cambiantes y hay una comunicación constante con los tutores del proyecto.

Velásquez Restrepo et al. (2019) identifican entre algunas de las metodologías ágiles más populares a Scrum, Metodología basada en componentes y Programación extrema (XP). Las últimas dos metodologías mencionadas poseen prácticas no aplicables al proyecto actual, por ejemplo, la reutilización de componentes previos y la programación en pareja, entre otras.

Finalmente se elige la metodología Scrum, aunque de también posee prácticas no aplicables, por ejemplo, los *daily scrum* (reuniones diarias), su estructura es adecuada para la naturaleza del proyecto.

Scrum utiliza un enfoque incremental con equipos autogestionados, multifuncionales que trabajan en iteraciones, como se puede abreviar en la Ilustración 5.1. En cada iteración se realiza una entrega al dueño del producto, con nuevas funcionalidades o modificaciones que el dueño del producto requiera (Navarro Cadavid et al., 2013).

Las iteraciones en Scrum se llaman *Sprints*, una ventana de tiempo donde se crea una versión utilizable del producto. Cada sprint se compone de los siguientes elementos: Reunión de planificación del Sprint, daily scrum, trabajo de desarrollo, revisión del Sprint y retrospectiva del Sprint.

Para organizar el proyecto se utilizan historias de usuario, cuantificadas a través de puntos de historia que miden el tamaño en función del esfuerzo necesario para realizarlas (Navarro Cadavid et al., 2013).

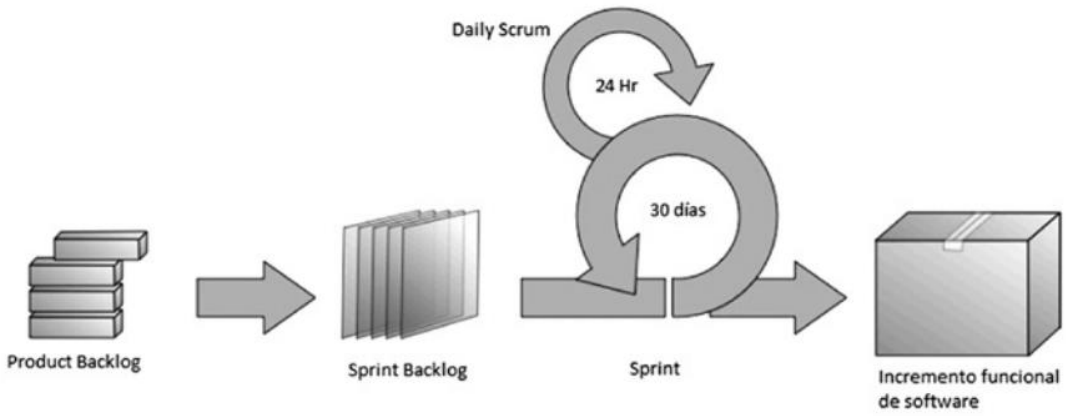


Ilustración 5.1: Estructura de una FCN, con capas de pooling en verde y capas de unpooling en naranja. Fuente: Cohn, M (2004).

5.1. Estimación de recursos y planificación

Para este proyecto, las revisiones y retrospectivas del Sprint serán realizadas con los tutores del TFG como dueños del producto. Al tener un equipo de desarrollo de una persona, las reuniones diarias no serán realizadas.

Se utilizará la secuencia de números de Fibonacci para asignar los puntos de historia de cada historia de usuario, incrementando en valor según la dificultad. Además de esto, se asigna una prioridad según la importancia que tenga la historia dentro del sprint actual, ubicando 1 para las tareas más importantes y 2 para las secundarias.

La organización de los sprints del proyecto se encuentra en la Tabla 5.1:

Tabla 5.1: Organización de sprints. Fuente: Elaboración propia.

ID	Sprint	Historia de Usuario	Prioridad	Puntos de Historia
1	1	Estudio del problema	1	3
2	1	Estudio de algoritmo K-Means	1	3
3	1	Prueba de segmentación no supervisada	1	2
4	1	Estudio de herramientas de anotación de imágenes	1	1
5	1	Creación de un dataset con VIA Annotator	1	5
6	1	Prueba de Faster R-CNN	2	2
7	2	Creación de un dataset con Apeer	1	13
8	2	Estudio de modelos de segmentación semántica	1	3
9	3	Prueba con U-Net	1	5
10	3	Generación de un dataset con Apeer	2	13
11	3	Prueba con dataset generado	2	3
12	4	Estudio de métricas para segmentación semántica	1	2
13	4	Cambio de métricas	1	1
14	4	Prueba con dataset generado	1	3
15	4	Prueba con dataset por clase	1	5
16	4	Estudio de transfer learning y fine tuning	2	3
17	5	Comparación de U-Net con otras arquitecturas pre-entrenadas	2	8
18	5	Segmentación del dataset completo	1	2
19	6	Prueba con dataset por clase	1	3
20	6	Comparación global de diferentes datasets	1	5
21	7	Pruebas con diferentes modelos de segmentación	2	8
22	8	Clasificación de imágenes con máscaras de segmentación	1	5
23	8	Concatenación de modelos de clasificación para imágenes y máscaras	1	13
24	9	Mejora del modelo de imágenes y máscaras	1	5

25	10	Balanceo de clases	1	5
26	10	Prueba de GAP	1	3
27	10	Concatenación de modelos de segmentación y clasificación para imágenes y máscaras	1	21

El product backlog contiene un total de 145 puntos de historia. El horario laboral para la realización del proyecto es de 9:00 a 13:00 de lunes a viernes, haciendo un total de 20 horas a la semana.

Estimando la duración del proyecto a 450 horas, por las 25 horas por crédito ECTS, podemos calcular el coste del punto de función, a través de la relación entre el total de puntos de historia y el total de horas a trabajar:

$$Pfh = \frac{Puntos\ de\ Función}{Horas} = \frac{145}{450} = 0.322\ pf/hora$$

Para visualizar la planificación del proyecto, en la Ilustración 5.2 se puede apreciar los puntos de función por sprint.

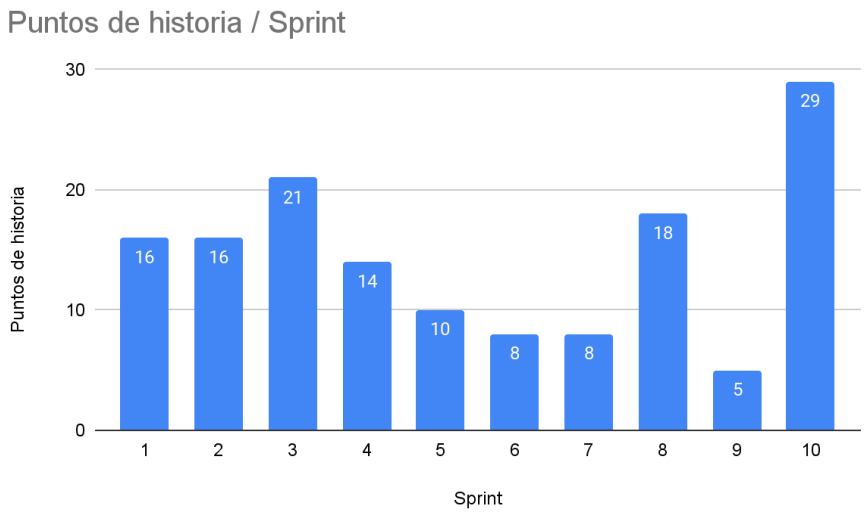


Ilustración 5.2: Gráfico de puntos de historia por sprint. Fuente: Elaboración propia.

5.2. Valoración de dedicación y coste económico

Para realizar el cálculo, se utilizará la media de salario por hora para un ingeniero de software en España, 18€ (Talent.com, n.d.). Dado que el entorno de desarrollo es gratuito y el software utilizado es de código abierto, el resto de los gastos sólo serán personales.

Al realizar la estimación de horas por sprint, es posible calcular el coste de cada sprint y del proyecto en su totalidad. El coste final es de **8.105,59 €**, en las Ilustraciones 5.3 y 5.4 se puede apreciar el desglose de horas y coste por sprint.

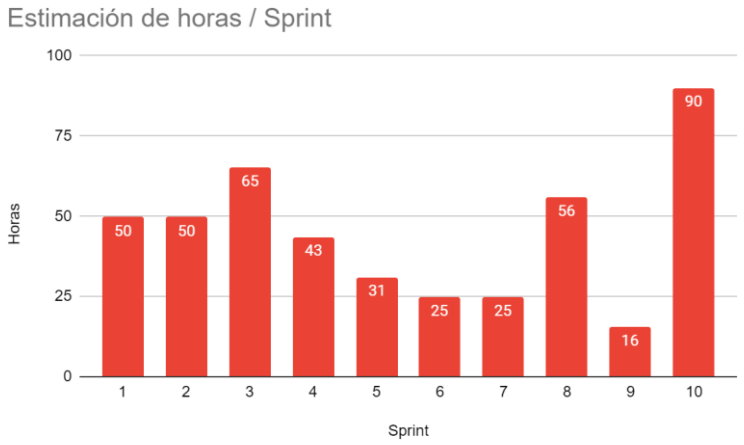


Ilustración 5.3: Gráfico de estimación de horas por sprint. Fuente: Elaboración propia.

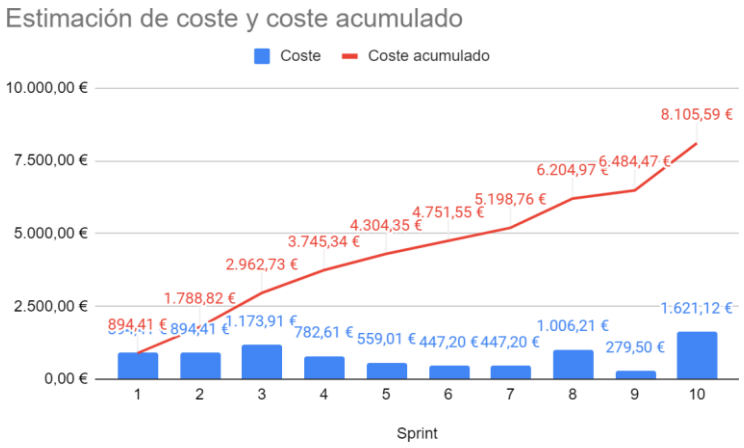


Ilustración 5.4: Gráfico de estimación de coste y coste acumulado por sprint. Fuente: Elaboración propia.

5.3. Tecnologías y herramientas utilizadas en el proyecto

5.3.1. Infraestructura de software

- **Python:** Es un lenguaje de programación orientado a objetos, comparable a Perl, Ruby o Java (Python, 2022). Este lenguaje es comúnmente utilizado para el desarrollo de aplicaciones de aprendizaje automático por su fácil implementación y gran variedad de librerías y frameworks.
- **TensorFlow:** Es una plataforma y librería de código abierto de extremo a extremo para el aprendizaje automático. Facilita la creación de modelos de aprendizaje automático en múltiples lenguajes como Python, JavaScript, C++ y Java. Permite el uso de GPUs y TPUs para incrementar la velocidad de entrenamiento e inferencia de modelos (TensorFlow, 2021).
- **Keras:** Es una API de código abierto escrita en Python, que se ejecuta sobre la plataforma TensorFlow. Está orientada para realizar experimentos de aprendizaje profundo de manera rápida y sencilla (Keras, 2021). Incluye múltiples modelos de aprendizaje profundo ya definidos fácilmente accesibles.
- **Jupyter notebook:** Aplicación web diseñada para el desarrollo y presentación de software, con la capacidad de mostrar gráficas, ecuaciones y texto, además de código (Project Jupyter, n.d.).
- **Google Colab:** Entorno web en la nube basado en Jupyter notebook que permite desarrollar y ejecutar código Python sobre una máquina virtual. Permite acceso gratuito a GPUs (con ciertas limitaciones) e integración con Google Drive (Google, n.d.). Este es el entorno por utilizar para el entretenimiento de modelos de inteligencia artificial.

5.3.2. Infraestructura de hardware

El hardware por utilizar durante el proyecto para el entrenamiento de modelos de inteligencia artificial es el proporcionado por Google Colab en el plan

gratuito. Los detalles precisos del sistema son variables porque Google asigna de manera dinámica los recursos, sin importar el tipo de plan (Google, n.d.).

Pero al inspeccionar el sistema podemos encontrar las siguientes especificaciones:

- **CPU:** Intel Xeon 2GHz
- **Memoria:** 12 GB
- **Disco duro:** 52 GB
- **GPU:** NVIDIA Tesla T4
    - 2560 núcleos de NVIDIA CUDA
    - 16 GB de memoria GDDR6

6. DESARROLLO DEL PROYECTO

Para la realización de la segmentación de imágenes se probaron múltiples métodos hasta encontrar el más adecuado para la problemática.

El objetivo de realizar la segmentación de las células es generar una imagen compuesta por máscaras segmentadas una por cada clase a segmentar, esto es llamado Segmentación Semántica, en este caso son 3 clases: el núcleo, el citoplasma y el fondo.

El dataset inicial para este TFG es un total de 1863 imágenes de tamaño 224 x 224 píxeles en formato .tiff dividido en 4 clases para la clasificación de células de diferentes grados de cáncer. La distribución de las imágenes no es uniforme como se ve en la Ilustración 6.1.

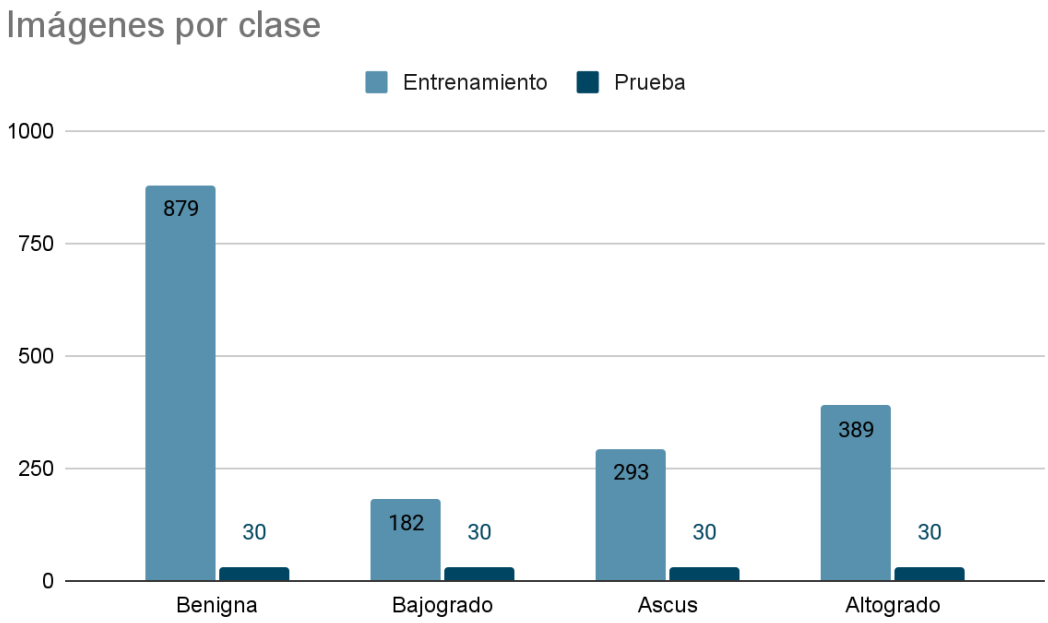


Ilustración 6.1: Distribución del dataset original. Fuente: Elaboración propia.

El entorno para todas las siguientes pruebas es Google Colab con un plan gratuito, esto implica que por la distribución dinámica de recursos para los planes no de pago, el hardware utilizado es variable pero se utiliza computación con GPU para evitar tiempos largos de entrenamiento.

6.1. Sprint 1

En esta fase del proyecto iniciamos con el estudio inicial del problema y los posibles métodos para solucionarlo. Principalmente, antes de empezar a entrenar un modelo de segmentación, es necesario crear un dataset.

Tabla 6.1: Organización de sprint 1. Fuente: Elaboración propia.

ID	Sprint	Historia de Usuario	Prioridad	Puntos de Historia
1	1	Estudio del problema	1	3
2	1	Estudio de algoritmo K-Means	1	3
3	1	Prueba de segmentación no supervisada	1	2
4	1	Estudio de herramientas de anotación de imágenes	1	1
5	1	Creación de un dataset con VIA Annotator	1	5
6	1	Prueba de Faster R-CNN	2	2

6.1.1. Segmentación no supervisada con K-Means

Como dice Sharma (2019), la segmentación basada en clusterización con K-Means funciona muy bien en datasets pequeños y con puntos de datos convexos al ser un algoritmo basado en distancia.

La segmentación utilizando K-Means utiliza como entrada la imagen transformada en un array unidimensional, generando K clusters aleatorios y clasificando cada píxel en uno de estos clusters.

Esta fue la primera prueba a realizar porque no requiere tener un dataset previamente segmentado, así que pude utilizar el dataset original de células clasificadas.

Para utilizar K-Means, la librería de Python, OpenCV tiene un método que nos permite ingresar el vector de la imagen, el número K de clusters aleatorios, el número de intentos, diferentes flags para el método, de las cuales se utiliza cv.KMEAN_RANDOM_CENTERS para variar los centros de los clusters y el criterio de finalización, el cual consiste en 3 parámetros: el número de iteraciones máximo, en mi caso 10, la precisión requerida/épsilon, en mi caso 1.0 y una flag para la condición de finalización del método, en mi caso

cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, para finalizar cuando se alcance el épsilon indicado o el número de iteraciones.

En la Ilustración 6.2 se puede ver que solamente utilizando K-Means no es posible generar una buena segmentación, las imágenes de las células son complejas, en parte porque cada célula posee varios orgánulos más que el núcleo y en parte porque las imágenes son una visión 2D de células 3D, causando superposición de células con más puntos oscuros que el núcleo.

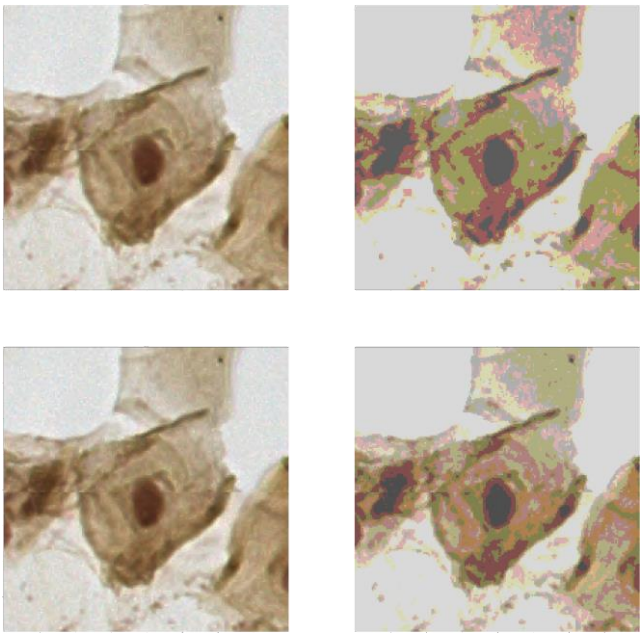


Ilustración 6.2: Células originales (izquierda) y segmentadas (derecha) con K-Means, K=3 (arriba) y K=4 (abajo). Fuente: Elaboración propia.

Considerando el funcionamiento y la estructura de los datos, es de esperar que no logre llegar a una segmentación uniforme, podría funcionar mejor para casos más específicos con imágenes cuyas clases sean de colores uniformes.

La conclusión de esta prueba es que la segmentación de imágenes de células con el algoritmo de K-Means no es viable, dado que las 3 clases a segmentar no poseen un color homogéneo cada una, como podemos observar en la Ilustración 6.2.

6.1.2. Generación de Dataset de máscaras con VIA

El resto de los modelos de segmentación de imágenes utilizan aprendizaje profundo para aprender de ejemplos las características a fijarse o ignorar al momento de realizar la segmentación, para utilizar uno de estos necesitamos un dataset con imágenes y las máscaras segmentadas correspondientes.

Para iniciar con la anotación de las imágenes del dataset, utilicé la herramienta VIA Annotator, que permite segmentar manualmente las imágenes en las clases deseadas. Con esta herramienta generé un dataset de 62 imágenes, 50 de entrenamiento y 12 de prueba, se puede apreciar un conjunto de ejemplo en la Ilustración 6.3.

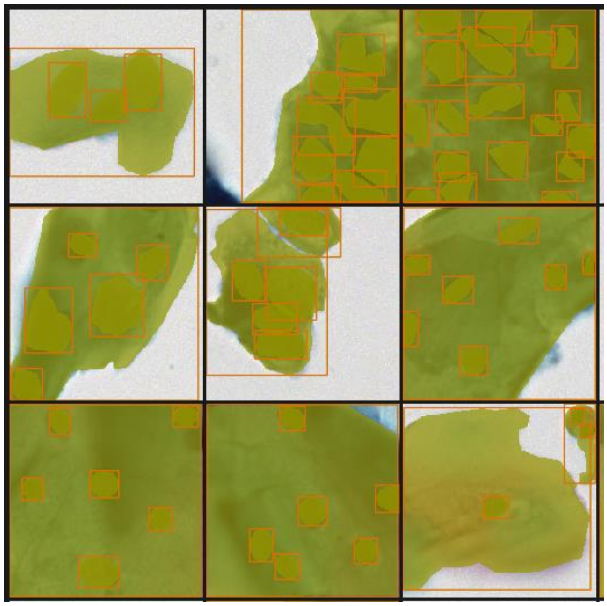


Ilustración 6.3: Células segmentadas entre citoplasma y núcleo. Fuente: Elaboración propia.

Lastimosamente, el formato de los datos generados por esta herramienta es muy difícil de manejar, ya que están orientados a Segmentación de Instancias, es decir, identifica las coordenadas de cada instancia de la clase identificada individualmente y luego agrupa los píxeles correspondientes a la instancia.

A pesar de las dificultades, fue posible realizar una prueba con este dataset utilizando el modelo Faster R-CNN, generando resultados poco precisos, como es de esperar con un dataset tan limitado, apreciables en la Ilustración 6.4.

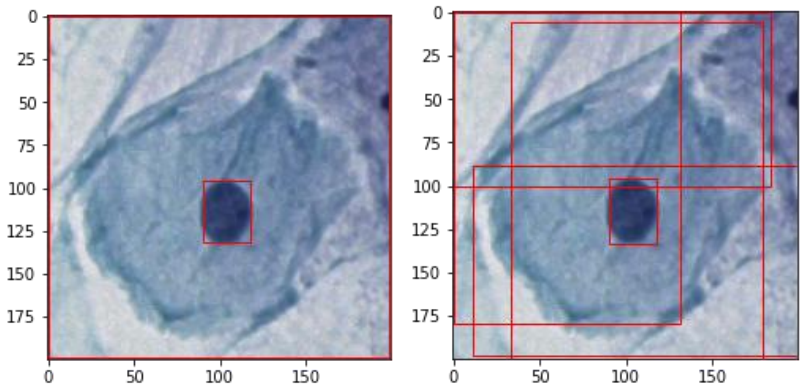


Ilustración 6.4: Células segmentadas entre citoplasma y núcleo, manualmente (izquierda) y predicción (derecha). Fuente: Elaboración propia.

6.1.3. Retrospectiva del sprint

Durante el desarrollo de este sprint se han encontrado dos problemas, primero cómo generar un dataset de máscaras segmentadas con las imágenes del dataset original y segundo aprender a implementar, medir e interpretar algoritmos de segmentación de imágenes.

Estos problemas son notables principalmente en la prueba de Faster R-CNN, donde no se ha sido capaz de visualizar adecuadamente la máscara generada con VIA ni la predicción, mostrando solamente las *bounding boxes* de la máscara.

6.2. Sprint 2

Durante este segundo sprint se va a continuar con la exploración de herramientas para la creación del dataset, específicamente la herramienta de Apeer que provee muchas facilidades en el área de *Computer Vision*, y con el estudio de los diferentes modelos de segmentación semántica actuales que mejor se apliquen a la problemática.

Tabla 6.2: Organización de sprint 2. Fuente: Elaboración propia.

ID	Sprint	Historia de Usuario	Prioridad	Puntos de Historia
7	2	Creación de un dataset con Apeer	1	13
8	2	Estudio de modelos de segmentación semántica	1	3

6.2.1. Generación de Dataset de máscaras con Apeer

Vuelvo a generar 62 imágenes manualmente segmentadas, esta vez con la herramienta Apeer, una herramienta basada en la web especializada en el área de visión de computadores. Dos ejemplos del resultado de etiquetado se muestran en la Ilustración 6.5.

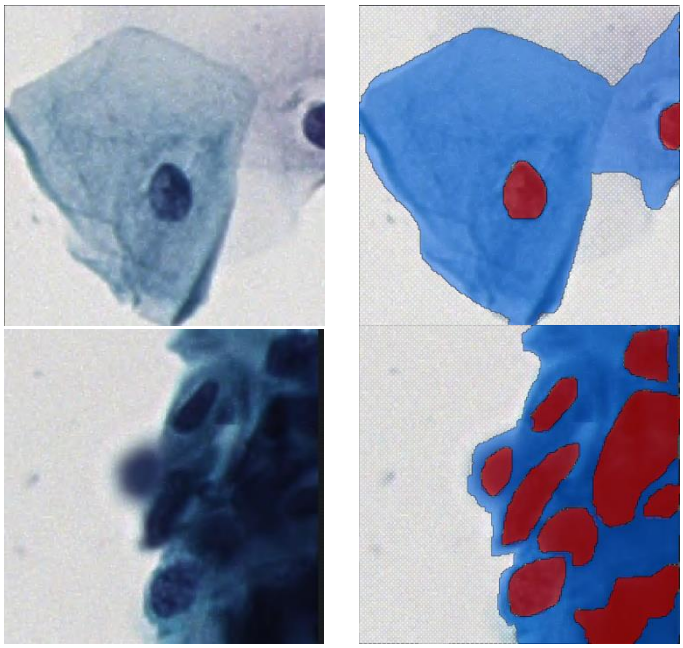


Ilustración 6.5: Células originales (izquierda) y segmentadas (derecha). Benigna (arriba) y altogrado (abajo) Fuente: Elaboración propia.

A pesar del gran consumo de tiempo que tarda manualmente segmentar, la ventaja de esta herramienta es que, además de generar las imágenes segmentadas en un formato adecuado, es posible entrenar su propio modelo basado en U-Net y segmentar más imágenes.

Utilizando las 50 imágenes de entrenamiento y 12 de prueba, entrené el modelo de Apeer para segmentar automáticamente las 1863 imágenes del

dataset original. Sin embargo, no todas las imágenes resultantes de esta segmentación eran de calidad adecuada así que fue necesario filtrar manualmente para obtener un dataset adecuado.

6.2.2. Buscando un modelo de segmentación

Partiendo de mi conocimiento sobre redes neuronales convolucionales para la clasificación de imágenes, buscaba un modelo de segmentación eficiente y simple para evitar periodos largos de entrenamiento y poder comprender completamente al modelo.

El modelo más simple para segmentación es una *Fully Convolutional Network*, una red completamente convolucional. Ambas las FCN y las CNN para clasificación de imágenes comparten una capa de entrada seguida de capas convolucionales y de *MaxPooling* para extraer características de la imagen, pero en vez de utilizar capas completamente conectadas y generar una salida prediciendo la clase, las FCN utilizan un decodificador para retornar la imagen al tamaño original, como se muestra en la Ilustración 6.6.

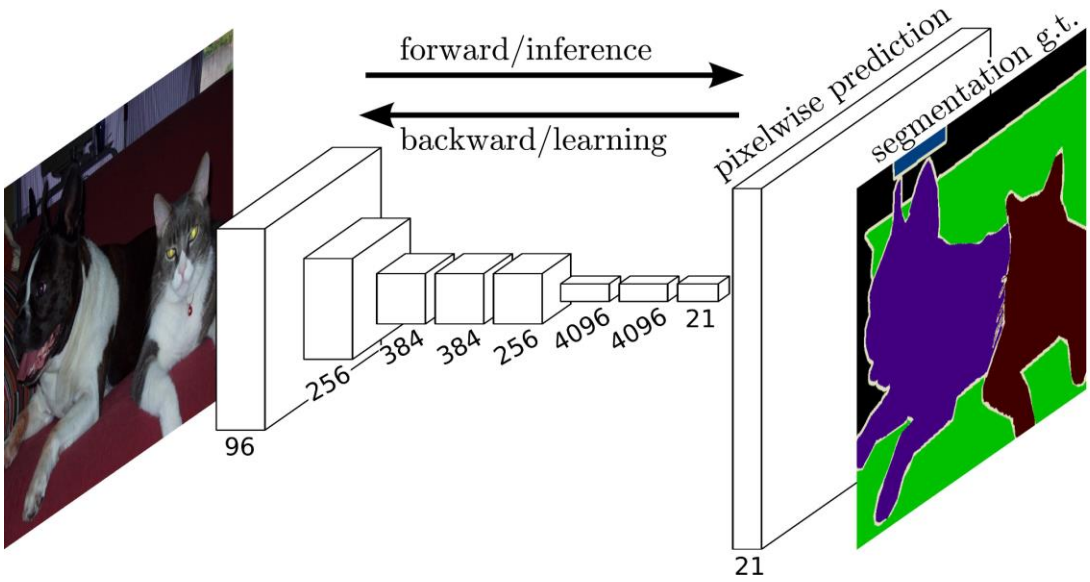


Ilustración 6.6: Arquitectura de una FCN. Fuente: Jonathan Long et al.

Este tipo de modelos de segmentación sufren de un problema, al ir extrayendo características de la imagen, el tamaño de la salida se va reduciendo, generando mapas de características menos precisos a mayor profundidad,

entonces al utilizar el decodificador para devolver la salida al tamaño de la imagen, la salida pierde mucha precisión.

Entre los modelos actuales posteriores a las FCN tradicionales, se eligió U-Net desarrollado por Ronneberger et al. (2015) Esta red, originalmente diseñada para identificar tumores de pulmón y cerebro, es una modificación de las FCN.

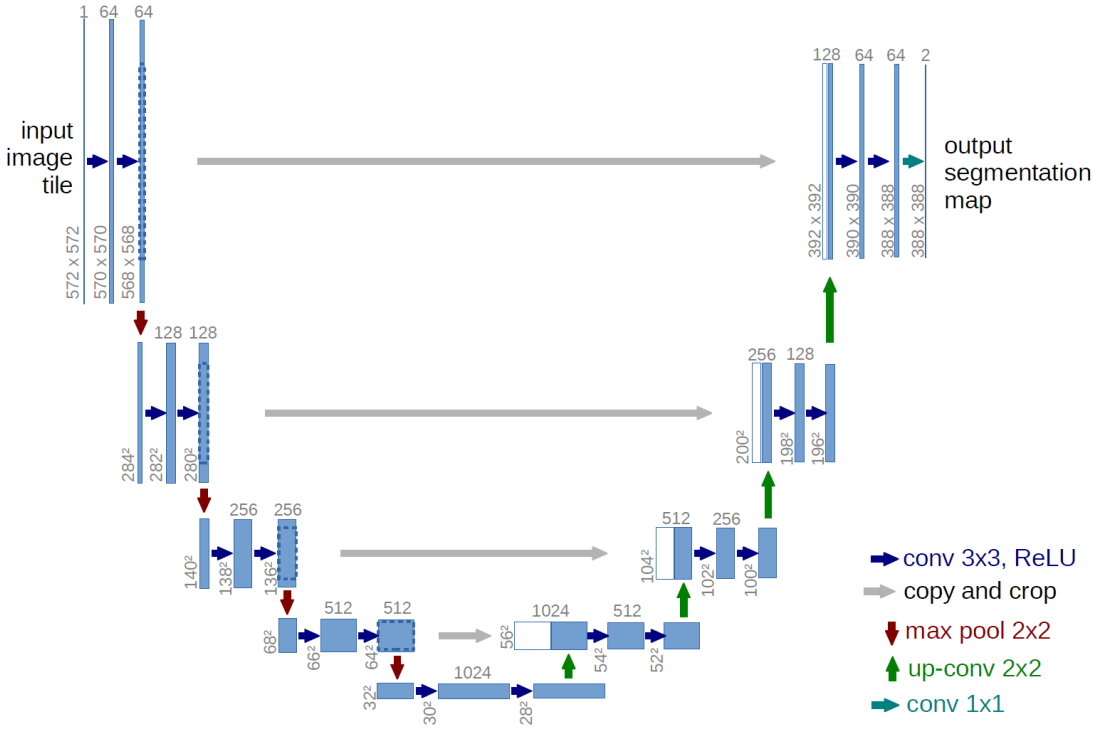


Ilustración 6.7: Arquitectura de U-Net. Fuente: Olaf Ronnenberg et al.

En la Ilustración 6.7, se puede ver que U-Net utiliza bloques convolucionales y MaxPooling como extractor de características y bloques convolucionales hacia arriba para realizar el sobre muestreo, pero para evitar el problema de las FCN, U-Net concatena la salida del bloque convolucional con la entrada del bloque convolucional hacia arriba de tamaño correspondiente, para no perder información espacial.

Otra ventaja que ofrece U-Net es su eficacia, al ser relativamente pequeña y simple, genera resultados de muy buena calidad en un tiempo muy corto. Por estas razones, decidí utilizar este modelo para mi problemática.

En el paper original de U-Net realizan su implementación con la librería de PyTorch, pero en el proyecto se tiene mayor experiencia con TensorFlow y Keras, así que se utiliza la implementación de Bhattiprolu (2020) en Keras, dejando solamente las últimas 2 capas de Dropout, con la meta de aprender más del dataset limitado.

6.2.3. Retrospectiva del sprint

En este sprint fue posible dar dos pasos muy importantes, encontrar una plataforma para generar el dataset y encontrar una implementación en Keras del modelo a utilizar.

A pesar de estos avances, el progreso es costoso y gradual, ya que la etiquetación manual de las imágenes y la precisión de ésta es clave para entrenar un modelo de segmentación de calidad. El alto coste temporal que tiene el etiquetado manual de imágenes lleva a la conclusión de dividir la tarea a más sprints o reducirla lo más posible, ya que el dataset original posee demasiadas imágenes para etiquetar manualmente todas.

6.3. Sprint 3

Durante este sprint se va a hacer una prueba con el modelo de segmentación U-Net seleccionado previamente y con las imágenes manualmente segmentadas con Apeer, también se buscará aprovechar otra funcionalidad de la herramienta de Apeer para automáticamente etiquetar el resto de las imágenes y también se realizará una prueba con estas máscaras.

Tabla 6.3: Organización de sprint 3. Fuente: Elaboración propia.

ID	Sprint	Historia de Usuario	Prioridad	Puntos de Historia
9	3	Prueba con U-Net	1	5
10	3	Generación de un dataset con Apeer	2	13
11	3	Prueba con dataset generado	2	3

6.3.1. Pruebas con U-Net

Con un modelo ya definido y dos datasets, el pequeño manualmente segmentado y el automáticamente segmentado con Apeer, empiezo a entrenar el modelo y evaluarlo.

Al momento de compilar todos los siguientes modelos se utilizan los siguientes parámetros:

- Optimizer: ‘adam’
- Learning rate: 1e-3
- Loss: ‘categorical_crossentropy’

Y para el entrenamiento de todos los siguientes modelos utilizo un batch size de 32 con un callback de *Early Stopping* monitorizando *‘val_loss’* para optimizar la longitud de mis entrenamientos, en este caso con paciencia de 20 épocas.

El formato en el que se descargan las máscaras generadas en Apeer asigna una clase (0, 1 o 2) a cada píxel de la imagen, dejando solamente un canal de imagen, a diferencia de una imagen RGB que posee 3 canales, uno por cada color.

Para formatear las máscaras para ser aceptables por U-Net, es necesario convertir la máscara a 3 máscaras binarias, una por clase, cada una en canal individual, mostrando 0 o 1 en cada píxel de cada canal para indicar si es una instancia de la clase o no.

Para realizar el formateo de máscaras es necesario expandir las dimensiones para hacer espacio para los canales con el método ‘expand_dims’ de la librería NumPy, convertir la imagen a máscaras binarias utilizando el método ‘to_categorical’ de Keras, y luego darle la forma correcta utilizando el método ‘reshape’ de la librería NumPy, indicando la misma forma original más la nueva dimensión de longitud igual a número de clases.

6.3.1.1. Prueba con dataset segmentado manual

Entrenando con el dataset pequeño se puede apreciar que el modelo aprende las diferentes estructuras del modelo rápidamente, diferenciando

fácilmente entre el fondo y el citoplasma, pero sin ser capaz de identificar un núcleo con suficiente seguridad para clasificarlo, como se puede ver en la Ilustración 6.8.

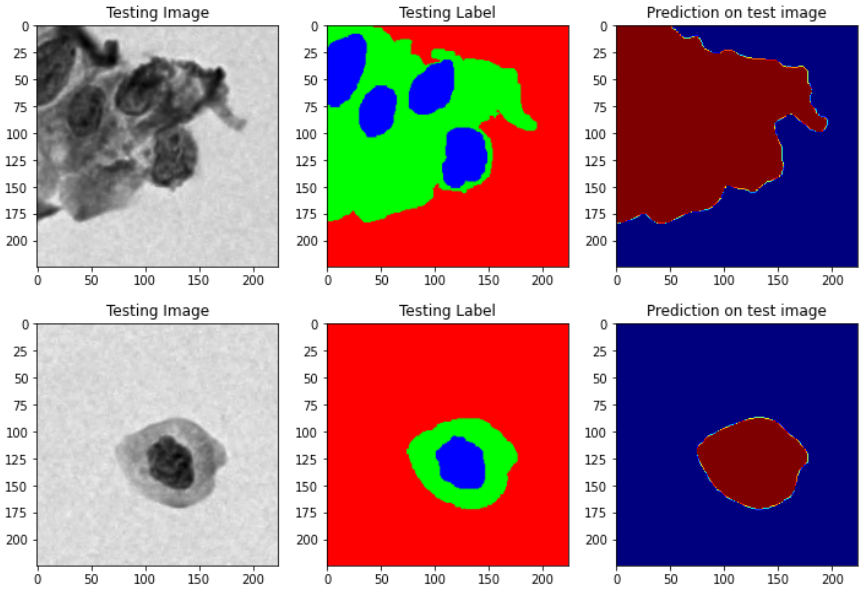


Ilustración 6.8: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia.

A pesar de no identificar una clase entera, la tasa de aciertos al realizar la predicción del dataset de prueba es de un 87.65%, pero como se ve en las gráficas del entrenamiento de la Ilustración 6.9, la pérdida no baja. Tiene sentido ya que los núcleos son la clase minoritaria en todas las imágenes, así que fallar en estos no afecta tanto a esta métrica.

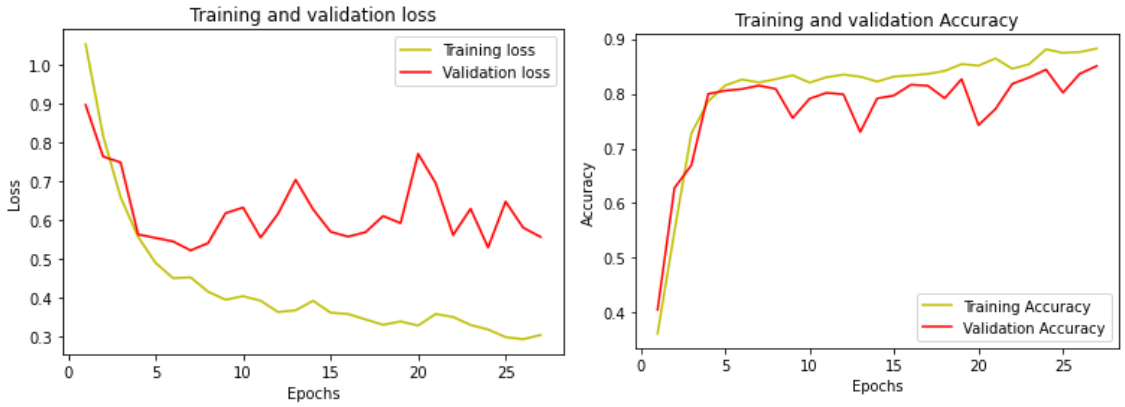


Ilustración 6.9: Gráficas de entrenamiento. (Izquierda) Pérdida. (Derecha) Tasa de aciertos. Fuente: Elaboración propia.

```
1/1 [==============================] - 1s 626ms/step - loss: 0.3488 - accuracy: 0.8765
Accuracy is =  87.64798045158386 %
```

Ilustración 6.10: Resultados al segmentar las imágenes del conjunto de prueba. Fuente: Elaboración propia.

Como conclusión de esta prueba, podemos ver que el modelo es capaz de aprender rápidamente la diferencia entre las clases, principalmente las clases mayoritarias de fondo y citoplasma, pero fallando al momento de reconocer los núcleos.

6.3.1.2. Prueba con dataset segmentado automáticamente

Entrenando con el dataset automáticamente segmentado por Apeer de 200 imágenes, en este instante solamente había filtrado alrededor de un tercio de las imágenes. En este entrenamiento ya podemos ver al modelo aprender a reconocer los núcleos, aunque aún no tan precisamente, como se muestra en la Ilustración 6.11.

Para esta prueba además de utilizar el dataset automáticamente segmentado para entrenar, también utilizo el dataset manualmente segmentado como set de validación, para ayudar con la precisión.

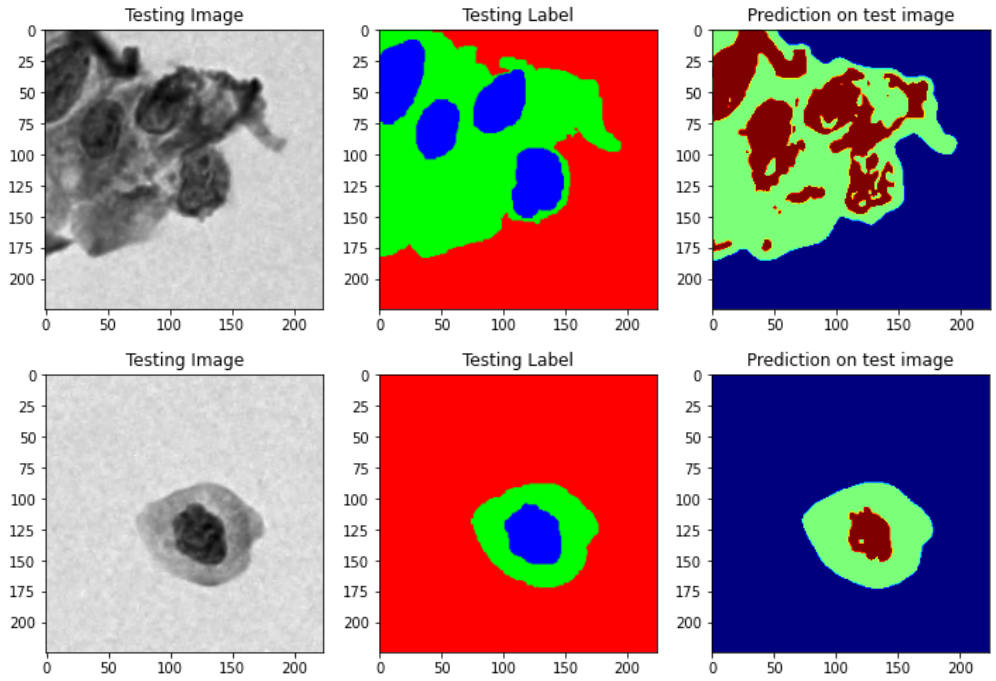


Ilustración 6.11: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia.

En las gráficas del entrenamiento de la Ilustración 6.12 se pueden apreciar aún dificultades con el aprendizaje y comportamientos erráticos, pero aun así la tasa de aciertos en el dataset de prueba es de 93.53%.

Ilustración 6.12: Gráficas de entrenamiento. (Izquierda) Pérdida. (Derecha) Tasa de aciertos. Fuente: Elaboración propia.

```
1/1 [==============================] - 0s 62ms/step - loss: 0.2513 - accuracy: 0.9353
Accuracy is =  93.53376030921936 %
```

Ilustración 6.13: Resultados al segmentar las imágenes del conjunto de prueba. Fuente: Elaboración propia.

Como conclusión de esta prueba, podemos ver un problema, al tener un dataset de validación diferente al de entrenamiento, el modelo no aprende de manera consistente, pero los resultados son mucho mejores que con el dataset pequeño, confirmando nuestra necesidad de aumentar el tamaño del dataset.

6.3.2. Retrospectiva del sprint

Durante este sprint se define la mayoría de la estructura a utilizar durante muchas de las siguientes pruebas, además de ser realizadas las primeras pruebas del modelo U-Net. También se ha logrado definir la manera de mostrar los resultados segmentados de manera semántica, en vez de las *bounding boxes* mostradas previamente.

También se confirma la necesidad de aumentar el tamaño del dataset, porque utilizando el dataset manual, aunque de muy buena calidad, es casi imposible entrenar un modelo que genere resultados aceptables pero generalizables a cualquier imagen de célula, pero sí encontramos un avance significativo al utilizar el dataset generado automáticamente en Apeer, a pesar de no haber utilizado todas las imágenes generadas aún, por falta de tiempo.

6.4. Sprint 4

En este sprint se terminará de filtrar las máscaras generadas en Apeer para completar el dataset, estudiar métricas específicas para segmentación para entender de manera más precisa, estudiar los posibles usos de *transfer learning* y *fine tuning* para los modelos y realizar más pruebas de U-Net con el dataset después de terminar de filtrar las imágenes.

Tabla 6.4: Organización de sprint 4. Fuente: Elaboración propia.

ID	Sprint	Historia de Usuario	Prioridad	Puntos de Historia
12	4	Estudio de métricas para segmentación semántica	1	2
13	4	Cambio de métricas	1	1
14	4	Prueba con dataset generado	1	3
15	4	Prueba con dataset por clase	1	5
16	4	Estudio de transfer learning y fine tuning	2	3

6.4.1. Cambio de métricas y más pruebas con U-Net

La métrica de la tasa de aciertos no es la más adecuada para entender el comportamiento de un modelo de segmentación, así que cambiamos la métrica a la Intersección entre la Unión o *IoU* (*Intersection over Union*), también conocida como el índice o coeficiente de Jaccard.

La métrica mide el grado de similitud entre dos conjuntos, calculado de la siguiente manera:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Fórmula 6.1: Índice de Jaccard. Fuente: Wikipedia.

Utilizando esta métrica podremos medir qué tanto se acerca la máscara predicha a la original, de manera global y por cada clase. Esta métrica está incluida en la librería Sklearn como ‘jaccard_score’.

En este momento se ha terminado de filtrar todas las imágenes segmentadas automáticamente por Apeer, definiendo el dataset a 669 imágenes automáticamente segmentadas.

6.4.1.1. Prueba con dataset segmentado automáticamente

Para asegurar mejores resultados he utilizado las 669 imágenes del nuevo dataset para el entrenamiento y las 50 del dataset manual para validación, realizando la prueba final en las mismas 12 imágenes del dataset manual.

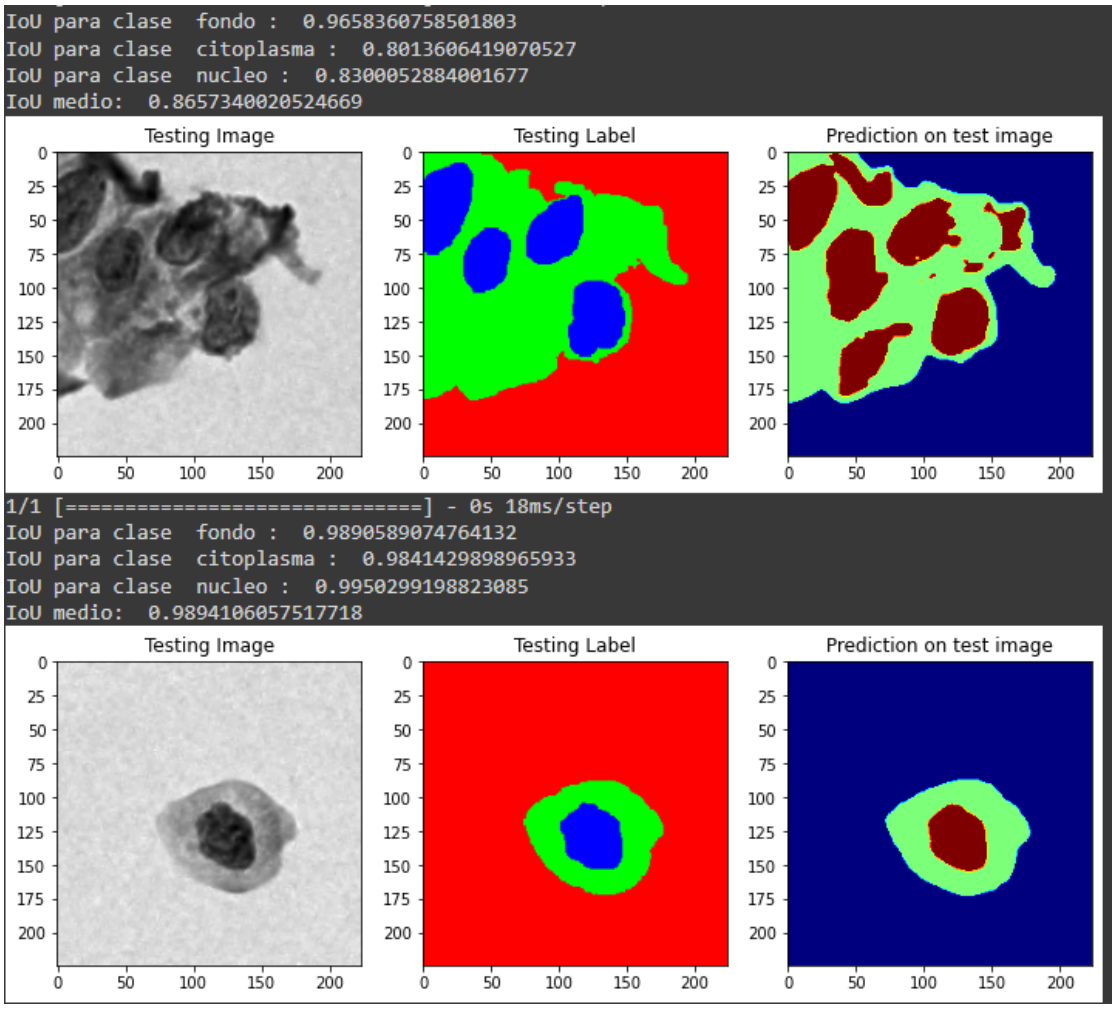


Ilustración 6.14: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia.

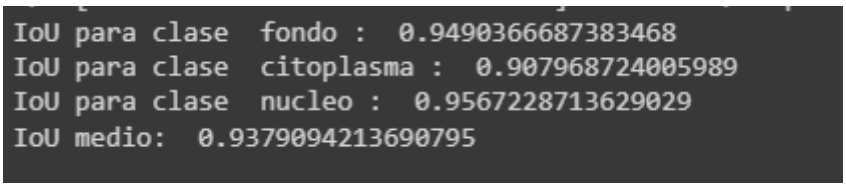


Ilustración 6.15: Métricas de segmentación. Fuente: Elaboración propia.

Ahora con la métrica de IoU se puede ver en las Ilustraciones 6.14 y 6.15 la capacidad del modelo más claramente, en células con núcleos definidos es capaz de reconocerlos sin dificultad pero aún hace falta precisión en imágenes más complejas, esto no es aparente en las imágenes de prueba porque la mayoría son sencillas.

La conclusión de esta prueba es que el modelo mejora bastante al aumentar el número de imágenes en el dataset, en caso de las imágenes sencillas, principalmente las benignas, es completamente capaz de reconocer las formas y segmentar las clases. Pero al momento de segmentar imágenes complejas, como las altogrado, ya no es capaz de identificar el núcleo a diferencia de otros orgánulos.

6.4.2. Prueba solo con imágenes de cada clase

Al ver que el modelo aún muestra dificultades segmentando imágenes de clases complejas, la siguiente prueba es realizada entrenando sólamente con las imágenes de cada clase, generando 4 modelos diferentes, se puede ver la distribución de datos en la Ilustración 6.16. El conjunto de imágenes de validación sigue siendo las imágenes manualmente segmentadas.

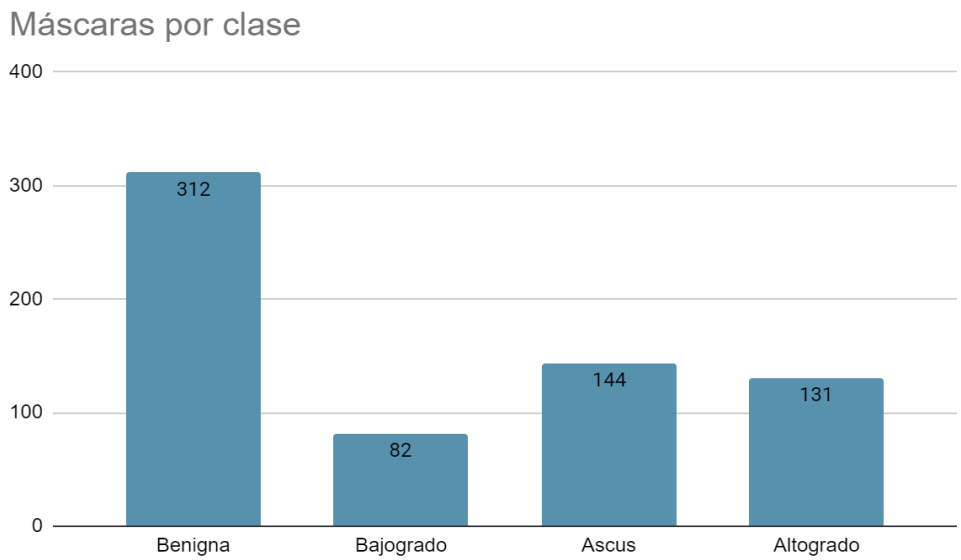


Ilustración 6.16: Distribución de las máscaras segmentadas automáticamente. Fuente: Elaboración propia.

6.4.2.1. Resultados Benigna

Los resultados del modelo con células benignas (Ilustraciones 6.17 y 6.18) son peores de lo esperado, generando zonas irregulares en los núcleos, a pesar de ser el conjunto de imágenes más grande, posiblemente sea porque son imágenes muy sencillas en comparación al resto.

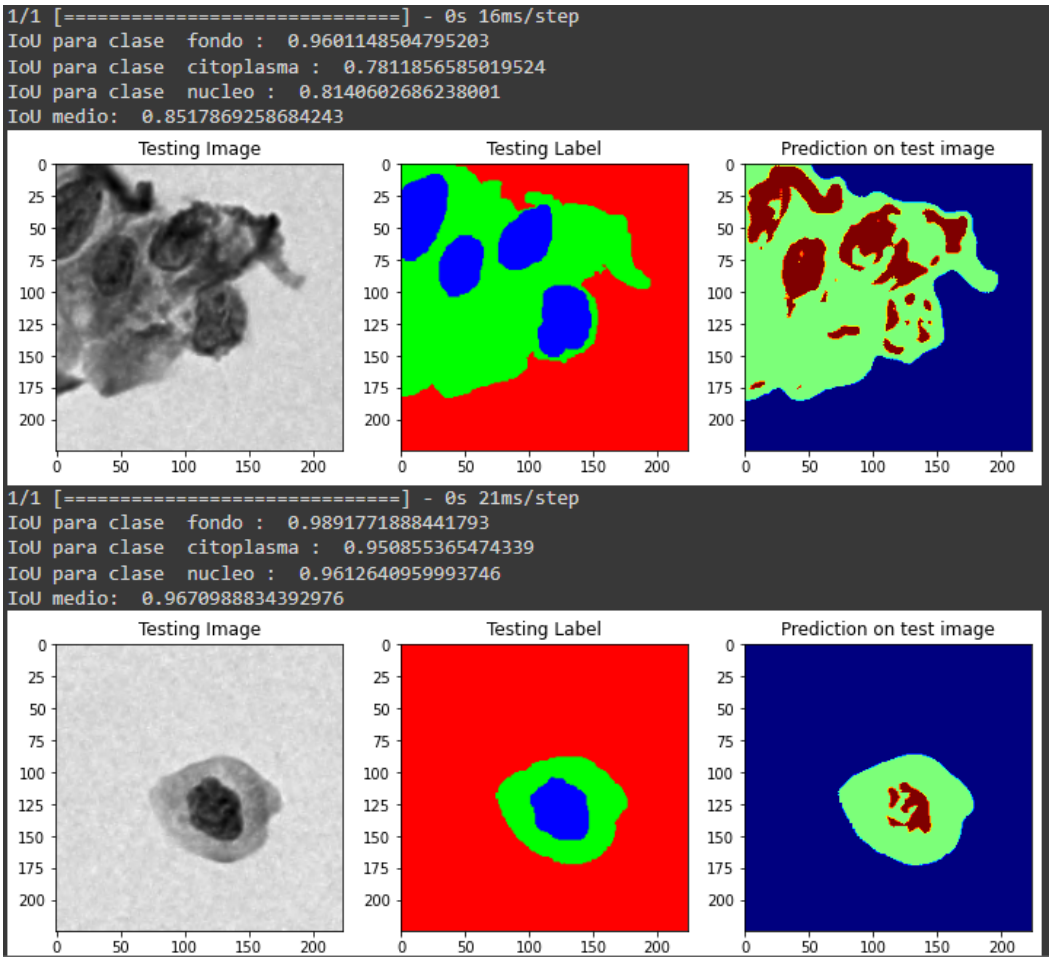


Ilustración 6.17: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia.

```
1/1 [==============================] - 0s 225ms/step
IoU para clase  fondo :  0.9545473727264759
IoU para clase  citoplasma :  0.889601280421786
IoU para clase  nucleo :  0.9320444736799884
IoU medio:  0.9253977089427501
```

Ilustración 6.18: Métricas de segmentación. Fuente: Elaboración propia.

6.4.2.2. Resultados Bajogrado

Al contrario, el conjunto de imágenes bajogrado logró generar mejores resultados (Ilustraciones 6.19 y 6.20) que el modelo de benignas, a pesar de ser el más pequeño de todo el dataset. Aun así, continúa teniendo problemas con los núcleos.

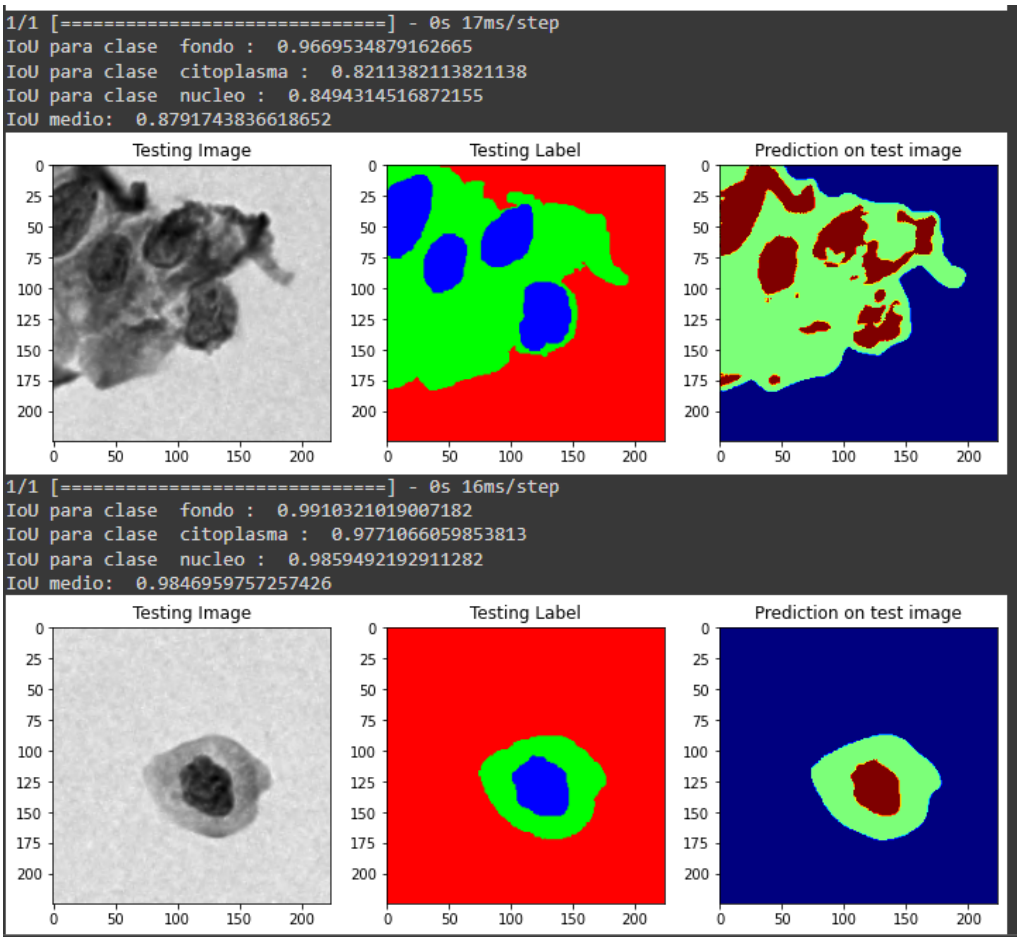


Ilustración 6.19: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia.

```
1/1 [==============================] - 0s 208ms/step
IoU para clase  fondo :  0.9403189007334459
IoU para clase  citoplasma :  0.8899838502356547
IoU para clase  nucleo :  0.946525758381516
IoU medio:  0.9256095031168723
```

Ilustración 6.20: Métricas de segmentación. Fuente: Elaboración propia.

6.4.2.3. Resultados Ascus

El modelo de ascus parece ser igual a los previos a simple vista pero en el conjunto total de prueba muestra peores resultados (Ilustraciones 6.21 y 6.22) que los anteriores.

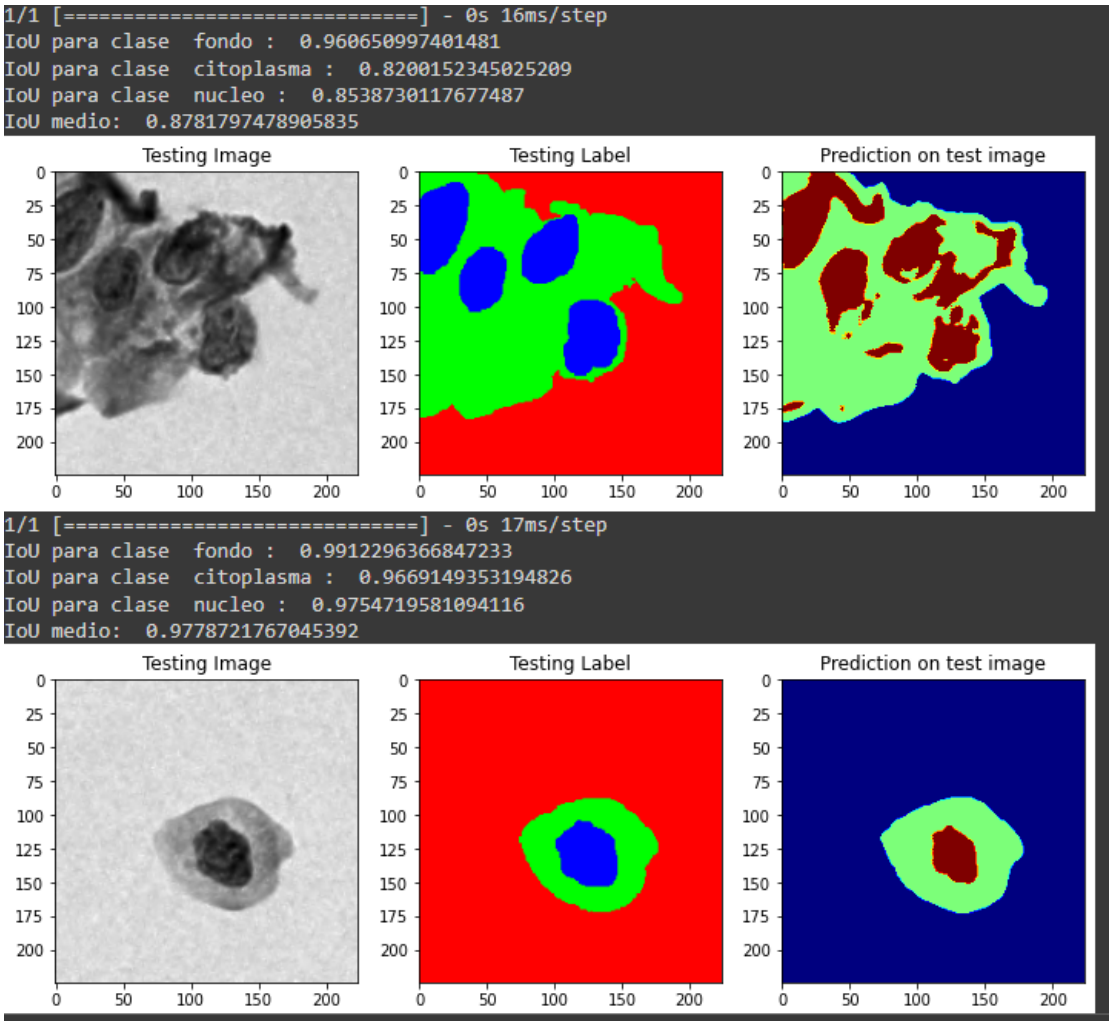


Ilustración 6.21: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia.

```
1/1 [==============================] - 0s 211ms/step
IoU para clase  fondo :  0.9169833186614814
IoU para clase  citoplasma :  0.8515328424506414
IoU para clase  nucleo :  0.9281590147082841
IoU medio:  0.898891725273469
```

Ilustración 6.22: Métricas de segmentación. Fuente: Elaboración propia.

6.4.2.4. *Resultados Altogrado*

El modelo altogrado se comporta prácticamente igual que el modelo ascus, con una leve mejora en las métricas finales (Ilustraciones 6.23 y 6.24).

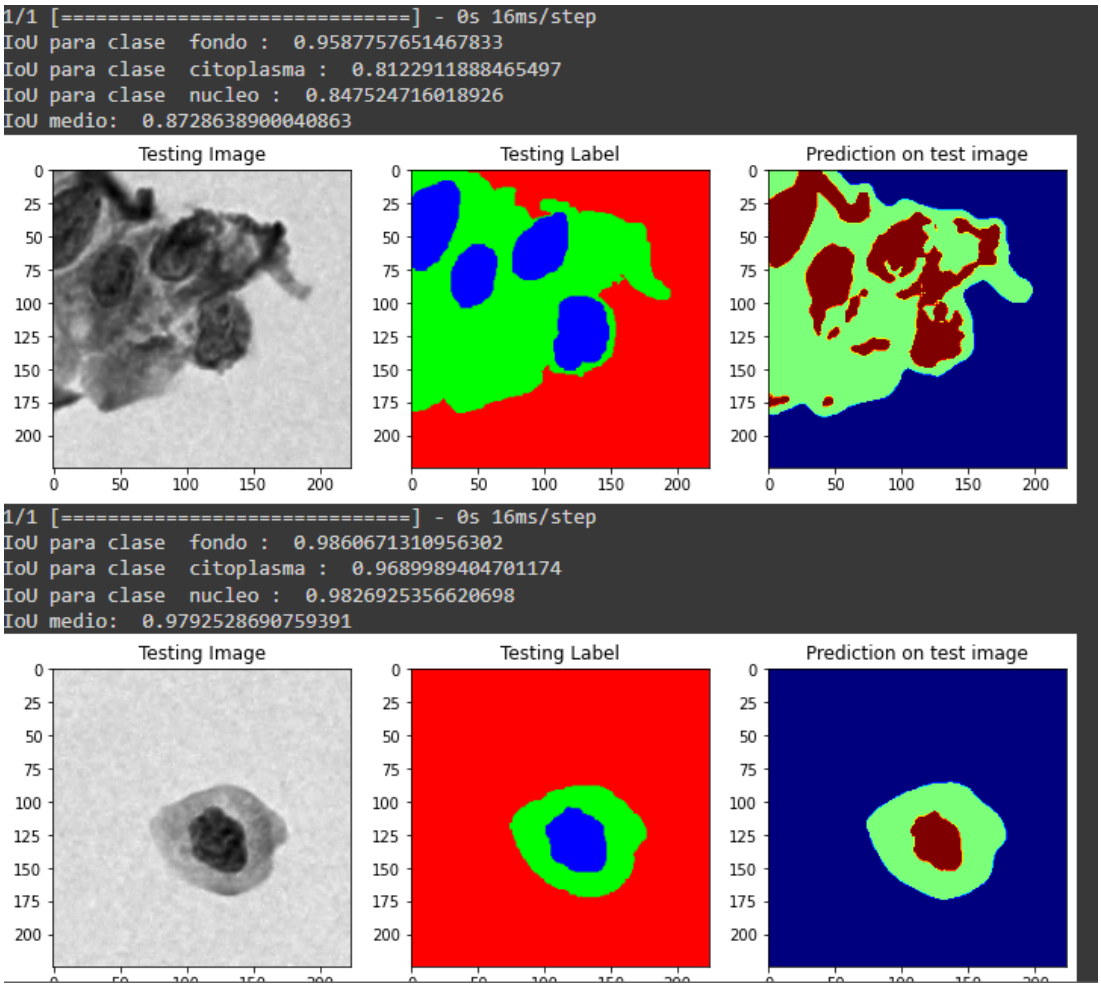


Ilustración 6.23: Resultados de segmentación. (Columna izquierda) imágenes de entrada. (Columna central) Máscara original. (Columna derecha) Máscara predicha. Fuente: Elaboración propia.

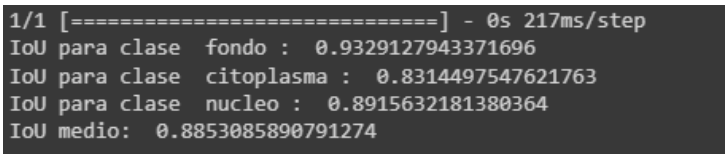


Ilustración 6.24: Métricas de segmentación. Fuente: Elaboración propia.

Como conclusión de las pruebas podemos ver que el modelo que utiliza las 669 imágenes para entrenamiento genera por media mejores resultados, así que para mejorar es necesario aumentar el dataset o mejorar la calidad de este. En la Ilustración 6.25 podemos observar la comparación de los resultados.

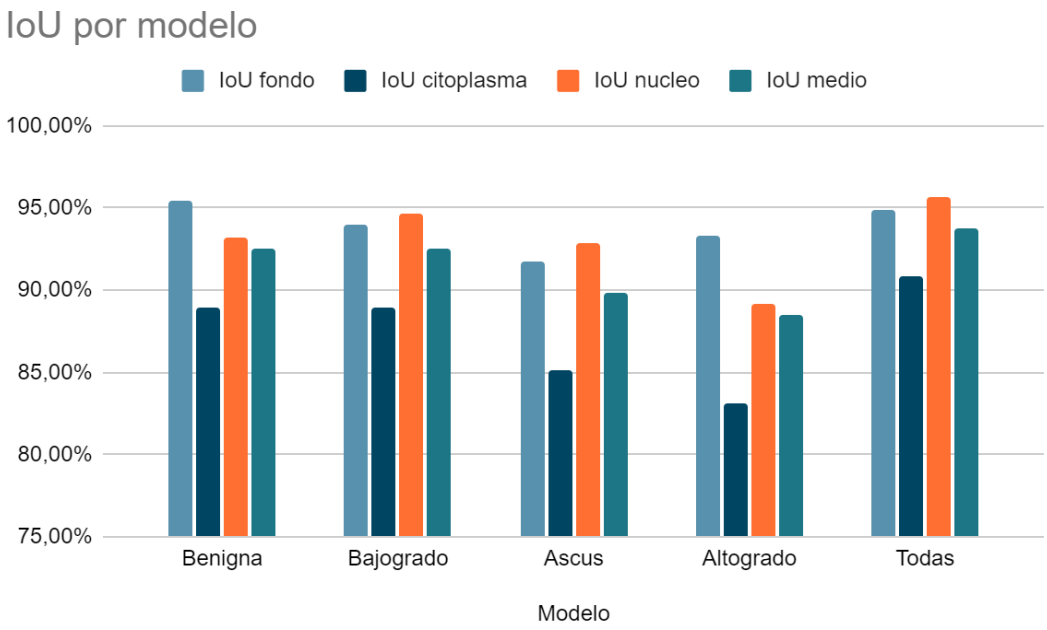


Ilustración 6.25: Gráfico de barras de comparación de resultados de IoU entre modelos entrenados. Fuente: Elaboración propia.

6.4.3. Retrospectiva del sprint

Durante este sprint fue posible identificar e implementar una métrica para la segmentación semántica que nos permite apreciar de una manera más intuitiva los resultados. Con la nueva métrica es más fácil entender el comportamiento del modelo en las pruebas realizadas, encontrando que al final nuestro modelo aporta mejores resultados utilizando el dataset más grande y es capaz de generalizar el conocimiento de las 669 imágenes.

A partir del estudio de *fine tuning* y *transfer learning* se encontró un repositorio de GitHub llamado Segmentation Models que será utilizada para posteriores pruebas.

6.5. Sprint 5

Durante este sprint las metas son seguir mejorando el modelo y aumentando el tamaño y calidad del dataset, así que se realiza una prueba comparando el modelo actual con otros modelos que han aprendido a partir de otro dataset y luego son reentrenados realizando *fine tuning*, siendo el propósito de la prueba entender las potenciales ventajas que puede ofrecer el *fine tuning* para esta problemática.

Para incrementar el dataset utilizado, dado que el modelo actual se comporta muy bien, se decide utilizarlo para que segmente todas las imágenes del dataset.

Tabla 6.5: Organización de sprint 5. Fuente: Elaboración propia.

ID	Sprint	Historia de Usuario	Prioridad	Puntos de Historia
17	5	Comparación de U-Net con otras arquitecturas pre-entrenadas	2	8
18	5	Segmentación del dataset completo	1	2

6.5.1. Comparaciones de U-Net utilizando modelos pre-entrenados

El modelo muestra buenos resultados, pero aún deja espacio para mejorar, así que realizamos una comparación con otros modelos.

Para comparar nuestra arquitectura con otros modelos utilizamos una librería llamada Segmentation Models por Iakubovskii (2019) que permite reemplazar el *feature extractor* o codificador de U-Net con el codificador de otros modelos de clasificación de imágenes, llamados *backbones*.

En este caso utilizo tres backbones de diferentes modelos de aprendizaje profundo, Resnet34, Inception V3 y VGG16, comparados con la arquitectura original de U-Net previamente entrenada. Estos modelos han mostrado buenos resultados en tareas de clasificación de imágenes.

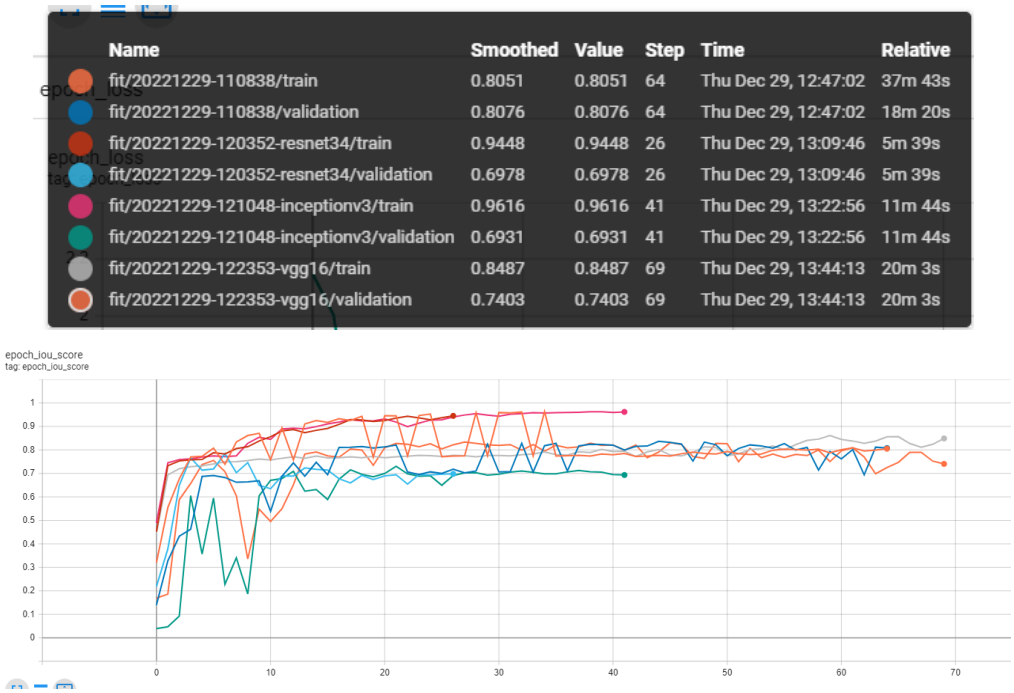
Los modelos son inicializados con los pesos de la clasificación de imágenes previa con el dataset de ImageNet, una colección de miles de imágenes etiquetadas de diferentes clases pública para investigación.

Además de esto, se realizan cambios en los parámetros del modelo:

- Learning rate: 1e-4
- Loss: Dice Loss + Categorical Focal Loss

El cambio de la tasa de aprendizaje es para aprovechar los pesos previos de los modelos pero permitir que aprenda de las nuevas imágenes. El cambio de la pérdida es para ayudar con el desbalance de instancias de clases en las imágenes, ya que los núcleos son una minoría.

La librería de Segmentation Models incluye las nuevas pérdidas y las métricas de IoU y la F-1 score para tener mejor visualización del aprendizaje del modelo durante el entrenamiento. El dataset de entrenamiento y de validación continúan siendo los mismos. En las siguientes Ilustraciones 6.26 y 6.27 se puede ver el comportamiento de los modelos durante el entrenamiento.



Name	Smoothed	Value	Step	Time	Relative
fit/20221229-110838/train	0.8051	0.8051	64	Thu Dec 29, 12:47:02	37m 43s
fit/20221229-110838/validation	0.8076	0.8076	64	Thu Dec 29, 12:47:02	18m 20s
fit/20221229-120352-resnet34/train	0.9448	0.9448	26	Thu Dec 29, 13:09:46	5m 39s
fit/20221229-120352-resnet34/validation	0.6978	0.6978	26	Thu Dec 29, 13:09:46	5m 39s
fit/20221229-121048-inceptionv3/train	0.9616	0.9616	41	Thu Dec 29, 13:22:56	11m 44s
fit/20221229-121048-inceptionv3/validation	0.6931	0.6931	41	Thu Dec 29, 13:22:56	11m 44s
fit/20221229-122353-vgg16/train	0.8487	0.8487	69	Thu Dec 29, 13:44:13	20m 3s
fit/20221229-122353-vgg16/validation	0.7403	0.7403	69	Thu Dec 29, 13:44:13	20m 3s

Ilustración 6.26: IoU durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia.

Name	Smoothed	Value	Step	Time	Relative
fit/20221229-110838/train	0.7319	0.7319	64	Thu Dec 29, 12:47:02	37m 43s
fit/20221229-110838/validation	0.7335	0.7335	64	Thu Dec 29, 12:47:02	31m 4s
fit/20221229-120352-resnet34/train	0.686	0.686	26	Thu Dec 29, 13:09:46	5m 39s
fit/20221229-120352-resnet34/validation	0.7996	0.7996	26	Thu Dec 29, 13:09:46	5m 39s
fit/20221229-121048-inceptionv3/train	0.6808	0.6808	41	Thu Dec 29, 13:22:56	11m 44s
fit/20221229-121048-inceptionv3/validation	0.8205	0.8205	41	Thu Dec 29, 13:22:56	11m 44s
fit/20221229-122353-vgg16/train	0.7177	0.7177	69	Thu Dec 29, 13:44:13	20m 3s
fit/20221229-122353-vgg16/validation	0.7555	0.7555	69	Thu Dec 29, 13:44:13	20m 3s

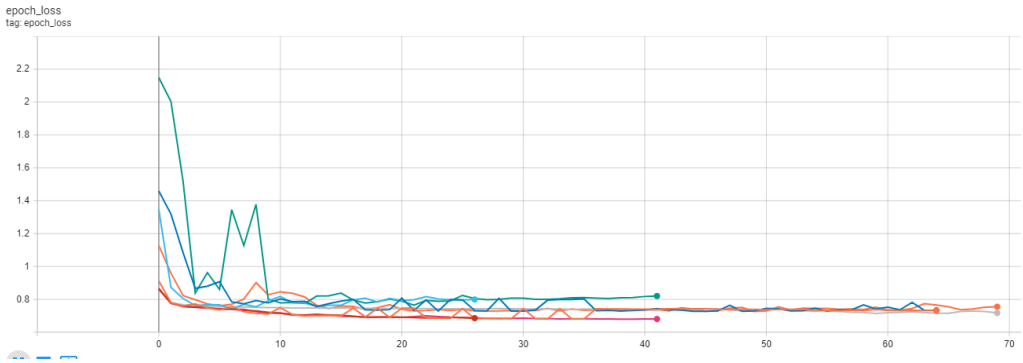


Ilustración 6.27: Pérdida total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia.

```
1/1 [==============================] - 0s 20ms/step
IoU para clase  fondo :  0.9315920158604893
IoU para clase  citoplasma :  0.7762044674147758
IoU para clase  nucleo :  0.829871811236119
IoU medio:  0.8458894315037947
1/1 [==============================] - 0s 26ms/step
IoU para clase  fondo :  0.9411571271060216
IoU para clase  citoplasma :  0.866631945090307
IoU para clase  nucleo :  0.9209800918836141
IoU medio:  0.909589721359981
1/1 [==============================] - 0s 18ms/step
IoU para clase  fondo :  0.9294751009421265
IoU para clase  citoplasma :  0.825013184934621
IoU para clase  nucleo :  0.8879837450379094
IoU medio:  0.8808240103048856
1/1 [==============================] - 0s 23ms/step
IoU para clase  fondo :  0.9331548226772746
IoU para clase  citoplasma :  0.8823529411764706
IoU para clase  nucleo :  0.9455980146959033
IoU medio:  0.9203685928498828
```

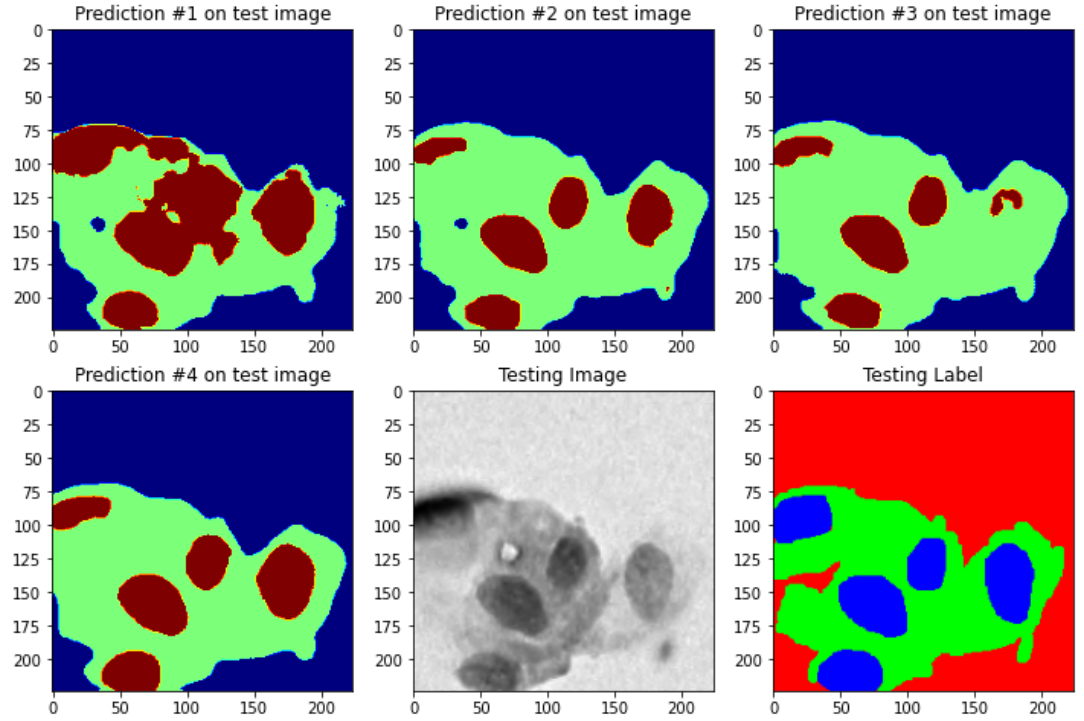


Ilustración 6.28: Métrica de IoU (arriba). Resultados de segmentación (abajo). (1) Predicción con Resnet34. (2) Predicción con InceptionV3. (3) Predicción con VGG16. (4) Predicción con U-Net. Fuente: Elaboración propia.

Comparando los resultados (Ilustración 6.28) con las imágenes de prueba los modelos con diferente backbones se comportan igual o peor que el modelo original. Para verificar cómo se comportarían con otras imágenes más complejas se realizó la prueba con imágenes sin máscara de segmentación (Ilustración 6.29).

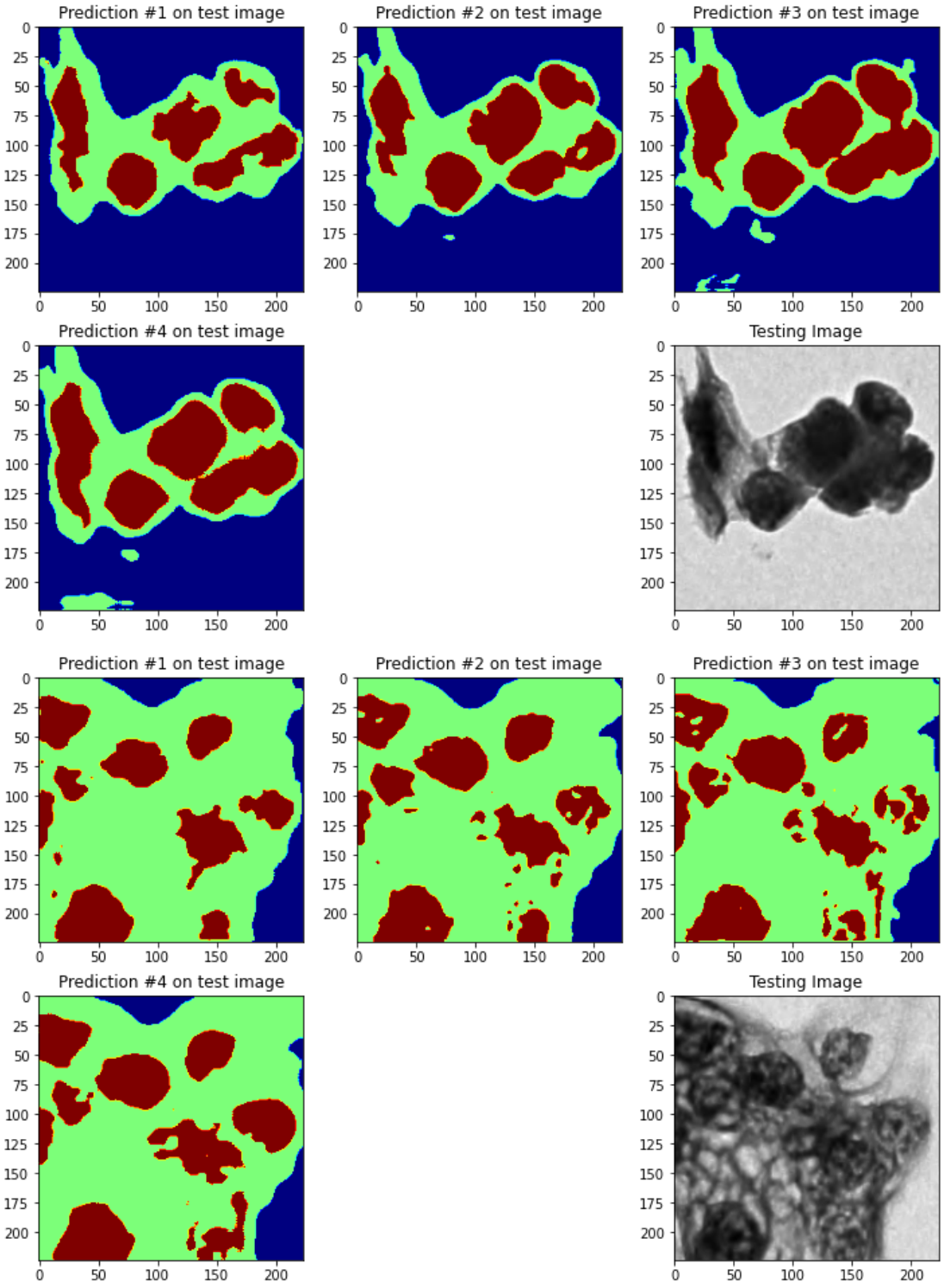


Ilustración 6.29: Resultados de segmentación. (1) Predicción con Resnet34. (2) Predicción con InceptionV3. (3) Predicción con VGG16. (4) Predicción con U-Net. Fuente: Elaboración propia.

Al utilizar los modelos con imágenes más complejas del dataset encontramos que hace falta mayor representación de imágenes con núcleos deformados en el dataset de entrenamiento, porque muchas veces solo segmenta cualquier forma oscura. A pesar de esto, el modelo original U-Net se comporta mejor que los otros con diferentes backbones.

6.5.2. Generación del dataset completo

Ahora con el modelo original de U-Net generamos las máscaras para todas las imágenes del dataset, así casi triplicamos el tamaño del dataset. Para visualizar mejor cómo quedará el dataset, en las Ilustraciones 6.30 - 6.33 se puede apreciar la máscara generada para cada clase del dataset original.

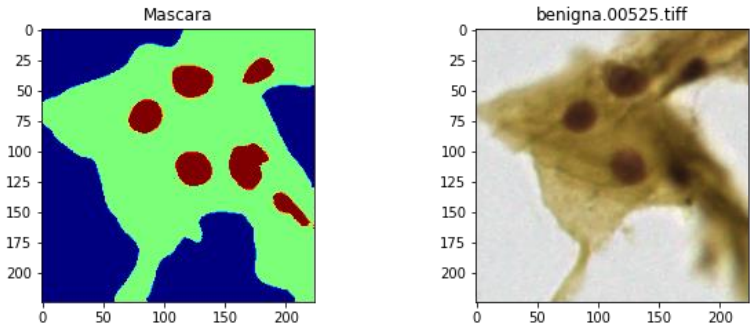


Ilustración 6.30: Máscara y célula benigna. Fuente: Elaboración propia.

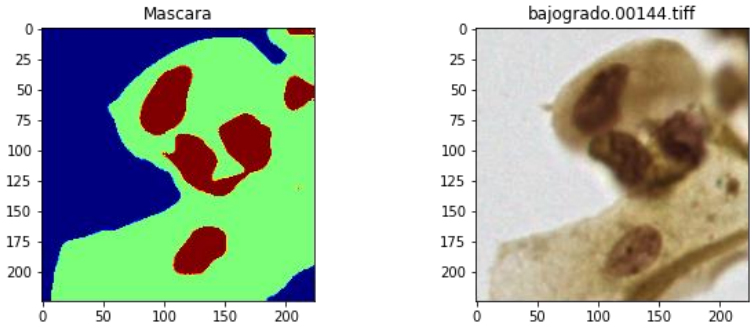


Ilustración 6.31: Máscara y célula bajogrado. Fuente: Elaboración propia.

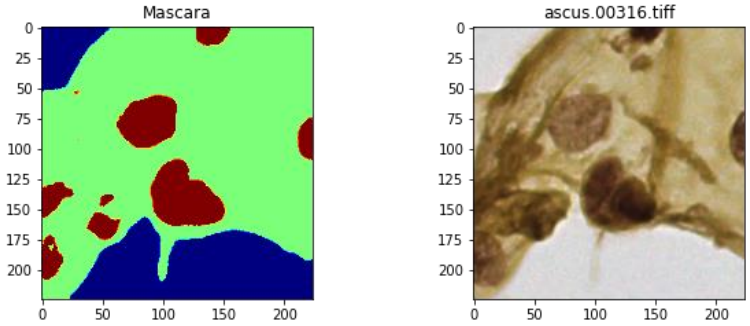


Ilustración 6.32: Máscara y célula ascus Fuente: Elaboración propia.

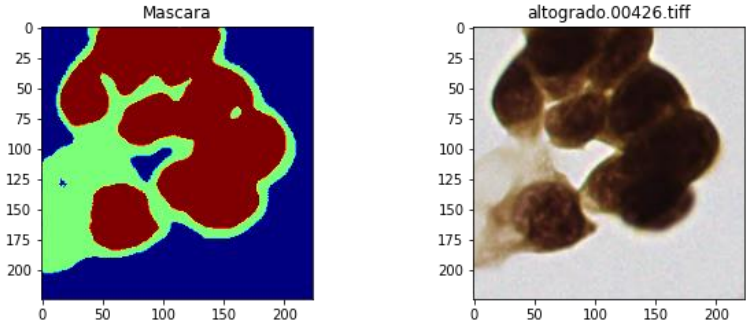


Ilustración 6.33: Máscara y célula altogrado. Fuente: Elaboración propia.

6.5.3. Retrospectiva del sprint

Durante este sprint fue posible mejorar el modelo levemente, al utilizar un conjunto de métodos de pérdida dirigidos a la segmentación, pero en la prueba no se logró encontrar algo que ayude a mejorar el modelo.

Lo que sí fue posible es incrementar el tamaño del dataset, ya que al ver que el modelo actual de U-Net es capaz de generar resultados aceptables con un IoU medio de casi 95%, se decidió que ya era momento de segmentar todo el dataset y utilizarlo para posteriores entrenamientos y pruebas.

La calidad aparente de las máscaras generadas es suficientemente adecuada, perdiendo precisión solo en las células más complejas. Considerando que se parte del etiquetado manual realizado por el autor de este TFG y no por un oncólogo o ginecólogo, existe un grado de error intrínseco que se considera aceptable para esta investigación.

6.6. Sprint 6

Dado que en los previos sprints se han ido agrupando diferentes datasets con diferentes calidades, en este sprint se realizarán pruebas con todos estos para identificar si es posible mejorar la calidad de las predicciones utilizando alguna combinación de estos.

También se realizará la prueba de utilizar todas las imágenes de cada clase para entrenar varios modelos, a pesar de haberla hecho previamente, ahora que está todo el dataset segmentado, se poseen más imágenes de entrenamiento, especialmente en las clases más complejas como altogrado y bajogrado.

Tabla 6.6: Organización de sprint 6. Fuente: Elaboración propia.

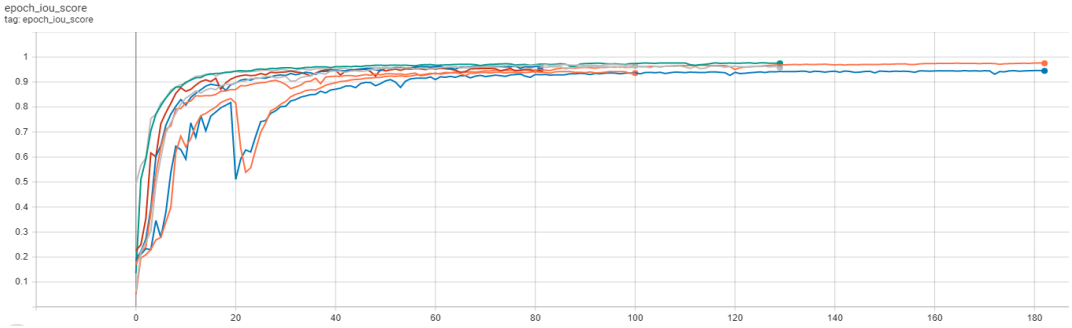
ID	Sprint	Historia de Usuario	Prioridad	Puntos de Historia
19	6	Prueba con dataset por clase	1	3
20	6	Comparación global de diferentes datasets	1	5

6.6.1. Pruebas con dataset por clase y comparación global

En este momento tenemos tres datasets diferentes, el dataset manual con 62 imágenes que tiene mejor calidad, el dataset de Apeer con 669 imágenes de calidad regular y el completo con 1863 imágenes con calidad variable.

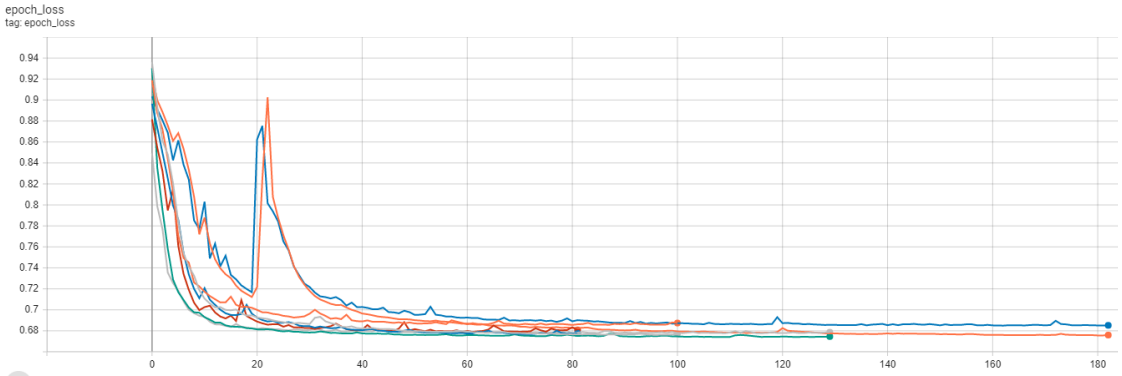
En la siguiente prueba utilizamos el dataset completo para entrenamiento y tres validaciones diferentes; con 20% del dataset completo, con ese 20% pero remplazado con las máscaras del dataset de Apeer, para evitar duplicados, y con el dataset manual. Los parámetros de entrenamiento y del modelo son iguales a la prueba previa.

Además de estos datasets, también es posible entrenar el modelo con sólamente las imágenes de una clase, así que también se incluyen las cuatro pruebas de cada clase utilizando el 20% para validación. El comportamiento de los modelos se puede visualizar en las Ilustraciones 34 - 37.



Name	Smoothed	Value	Step	Time	Relative
fit/20230217-135809-ascus/train	0.9647	0.9647	100	Fri Feb 17, 15:02:17	4m 0s
fit/20230217-135809-ascus/validation	0.9356	0.9356	100	Fri Feb 17, 15:02:17	4m 0s
fit/20230217-140236-altogrado/train	0.9621	0.9621	81	Fri Feb 17, 15:06:50	4m 5s
fit/20230217-140236-altogrado/validation	0.9478	0.9478	81	Fri Feb 17, 15:06:50	4m 5s
fit/20230217-142032-benigna/train	0.9749	0.9749	129	Fri Feb 17, 15:33:25	12m 41s
fit/20230217-142032-benigna/validation	0.9584	0.9584	129	Fri Feb 17, 15:33:25	12m 41s
fit/20230217-143358-bajogrado/train	0.975	0.975	182	Fri Feb 17, 15:39:33	5m 29s
fit/20230217-143358-bajogrado/validation	0.9446	0.9446	182	Fri Feb 17, 15:39:33	5m 29s

Ilustración 6.34: IoU durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia.



Name	Smoothed	Value	Step	Time	Relative
fit/20230217-135809-ascus/train	0.6772	0.6772	100	Fri Feb 17, 15:02:17	4m 0s
fit/20230217-135809-ascus/validation	0.6874	0.6874	100	Fri Feb 17, 15:02:17	4m 0s
fit/20230217-140236-altogrado/train	0.6776	0.6776	81	Fri Feb 17, 15:06:50	4m 5s
fit/20230217-140236-altogrado/validation	0.6808	0.6808	81	Fri Feb 17, 15:06:50	4m 5s
fit/20230217-142032-benigna/train	0.6745	0.6745	129	Fri Feb 17, 15:33:25	12m 41s
fit/20230217-142032-benigna/validation	0.6786	0.6786	129	Fri Feb 17, 15:33:25	12m 41s
fit/20230217-143358-bajogrado/train	0.6759	0.6759	182	Fri Feb 17, 15:39:33	5m 29s
fit/20230217-143358-bajogrado/validation	0.6852	0.6852	182	Fri Feb 17, 15:39:33	5m 29s

Ilustración 6.35: Pérdida total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia.

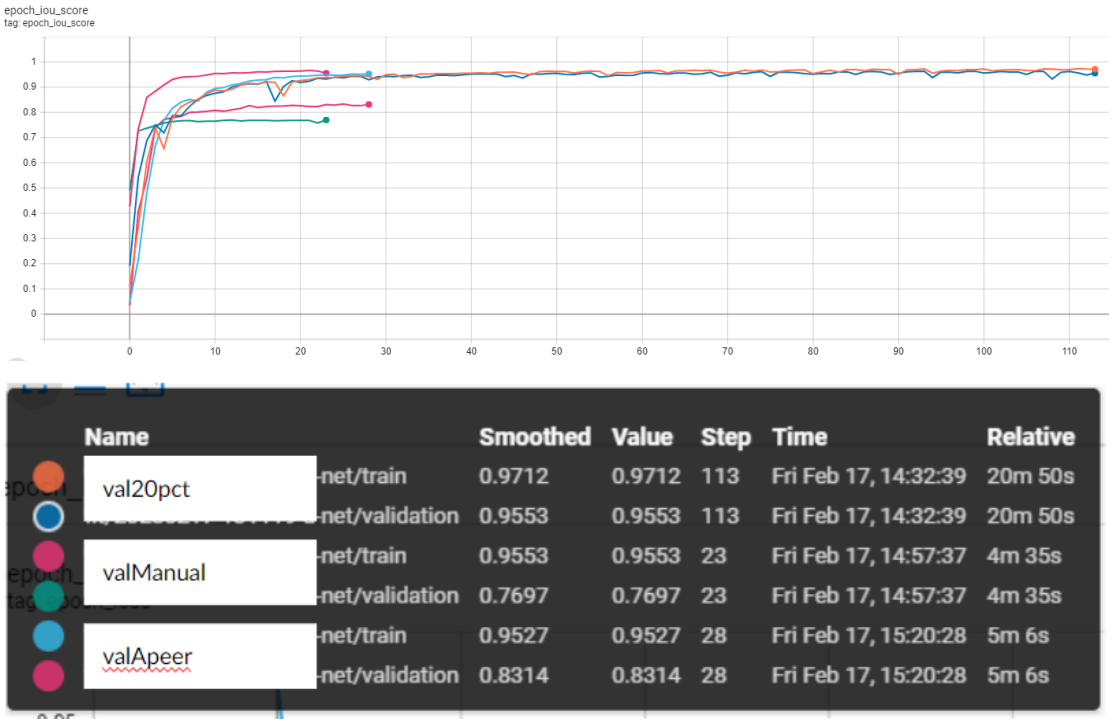


Ilustración 6.36: IoU durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia.

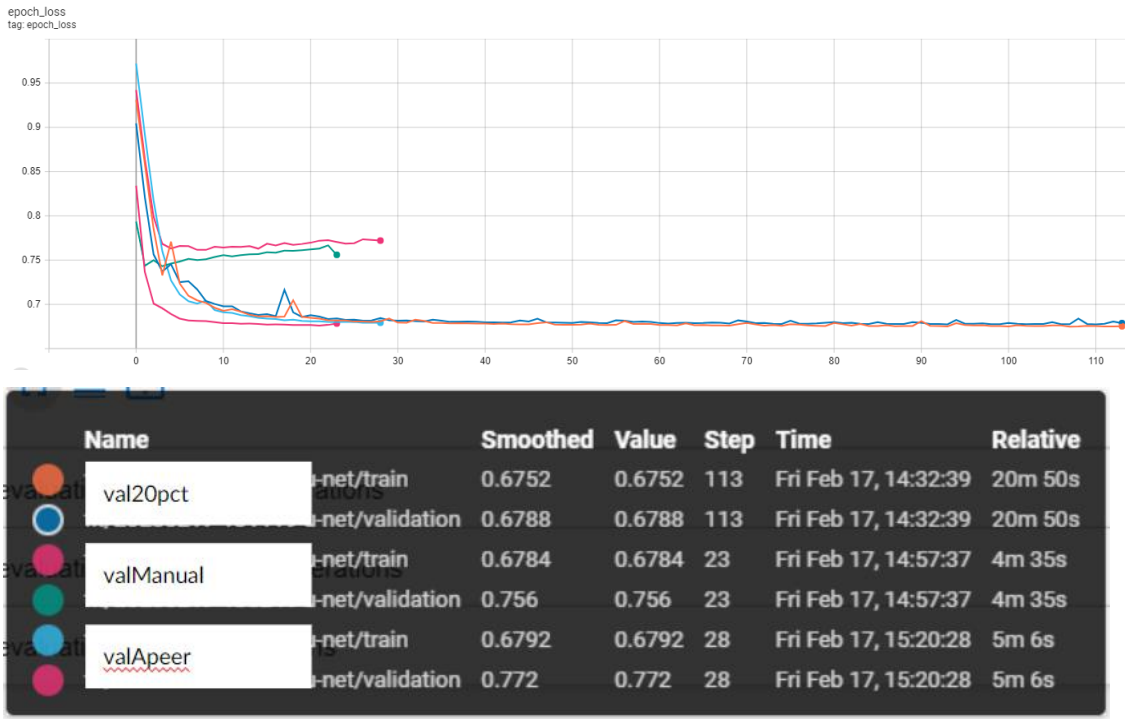


Ilustración 6.37: Pérdida total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia.

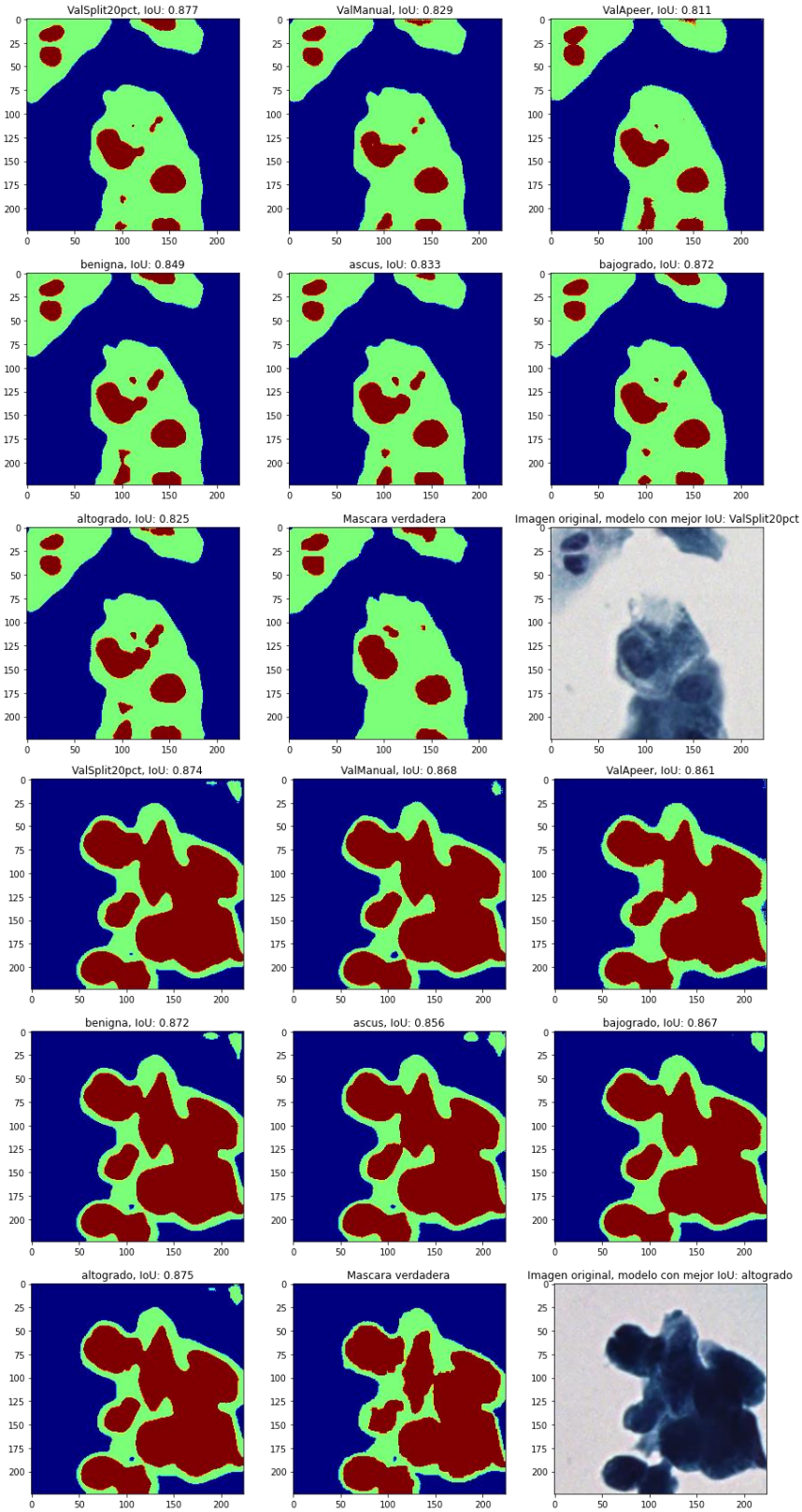


Ilustración 6.38: Resultados de segmentación. Células bajogrado (arriba) y altogrado (abajo) Fuente: Elaboración propia.

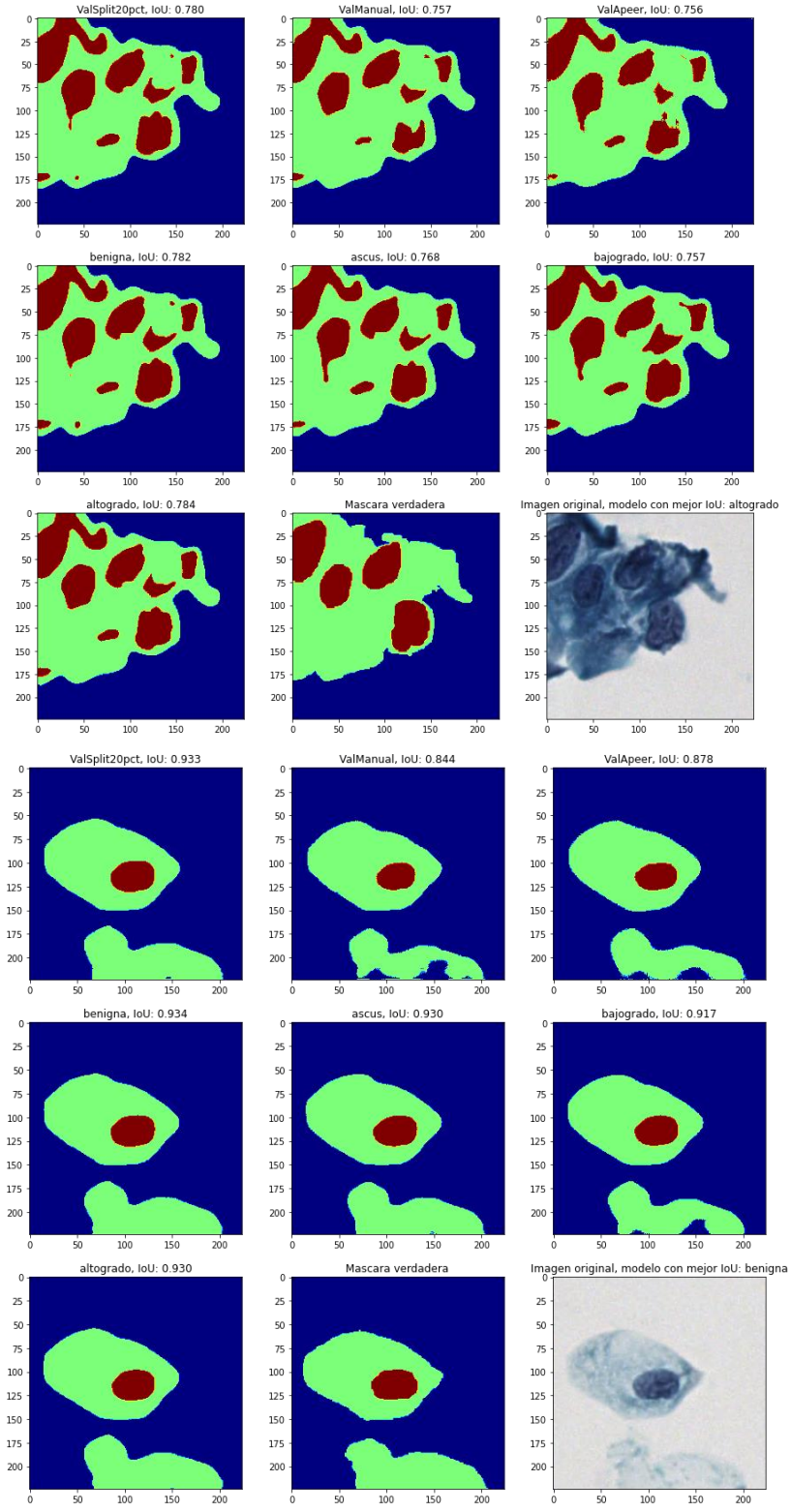


Ilustración 6.39: Resultados de segmentación. Células altogrado (arriba) y benigna (abajo)
Fuente: Elaboración propia.

Model	IoU para fondo	IoU para citoplasma	IoU para nucleo	IoU medio
ValSplit20pct	0.7765	0.8927	0.791	0.8201
ValManual	0.7725	0.8738	0.7362	0.7942
ValApeer	0.7665	0.8717	0.7328	0.7903
benigna	0.7779	0.8941	0.7915	0.8212
ascus	0.7754	0.8863	0.7748	0.8122
bajogrado	0.7745	0.8786	0.7483	0.8005
altogrado	0.7763	0.8857	0.7632	0.8084

Ilustración 6.40: Métricas con el dataset de prueba manual. Fuente: Elaboración propia.

Model	IoU para fondo	IoU para citoplasma	IoU para nucleo	IoU medio
ValSplit20pct	0.7635	0.8471	0.7238	0.7781
ValManual	0.7624	0.8302	0.6745	0.7557
ValApeer	0.7531	0.8284	0.6689	0.7501
benigna	0.764	0.8457	0.7213	0.777
ascus	0.7622	0.8396	0.7027	0.7682
bajogrado	0.7652	0.8372	0.6934	0.7652
altogrado	0.7613	0.8456	0.719	0.7753

Ilustración 6.41: Métricas con el dataset de entrenamiento manual. Fuente: Elaboración propia.

En los resultados (Ilustraciones 6.38 - 6.41) de esta prueba la métrica es más estricta a pesar de generar buenos resultados a la vista. Los modelos con mayor dataset y menor diferencia con la validación parecen generar mejores resultados, específicamente el modelo con el dataset completo y 20% para validación y el modelo entrenado sólo con células de clase benigna.

Podemos concluir de esta prueba que con el dataset actual, el modelo es capaz de diferenciar las clases mayoritarias, fondo y citoplasma, sin problema, mientras que los núcleos son casi todos identificados, pero también cualquier

mancha oscura en la imagen. Sería necesario un preprocesamiento de las máscaras de entrenamiento para generar un dataset más adecuado.

6.6.2. Retrospectiva del sprint

En este sprint se confirma otra vez que lo más importante es tener un dataset amplio con calidad consistente, sin aportar mejores resultados al momento de usar los otros como validación. La prueba de entrenar modelos por clase tampoco genera ninguna ventaja aparente.

Considerando que ya se han segmentado todas las imágenes del dataset original, no es posible incrementarlo más, así que para generar mejores resultados es necesario mejorar el modelo o la calidad del dataset.

6.7. Sprint 7

Durante la investigación de modelos de segmentación se encontraron múltiples arquitecturas de modelos diferentes, pero también muchas arquitecturas basadas en U-Net pero más complejas. En un principio se eligió U-Net por su sencillez y parecido a una red neuronal convolucional, pero en este sprint lo compararemos con otros modelos de segmentación.

Tabla 6.7: Organización de sprint 7. Fuente: Elaboración propia.

ID	Sprint	Historia de Usuario	Prioridad	Puntos de Historia
21	7	Pruebas con diferentes modelos de segmentación	2	8

6.7.1. Pruebas de diferentes modelos de segmentación

Al tener un dataset amplio de imágenes y máscaras, es posible comparar la capacidad de segmentación del mejor modelo U-Net entrenado hasta el momento, el que utilizaba todo el dataset y un 20% para validación, con otros modelos de segmentación actuales.

Los otros modelos de segmentación actuales proveen de un repositorio en GitHub llamado *Keras U-net Collection* (Sha, 2021), que posee la implementación de 10 arquitecturas diferentes (Ilustración 6.42).

keras_unet_collection.models	Name	Reference
unet_2d	U-net	Ronneberger et al. (2015)
vnet_2d	V-net (modified for 2-d inputs)	Milletari et al. (2016)
unet_plus_2d	U-net++	Zhou et al. (2018)
r2_unet_2d	R2U-Net	Alom et al. (2018)
att_unet_2d	Attention U-net	Oktay et al. (2018)
resunet_a_2d	ResUnet-a	Diakogiannis et al. (2020)
u2net_2d	U^2-Net	Qin et al. (2020)
unet_3plus_2d	UNET 3+	Huang et al. (2020)
transunet_2d	TransUNET	Chen et al. (2021)
swin_unet_2d	Swin-UNET	Hu et al. (2021)

Ilustración 6.42: Modelos de Keras U-Net Collection. Fuente: Sha, Y. 2021

El modelo de U-Net a utilizar es con el que se ha trabajado durante todo el proyecto entrenado con el dataset completo. Los modelos *R2U-Net, Attention U-net y TransUNET* no fueron capaces de ser implementados dado que estaban orientados a segmentación semántica binaria, no multiclase que es nuestro caso, así que no fueron incluidos en las pruebas por malos resultados.

Todos los modelos además de U-Net poseen un número considerablemente mayor de neuronas así que al momento de entrenar y predecir requieren mayor cantidad de recursos, por lo cual no se han almacenado y comparado los datos de entrenamiento y validación.

Model	IoU para fondo	IoU para citoplasma	IoU para nucleo	IoU medio
unet_plus	0.8641	0.88	0.8822	0.8754
unet_3_plus	0.8661	0.8765	0.8851	0.8759
vnet	0.8675	0.8822	0.8897	0.8798
resunet_a	0.8665	0.8824	0.8882	0.8791
u2net	0.8677	0.8779	0.8758	0.8738
swim_unet	0.6741	0.6936	0.8778	0.7485
unet	0.8654	0.878	0.885	0.8761

Ilustración 6.43: Métricas con el dataset de prueba manual. Fuente: Elaboración propia.

Model	IoU para fondo	IoU para citoplasma	IoU para nucleo	IoU medio
unet_plus	0.8509	0.8391	0.8311	0.8404
unet_3_plus	0.8555	0.8451	0.8358	0.8455
vnet	0.8539	0.8421	0.8311	0.8424
resunet_a	0.8495	0.8378	0.8325	0.8399
u2net	0.854	0.8429	0.8356	0.8442
swim_unet	0.6771	0.6859	0.8309	0.7313
unet	0.8533	0.8427	0.8354	0.8438

Ilustración 6.44: Métricas con el dataset de entrenamiento manual. Fuente: Elaboración propia.

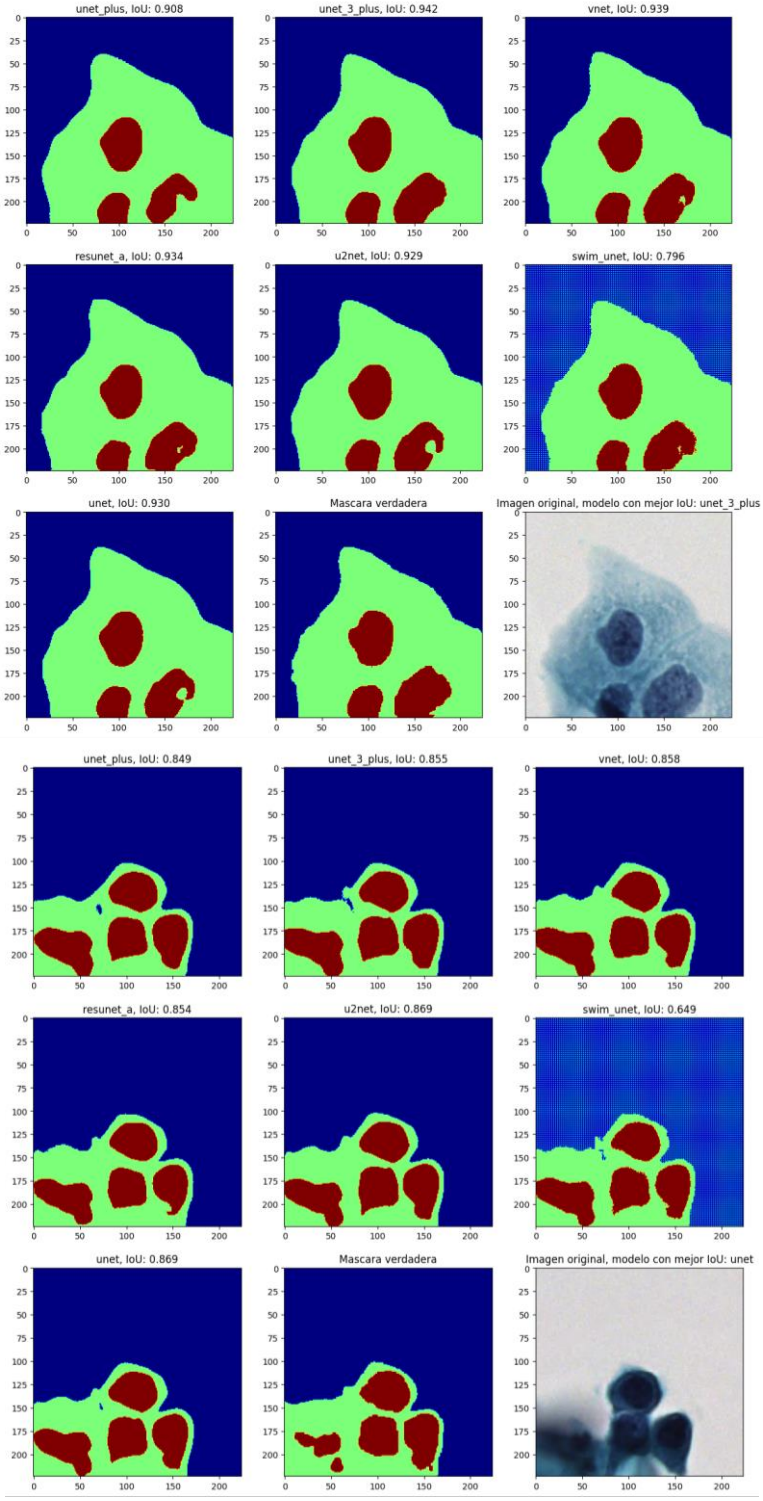


Ilustración 6.45: Resultados de segmentación. Células bajogrado (arriba) y altogrado (abajo) Fuente: Elaboración propia.

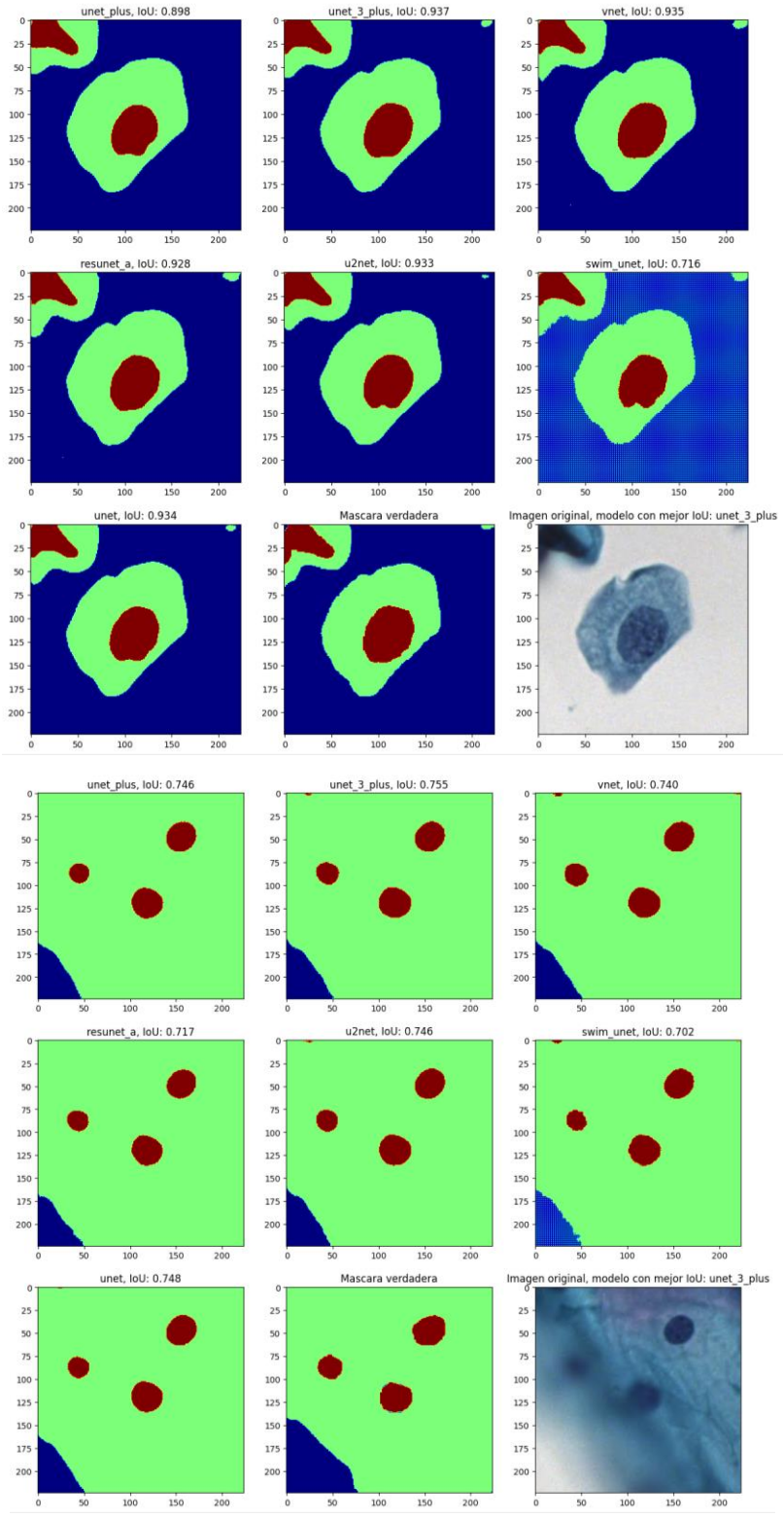


Ilustración 6.46: Resultados de segmentación. Células ascus (arriba) y benigna (abajo)
Fuente: Elaboración propia.

Al ver los resultados (Ilustraciones 43 - 46) podemos confirmar que los modelos de segmentación de la colección U-Net son completamente

comparables con el modelo original U-Net, tanto que en las pruebas UNET 3+, V-Net y ResUnet-a superan la media de U-Net, aunque no por mucho.

Es de tomar en consideración que al no haber trabajado por tanto tiempo con los nuevos modelos estos no están optimizados a su mayor potencial, así que es posible que puedan aportar mejores resultados que U-Net o que se haya alcanzado lo máximo posible de aprendizaje a partir de las imágenes de entrenamiento.

6.7.2. Retrospectiva del sprint

Durante este sprint logramos confirmar que el modelo actual de U-Net genera resultados de alta calidad comparables con otros modelos de segmentación más potentes y avanzados, aportando la ventaja de tener un gasto de recursos menor.

6.8. Sprint 8

El propósito de crear un modelo de segmentación de células es asistir a la clasificación de células, sea esta clasificación realizada manualmente por un experto o automáticamente a través de una red neuronal.

Durante este sprint se explorará el uso de las máscaras para la clasificación de dos maneras, clasificando solamente máscaras y su comparación con células y clasificando máscaras y células a la vez en un modelo.

Tabla 6.8: Organización de sprint 8. Fuente: Elaboración propia.

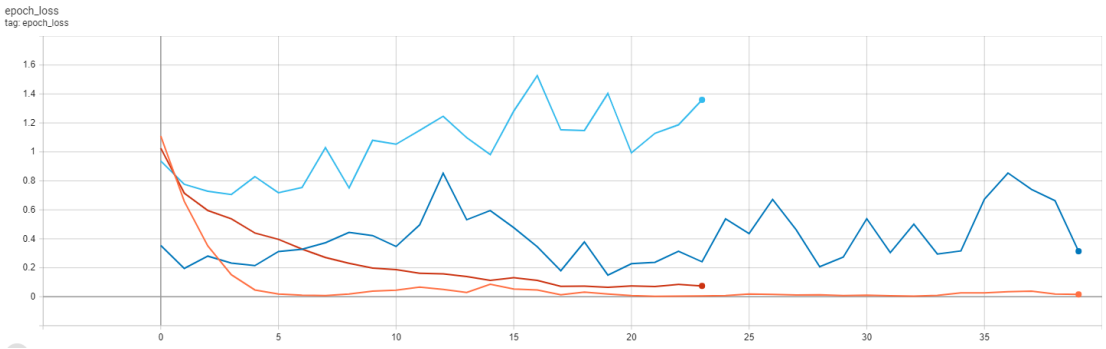
ID	Sprint	Historia de Usuario	Prioridad	Puntos de Historia
22	8	Clasificación de imágenes con máscaras de segmentación	1	5
23	8	Concatenación de modelos de clasificación para imágenes y máscaras	1	13

6.8.1. Clasificación de imágenes con máscaras de segmentación

Para la clasificación de imágenes, se utiliza la arquitectura *Xception* (Chollet, 2016) dado que está integrada en la librería de Keras. Del dataset completamente segmentado se ha utilizado solamente las máscaras en un entrenamiento y solamente las imágenes en otro.

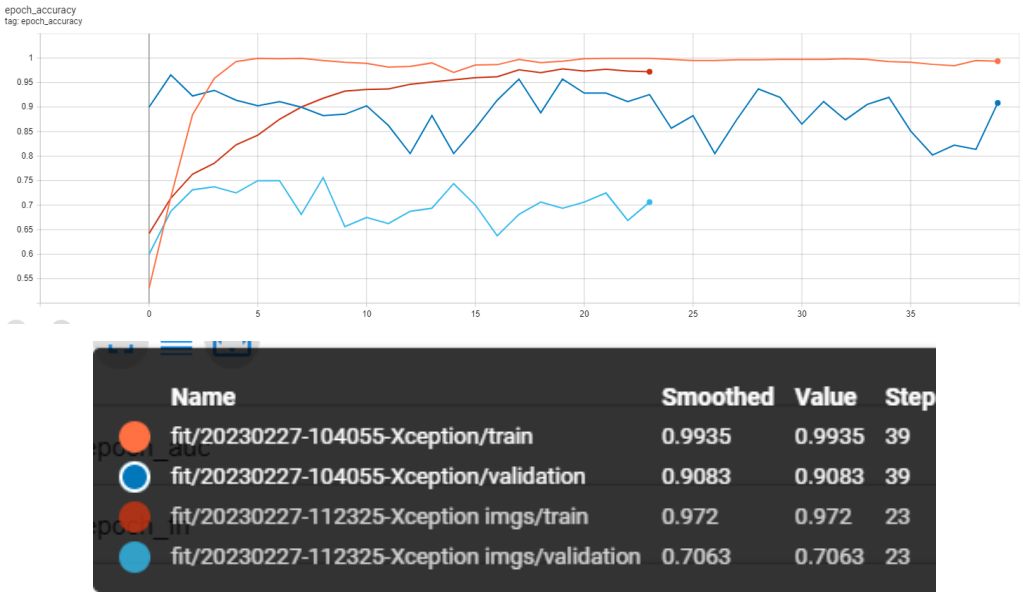
Para asegurar que los resultados son replicables, se utiliza un método que asigna la misma *seed* o semilla para los generadores de número aleatorios en todas las librerías utilizadas, la seed utilizada en todas las ejecuciones es 100.

En las Ilustraciones 6.47 y 6.48, en las gráficas de entrenamiento se llama "fit/20230227-104055-Xception" al modelo que entrena con máscaras solamente.



Name	Smoothed	Value	Step
fit/20230227-104055-Xception/train	0.01625	0.01625	39
fit/20230227-104055-Xception/validation	0.3143	0.3143	39
fit/20230227-112325-Xception imgs/train	0.07483	0.07483	23
fit/20230227-112325-Xception imgs/validation	1.359	1.359	23

Ilustración 6.47: Pérdida total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia.



Name	Smoothed	Value	Step
fit/20230227-104055-Xception/train	0.9935	0.9935	39
fit/20230227-104055-Xception/validation	0.9083	0.9083	39
fit/20230227-112325-Xception imgs/train	0.972	0.972	23
fit/20230227-112325-Xception imgs/validation	0.7063	0.7063	23

Ilustración 6.48: Tasa de aciertos total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo). Fuente: Elaboración propia.

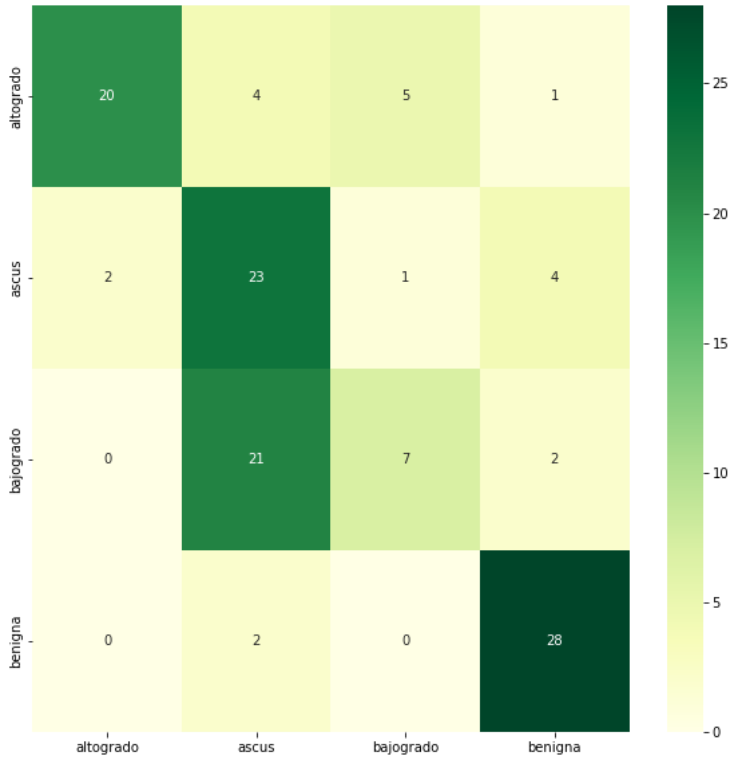


Ilustración 6.49: Matriz de confusión de entrenamiento con imágenes. Fuente: Elaboración propia.

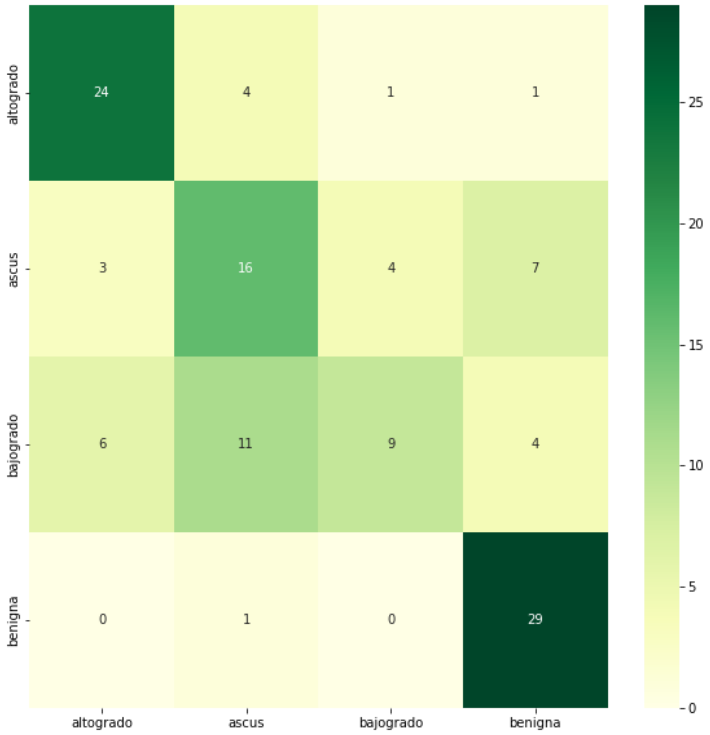


Ilustración 6.50: Matriz de confusión de entrenamiento con máscaras. Fuente: Elaboración propia.

En estos resultados (Ilustraciones 49 y 50) podemos ver que el modelo entrenado solamente con imágenes es capaz de reconocer fácilmente las células de clase altogrado, ascus y benigna, pero fallando con las bajogrado por su apariencia tan parecida.

En comparación, el modelo entrenado solamente con máscaras aumenta la precisión en las clases altogrado y benigna, pero en las clases intermedias ascus y bajogrado pierde casi completamente la capacidad de diferenciar. Al calcular la tasa de aciertos de cada uno de los modelos tenemos la misma cifra, un 65% de aciertos.

Como conclusión de esta prueba podemos ver que sí existe alguna ventaja en el uso de las máscaras para la clasificación de imágenes de células, pero al utilizar solamente la máscara se pierde información utilizada para identificar las clases intermedias.

6.8.2. Concatenación de modelos de clasificación para imágenes y máscaras

Para poder aprovechar la información de ambas imágenes se decide concatenar dos modelos de clasificación de imágenes, con la arquitectura

mostrada en la Ilustración 6.51. Como segundo modelo se elige *ResNet50* (He et al., 2015) dedicado a las máscaras y Xception dedicado a las imágenes.

La concatenación de estos modelos es sencilla dado que los métodos que aporta Keras para importar sus arquitecturas también permiten no incluir la cabecera de los modelos, es decir, su clasificador, así que es posible tomar la salida de ambos modelos y unificarlos antes de llegar al clasificador.

En esta prueba inicial se configuran como no entrenables o “congelan” las primeras 80 capas del modelo compuesto, dado que los modelos vienen con pesos iniciales de entrenamiento con el dataset “imagenet”, porque estas capas iniciales potencialmente tienen el conocimiento para distinguir formas simples. Las gráficas de entrenamiento están en las Ilustraciones 6.52 y 6.53.

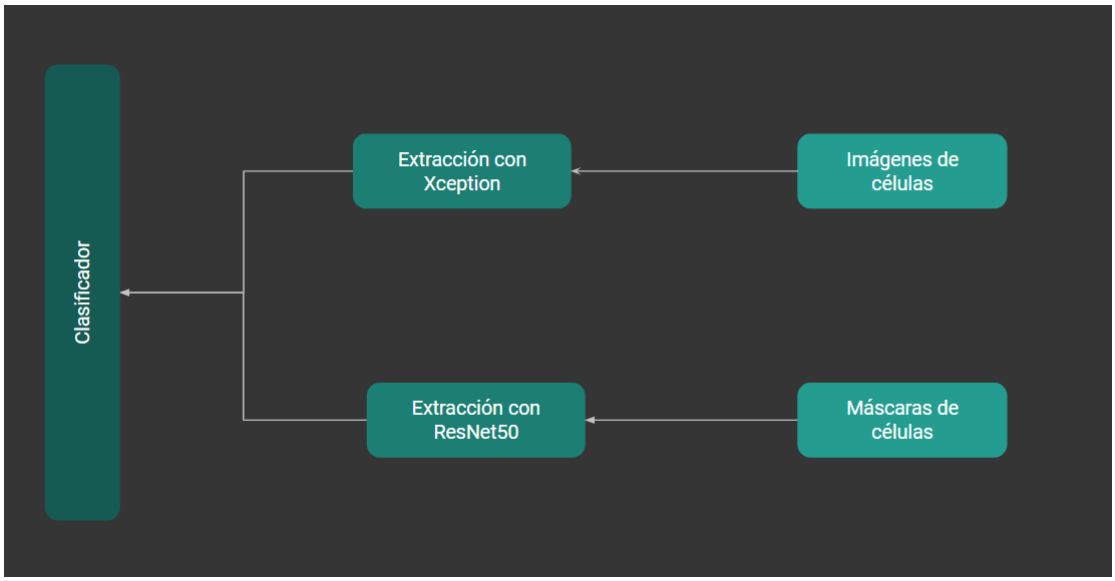


Ilustración 6.51: Modelo compuesto para clasificación. Fuente: Elaboración propia

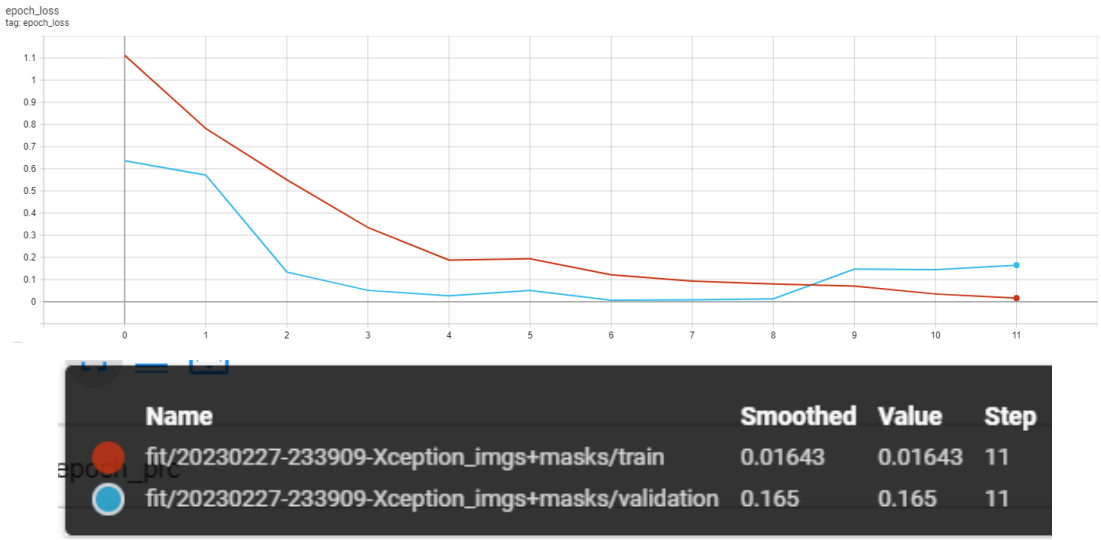


Ilustración 6.52: Pérdida total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia.

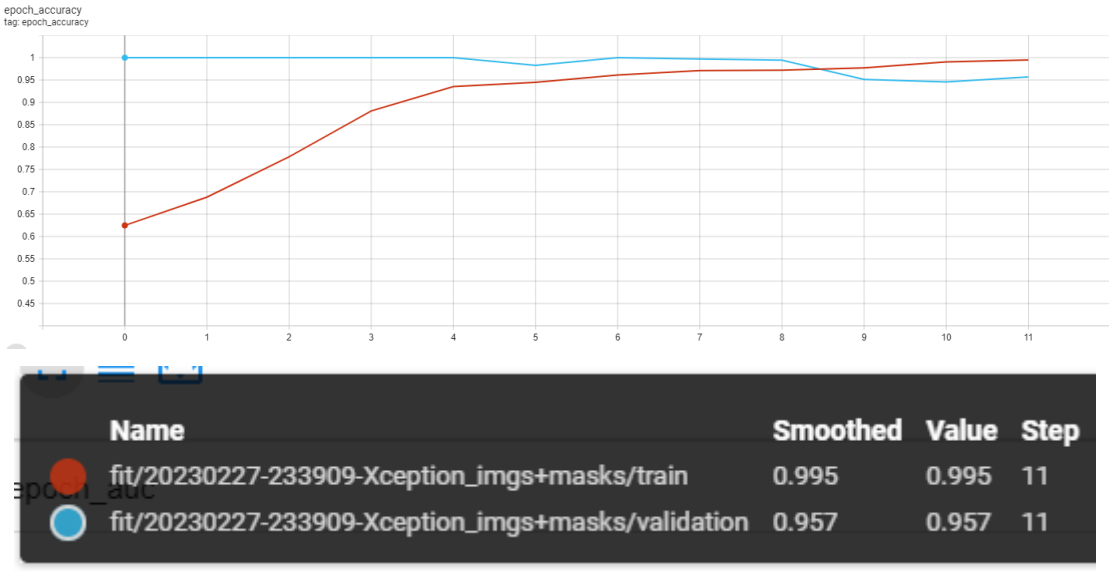


Ilustración 6.53: Tasa de aciertos total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia.

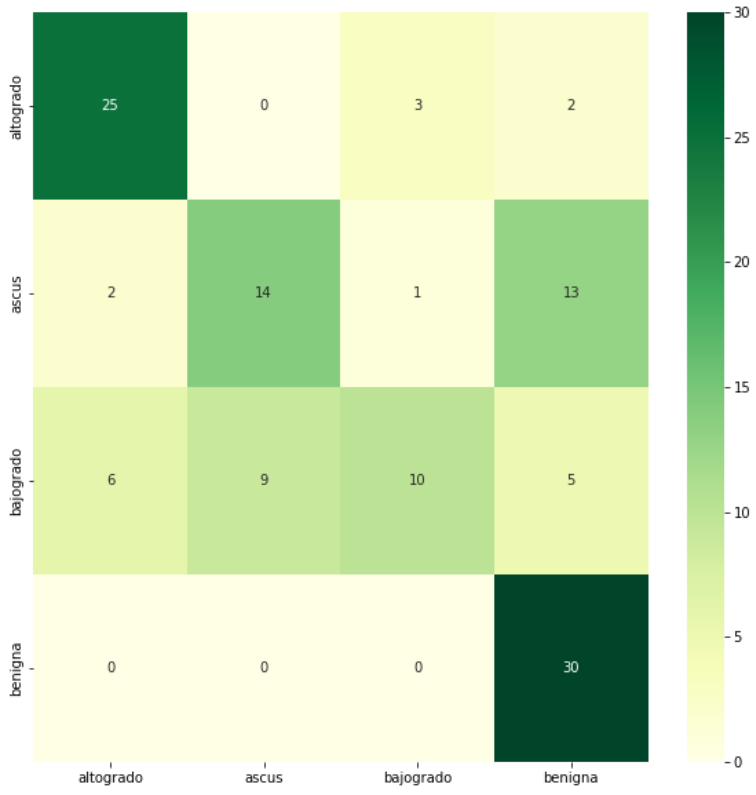


Ilustración 6.54: Matriz de confusión de entrenamiento con imágenes y máscaras. Fuente: Elaboración propia.

El modelo compuesto no presenta resultados (Ilustración 6.54) ideales aún pero sí logra el objetivo original de integrar las características de ambas imágenes, se puede apreciar en las clases mayoritarias de altogrado y benigna que el modelo tiene una mayor cantidad de aciertos. En las clases minoritarias de bajogrado y ascus aún podemos ver problemas de diferenciación con respecto a las demás clases.

A pesar de esto, al calcular la tasa de aciertos, se encuentra que sí mejora a los modelos originales de sólo imágenes o sólo máscaras, con una tasa de 65.83%.

6.8.3. Retrospectiva del sprint

Durante este sprint se ha iniciado la exploración del aporte que puede tener la segmentación de células en la detección de cáncer, aún hace falta optimizar el modelo para tener mejores resultados, pero los actuales demuestran el potencial de la idea.

6.9. Sprint 9

Durante el siguiente sprint se realizarán pruebas para encontrar los parámetros óptimos para generar mejores resultados, entre estas cambiar el número y orden de las capas congeladas y el uso de pesos de clases para complementar el desbalance de clases en el entrenamiento.

Tabla 6.9: Organización de sprint 9. Fuente: Elaboración propia.

ID	Sprint	Historia de Usuario	Prioridad	Puntos de Historia
24	9	Mejora del modelo de imágenes y máscaras	1	5

6.9.1. Mejora del modelo de imágenes y máscaras

La primera prueba (Ilustración 6.55) a realizar en este sprint es sin congelar ninguna capa del modelo, con la esperanza de que el conocimiento del entrenamiento previo con 'imagenet' sea útil como punto de partida, pero no fijo.

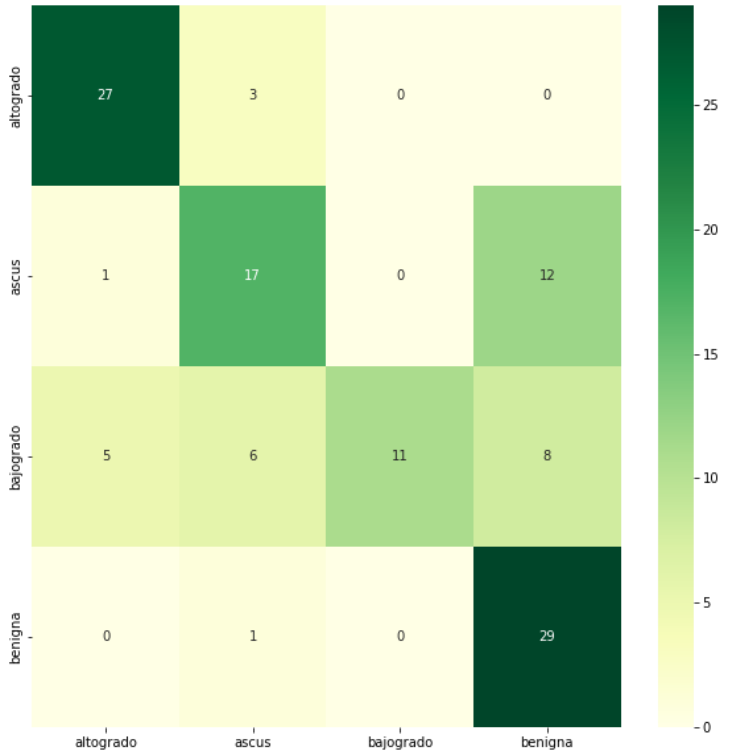


Ilustración 6.55: Matriz de confusión de entrenamiento con imágenes y máscaras. Fuente: Elaboración propia.

La siguiente prueba (Ilustración 6.56) realizada igualmente es con todas las capas sin congelar pero utilizando pesos de clases para tratar el desbalance de clases, los pesos asignados son proporcionales al número de veces que cabe el conjunto de imágenes de cada clase en el conjunto de imágenes de la clase mayoritaria.

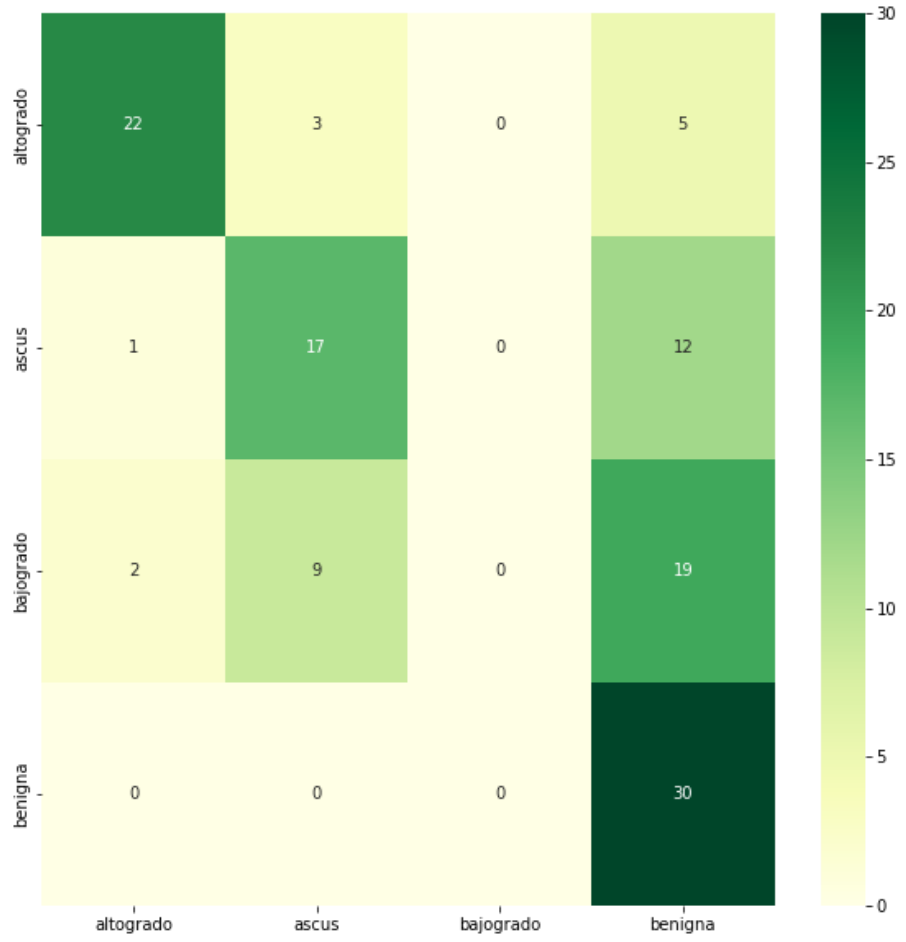


Ilustración 6.56: Matriz de confusión de entrenamiento con imágenes y máscaras. Fuente: Elaboración propia.

En la próxima prueba (Ilustración 6.57), se vuelve a congelar las capas iniciales, pero en este caso congelando las primeras 40 capas de cada modelo antes de concatenarlos, porque en la primera prueba estaba congelando más capas de un modelo que del otro, por la manera en la que se cargan.

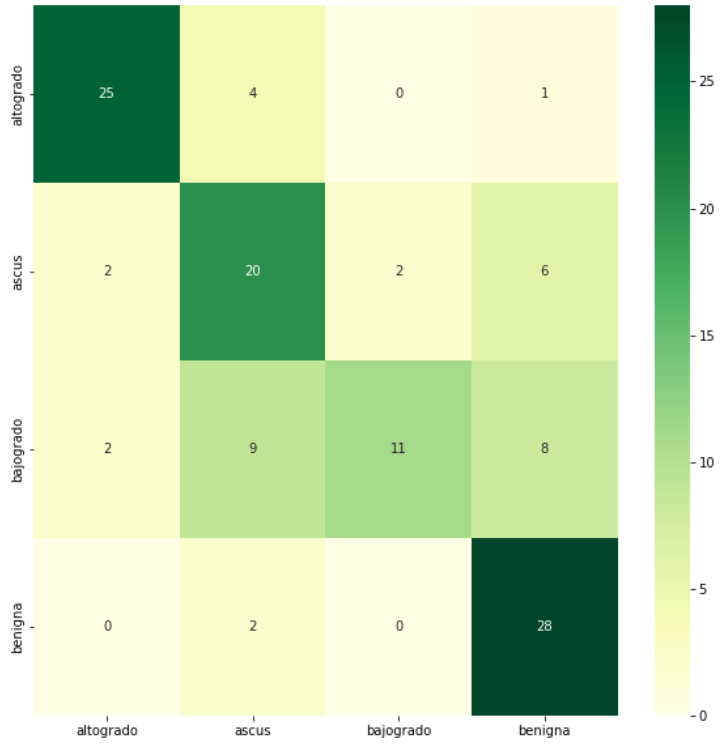


Ilustración 6.57: Matriz de confusión de entrenamiento con imágenes y máscaras. Fuente: Elaboración propia.

Y en la prueba final (Ilustración 6.58) de este sprint, se entrena a los modelos pero congelando completamente a los modelos base y solo dejando al clasificador aprender.

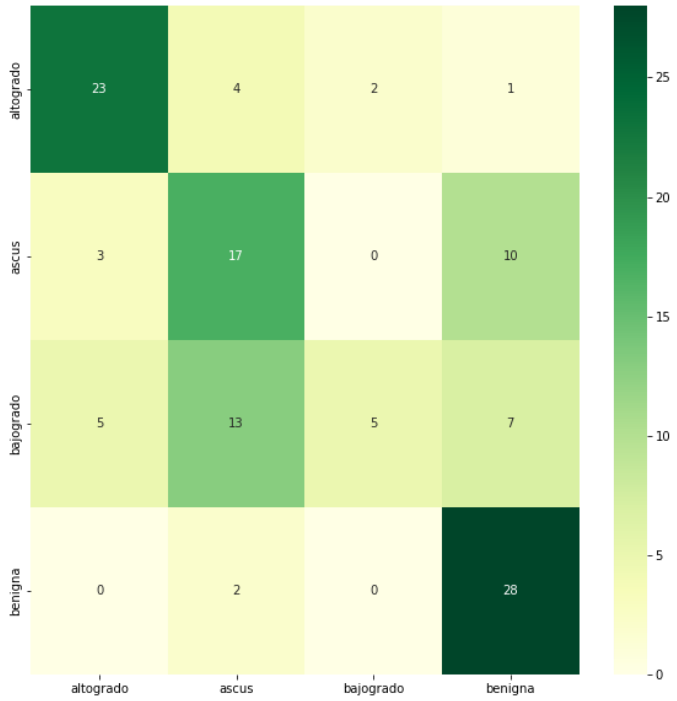


Ilustración 6.58: Matriz de confusión de entrenamiento con imágenes y máscaras. Fuente: Elaboración propia.

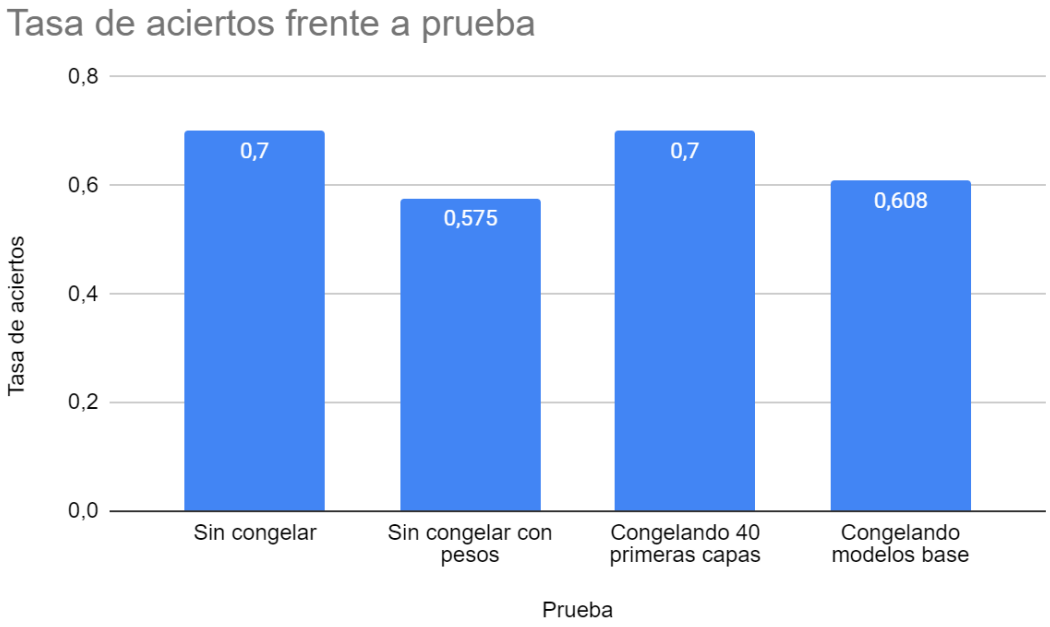


Ilustración 6.59: Tasas de acierto por prueba en sprint 9. Fuente: Elaboración propia

En la Ilustración 6.59 se puede observar una comparación de las tasas de acierto de las pruebas previas. Las pruebas de pesos de clases y congelando completamente los modelos base no logran aportar resultados adecuados, se nota que algo logran aprender pero no suficiente para ser modelos exitosos.

Por otro lado, los modelos sin congelar y congelando las primeras 40 capas de los modelos base, son los que mejores resultados aportan y al tener la misma tasa de acierto, es necesario inspeccionar las matrices de confusión para elegir la mejor.

Al apreciar las matrices de confusión, podemos ver que el modelo con las primeras 40 capas congeladas de los modelos base identifica menos células cancerígenas como benignas, además de tener mayor precisión con la clase ascus. Para futuras pruebas se continuará congelando las primeras 40 capas.

6.9.2. Retrospectiva del sprint

Durante este sprint se ha buscado optimizar los resultados del modelo compuesto, alcanzando una tasa de aciertos de 70%, un 5% mejor que el modelo solamente entrenado con imágenes.

Aunque la prueba de pesos de clase no logró aportar mejoras, el problema del desbalance de clases continúa afectando al aprendizaje, es necesario buscar alternativas viables.

6.10. Sprint 10

Para este sprint es necesario una solución al problema del desbalance de datos, específicamente se probará con el incremento de instancias de cada clase, hasta tener un balance de clases. Además de esto, también se probará con otro clasificador para ver si es posible mejorar.

Tabla 6.10: Organización de sprint 10. Fuente: Elaboración propia.

ID	Sprint	Historia de Usuario	Prioridad	Puntos de Historia
25	10	Balanceo de clases	1	5
26	10	Prueba de GAP	1	3
27	10	Concatenación de modelos de segmentación y clasificación para imágenes y máscaras	1	21

6.10.1. Balanceo de clases

Al momento de balancear instancias de clases lo que realmente se hace es copiar tantas imágenes como sean necesarias y añadirlas al dataset, no integra información nueva, para combatir esto, los métodos actuales realizan zoom, rotaciones de la imagen o vueltas verticales o horizontales.

El problema en este caso es que el modelo recibe 2 imágenes que están enlazadas realmente, si se rota la imagen y no la máscara, los resultados de cada modelo serán completamente diferentes.

Para solucionar este problema, es necesario aplicar la modificación a ambas imágenes, así que utilizando la librería *Albumentations* (Buslaev et al., 2020) es posible realizar las modificaciones simultáneamente, así que se utilizan 2 modificaciones, *randomCrop()* que recorta una parte de la imagen aleatoria y *horizontalFlip()* que voltea la imagen de manera horizontal.

Con estas modificaciones se genera un dataset balanceado de 3515 imágenes de células y 3515 máscaras segmentadas, junto a las 120 imágenes y máscaras de prueba.

Lastimosamente, al momento de cargar el dataset, el modelo compuesto a la vez e intentar entrenar, el entorno de ejecución se queda sin memoria RAM antes de terminar una sola época.

Se intentó utilizar la herramienta de Keras *ImageDataGenerator* que lee dinámicamente los datos desde un directorio, pero no se logró leer la imagen y la máscara correspondiente a la vez. A consecuencia de esto, se decide modificar el modelo.

6.10.2. Prueba de GAP

Antes de modificar el modelo, era necesario realizar una prueba más. En la siguiente prueba se compara el modelo compuesto de Xception y ResNet con el clasificador de neuronas densamente conectadas contra una capa de *GlobalAveragePooling* (Lin et al., 2013) como clasificador. GAP ofrece la ventaja a un clasificador densamente conectado de continuar trabajando con los mapas de características extraídos por los modelos base en 2D, en vez de tener que aplanarlos, esto aporta información espacial al modelo. Además de esto los modelos Xception y ResNet se cargan con los pesos de "imagenet" y se congelan las primeras 40 capas, ya que la prueba más exitosa en el sprint previo.

Las gráficas de entrenamiento están en las Ilustraciones 6.61 y 6.62.

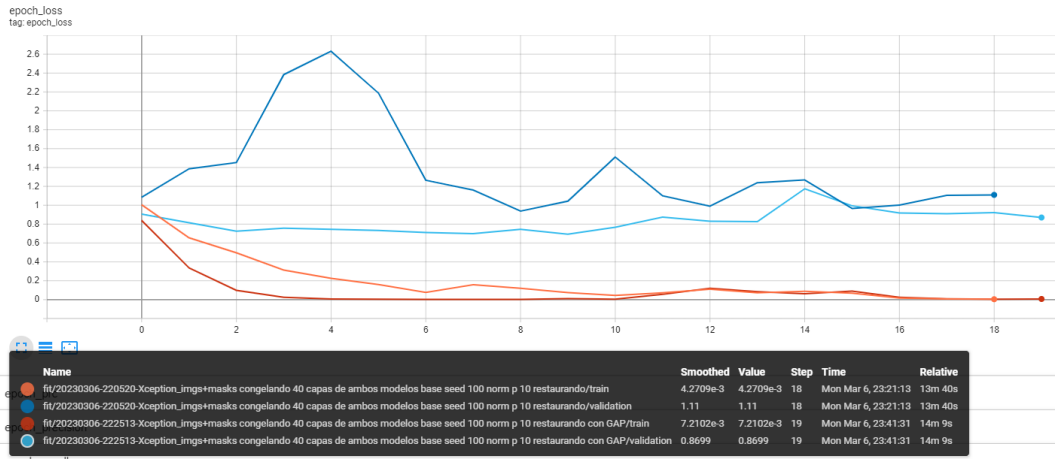


Ilustración 6.61: Pérdida total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia.

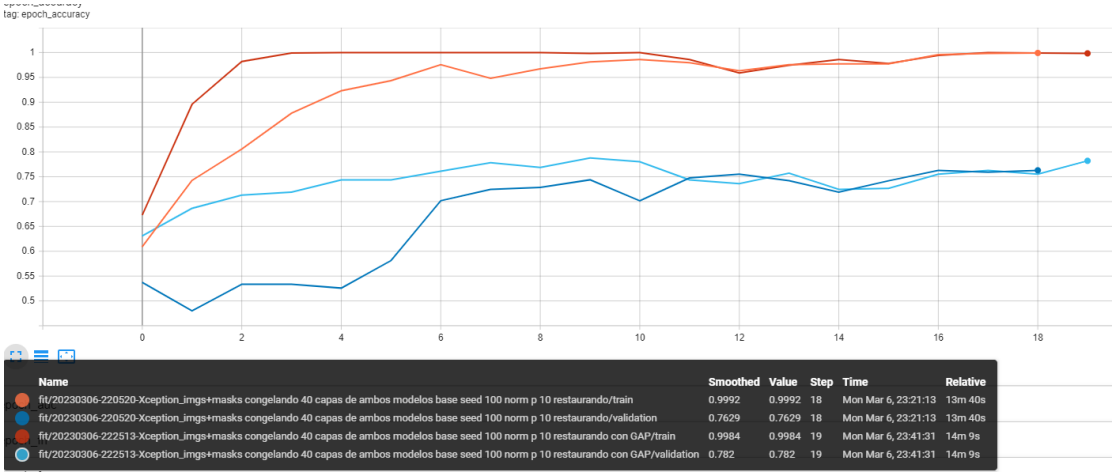


Ilustración 6.62: Tasa de aciertos total durante el entrenamiento por modelo (arriba). Valores finales del entrenamiento (abajo) Fuente: Elaboración propia.

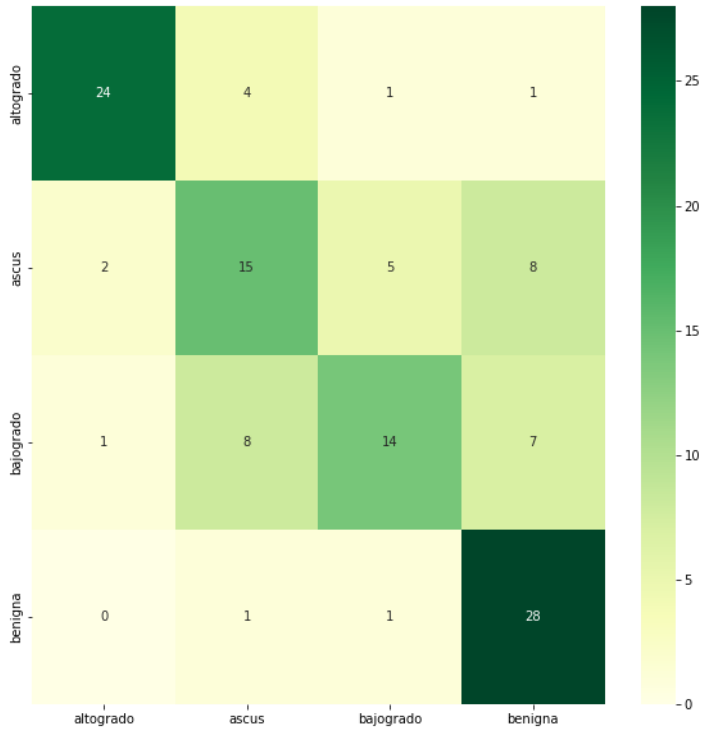


Ilustración 6.63: Matriz de confusión de entrenamiento con imágenes y máscaras, clasificador tradicional. Fuente: Elaboración propia.

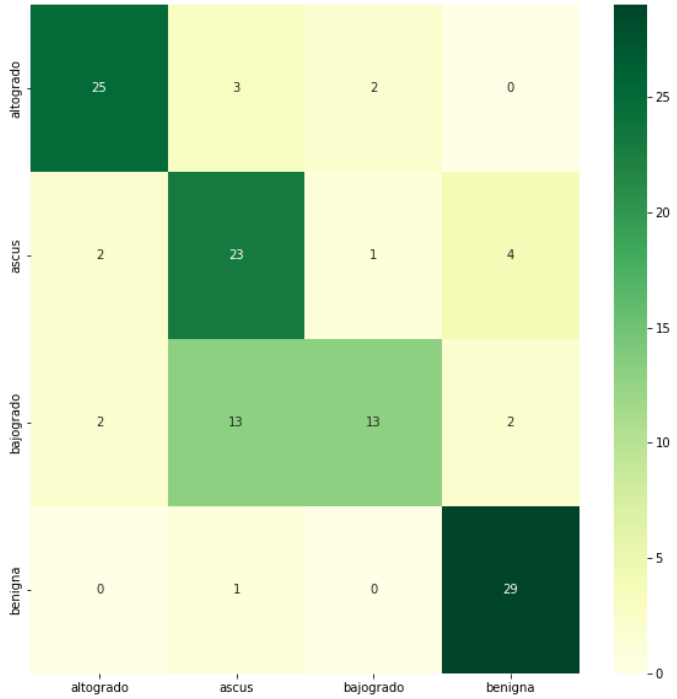


Ilustración 6.64: Matriz de confusión de entrenamiento con imágenes y máscaras, clasificador GAP. Fuente: Elaboración propia.

Además de las matrices de confusión para cuatro clases (Ilustraciones 6.63 y 6.64), es también relevante ver el comportamiento del modelo si lo interpretamos para dos clases, cancerígena o benigna (Ilustraciones 6.65 y 6.66).

		Truth data		
		Class 1	Class 2	Classification overall	User's accuracy (Precision)
Classifier results	Class 1	74	16	90	82.222%
	Class 2	2	28	30	93.333%
	Truth overall	76	44	120	
	Producer's accuracy (Recall)	97.368%	63.636%		
Overall accuracy (OA):		85%			

Ilustración 6.65: Matriz de confusión 2 clases de entrenamiento con imágenes y máscaras, clasificador tradicional. Fuente: Elaboración propia.

		Truth data			
		Class 1	Class 2	Classification overall	User's accuracy (Precision)
Classifier results	Class 1	84	6	90	93.333%
	Class 2	1	29	30	96.667%
	Truth overall	85	35	120	
	Producer's accuracy (Recall)	98.824%	82.857%		
Overall accuracy (OA):		94.167%			

Ilustración 6.66: Matriz de confusión 2 clases de entrenamiento con imágenes y máscaras, clasificador GAP. Fuente: Elaboración propia.

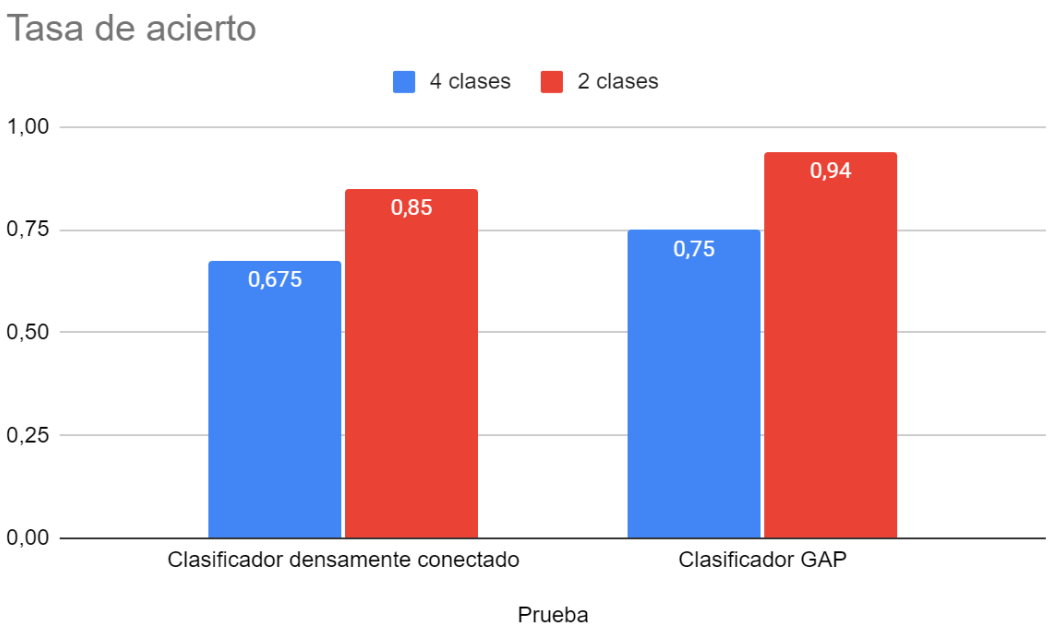


Ilustración 6.67: Tasas de acierto por prueba en sprint 10. Fuente: Elaboración propia

En la Ilustración 6.67, se puede apreciar que con el nuevo modelo compuesto utilizando el clasificador densamente conectado de antes no se logra alcanzar el 70% que se logró en las pruebas anteriores, pero el clasificador GAP

sí logra superar las pruebas previas, con una matriz de confusión muy limpia, a pesar del continuo problema de diferenciación de las clases minoritarias.

Igualmente, con las tasas de acierto a dos clases, es posible observar una puntuación casi perfecta al equivocarse en 7 de las 120 muestras.

6.10.3. Concatenación de modelos de segmentación y clasificación para imágenes y máscaras

Para evitar el problema de la memoria, se decide integrar el último modelo de U-Net al modelo complejo, así reduciendo la entrada a solo una imagen, como un modelo de clasificación de imágenes tradicional. La arquitectura sería como en la Ilustración 6.68.

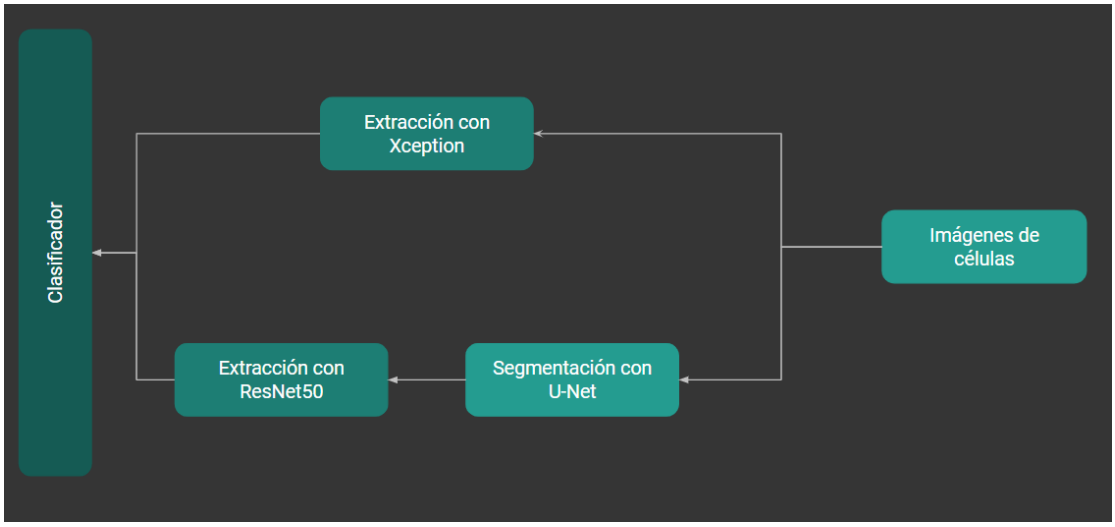


Ilustración 6.68: Modelo compuesto para segmentación y clasificación. Fuente: Elaboración propia

Este modelo recibe la imagen de la célula y la envía en dos direcciones, a la red Xception para extraer características y a la red U-Net para segmentar. Después de tener la máscara segmentada, esta es enviada a la red ResNet50 para la extracción de características, finalmente las características se concatenan y se pasan al clasificador.

Con esta estructura, es posible utilizar ImageDataGenerator para leer las 3515 imágenes de las células, así realizando un entrenamiento con un dataset balanceado.

Para las pruebas de este modelo se congelan los pesos de U-Net, así evitando que se modifique su capacidad de segmentación mientras se entrena el resto de la red. Las redes Xception y ResNet no están congeladas ni tienen pesos cargados, entrenan completamente junto al clasificador. En estas pruebas también se compara entre el clasificador densamente conectado y GAP.

En las ilustraciones 6.96 y 6.70 se pueden ver las matrices de confusión resultantes, la tasa de aciertos con el clasificador denso es de 70% y de clasificador GAP es de 71,7%.

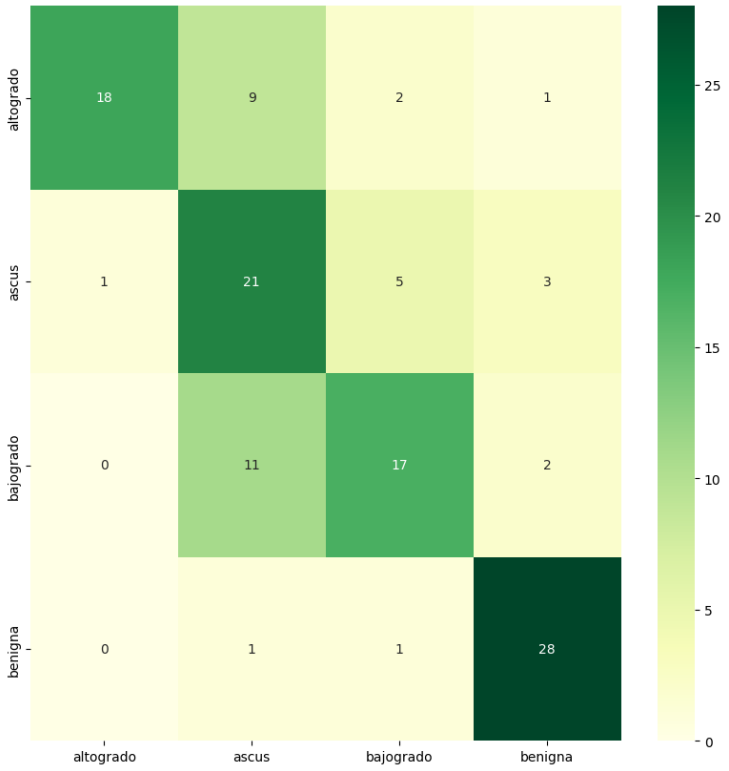


Ilustración 6.69: Matriz de confusión de entrenamiento con imágenes y máscaras, clasificador tradicional. Fuente: Elaboración propia.

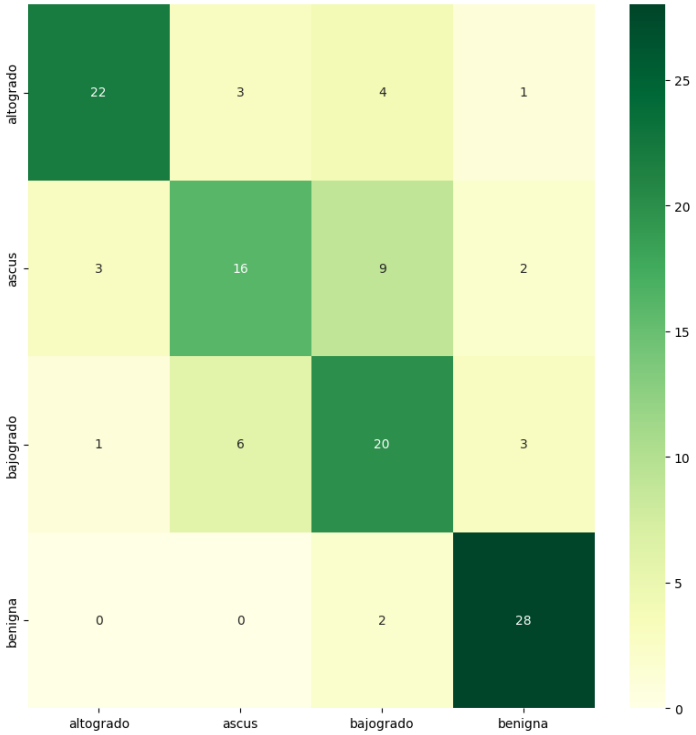


Ilustración 6.70: Matriz de confusión de entrenamiento con imágenes y máscaras, clasificador GAP. Fuente: Elaboración propia.

Dado que los resultados no han mejorado a comparación de la prueba previa, se ha decidido realizar una prueba más, pero con los parámetros que generaron el mejor resultado, los modelos de clasificación cargados con sus pesos de “imagenet” y sus primeras 40 capas congeladas. También fue generada la matriz de confusión de dos clases para ver su tasa de aciertos.

En las ilustraciones 6.71 y 6.72 se pueden apreciar las matrices de confusión, la tasa de aciertos para cuatro clases es de 72,5% y para dos clases es de 94,17%.

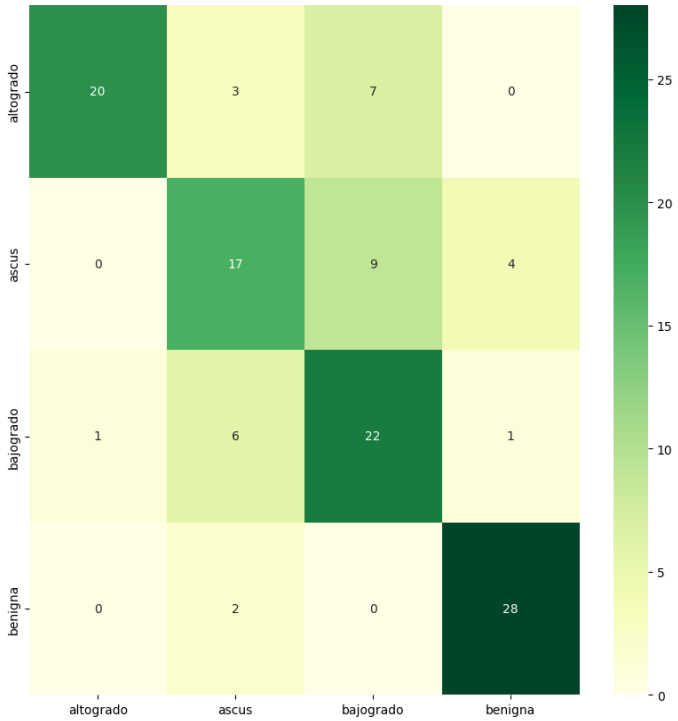


Ilustración 6.71: Matriz de confusión para cuatro clases de entrenamiento con imágenes y máscaras, clasificador GAP. Fuente: Elaboración propia.

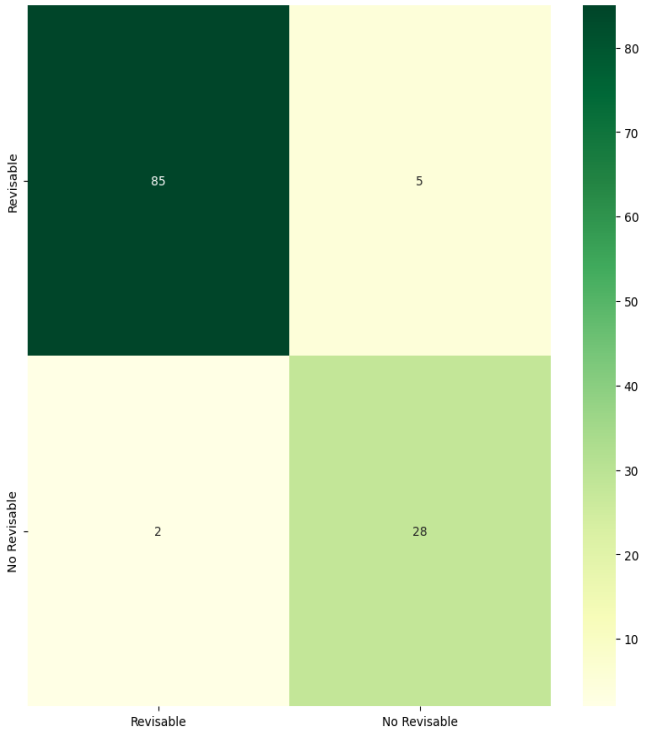


Ilustración 6.72: Matriz de confusión de dos clases de entrenamiento con imágenes y máscaras, clasificador GAP. Fuente: Elaboración propia.

A pesar de que la tasa de acierto de esta prueba es menor a la mejor tasa de aciertos previa en cuatro clases, es igual en dos clases. También se puede apreciar que, a pesar de tener menor tasa de aciertos, el modelo es más capaz de diferenciar entre las clases ascus y bajogrado, que son las más difíciles de diferenciar.

6.10.4. Retrospectiva del sprint

Durante este sprint se ha tenido que luchar contra varios problemas y encontrar soluciones más creativas, a través de las pruebas realizadas en este sprint se confirma que las máscaras de segmentación aportan a la clasificación de las células de manera positiva. Es necesario hacer una comparación más profunda de los modelos entrenados para encontrar sus ventajas y desventajas.

7. RESULTADOS

Dado que este estudio se enfoca en dos áreas diferentes de la inteligencia artificial, se presentan primero los resultados de los modelos de segmentación semántica y luego los resultados de los modelos de clasificación de imágenes.

7.1. Segmentación semántica

Para la comparación final de los métodos de segmentación semántica se compara el modelo U-Net que se ha ido mejorando iterativamente con los seis modelos utilizados de la "U-Net collection", específicamente U-Net++ (Zhou et al., 2018), ResUNet-a (Diakogiannis et al., 2020), U^2-Net (Qin et al., 2020), UNet 3+ (Huang et al., 2020), Swin-Unet (Cao et al., 2021) y V-Net (Milletari et al., 2016). Estos modelos son los mismos utilizados en la prueba del sprint 7.

Dado que en las pruebas del sprint 7 se muestra y comparan las máscaras generadas por cada modelo, en este apartado se presenta una comparación a través de métricas comúnmente utilizadas para segmentación semántica y clasificación. Específicamente:

- *mIoU*: *Mean Intersection over Union*, es la misma métrica que se ha utilizado hasta ahora para segmentación, específicamente la media de la división entre la intersección y la unión de la predicción y la máscara verdadera, de todas las clases.
- *Pixel Accuracy*: Tasa de acierto por píxel y es definida como el número de predicciones correctas divididas por el número total de predicciones.
- *Precision*: Precisión, definida como la división del número de instancias correctamente predichas en una clase entre la suma de todas las instancias predichas de esa misma clase.
- *Recall*: Traducible como cobertura, es definida como la división del número de instancias correctamente predichas en una clase entre la suma de todas las instancias verdaderas de esa clase.
- *F1 Score*: Métrica que integra la precisión y la cobertura en una, definida como la media armónica de la precisión y la cobertura.

Para el cálculo de la métrica se utiliza el dataset manualmente segmentado. Además del cálculo global de estas métricas en la Ilustración 7.1, también se ha calculado la precisión, la cobertura y la F1 score por cada clase en las Ilustraciones 7.2, 7.3 y 7.4, para profundizar en los resultados.

Model	mIoU	Pixel Accuracy	Precision	Recall	F1-Score
unet_plus	0.823049	0.926602	0.927242	0.926602	0.926155
unet_3_plus	0.830548	0.929577	0.929824	0.929577	0.929399
vnet	0.825152	0.9277	0.928085	0.9277	0.927294
resunet_a	0.820333	0.925832	0.926716	0.925832	0.92518
u2net	0.828164	0.928085	0.928847	0.928085	0.928107
swim_unet	0.705783	0.835462	0.859563	0.835462	0.833473
unet	0.827344	0.927978	0.928572	0.927978	0.927837

Ilustración 7.1: Métricas globales con el dataset manual. Fuente: Elaboración propia.

Model	fondo	citoplasma	nucleo
unet_plus	0.9771	0.9041	0.8543
unet_3_plus	0.9732	0.9154	0.844
vnet	0.9755	0.9079	0.8518
resunet_a	0.9785	0.8995	0.8635
u2net	0.9777	0.9125	0.8324
swim_unet	0.9735	0.7644	0.8596
unet	0.9768	0.9106	0.8409

Ilustración 7.2: Precisión por clase con el dataset de entrenamiento manual. Fuente: Elaboración propia.

Model	fondo	citoplasma	nucleo
unet_plus	0.9367	0.9568	0.7743
unet_3_plus	0.9445	0.9495	0.8017
vnet	0.9407	0.9544	0.7795
resunet_a	0.9347	0.9607	0.7587
u2net	0.9385	0.9497	0.8084
swim_unet	0.707	0.9595	0.7689
unet	0.9389	0.9519	0.7978

Ilustración 7.3: Cobertura por clase con el dataset de entrenamiento manual. Fuente: Elaboración propia.

Model	fondo	citoplasma	nucleo
unet_plus	0.9564	0.9297	0.8123
unet_3_plus	0.9586	0.9321	0.8223
vnet	0.9578	0.9306	0.814
resunet_a	0.9561	0.9291	0.8077
u2net	0.9577	0.9308	0.8202
swim_unet	0.8192	0.8509	0.8117
unet	0.9574	0.9308	0.8188

Ilustración 7.4: F-1 Score por clase con el dataset de entrenamiento manual. Fuente: Elaboración propia.

En la Ilustración 7.1 se puede apreciar que los mejores tres modelos son UNet 3+, U^2 Net y U-Net, con diferencias de 0.1% en mIoU y valores casi idénticos en la tasa de aciertos por píxel, la precisión, cobertura y F1 score. En términos globales, la red UNet 3+ genera mejores resultados, con U^2 Net y U-Net justo por detrás.

En las Ilustraciones 7.2, 7.3 y 7.4 se puede apreciar el mismo comportamiento, los modelos generan resultados muy cercanos, pero UNet 3+ genera mejores resultados en todas las clases al examinar la F1 score.

Pero en la clase núcleo, la precisión y la cobertura más altas no son por UNet 3+, sino que por ResUNet-a y U^2 Net, respectivamente.

Al inspeccionar estos resultados es importante considerar el tamaño y la calidad del dataset de entrenamiento utilizado, este fue realizado por U-Net en el sprint 5 y es considerado de menor calidad que el dataset manualmente segmentado utilizado para estas pruebas. Es posible que con un dataset más amplio y de mayor calidad se pueda ver mayor diferencia entre los modelos.

7.2. Clasificación de imágenes

Para la comparación de los modelos de clasificación de imágenes se han elegido cuatro modelos, específicamente los modelos entrenados que han generado mejores matrices de confusión.

Los modelos por utilizar son:

- X2: Modelo definido en sprint 8, concatenación de modelos Xception y ResNet para recibir imágenes y máscaras a la vez, utilizando un clasificador de neuronas densamente conectadas. Entrenamiento realizado con los pesos iniciales de “imagenet” con las primeras 40 capas congeladas.
- X2 GAP: Modelo definido en sprint 10, modificación al modelo X2 para utilizar GAP como clasificador. Entrenamiento realizado con los pesos iniciales de “imagenet” con las primeras 40 capas congeladas.
- U-Net X2: Modelo definido en sprint 10, Concatenación de U-Net al modelo X2 para generar máscaras segmentadas durante la predicción. Entrenamiento realizado sin pesos iniciales ni capas congeladas en los modelos de clasificación.
- U-Net X2 GAP: Modelo definido en sprint 10, modificación al modelo U-Net X2 para utilizar GAP como clasificador. Entrenamiento realizado con

los pesos iniciales de “imagenet” con las primeras 40 capas congeladas de los modelos de clasificación.

Para esta comparación, las métricas utilizadas son accuracy o tasa de aciertos, precisión, cobertura y F1 score, estas han sido calculadas de manera global y luego tasa de acierto y precisión por cada clase. También se ha calculado las métricas en dos clases, Revisable (altogrado, bajogrado y ascus) y No Revisable (benigna).

Model	Accuracy	Precision	Recall	F1-Score
U-Net X2 GAP	0.725	0.746739	0.725	0.726617
U-Net X2	0.7	0.737724	0.7	0.702802
X2 GAP	0.75	0.769535	0.75	0.740531
X2	0.675	0.681908	0.675	0.666281

Ilustración 7.5: Métricas globales para cuatro clases. Fuente: Elaboración propia.

Model	Accuracy (altogrado)	Accuracy (ascus)	Accuracy (bajogrado)	Accuracy (benigna)
U-Net X2 GAP	0.666667	0.566667	0.733333	0.933333
U-Net X2	0.6	0.7	0.566667	0.933333
X2 GAP	0.833333	0.766667	0.433333	0.966667
X2	0.8	0.5	0.466667	0.933333

Ilustración 7.6: Tasa de acierto por clase para cuatro clases. Fuente: Elaboración propia.

Model	Precision (altogrado)	Precision (ascus)	Precision (bajogrado)	Precision (benigna)
U-Net X2 GAP	0.952381	0.607143	0.578947	0.848485
U-Net X2	0.947368	0.5	0.68	0.823529
X2 GAP	0.862069	0.575	0.8125	0.828571
X2	0.888889	0.535714	0.666667	0.636364

Ilustración 7.7: Precisión por clase para cuatro clases. Fuente: Elaboración propia.

En la Ilustración 7.5 se puede apreciar, como previamente visto, que el modelo X2 GAP genera los mejores resultados en las cuatro métricas, seguido cercanamente por el modelo U-Net X2 GAP.

La Ilustración 7.6 muestra la principal diferencia entre los modelos con mejores resultados, el modelo X2 GAP es acierta más células altogrado y células benignas, pero el modelo U-Net X2 GAP tiene mayor porcentaje de acierto en las células de anormalidades intermedias de ascus y bajogrado.

En la Ilustración 7.7 se puede observar que el modelo X2 GAP tiene resultados muy altos en todas las clases excepto en ascus, la clase con menor precisión a través de todos los modelos.

Model	Accuracy	Precision	Recall	F1-Score
U-Net X2 GAP	0.941667	0.94488	0.941667	0.942561
U-Net X2	0.933333	0.93844	0.933333	0.934659
X2 GAP	0.941667	0.948319	0.941667	0.943077
X2	0.85	0.889354	0.85	0.857864

Ilustración 7.8: Métricas globales para dos clases . Fuente: Elaboración propia.

Model	Accuracy (Revisable)	Accuracy (No Revisable)
U-Net X2 GAP	0.944444	0.933333
U-Net X2	0.933333	0.933333
X2 GAP	0.933333	0.966667
X2	0.822222	0.933333

Ilustración 7.9: Tasa de acierto por clase para dos clases . Fuente: Elaboración propia.

Model	Precision (Revisable)	Precision (No Revisable)
U-Net X2 GAP	0.977011	0.848485
U-Net X2	0.976744	0.823529
X2 GAP	0.988235	0.828571
X2	0.973684	0.636364

Ilustración 7.10: Precisión por clase para dos clases . Fuente: Elaboración propia.

En la Ilustración 7.8, los modelos U-Net X2 GAP y X2 GAP son casi idénticos al inspeccionarlos por dos clases, con X2 GAP continuando por encima en las métricas de precisión y F1 score. Y en las Ilustraciones 7.9 y 7.10 el modelo X2 GAP continúa mostrando mejores resultados, solamente quedándose atrás del modelo U-Net X2 GAP en la tasa de aciertos para la clase Revisable.

8. CONCLUSIONES

8.1. Objetivos alcanzados

El objetivo general de este estudio, la clasificación de imágenes médicas mediante máscaras de segmentación para células procedentes de citologías cervicovaginales, ha sido completado con éxito.

Para empezar a abordar el objetivo general, la clasificación de imágenes de células mediante máscaras de segmentación, se desglosó el proyecto en los diferentes objetivos específicos, con dos de estos siendo claves, el OE1, el entrenamiento de una red neuronal de segmentación semántica, y el OE5, el entrenamiento de una red neuronal de clasificación de imágenes y máscaras.

Para alcanzar el OE1 era necesario el OE2, la creación de un dataset de máscaras segmentadas por núcleo y citoplasma correspondiente al dataset de imágenes original. Este objetivo ha sido completado, de manera iterativa y gradual se han generado múltiples datasets de máscaras de segmentación correspondientes a las imágenes correspondientes a las imágenes del dataset original, con diferentes grados de cantidad y calidad hasta alcanzar un resultado aceptable y balanceado.

Con el OE2 alcanzado, se logra realizar el OE2, se ha entrenado principalmente el modelo U-Net hasta alcanzar resultados aceptables en las máscaras de segmentación para las tres clases, núcleo, citoplasma y fondo.

El OE3, el estudio de la relación entre la precisión del modelo de segmentación semántica y la calidad y cantidad del dataset de entrenamiento, fue completado, se ha estudiado el comportamiento del modelo de segmentación variando datasets de diferentes grados de calidad y tamaños, identificando la combinación óptima para el entrenamiento.

Para realizar el OE4, la comparación de la precisión y el gasto de recursos de diferentes modelos de segmentación semántica, se ha comparado los resultados de diferentes modelos y arquitecturas tomando en cuenta los tiempos de entrenamiento y ejecución y contrastándolo con sus resultados.

Finalmente, con los objetivos específicos previos cumplidos fue posible realizar el OE5, se han entrenado múltiples modelos y desarrollado modelos compuestos de clasificación entrenados en imágenes y máscaras de células.

El OE6, el estudio de los hiperparámetros necesarios para optimizar los modelos de inteligencia artificial, también fue completado durante el transcurso de todo el estudio para encontrar la combinación óptima en resultados.

8.2. Conclusiones del proyecto

Una de las principales dificultades del proyecto, la creación del dataset de máscaras, y su realización lleva la conclusión de que es posible crear y mejorar un dataset para segmentación utilizando una combinación de herramientas y predicciones de modelos de segmentación partiendo de un dataset de imágenes.

A partir de las diferentes pruebas realizadas, se puede concluir que el principal factor que afecta a la calidad de los resultados al entrenar modelos de segmentación y clasificación de imágenes es el tamaño del dataset a utilizar. A través de todas las pruebas comparando calidad, especificidad de clases y combinaciones de dataset en el conjunto de validación, el modelo entrenado sobre una mayor cantidad de datos tiende a aportar los mejores resultados.

Otra conclusión de este proyecto es que es posible entrenar modelos de segmentación semántica para la clasificación de núcleo y citoplasma en imágenes de células médicas de diferentes grados de anormalidades, generando resultados exitosos aún en células de alto grado de anormalidad. Esto presenta una potencial aplicación en el ámbito médico para evidenciar a mayor grado las irregularidades de la forma del núcleo.

Las máscaras de segmentación aplicadas para la clasificación de imágenes de células cervicovaginales permiten a los modelos mejorar en su precisión, principalmente en las clases de anormalidades intermedias como ascus y bajogrado. Esto sugiere que la información extraída por las redes completamente convolucionales es relevante para distinguir entre las diferentes categorías de células, y que la combinación de segmentación y clasificación puede aumentar la confiabilidad del diagnóstico.

Para continuar mejorando los resultados en segmentación, además de ampliar y balancear el conjunto inicial de imágenes, sería necesaria la realización del etiquetado de núcleos y citoplasmas por un experto en el campo hasta alcanzar un conjunto balanceado de datos menor al inicial, para asegurar la mayor calidad inicial posible, y aplicar las técnicas y herramientas utilizadas en este proyecto para finalizar el segmentado del resto del dataset. De esta manera se podría asegurar un sistema de una calidad aceptable para un entorno real.

Para mejorar los resultados de clasificación, una ruta posible sería la creación de modelos compuestos utilizando otros modelos de clasificación y segmentación diferentes a los explorados en este estudio, dado que el alcance de los modelos de clasificación explorados en este estudio fue corto.

Las técnicas de inteligencia artificial utilizadas en este proyecto son capaces de asistir a patólogos profesionales en el diagnóstico y detección de cáncer cervical, a pesar de no generar resultados perfectos. Esto implica que la visión artificial puede ser una herramienta útil para mejorar la precisión y la eficiencia de los exámenes citológicos, así como para reducir el tiempo y el costo de estos.

9. REFLEXIÓN Y VALORACIÓN PERSONAL

Las dificultades presentadas durante el estudio han sido muy amplias, ya que en el campo de segmentación de imágenes yo no tenía ningún tipo de conocimiento previo, solo un poco en clasificación de imágenes, así que fue necesario leer y probar mucho hasta poder entender y generar algo que pareciera funcional.

Otra dificultad muy amplia fue el crear un dataset, por suerte no empezaba desde cero, pero el trabajo de manualmente etiquetar las imágenes de la mejor manera posible fue muy difícil y lento. Las células de grados mayores de anormalidades son las más complejas por la deformación del núcleo y mi falta de conocimientos en el área de la patología.

Posteriormente el desarrollo de los modelos compuestos también fue un reto grande, porque al trabajar con arquitecturas ya definidas es difícil modificarlas sin causar que pierda capacidad. A pesar de ello, me resultó profundamente satisfactorio el ver cómo mejoraba el modelo al integrar las máscaras en la clasificación.

Para aprovechar completamente las herramientas desarrolladas sería necesario realizar otro entrenamiento de los modelos con más datos, más balanceados y con máscaras validadas por profesionales, así podrían alcanzar su mejor versión, igualmente considero que los resultados alcanzados son completamente satisfactorios.

El desarrollo de este trabajo me ha abierto al mundo de la inteligencia artificial y el aprendizaje profundo en específico, un campo gigantesco y complejo pero que tiene la capacidad de aportar mucho a la sociedad. He disfrutado mucho el proceso, a pesar de sus dificultades, y, si tuviera más tiempo, alegremente continuaría iterando los modelos para mejorar resultados. Por esto estoy orgulloso de presentar múltiples meses de investigación, estudio y trabajo.

10. REFERENCIAS

Bhattiprolu, S. (2020, Enero 31). *Defining U-net in Python using Keras.py at*. GitHub. Retrieved Febrero 3, 2023, from https://github.com/bnsreenu/python_for_microscopists/blob/master/074-Defining%20U-net%20in%20Python%20using%20Keras.py

Iakubovskii, P. (2019). *Segmentation Models*. GitHub. Retrieved Febrero 2, 2023, from https://github.com/qubvel/segmentation_models

Long, J., Shelhamer, E., & Darrell, T. (2016, May 20). *[1605.06211] Fully Convolutional Networks for Semantic Segmentation*. arXiv. Retrieved Diciembre 22, 2022, from https://arxiv.org/abs/1605.06211

Ronneberger, O., Fischer, P., & Brox, T. (2015, Mayo 18). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. Computer Vision Group, Freiburg. Retrieved Diciembre 22, 2022, from https://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a/

Sharma, P. (2019, Julio 22). *Image Segmentation Python | Implementation of Mask R-CNN*. Analytics Vidhya. Retrieved Noviembre 28, 2022, from https://www.analyticsvidhya.com/blog/2019/07/computer-vision-implementing-mask-r-cnn-image-segmentation/

Sharma, P. (2019, Abril 1). *Image Segmentation | Types Of Image Segmentation*. Analytics Vidhya. Retrieved Noviembre 27, 2022, from https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/

Buslaev, A., Iglovikov, V., Khvedchenya, E., Parinov, A., Druzhinin, M., & Kalinin, A. A. (2020, Febrero 24). Albumentations: Fast and Flexible Image Augmentations. Information, 11(2), article number: 125. 10.3390/info11020125

Caicedo, J. C., Roth, J., Goodman, A., Becker, T., Karhohs, K. W., Broisin, M., Molnar, C., McQuin, C., Singh, S., Theis, F. J., & Carpenter, A. E. (2019). Evaluation of Deep Learning Strategies for Nucleus Segmentation in Fluorescence Images. Cytometry. Part A : the journal of the International Society for Analytical Cytology, 95(9), 952-965. 10.1002/cyto.a.23863

Cao, H., Wang, Y., Chen, J., Jiang, D., Zhang, X., Tian, Q., & Wang, M. (2021, May 12). Swin-Unet: Unet-like Pure Transformer for Medical Image Segmentation. ECCV Workshops. https://arxiv.org/abs/2105.05537

Chollet, F. (2016, Octubre 7). [1610.02357] Xception: Deep Learning with Depthwise Separable Convolutions. arXiv. Retrieved Febrero 28, 2023, from https://arxiv.org/abs/1610.02357

Cohn, M. (2004). User Stories Applied. Addison-Wesley. https://books.google.es/books/about/User_Stories_Applied.html

Diakogiannis, F. I., Waldner, F., Caccetta, P., & Wu, C. (2020, Abril). ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data. ISPRS Journal of Photogrammetry and Remote Sensing, 162, 94-114. 10.1016/j.isprsjprs.2020.01.013

Google. (n.d.). Google Colab. Google Research. https://research.google.com/colaboratory/faq.html

Han, S.-H., Kim, K. W., Kim, S., & Youn, Y. C. (2018). Artificial Neural Network: Understanding the Basic Concepts without Mathematics. Dement Neurocogn Disord, 17(3), 83–89. 10.12779/dnd.2018.17.3.83

He, K., Zhang, X., Ren, S., & Sun, J. (2015, Diciembre 10). [1512.03385] Deep Residual Learning for Image Recognition. arXiv. Retrieved Abril 22, 2023, from https://arxiv.org/abs/1512.03385

Huang, H., Lin, L., Tong, R., Hu, H., Zhang, Q., Iwamoto, Y., Han, X., Chen, Y.-W., & Wu, J. (2020, April 19). UNet 3+: A Full-Scale Connected UNet for Medical Image Segmentation. 2020 IEEE International Conference on Acoustics, Speech and Signal Processing. https://arxiv.org/abs/2004.08790

IBM. (n.d.). ¿Qué es Deep Learning? IBM. Retrieved Marzo 24, 2023, from https://www.ibm.com/es-es/topics/deep-learning

IBM. (n.d.). ¿Qué es el aprendizaje supervisado? IBM. Retrieved Marzo 24, 2023, from https://www.ibm.com/es-es/topics/supervised-learning

IBM. (n.d.). ¿Qué es la inteligencia artificial (IA)? IBM. Retrieved Marzo 24, 2023, from https://www.ibm.com/es-es/topics/artificial-intelligence

Janiesch, C., Zschench, P., & Heinrich, K. (2021). Machine learning and deep learning. Electronic Markets, 31(3), 685-695. https://arxiv.org/abs/2104.05314

Keras. (2021). About Keras. Keras. https://keras.io/about/

Lin, M., Chen, Q., & Yan, S. (2013, Diciembre 16). Network In Network. Papers With Code. Retrieved Abril 10, 2023, from https://paperswithcode.com/paper/network-in-network

MathWorks. (n.d.). Segmentación semántica - MATLAB & Simulink. MathWorks. Retrieved Marzo 28, 2023, from https://es.mathworks.com/solutions/image-video-processing/semantic-segmentation.html

Milletari, F., Navab, N., & Ahmadi, S.-A. (2016, June 15). V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. 2016 Fourth International Conference on 3D Vision. https://arxiv.org/abs/1606.04797

Navarro Cadavid, A., Fernández Martínez, J. D., & Morales Vélez, J. (2013, julio - diciembre). Revisión de metodologías ágiles para el desarrollo de software. PROSPECTIVA, 11(2), 30-39. https://www.redalyc.org/pdf/4962/496250736004.pdf

Nayar, R., & Wilbur, D. C. (2015, Mayo 1). The Pap test and Bethesda 2014. Cancer Cytopathol, 123(5), 271–281. 10.1002/cncy.21521

Orr, G., Müller, K.-R., & Montavon, G. (Eds.). (2012). Neural Networks: Tricks of the Trade. Springer Berlin Heidelberg. 10.1007/978-3-642-35289-8

O'Shea, K., & Nash, R. (2015, Noviembre 26). An Introduction to Convolutional Neural Networks. arXiv. https://arxiv.org/abs/1511.08458

Project Jupyter. (n.d.). Jupyter Project Documentation. Project Jupyter | Home. https://docs.jupyter.org/en/latest/

Python. (2022, Noviembre 22). Descripción general. Python Wiki. https://wiki.python.org/moin/BeginnersGuide/Overview

Qin, X., Zhang, Z., Huang, C., Dehghan, M., Zaiane, O. R., & Jagersand, M. (2020, May 18). U^2-Net: Going Deeper with Nested U-Structure for Salient Object Detection. Pattern Recognition. https://arxiv.org/abs/2005.09007

Rashid, T. A. (2003, Septiembre). A simple recurrent neural network with real time recurrent learning process. 14th Irish conference on Artificial Intelligence and Cognitive Science AICS 2003, 1(1). 10.13140/2.1.5092.7680

Schreiber, G. (2000). Knowledge Engineering and Management: The CommonKADS Methodology. MIT Press. https://nixis.de/~nikku/uni/ws201011/knowledge-engineering/CommonKADS_methodology.pdf

Sha, Y. (2021, Septiembre 4). yingkaisha/keras-unet-collection: v0.1.12. Zenodo. Retrieved Abril 22, 2023, from https://zenodo.org/record/5449801

Talent.com. (n.d.). Salario para Ingeniero De Software en España - Salario Medio. Talent.com. https://es.talent.com/salary?job=ingeniero+de+software

Taye, M. M. (2023). Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions. Computation, 11(3), no. 3. 10.3390/computation11030052

TensorFlow. (2021). TensorFlow Core | Aprendizaje automático para principiantes y expertos. TensorFlow. https://www.tensorflow.org/overview?hl=es-419

Velásquez Restrepo, S. M., Vahos-Montoya, J. D., Gómez-Adasme, M. E., Pino – Martínez, A. A., Restrepo-Zapata, E. J., & Londoño-Marí, S. (2019, julio - diciembre). Una revisión comparativa de la literatura acerca de metodologías tradicionales y modernas de desarrollo de software. Revista CINTEX, 24(2), 13-23. 10.33131/24222208.334

Zhou, Z., Rahman Siddiquee, M. M., Tajbakhsh, N., & Liang, J. (2018, Septiembre 20). UNet++: A Nested U-Net Architecture for Medical Image Segmentation. Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support. Lecture Notes in Computer Science, 11045, 3-11. 10.1007/978-3-030-00889-5_1

11. ANEXOS

11.1. Anexo 1. Explicación de ficheros anexos

En la carpeta comprimida de “Anexos TFG” se encuentran dos carpetas más, una para cada parte de este estudio, “Segmentación” y “Clasificación”. Dentro de estas carpetas se encuentran los ficheros .ipynb, formato para notebooks o cuadernos de python, ejecutables en Jupyter Notebook o importables en Google Colab. Se sugiere importar a Google Colab para aprovechar la GPU gratuita.

Los cuadernos encontrados en cada carpeta son los entornos donde se han realizado las diferentes pruebas de este estudio y una comparación final con los mejores modelos de cada problemática.

Además de los cuadernos, cada carpeta contiene una carpeta de pesos, con los pesos correspondientes a los mejores modelos y entrenamientos.

Dada la naturaleza privada de las imágenes médicas utilizadas para entrenamiento y prueba, el dataset utilizado no viene incluido en los anexos. De la misma manera, las rutas utilizadas en los cuadernos son las utilizadas durante el estudio, no son funcionales a menos que se recree la misma estructura de datos.

En el caso de los modelos de segmentación, sí es posible realizar una prueba puntual para comprobar su comportamiento, esto se puede hacer en el apartado “Segmentar una imagen” del cuaderno “Comparación final segmentación U-Net contra u-net collection”, al subir una imagen de una célula a la carpeta “/content/sample_data”. Por otro lado, los modelos de clasificación no se pueden probar sin recrear la estructura del dataset original con más imágenes.

Para demostrar el comportamiento de los modelos al momento de realizar predicciones, se han realizado dos vídeos, uno para cada problemática:

- Segmentación: https://youtu.be/P1MFTsDXo-0
- Clasificación: https://youtu.be/XzIshx4nSU4

Detalle de los ficheros anexos:

Cuadernos:

- Segmentacion_no_supervisada_K_means
    - Prueba con K-Means para generar imágenes segmentadas automáticamente. Utilizado durante sprint 1.
- Prueba_con_FasterRCNN
    - Prueba entrenando Faster RCNN con un dataset segmentado manualmente. Utilizado durante sprint 1.
- U_Net_segmentacion
    - Primeras pruebas de entrenar U-Net con los diferentes datasets. Utilizado durante sprint 3 y 4.
- U_Net_segmentacion_por_clase
    - Prueba de entrenar U-Net con imágenes de cada clase. Utilizado durante sprint 4.
- U_Net_segmentacion_con_diferentes_backbones
    - Prueba de comparar U-Net con diferentes arquitecturas. Utilizado durante sprint 5.
- U_Net_prueba_con_4_clases_fulldataset
    - Prueba de entrenar U-Net con imágenes de cada clase del dataset completo segmentado. Utilizado durante sprint 6.
- U_Net_prueba_con_4_clases_fulldataset_vs_diferentes_validaciones
    - Prueba de entrenar U-Net mezclando los diferentes datasets para validación. Utilizado durante sprint 6.
- U_Net_prueba_contra_u_net_collection
    - Prueba de entrenar múltiples modelos de segmentación con el dataset completo. Utilizado durante sprint 7.
- Comparación_final_segmentación_U_Net_contra_u_net_collection
    - Comparación final de los mejores modelos de segmentación. Contiene implementación para segmentar una imagen al final.
- Clasificación_4_clases_con_máscaras
    - Prueba de clasificar utilizando solamente máscaras o imágenes. Utilizado durante sprint 8.
- Clasificación_4_clases_con_máscaras_+_imgs

    - Prueba de concatenar modelos de clasificación entrenando con imágenes y máscaras. Utilizado durante sprint 8, 9 y 10.
- Clasificación_+_Segmentacion_4_clases_con_máscaras_+_imgs
    - Prueba de concatenar modelos de segmentación y clasificación entrenando con imágenes. Utilizado durante sprint 10.
- Comparación_final_clasificación_máscaras_+_imgs
    - Comparación final de los mejores modelos de clasificación.

Pesos/Modelos:

- altogrado_simple_unet_200epochs_all_imgs
    - Pesos para U-Net, entrenado con imágenes de clase altogrado.
- ascus_simple_unet_200epochs_all_imgs
    - Pesos para U-Net, entrenado con imágenes de clase ascus.
- bajogrado_simple_unet_200epochs_all_imgs
    - Pesos para U-Net, entrenado con imágenes de clase bajogrado.
- benigna_simple_unet_200epochs_all_imgs
    - Pesos para U-Net, entrenado con imágenes de clase benigna.
- ValApeer_simple_unet_500epochs_all_imgs
    - Pesos para U-Net, entrenado con todas las imágenes, validando con dataset generado con Apeer.
- ValManual_simple_unet_500epochs_all_imgs
    - Pesos para U-Net, entrenado con todas las imágenes, validando con dataset creado manualmente.
- ValSplit20pct_simple_unet_500epochs_all_imgs
    - Pesos para U-Net, entrenado con todas las imágenes, validando con 20% del dataset de entrenamiento.
- resunet_a_all_imgs
    - Pesos para ResUNet-a.
- swim_unet_all_imgs
    - Pesos para Swin-Unet .
- u2net_all_imgs
    - Pesos para U^2-Net.
- unet_3_plus_all_imgs

    - Pesos para UNet 3+.
- unet_plus_all_imgs
    - Pesos para U-Net++.
- vnet_all_imgs
    - Pesos para V-Net.
- Unet+Xception+ResNet fully connected classifier dataset augmentado20230522-205259-comparacion
    - Pesos para mejor modelo U-Net X2
- Unet+Xception+ResNet GAP classifier dataset augmentado 40 capas base congeladas paciencia 1520230524-123749-comparacion
    - Pesos para mejor modelo U-Net X2 GAP
- Xception_imgs+masks congelando 40 capas de ambos modelos base seed 100 norm p 10 restaurando con GAP20230306-224147-_4clases_masks_imgs
    - Pesos para mejor modelo X2 GAP
- Xception_imgs+masks congelando 40 capas de ambos modelos base seed 100 norm p 10 restaurando20230306-222129-_4clases_masks_imgs
    - Pesos para mejor modelo X2

11.2. Anexo 2. Importar ficheros .ipynb a Google Colab

Para importar ficheros .ipynb a Google Colab es necesario tener una cuenta de Google y acceder con esta a la página https://colab.research.google.com/.

Ilustración 11.1: Captura de Google Colab pantalla inicial . Fuente: Elaboración propia.

Al acceder, aparece un menú con opciones para elegir la fuente del fichero a abrir, es necesario seleccionar la pestaña “Upload” y subir ahí el fichero deseado, como aparece en la Ilustración 11.1.

Dado que los cuadernos de este TFG han sido creados en Google Colab se debería de importar automáticamente la configuración del entorno de ejecución con GPU, pero por si acaso es necesario verificarlo.

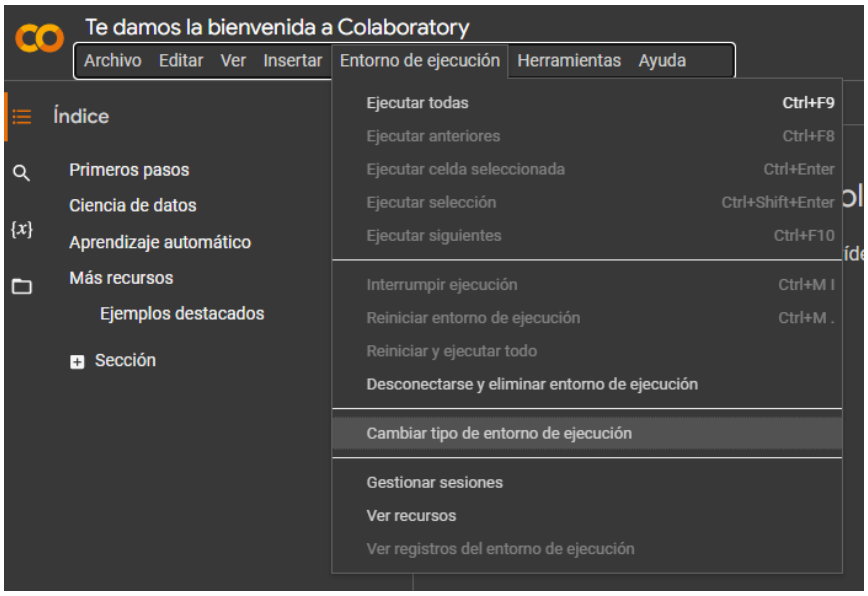


Ilustración 11.2: Captura de Google Colab cambio de entorno de ejecución . Fuente: Elaboración propia.

La opción para cambiar el entorno de ejecución está en el menú de "Entorno de ejecución", como aparece en la Ilustración 11.2.

En el menú de "Configuración del cuaderno" es necesario confirmar los valores de "Python 3" para tipo de entorno y "GPU" para el acelerador por hardware, como aparece en la Ilustración 11.3. Con esto ya es posible ejecutar el código del cuaderno.

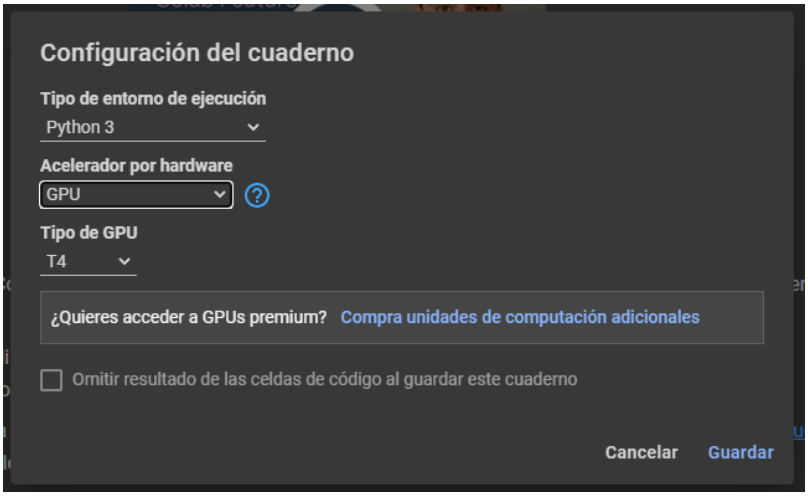


Ilustración 11.3: Captura de Google Colab cambio de entorno de ejecución . Fuente: Elaboración propia.

Segmentación de imágenes de células procedentes de citologías cervicovaginales