Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

SEGMENTACIÓN AUTOMÁTICA DE IMÁGENES DE RESONANCIA MAGNÉTICA CEREBRAL MEDIANTE REDES NEURONALES CONVOLUCIONALES

Autor:

Ayuso Lera, Adrián

Tutor:

de la Fuente López, Eusebio
Ingeniería en Sistemas y Automática

Valladolid, mayo 2022.

Resumen

Actualmente, el uso de redes neuronales convolucionales para la segmentación de imágenes se ha incrementado en diversas áreas, debido a que en muchos casos el uso de esta tecnología proporciona mejores resultados que los métodos tradicionales de procesamiento de imágenes.

En el sector médico, la segmentación automática de imágenes puede resultar de gran ayuda a la hora de realizar diagnósticos, disminuir riesgos en cirugías al proporcionar información relevante sobre la zona a operar, y también puede resultar de gran utilidad en futuros sistemas de cirugía autónomos.

En este Trabajo de Fin de Grado se expone la creación de una aplicación de segmentación automática de imágenes de resonancia magnética que sirva como apoyo a los cirujanos a la hora de realizar un proceso de cirugía endoscópica endonasal transesfenoidal, esta aplicación permitirá identificar automáticamente algunas de las estructuras anatómicas a evitar durante la operación para prevenir lesiones en el paciente.

Palabras clave

Imágenes de resonancia magnética

Cirugía endoscópica endonasal transesfenoidal

Redes neuronales convolucionales

Segmentación de imágenes médicas

Visión artificial

Abstract

Nowadays, the use of convolutional neural networks for image segmentation has increased in various areas, because in many cases, the use of this technology provides better results than traditional image processing methods.

In the medical sector, automatic image segmentation could be of great help when making diagnoses, reducing risks in surgeries by providing relevant information about the area to be operated on, and it can also be very useful in future autonomous surgery systems.

In this Final Degree Project, we expose the creation of an application for automatic segmentation of magnetic resonance images that can serve as support to surgeons when performing a process of endonasal transsphenoidal endoscopic surgery, this application will automatically identify some of the anatomical structures to avoid during the operation to avoid injury to the patient.

Keywords

Magnetic Resonance Imaging (MRI)

Endoscopic endonasal transsphenoidal surgery

Convolutional Neural Networks (CNN)

Medical image segmentation

Computer Vision

Índices

Índice

Resumen 2
Palabras clave 2
Abstract 3
Keywords 3
Índices 4
Índice 4
Índice de figuras 7
1- Introducción y objetivos 11
1.1- Cirugía endoscópica endonasal transesfenoidal 11
1.2- Segmentación de imágenes médicas 13
1.3- Objetivos del TFG 14
2- Segmentación de imágenes médicas con redes neuronales 15
2.1- Estado del arte: segmentación automática de imágenes médicas mediante redes neuronales convolucionales 15
2.1.1- Segmentación automática de imágenes de tomografía axial computarizada para procesos de cirugía endoscópica endonasal transesfenoidal. 15
2.1.2- Segmentación automática de tumores cerebrales en imágenes de resonancia magnética. 16
2.2- Imágenes médicas 17
2.2.1- Tomografía Axial Computarizada (TAC) 17
2.2.2- Imágenes de resonancia magnética (IRM). 19
2.3- Redes neuronales artificiales 21
2.3.1- Definición de red neuronal artificial 21
2.3.2- Funcionamiento de la neurona artificial 22
2.3.3- Estructura de la red neuronal artificial 24
2.3.4- Aprendizaje en redes neuronales artificiales 26
2.3.5- Descenso del gradiente y el algoritmo de “retropropagación de errores” o “backpropagation” 27
2.3.6- Evaluación del aprendizaje 30
2.4- Redes neuronales convolucionales. 32
2.4.1- Análisis de imágenes mediante redes neuronales artificiales. 32

2.4.2- La convolución.....33
2.4.3- Componentes de una red neuronal convolucional.....37
2.4.4 Arquitecturas de redes neuronales convolucionales. 41
2.4.5- Redes neuronales convolucionales de segmentación.....45
3- Desarrollo del proyecto 48
3.1- Recursos utilizados.....48
3.1.1- Dataset OASIS-1 Brains. 48
3.1.2- Entorno de programación Google colaboratory. 50
3.1.3- 3DSlicer 51
3.2- Proceso de etiquetado de imágenes con 3DSlicer. 51
3.2.1- Cargar la secuencia de imágenes IRM de un paciente.....52
3.2.2- Crear las etiquetas.....53
3.2.3- Creación del dataset. Estructura de directorios. 56
3.3- Aplicación de entrenamiento de modelos. 58
3.3.1- Instalación e importación de librerías. 59
3.3.2- Proceso de carga del dataset etiquetado. 61
3.3.3- Definición de la red y preprocesamiento del dataset.....68
3.3.4- Entrenamiento de la red.....70
3.3.5- Visualización de inferencia con datos de validación.....73
3.4- Aplicación de inferencia de modelos. 74
3.4.1- Instalación e importación de librerías. 74
3.4.2- Descarga del modelo entrenado.....76
3.4.3- Descarga de nuevas imágenes del dataset OASIS.....77
3.4.4- Inferencia del modelo.....83
3.4.5- Visualización de resultados.....84
4- Resultados obtenidos.....85
4.1- Entrenamiento de modelos.....85
4.2- Inferencia de los modelos entrenados.....93
4.3- Entrenamiento del modelo de segmentación exclusiva de las arterias carótidas 107
4.4- Inferencia del modelo de segmentación exclusiva de las arterias carótidas. 111
4.5- Resumen de resultados. 119
5- Conclusiones y líneas futuras 121

6- Bibliografía 123
7- Anexos 125
7.1- Script Matlab para visualizar etiquetas de nervios ópticos y arterias carótidas en tres dimensiones. 125
7.2- Script Matlab para visualizar etiquetas de arterias carótidas en tres dimensiones. 125
7.3- Enlace a aplicación de entrenamiento de modelos. 125
7.4- Enlace a aplicación de inferencia de modelos 125

Índice de figuras

Figura 1: Localización de la silla turca. 11
Figura 2. Anatomía de la zona cercana a la hipófisis. 12
Figura 3. Escala de unidades Hounsfield. 18
Figura 4. Diferencias entre imágenes de resonancia magnética ponderadas en T1 y en T2. 20
Figura 5. Tabla de coloración de los distintos tejidos y elementos en secuencias T1 y T2 20
Figura 6. Representación de la neurona artificial 22
Figura 7. Resultado de clasificador binario con una única neurona. 23
Figura 8. Varios tipos de funciones de activación existentes. 24
Figura 9. Representación de red neuronal multicapa. 24
Figura 10. Resultado de clasificación binaria para datos de entrada más complejos. 25
Figura 11. Representación del gradiente en un espacio tridimensional. 27
Figura 12. Representación del método de descenso de gradiente. 28
Figura 13. Ejemplo de curvas de error en modelo con sobreentrenamiento. 31
Figura 14. Ejemplo de generación de conceptos abstractos a partir de conceptos simples. 32
Figura 15. Ejemplo de flipping de un kernel. 33
Figura 16. Ejemplo de una convolución 2D sin flipping del kernel. 34
Figura 17. Ejemplo de interacciones dispersas de una red neuronal convolucional. 35
Figura 18. Ejemplo de estructura con cinco mapas de características en la capa oculta. 36
Figura 19. Ejemplo de la aplicación del "zero padding" para mantener la dimensión de la salida. 37
Figura 20. Ejemplo de una convolución usando un "stride" de 2 y un "padding" de 0. 38
Figura 21. Ejemplo de agrupamiento basado en "max pooling". 39
Figura 22. Representación de la arquitectura de la red neuronal convolucional VGG-16. 41
Figura 23. Residual ResNet. 42
Figura 24. Representación por bloques de la arquitectura ResNet-50. 43
Figura 25. Bloque de convolución de Resnet-50 44
Figura 26. Bloque de identidad de ResNet-50. 44
Figura 27. Ejemplo de Fully Convolutional Network para segmentación de imágenes. 45
Figura 28. Representación de la arquitectura U-Net. 46
Figura 29. Representación de la arquitectura 3D U-Net. 47
Figura 30. Estructura de carpetas para cada paciente del dataset OASIS-1. 49
Figura 31. Vista general del programa con una secuencia de imágenes cargada. 52
Figura 32. Pestaña de creación de etiquetas. 53
Figura 33. Utilización de la herramienta "threshold". 54
Figura 34. Etiquetado de la arteria carótida derecha 55
Figura 35. Apartado "Export to files" de la pestaña "Segmentations" 55
Figura 36. Estructura de directorios del dataset. 56
Figura 37. Archivos de secuencia de imágenes y etiquetas que se encuentran dentro de la carpeta de uno de los pacientes del dataset. 57
Figura 38. Carpeta dataset generada en el espacio de almacenamiento del computador de Google. 62
Figura 39. Interfaz gráfica del apartado "cargar imágenes y etiquetas del dataset" de la aplicación de entrenamiento de modelos. 63
Figura 40. Interfaz gráfica del apartado "Visualizar imágenes y etiquetas del dataset" de la aplicación de entrenamiento de modelos 66
Figura 41. Ejemplo de visualización de resultados de inferencia. 84
Figura 42. Selección de dataset con nervios ópticos en la interfaz gráfica de la aplicación de entrenamiento de modelos. 85

FIGURA 43. AJUSTE DE PARÁMETROS EN LA INTERFAZ GRÁFICA DEL APARTADO "CARGAR IMÁGENES Y ETIQUETAS DEL DATASET" DE LA APLICACIÓN DE ENTRENAMIENTO DE MODELOS PARA EL ENTRENAMIENTO CON EL DATASET CON NERVIOS ÓPTICOS. 85
FIGURA 44. AJUSTE DE PARÁMETROS EN LA INTERFAZ GRÁFICA DEL APARTADO " DEFINICIÓN DE LOS PARÁMETROS DEL MODELO Y PREPROCESADO DEL DATASET" EN LA APLICACIÓN DE ENTRENAMIENTO DE MODELOS. MODELO CON ETAPA DECODIFICADORA RESNET-50. 86
FIGURA 45. AJUSTE DE PARÁMETROS EN LA INTERFAZ GRÁFICA DEL APARTADO " ENTRENAR MODELO" EN LA APLICACIÓN DE ENTRENAMIENTO DE MODELOS. 86
FIGURA 46. REPRESENTACIÓN GRÁFICA DEL VALOR DEL ERROR DE ENTRENAMIENTO Y EL ERROR DE VALIDACIÓN POR ÉPOCAS. MODELO CON ETAPA DECODIFICADORA RESNET-50 87
FIGURA 47. REPRESENTACIÓN GRÁFICA DE LA MEDIDA DE "IOU" DE ENTRENAMIENTO Y VALIDACIÓN POR ÉPOCAS. MODELO CON ETAPA DECODIFICADORA RESNET-50. 87
FIGURA 48. RESULTADO DE VALIDACIÓN PARA LA PRIMERA SECUENCIA DE IMÁGENES DEL DATASET DE VALIDACIÓN. MODELO CON ETAPA DECODIFICADORA RESNET-50. 88
FIGURA 49. RESULTADO DE VALIDACIÓN PARA LA SEGUNDA SECUENCIA DE IMÁGENES DEL DATASET DE VALIDACIÓN. MODELO CON ETAPA DECODIFICADORA RESNET-50. 88
FIGURA 50. RESULTADO DE VALIDACIÓN PARA LA SEGUNDA SECUENCIA DE IMÁGENES DEL DATASET DE VALIDACIÓN. MODELO CON ETAPA DECODIFICADORA RESNET-50. 88
FIGURA 51. AJUSTE DE PARÁMETROS EN LA INTERFAZ GRÁFICA DEL APARTADO "DEFINICIÓN DE LOS PARÁMETROS DEL MODELO Y PREPROCESADO DEL DATASET" EN LA APLICACIÓN DE ENTRENAMIENTO DE MODELOS. MODELO CON ETAPA DECODIFICADORA VGG-16 89
FIGURA 52. REPRESENTACIÓN GRÁFICA DEL VALOR DEL ERROR DE ENTRENAMIENTO Y EL ERROR DE VALIDACIÓN POR ÉPOCAS. MODELO CON ETAPA DECODIFICADORA VGG-16. 90
FIGURA 53. REPRESENTACIÓN GRÁFICA DE LA MEDIDA DE "IOU" DE ENTRENAMIENTO Y VALIDACIÓN POR ÉPOCAS. MODELO CON ETAPA DECODIFICADORA VGG-16. 90
FIGURA 54. RESULTADO DE VALIDACIÓN PARA LA PRIMERA SECUENCIA DE IMÁGENES DEL DATASET DE VALIDACIÓN. MODELO CON ETAPA DECODIFICADORA VGG-16. 91
FIGURA 55. RESULTADO DE VALIDACIÓN PARA LA SEGUNDA SECUENCIA DE IMÁGENES DEL DATASET DE VALIDACIÓN. MODELO CON ETAPA DECODIFICADORA VGG-16. 91
FIGURA 56. RESULTADO DE VALIDACIÓN PARA LA TERCERA SECUENCIA DE IMÁGENES DEL DATASET DE VALIDACIÓN. MODELO CON ETAPA DECODIFICADORA VGG-16. 92
FIGURA 57. SELECCIÓN DE LA DESCARGA DEL DISCO 3 DEL DATASET OASIS1 BRAINS EN LA INTERFAZ GRÁFICA DEL APARTADO "DESCARGA DE DISCOS DATASET OASIS1 BRAINS" EN LA APLICACIÓN DE INFERENCIA DE MODELOS. 93
FIGURA 58. SELECCIÓN DE LA CARGA DEL DISCO 3 DEL DATASET OASIS1 BRAINS EN LA INTERFAZ GRÁFICA DEL APARTADO "CARGAR SECUENCIAS DE IMÁGENES" EN LA APLICACIÓN DE INFERENCIA DE MODELOS. 93
FIGURA 59. SELECCIÓN DE LA DESCARGA DEL MODELO PREENTRENADO CON ETAPA DECODIFICADORA RESNET-50 EN LA INTERFAZ GRÁFICA DEL APARTADO "DESCARGA EL MODELO ENTRENADO" DE LA APLICACIÓN DE INFERENCIA DE MODELOS. 93
FIGURA 60. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0081_MR1 UTILIZANDO RESNET-50. 94
FIGURA 61. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS Y LOS NERVIOS ÓPTICOS PARA EL PACIENTE OAS1_0081_MR1 UTILIZANDO RESNET-50. 94
FIGURA 62. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0081_MR1 UTILIZANDO RESNET-50. 94
FIGURA 63. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0086_MR1 UTILIZANDO RESNET-50. 95
FIGURA 64. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS Y LOS NERVIOS ÓPTICOS PARA EL PACIENTE OAS1_0086_MR1 UTILIZANDO RESNET-50. 95
FIGURA 65. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0086_MR1 UTILIZANDO RESNET-50. 95

FIGURA 66. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0092_MR1 UTILIZANDO RESNET-50. 97
FIGURA 67. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS Y LOS NERVIOS ÓPTICOS PARA EL PACIENTE OAS1_0092_MR1 UTILIZANDO RESNET-50. 97
FIGURA 68. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0092_MR1 UTILIZANDO RESNET-50. 97
FIGURA 69. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0095_MR1 UTILIZANDO RESNET-50. 98
FIGURA 70. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS Y LOS NERVIOS ÓPTICOS PARA EL PACIENTE OAS1_0095_MR1 UTILIZANDO RESNET-50. 98
FIGURA 71. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0095_MR1 UTILIZANDO RESNET-50. 98
FIGURA 72. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0114_MR1 UTILIZANDO RESNET-50. 100
FIGURA 73. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS Y LOS NERVIOS ÓPTICOS PARA EL PACIENTE OAS1_0114_MR1 UTILIZANDO RESNET-50. 100
FIGURA 74. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0114_MR1 UTILIZANDO RESNET-50. 100
FIGURA 75. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0082_MR1 UTILIZANDO RESNET-50. 101
FIGURA 76. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS Y LOS NERVIOS ÓPTICOS PARA EL PACIENTE OAS1_0082_MR1 UTILIZANDO RESNET-50. 101
FIGURA 77. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0082_MR1 UTILIZANDO RESNET-50. 101
FIGURA 78. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0081_MR1 UTILIZANDO VGG-16. 103
FIGURA 79. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS Y LOS NERVIOS ÓPTICOS PARA EL PACIENTE OAS1_0081_MR1 UTILIZANDO VGG-16. 103
FIGURA 80. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0081_MR1 UTILIZANDO VGG-16. 103
FIGURA 81. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0086_MR1 UTILIZANDO VGG-16. 104
FIGURA 82. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS Y LOS NERVIOS ÓPTICOS PARA EL PACIENTE OAS1_0086_MR1 UTILIZANDO VGG-16. 104
FIGURA 83. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0086_MR1 UTILIZANDO VGG-16. 104
FIGURA 84. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0092_MR1 UTILIZANDO VGG-16. 105
FIGURA 85. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS Y LOS NERVIOS ÓPTICOS PARA EL PACIENTE OAS1_0092_MR1 UTILIZANDO VGG-16. 105
FIGURA 86. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0092_MR1 UTILIZANDO VGG-16. 105
FIGURA 87. SELECCIÓN DE DATASET SIN NERVIOS ÓPTICOS EN LA INTERFAZ GRÁFICA DE LA APLICACIÓN DE ENTRENAMIENTO DE MODELOS. 107
FIGURA 88. AJUSTE DE PARÁMETROS EN LA INTERFAZ GRÁFICA DEL APARTADO "CARGAR IMÁGENES Y ETIQUETAS DEL DATASET" DE LA APLICACIÓN DE ENTRENAMIENTO DE MODELOS PARA EL ENTRENAMIENTO CON EL DATASET SIN NERVIOS ÓPTICOS. 107
FIGURA 89. VISUALIZACIÓN DEL RECORTE DE LA ZONA DE BÚSQUEDA DE LAS ARTERIAS CARÓTIDAS. 108
FIGURA 90. AJUSTE DE PARÁMETROS EN LA INTERFAZ GRÁFICA DEL APARTADO " DEFINICIÓN DE LOS PARÁMETROS DEL MODELO Y PREPROCESADO DEL DATASET" EN LA APLICACIÓN DE ENTRENAMIENTO DE MODELOS. MODELO DE SEGMENTACIÓN EXCLUSIVA DE LAS ARTERIAS CARÓTIDAS. 108

FIGURA 91. REPRESENTACIÓN GRÁFICA DEL VALOR DEL ERROR DE ENTRENAMIENTO Y EL ERROR DE VALIDACIÓN POR ÉPOCAS. MODELO DE SEGMENTACIÓN EXCLUSIVA DE LAS ARTERIAS CARÓTIDAS. 109
FIGURA 92. REPRESENTACIÓN GRÁFICA DE LA MEDIDA DE "IOU" DE ENTRENAMIENTO Y VALIDACIÓN POR ÉPOCAS. MODELO DE SEGMENTACIÓN EXCLUSIVA DE LAS ARTERIAS CARÓTIDAS. 110
FIGURA 93. SELECCIÓN DE LA DESCARGA DEL MODELO PREENTRENADO DE SEGMENTACIÓN EXCLUSIVA DE LAS ARTERIAS CARÓTIDAS EN LA INTERFAZ GRÁFICA DEL APARTADO "DESCARGA EL MODELO ENTRENADO" DE LA APLICACIÓN DE INFERENCIA DE MODELOS 111
FIGURA 94. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0081_MR1 UTILIZANDO EL MODELO DE SEGMENTACIÓN EXCLUSIVA DE ARTERIAS CARÓTIDAS. 112
FIGURA 95. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0081_MR1 UTILIZANDO EL MODELO DE SEGMENTACIÓN EXCLUSIVA DE ARTERIAS CARÓTIDAS. 112
FIGURA 96. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0095_MR1 UTILIZANDO EL MODELO DE SEGMENTACIÓN EXCLUSIVA DE ARTERIAS CARÓTIDAS. 113
FIGURA 97. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0095_MR1 UTILIZANDO EL MODELO DE SEGMENTACIÓN EXCLUSIVA DE ARTERIAS CARÓTIDAS. 113
FIGURA 98. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0114_MR1 UTILIZANDO EL MODELO DE SEGMENTACIÓN EXCLUSIVA DE ARTERIAS CARÓTIDAS. 114
FIGURA 99. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0114_MR1 UTILIZANDO EL MODELO DE SEGMENTACIÓN EXCLUSIVA DE ARTERIAS CARÓTIDAS. 114
FIGURA 100. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0086_MR1 UTILIZANDO EL MODELO DE SEGMENTACIÓN EXCLUSIVA DE ARTERIAS CARÓTIDAS. 116
FIGURA 101. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0086_MR1 UTILIZANDO EL MODELO DE SEGMENTACIÓN EXCLUSIVA DE ARTERIAS CARÓTIDAS. 116
FIGURA 102. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0092_MR1 UTILIZANDO EL MODELO DE SEGMENTACIÓN EXCLUSIVA DE ARTERIAS CARÓTIDAS. 117
FIGURA 103. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0092_MR1 UTILIZANDO EL MODELO DE SEGMENTACIÓN EXCLUSIVA DE ARTERIAS CARÓTIDAS. 117
FIGURA 104. RESULTADO DE LA SEGMENTACIÓN DE LAS ARTERIAS CARÓTIDAS PARA EL PACIENTE OAS1_0082_MR1 UTILIZANDO EL MODELO DE SEGMENTACIÓN EXCLUSIVA DE ARTERIAS CARÓTIDAS. 118
FIGURA 105. REPRESENTACIÓN 3D DE LAS ETIQUETAS GENERADAS POR LA RED NEURONAL PARA EL PACIENTE OAS1_0082_MR1 UTILIZANDO EL MODELO DE SEGMENTACIÓN EXCLUSIVA DE ARTERIAS CARÓTIDAS. 118
FIGURA 106. TABLA RESUMEN DE RESULTADOS.....119

1- Introducción y objetivos

1.1- Cirugía endoscópica endonasal transesfenoidal

La cirugía endoscópica endonasal es una cirugía mínimamente invasiva para la eliminación de lesiones en la zona de asentamiento de la glándula pituitaria o hipófisis, esta región se denomina silla turca y se encuentra aproximadamente en la base del cráneo.

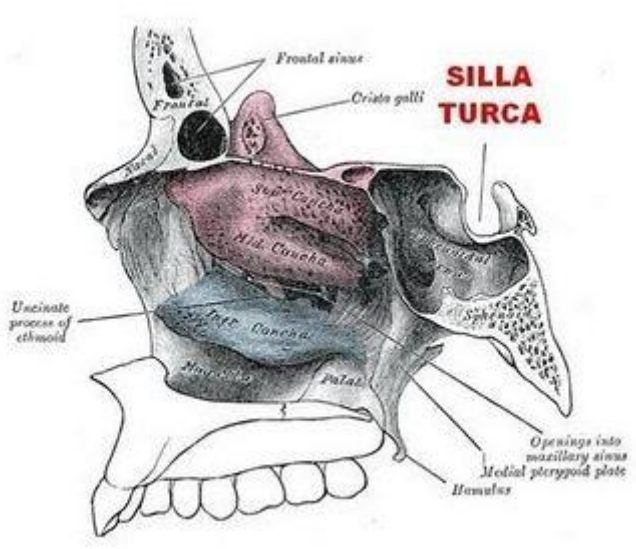

Figura 1: Localización de la silla turca.

Fuente http://neurocirugiacontemporanea.com/doku.php?id=silla_turca

Los adenomas hipofisarios son lesiones tumorales que aparecen normalmente en la parte anterior de la glándula pituitaria, son los tumores intracraneales más frecuentes, suponiendo un 15% del total [1]. La forma habitual de diagnosticar un adenoma hipofisario es la prueba de resonancia magnética nuclear, se obtiene una secuencia de imágenes de la cabeza que permite determinar el tipo de tumor hipofisario, su tamaño y su posición. Los pacientes diagnosticados con este tipo de lesión pueden presentar síntomas como pérdida de visión, dolor de cabeza, desorden endocrino, defectos en el campo de visión, etc. [2].

El proceso de cirugía consiste en introducir un endoscopio y otras herramientas quirúrgicas a través de las fosas nasales, mediante herramientas se abre el camino hasta alcanzar el ostium esfenoidal, (abertura natural al seno esfenoidal). Utilizando una herramienta fresadora, se amplía la abertura natural del ostium, permitiendo la entrada al seno esfenoidal. Una vez dentro del seno esfenoidal, se procede a identificar ciertas estructuras anatómicas, entre ellas la silla turca y se taladra con una herramienta fresadora la pared de la silla turca para acceder a la hipófisis. Con la abertura realizada, se procede a extraer el tumor mediante el uso de curetas y demás instrumental. Finalmente se comprueba que se han eliminado todos los restos tumorales, y se reconstruyen las aperturas realizadas durante la operación empleando fragmentos óseos del propio paciente y añadiendo productos selladores biológicos. [3]

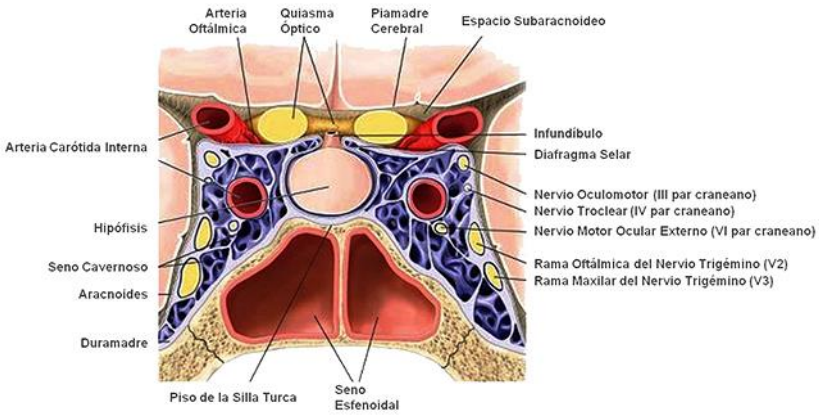

Figura 2. Anatomía de la zona cercana a la hipófisis.

Fuente: http://www.tumoresdehipofisis.com/la-glandula-hipofisis

Debido al proceso de cirugía pueden surgir varias complicaciones como la fístula de líquido cefalorraquídeo, lesión de la arteria carótida interna, sinusitis, perforación del septo nasal, lesión de los nervios ópticos, hemorragia intracerebral y alteraciones endocrinas.

1.2- Segmentación de imágenes médicas

Durante la segunda mitad del siglo XX se comenzaron a desarrollar tecnologías como la resonancia magnética y la tomografía computarizada, que permiten obtener imágenes del interior del cuerpo humano sin alterar la salud del paciente. El desarrollo de estas tecnologías ha permitido adquirir un mayor conocimiento anatómico del cuerpo humano y son tecnologías clave para realizar el diagnóstico y planificación del tratamiento de algunas enfermedades. [4]

La segmentación de imágenes médicas consiste en señalar o delinear elementos de interés en las imágenes médicas, de esta forma se añaden nuevas capas de información útil sobre las imágenes base. La segmentación se realiza a través de un computador, y el proceso puede realizarse de forma manual o se puede implementar un algoritmo de segmentación automática.

La segmentación manual de imágenes médicas suele ser un proceso lento y laborioso, además requiere de personal especializado con buenos conocimientos de la anatomía del órgano o estructura a segmentar.

En el caso de los algoritmos de segmentación automática el proceso de segmentación suele realizarse en unos pocos segundos. Estos programas requieren de personal especializado únicamente durante la etapa de desarrollo. Dependiendo de la complejidad de la zona o zonas a segmentar, un algoritmo de segmentación automática puede proporcionar resultados igual de buenos que la segmentación manual. Las aplicaciones de segmentación automática más tradicionales emplean técnicas de visión artificial, que se centran en la detección y extracción de características de la imagen; los algoritmos más actuales utilizan redes neuronales.

1.3- Objetivos del TFG.

Como se ha expuesto en el primer apartado, en el proceso de cirugía endoscópica se pueden producir daños en las arterias carótidas y en los nervios ópticos; debido a la falta de visibilidad durante la operación puede resultar complicado identificar estas estructuras, por lo que podría ser de gran utilidad la creación de un sistema que indique a los cirujanos la posición de estas estructuras durante la operación para que así puedan evitar acercar demasiado las herramientas quirúrgicas a estas estructuras. Para poder realizar un sistema de este tipo, en este proyecto, se plantea realizar una segmentación automática de las arterias carótidas y los nervios ópticos en una secuencia de imágenes de resonancia magnética obtenidas antes del proceso de cirugía, de esta forma quedarían delineadas estas estructuras y se podría conocer su posición durante la operación.

Con todo lo mencionado anteriormente, se define como principal objetivo del trabajo la creación de una aplicación capaz de identificar y segmentar automáticamente las arterias carótidas y los nervios ópticos a partir de una secuencia de imágenes de resonancia magnética.

2- Segmentación de imágenes médicas con redes neuronales

2.1- Estado del arte: segmentación automática de imágenes médicas mediante redes neuronales convolucionales

En la actualidad existen diversos proyectos que emplean redes neuronales convolucionales para segmentar automáticamente imágenes médicas como radiografías, secuencias de imágenes de resonancia magnética o secuencias de imágenes de tomografía axial computarizada...

El desarrollo de sistemas con redes neuronales convolucionales es posible en la actualidad sobre todo gracias al aumento de la potencia de procesamiento de las tarjetas gráficas o GPU.

2.1.1- Segmentación automática de imágenes de tomografía axial computarizada para procesos de cirugía endoscópica endonasal transesfenoidal.

En el artículo *"Surgical Navigation System for Transsphenoidal Pituitary Surgery Applying U-Net-Based Automatic Segmentation and Bendable Devices"[5]*, se describe la creación de un sistema de navegación que sirva para alertar durante la operación a los cirujanos de la presencia de órganos vitales como las arterias carótidas y nervios ópticos. Para la creación de este sistema, el estudio propone un algoritmo de segmentación automática basado en la arquitectura de red neuronal convolucional U-Net que segmente las arterias carótidas y nervios ópticos en imágenes de tomografía computarizada.

En el artículo también se describe la creación de un endoscopio flexible que permite acceder a partes dentro de la cavidad nasal y esfenoidal a las que un endoscopio rígido no puede acceder.

Para el entrenamiento de la red neuronal del algoritmo de segmentación de este proyecto se etiquetaron manualmente las arterias carótidas y nervios ópticos en las secuencias de imágenes de 6 pacientes, para generar más datos se aplicaron técnicas de aumento del dataset o "*data augmentation*", generando 11 nuevas secuencias por cada una de las secuencias originales.

2.1.2- Segmentación automática de tumores cerebrales en imágenes de resonancia magnética.

En el artículo “Brain tumor segmentation with self-ensembled,deeply-supervised 3D U-net neural networks: a BraTS 2020 challenge solution”[6] se describe la creación de una herramienta para segmentar automáticamente tumores cerebrales en imágenes de resonancia magnética cerebral. Para crear la herramienta en el artículo se propone utilizar una arquitectura de red neuronal convolucional 3D U-Net. El entrenamiento de la red se realiza utilizando el dataset “*Multimodal Brain Tumor Segmentation Challenge (BraTS)*” del año 2020

2.2- Imágenes médicas

En la actualidad existen diversas tecnologías para obtener imágenes del interior del cuerpo humano, más allá de la clásica imagen de radiografía de dos dimensiones, existen tecnologías como la Tomografía Axial Computarizada, (TAC) y la Resonancia Magnética (RM) que permiten obtener una representación en tres dimensiones de los tejidos y estructuras del cuerpo humano.

2.2.1- Tomografía Axial Computarizada (TAC)

La Tomografía Axial Computarizada o Tomografía Computarizada es una tecnología de obtención de imágenes médicas que emplea rayos X y computadores para obtener imágenes-sección del cuerpo humano.

La radiografía es una representación de la atenuación que sufren los rayos X debido a los tejidos y estructuras del cuerpo humano, para construir la imagen del mapa de atenuación, los valores de cada píxel se obtienen como resultado de traducir los valores de atenuación en cada punto a escala de grises.

En la Tomografía Computarizada, al proceso de obtención de la radiografía se le añade un procesamiento posterior de los datos mediante computador, los valores de los píxeles obtenidos se normalizan al coeficiente de atenuación lineal del agua, lo que permite establecer su valor en Unidades Hounsfield, (UH). Los distintos tejidos y estructuras del cuerpo tienen un valor en UH determinado, esto se representa en la Escala Hounsfield.[7]

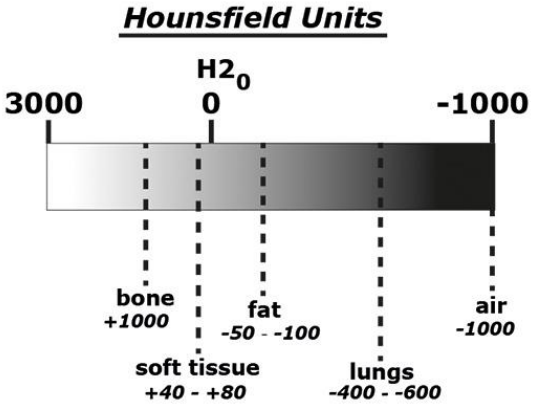

Figura 3. Escala de unidades Hounsfield.

Fuente: https://www.startradiology.com/internships/neurology/brain/ct-brain-hemorrhage/

En la zona del cráneo las imágenes resultado del proceso de TAC presentarán valores de píxel más altos en las zonas óseas; los tejidos cerebrales, el líquido cefalorraquídeo y algunas estructuras como las arterias carótidas y los nervios ópticos presentarán valores UH más cercanos a 0.

2.2.2- Imágenes de resonancia magnética (IRM).

El proceso de obtención de imágenes mediante resonancia magnética comienza con la aplicación de un campo magnético homogéneo de gran intensidad, la gran intensidad del campo generado provoca que los campos magnéticos de una pequeña parte de los protones o núcleos de hidrógeno presentes en los distintos componentes del cuerpo se alineen con el campo generado.

Una vez aplicado el campo magnético, se aplica un pulso de radiofrecuencia de una frecuencia determinada que modifica la orientación de los protones hasta que su momento magnético es perpendicular al campo magnético generado; al encontrarse en esta orientación, los momentos magnéticos inducirán un voltaje en los circuitos sensores de la máquina, que servirán para medir la cantidad de protones de una zona y el tiempo que tardan los protones en alinearse de nuevo con el campo magnético.

El número de protones y el tiempo de realineamiento de cada tejido, líquido o estructura del cuerpo es distinto, esto se recoge con los sensores y se procesa mediante ordenador, dando como resultado imágenes en blanco y negro en las que se pueden distinguir las diferentes estructuras. El valor de los píxeles dependerá de la composición de la zona del cuerpo que representa, las zonas del cuerpo con una composición similar se representarán con valores de píxel similares, esto permite que en las imágenes se puedan diferenciar las distintas estructuras del cuerpo por su composición.

Las imágenes obtenidas corresponden a la representación del corte transversal de la parte del cuerpo sobre la que se encuentra situado el escáner de resonancia magnética. Para obtener una representación tridimensional se desplaza linealmente al paciente por el escáner y se van obteniendo imágenes a medida que este avanza, generando una secuencia de imágenes que forma la representación tridimensional.

Generalmente se utilizan dos tipos de secuencias de resonancia magnética, la secuencia ponderada en T1 y la secuencia ponderada en T2

La secuencia ponderada en T1 mide el tiempo T1 para establecer el valor de los píxeles de la imagen, el tiempo T1 es una constante de tiempo específica del tejido que describe el retorno de la magnetización longitudinal al estado de equilibrio (63% de su valor máximo). La secuencia ponderada en T2 utiliza el tiempo T2, que es una constante de tiempo específica del tejido que describe la pérdida de la magnetización transversal (37% de su valor máximo). En la Figura 4 podemos observar la diferencia entre una imagen ponderada en T1 y una ponderada en T2.

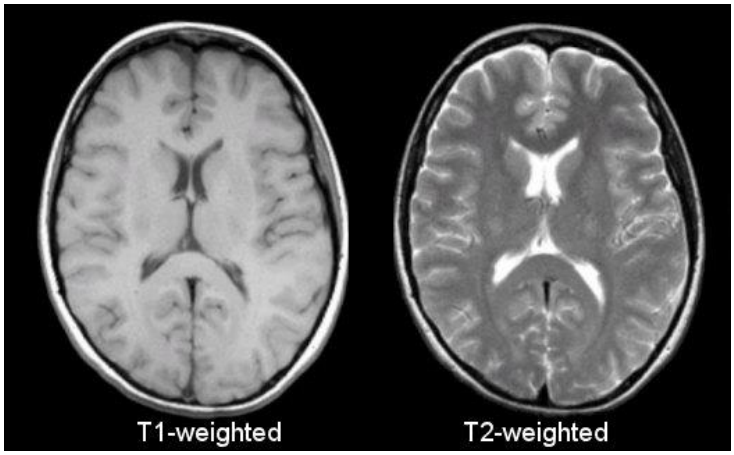

Figura 4. Diferencias entre imágenes de resonancia magnética ponderadas en T1 y en T2.

Fuente: https://case.edu/med/neurology/NR/MRI%20Basics.htm

En la tabla de la Figura 5 se indican los colores con los que se representan algunos tejidos o elementos en las secuencias ponderadas en T1 y en T2.

Tejido/elemento	Secuencia T1	Secuencia T2
Materia gris	Gris	Gris
Materia blanca	Blanco	Gris
Grasa	Blanco	Gris
Aire	Negro	Negro
Hueso cortical	Negro	Negro
Sangre en periodo subagudo	Blanco	Blanco
Sangre en periodo remoto	Negro	Negro

Figura 5. Tabla de coloración de los distintos tejidos y elementos en secuencias T1 y T2

2.3- Redes neuronales artificiales

2.3.1- Definición de red neuronal artificial

Una red neuronal artificial es un sistema de procesamiento de información basado en el funcionamiento del cerebro humano, que trata de imitar el aprendizaje a partir de la experiencia. Las redes neuronales artificiales están compuestas por nodos o neuronas interconectadas, cada una de estas neuronas es una unidad de procesamiento que pondera la información que obtiene en su entrada o entradas, la modificación de estas ponderaciones es la clave del aprendizaje de la red.

Las neuronas se organizan jerárquicamente en capas o niveles, una capa viene definida por un conjunto de neuronas cuyas entradas provienen de la misma fuente y cuyas salidas se dirigen al mismo destino; la agrupación de las distintas capas o niveles conforma el sistema neuronal completo.

“Las RNAs pueden considerarse modelos de cálculo caracterizados por algoritmos muy eficientes que operan de forma masivamente paralela y permiten desarrollar tareas cognitivas como el aprendizaje de patrones, la clasificación o la optimización” [8].

2.3.2- Funcionamiento de la neurona artificial

El funcionamiento de la neurona artificial es simple, la neurona realiza una suma ponderada de sus entradas y aplica sobre este resultado una función de activación, dando como resultado un valor de salida que se transmite al exterior o a otras neuronas. La neurona artificial se puede representar como se indica en la Figura 6.

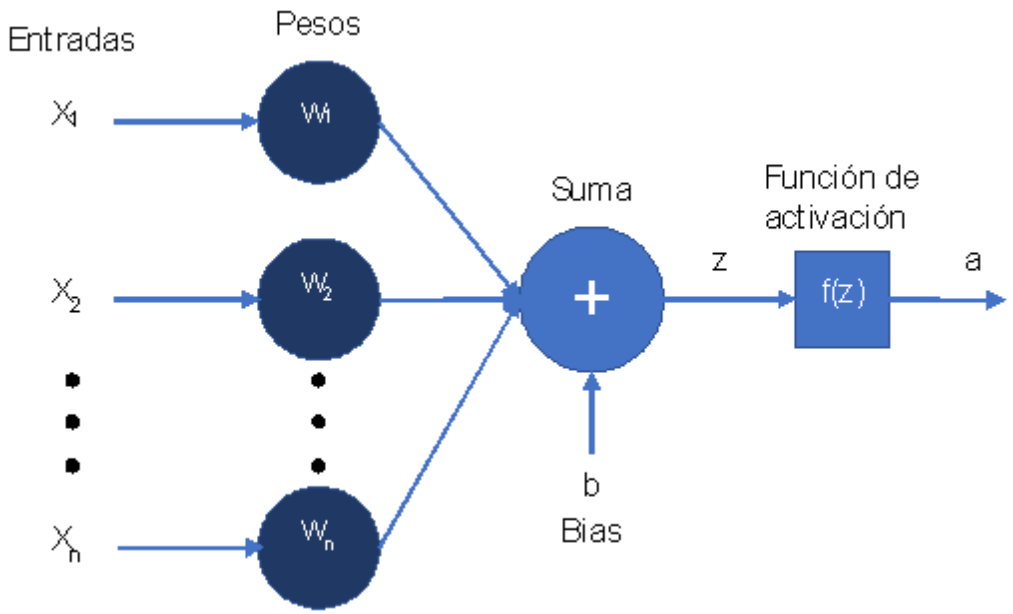

Figura 6. Representación de la neurona artificial

Fuente: elaboración propia.

El modelo matemático de la neurona artificial contiene los siguientes elementos:

- Entradas: Son los valores provenientes del exterior o de otras neuronas.
- Pesos: Son valores que se multiplican por cada una de las entradas, establecen la aportación de cada una de las entradas al valor de salida de la neurona.
- Suma: se realiza la suma de las entradas ponderadas y el bias, da como resultado el valor de la regresión lineal z.
- Bias: es el valor de umbral o threshold, es un parámetro que se añade a la suma para ajustar el valor de salida.
- Función de activación: es una función lineal o no lineal que se aplica sobre el valor resultante de la suma, proporciona el valor de salida de la neurona, (a). Esta función se selecciona al diseñar la red neuronal y dependerá de la tarea que vaya a desempeñar.

El valor de regresión lineal z viene dado por la siguiente expresión:

$$z = \sum_{i=1}^{i=n} X_i \cdot W_i + b$$

El valor de salida a, es el resultado de aplicar la función de activación al valor de regresión lineal, es decir:

$$a = f\left(\sum_{i=1}^{i=n} X_i \cdot W_i + b\right) = f(z)$$

Las funciones de activación pueden ser de diversos tipos, uno de los casos más sencillos es cuando la función de activación es f(z)=1 para z>0 y f(z)=0 para z<=0; en este caso la neurona funcionaría como un clasificador binario simple. En la Figura 7 se puede observar un ejemplo de clasificador binario simple que utiliza solamente una neurona artificial.

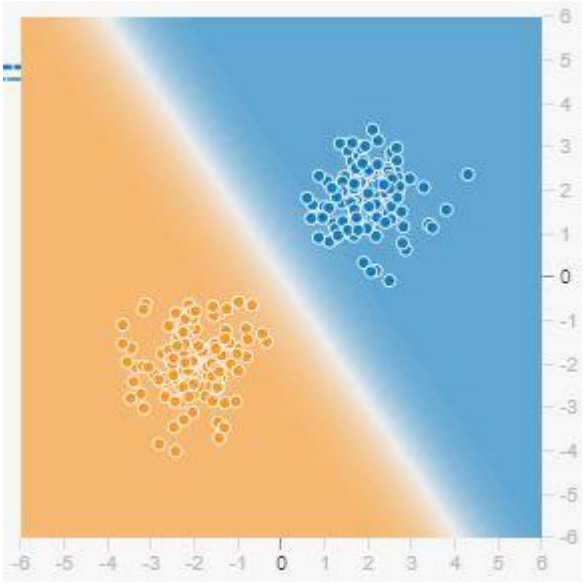

Figura 7. Resultado de clasificador binario con una única neurona.

Fuente: elaboración propia utilizando la herramienta playground tensorflow: http://playground.tensorflow.org/

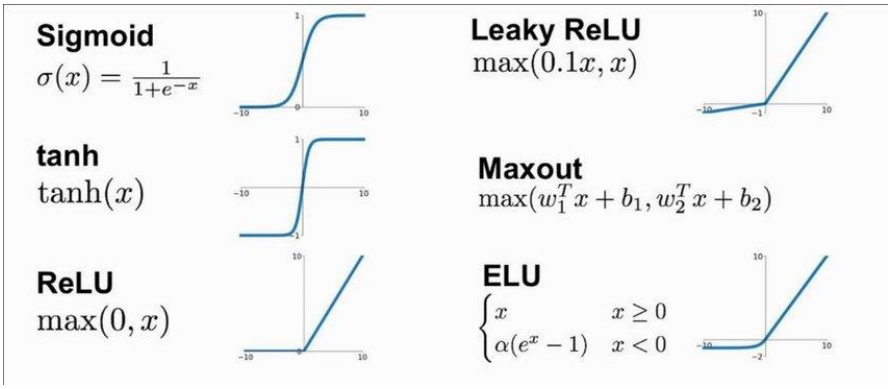

Figura 8. Varios tipos de funciones de activación existentes.

Fuente: https://www.researchgate.net/figure/Fig-3-The-basic-activation-functions-of-the-neural-networksNeural-Networks_fig3_350567223

2.3.3- Estructura de la red neuronal artificial

Como comentábamos anteriormente, en una red neuronal artificial, las neuronas se organizan jerárquicamente en capas o niveles, esto se denomina arquitectura multicapa y permite crear sistemas con mayor funcionalidad que permitan resolver problemas más complejos.

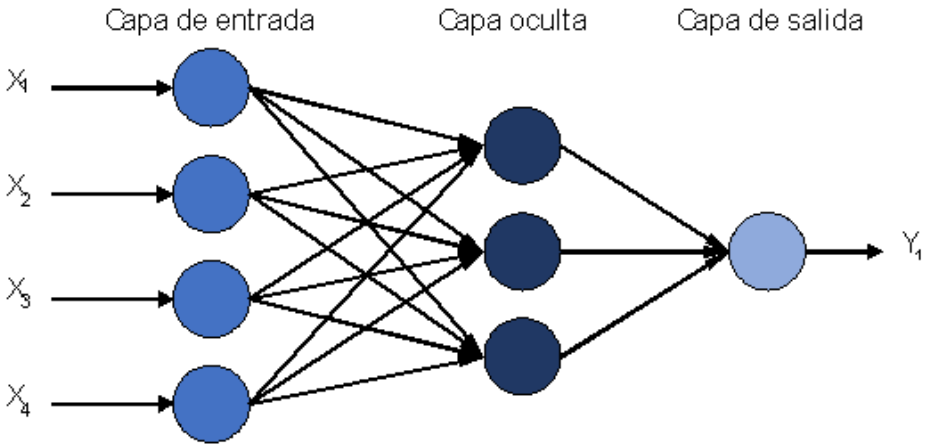

Figura 9. Representación de red neuronal multicapa.

Fuente: elaboración propia

El valor de salida es igual a la suma de los valores ponderados de las anteriores capas, si las funciones de activación fueran lineales, el resultado de la suma ponderada en la última neurona sería otra función lineal, lo que sería equivalente a utilizar una única neurona, para obtener una arquitectura multicapa funcional, las funciones de activación deben ser no lineales, de esta forma los valores de salida de las neuronas de las capas anteriores se tienen en cuenta para calcular el valor de la salida de la red y se pueden resolver problemas más complejos, como se puede observar en la Figura 10.

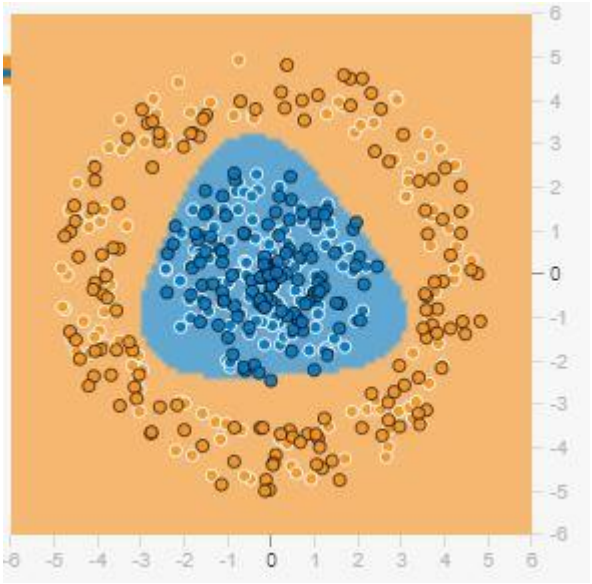

Figura 10. Resultado de clasificación binaria para datos de entrada más complejos.

Fuente: elaboración propia utilizando la herramienta playground tensorflow: http://playground.tensorflow.org/

2.3.4- Aprendizaje en redes neuronales artificiales

Según [8]: “el aprendizaje puede definirse como el proceso por el que una red neuronal crea, modifica o destruye sus conexiones (pesos) en respuesta a una información de entrada”. En redes neuronales artificiales existen dos modos principales de funcionamiento: el modo de entrenamiento o aprendizaje y el modo de ejecución u operación, para que el segundo modo funcione correctamente, previamente se debe ejecutar correctamente el proceso de entrenamiento.

Existen dos tipos de reglas de aprendizaje en redes neuronales artificiales: el aprendizaje supervisado y el aprendizaje no supervisado.

Aprendizaje supervisado: se caracteriza por la existencia de un agente externo que controla el proceso de entrenamiento, en este proceso son conocidos los valores de salida deseados ante unos determinados valores de entrada. De esta forma el agente supervisor compara los valores de salida con los valores deseados y ajusta los pesos de la red de forma iterativa hasta que los valores de salida se aproximan a los valores deseados, para este proceso se emplea el valor del error cometido en cada paso del entrenamiento. Las redes que se emplearán en este trabajo serán de este tipo.

Aprendizaje no supervisado: se caracterizan por no necesitar de información externa para ajustar los valores de los pesos de la red, es la propia red la que estima los pesos necesarios extrayendo de los datos de entrada rasgos o agrupando patrones según su similitud.

2.3.5- Descenso del gradiente y el algoritmo de “retropropagación de errores” o “backpropagation”

El gradiente es un vector de n-dimensiones que proporciona la dirección de máximo incremento de una función en un determinado punto, se obtiene evaluando la variación de la función respecto a todas sus variables independientes en un punto. El vector gradiente se obtiene derivando una función respecto a cada una de las variables de las que depende, el vector es de la siguiente forma:

$$grad(f) = \nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

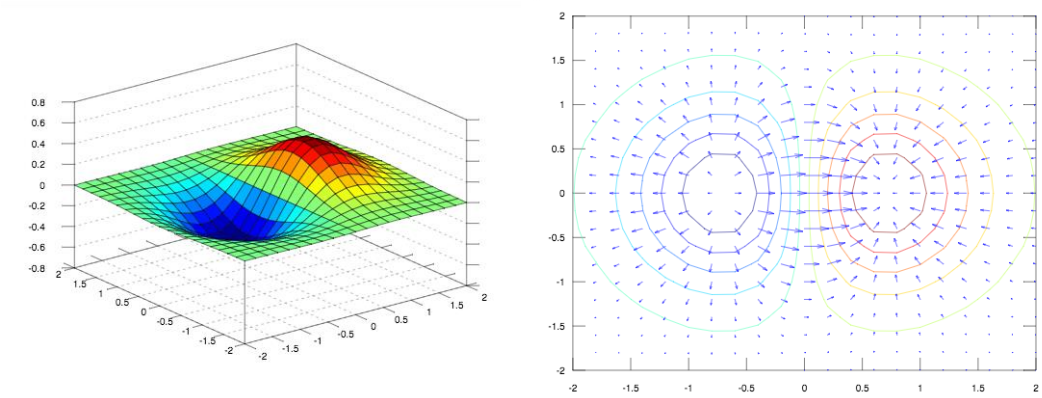

Figura 11. Representación del gradiente en un espacio tridimensional.

Fuente: https://www.geogebra.org/m/SUVruKxm

En el caso de las redes neuronales artificiales, el gradiente se emplea para obtener la dirección de crecimiento de la función de error de salida y así poder tomar medidas para alcanzar un mínimo, el gradiente tendrá tantas dimensiones como variables entrenables tenga la red neuronal.

Para obtener el mínimo se podría igualar el gradiente a cero, esto daría como resultado un sistema de n-ecuaciones que habría que resolver para obtener los valores óptimos de las variables, esto es posible cuando la función a optimizar es sencilla, para casos más complejos esto resulta prácticamente imposible, por lo que tenemos que recurrir a métodos que aproximen estos valores óptimos.

El método del descenso de gradiente se puede emplear para aproximar los valores óptimos de las variables que minimicen el error, es un método muy sencillo que se ejecuta de la siguiente forma:

1- Se obtiene el valor del error en un punto y se calcula el gradiente
2- Nos desplazamos a un nuevo punto a una distancia α en la dirección del gradiente calculado, en sentido descendente.
3- Obtenemos el valor del error en el nuevo punto, si es lo suficientemente pequeño paramos, si no, se vuelve a calcular el gradiente y se repite el paso 2.

En el método del descenso de gradiente, la convergencia al valor mínimo dependerá en parte del valor del tamaño de paso o tasa de aprendizaje α, si empleamos un valor de α demasiado grande, el valor del error puede no converger al mínimo y por lo tanto el método no funcionaría; para evitar esto podríamos emplear un valor α pequeño, de esta forma es más sencillo que el valor del error converja al mínimo. Cuanto más pequeño sea el tamaño de paso, más tiempo le llevará al algoritmo alcanzar el punto de error mínimo, por lo que conviene ajustar el valor para que el error converja y el tiempo no sea muy grande.

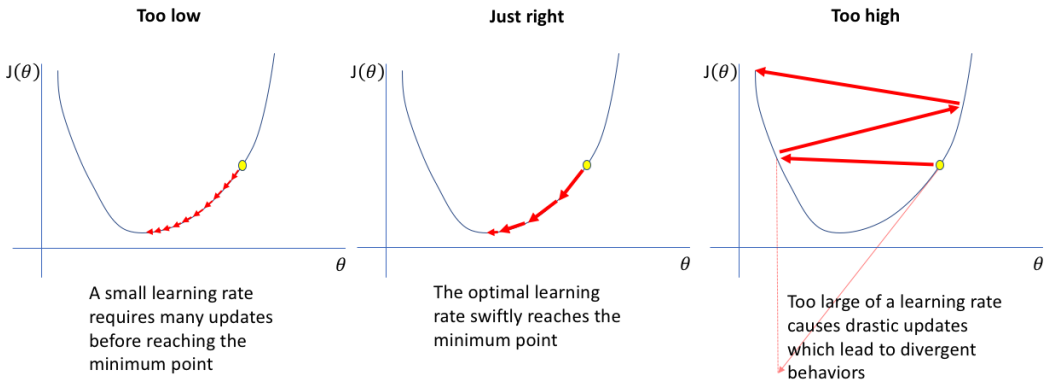

Figura 12. Representación del método de descenso de gradiente.

Fuente: https://www.jeremyjordan.me/content/images/2018/02/Screen-Shot-2018-02-24-at-11.47.09-AM.png

Para poder aplicar el método de descenso de gradiente, necesitamos actualizar los valores de los pesos de la red en todas las capas. Para esto se puede emplear el algoritmo de retropropagación de errores o backpropagation, cuyo funcionamiento se puede separar en dos fases:

1- Fase de aprendizaje “hacia adelante”: se introducen los valores de entrada en la primera capa, se propagan los valores resultantes capa por capa hasta obtener el valor o valores de salida de la red.
2- Fase de aprendizaje “hacia atrás”: se calcula el valor del error comparando la salida obtenida con la salida deseada, se asigna un valor de error a cada neurona de la capa de salida y se propaga hacia las neuronas de la capa anterior, que reciben una parte del error proporcional a su participación en la salida original. Este proceso se repite capa por capa hasta que cada neurona haya recibido un error que describa su aportación relativa a la salida. Obtenido el error para cada neurona, se procede a reajustar los valores de los pesos para continuar con el proceso de minimización del error.

2.3.6- Evaluación del aprendizaje

A la hora de desarrollar aplicaciones con redes neuronales artificiales uno de los principales intereses es que esta red sea capaz de proporcionar buenos resultados cuando se le muestran datos nuevos, esto se denomina capacidad de generalización y se comprueba durante la fase de validación o evaluación del aprendizaje.

Para realizar la evaluación del aprendizaje, se genera un conjunto de datos que no se utilizarán durante el entrenamiento y se introducen en la red para obtener el valor del error de evaluación o validación. Comparando el valor del error de validación con el valor del error obtenido con los datos de entrenamiento, podemos estimar la capacidad de generalización de la red.

Comparando los errores de entrenamiento y validación podemos obtener varias conclusiones sobre la generalización:

- Si el error de entrenamiento y el error de validación tienen valores similares y cercanos a 0: en este caso podemos concluir que los resultados que proporciona la red para los datos de validación son igual de buenos que los que proporciona para los datos de entrenamiento, y por lo tanto la capacidad de generalización es buena.
- Si el error de entrenamiento es mucho menor que el error de validación: en este caso los resultados que proporciona la red para los datos de validación son peores, por lo que la generalización no es buena. Además, si el error de entrenamiento es prácticamente 0 y el de validación es mucho mayor, es posible que se esté produciendo un problema de sobreentrenamiento o "overtraining" que consiste en que la red se ajusta demasiado a las particularidades de los datos de entrenamiento, lo que provoca una pérdida en la capacidad de generalización. Esto se puede ver en la Figura 13
- Si el error de entrenamiento y el error de validación tienen valores grandes, (mucho mayores que 0): En este caso la red neuronal no ha sido capaz de encontrar relaciones en los datos que reduzcan el error, es decir, el modelo no funciona y es posible que haya que modificar su estructura o seleccionar mejor los datos de entrenamiento.

Para evitar problemas como el sobreentrenamiento y obtener modelos de red neuronal con buena capacidad de generalización, lo que se suele hacer es evaluar el aprendizaje durante el entrenamiento, es decir, se realiza el proceso de evaluación en cada iteración o época justo después de que los parámetros de la red hayan sido modificados. De esta forma se puede ir visualizando la evolución del entrenamiento y se puede detectar el punto en el que el aprendizaje es óptimo.

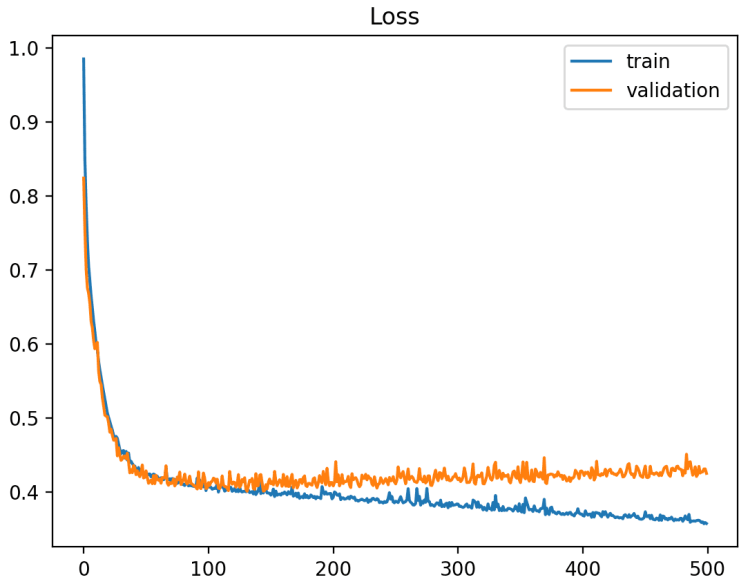

Figura 13. Ejemplo de curvas de error en modelo con sobreentrenamiento.

Fuente: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

2.4- Redes neuronales convolucionales.

2.4.1- Análisis de imágenes mediante redes neuronales artificiales.

Una de las aplicaciones de las redes neuronales artificiales es la de analizar imágenes, es decir, se pueden utilizar RNAs para realizar tareas como reconocer y posicionar objetos presentes en una imagen. Este tipo de tareas resultan generalmente sencillas para el ser humano, que de manera intuitiva es capaz de distinguir los distintos elementos en una imagen.

Uno de los principales problemas del aprendizaje en imágenes es que en la mayor parte de los casos no resulta fácil formalizar con exactitud las características visuales de un objeto. Debido a esto, es preferible crear modelos que sean capaces de extraer su propio conocimiento a partir de los datos, este tipo de modelos forman parte del área denominada machine learning o aprendizaje automático.

El Machine Learning o aprendizaje automático se define como el aprendizaje de una serie de reglas o patrones a partir de unos resultados definidos previamente. Los modelos de machine learning son capaces en muchos casos de aprender patrones a partir de los datos, pero no son capaces de aprender conceptos abstractos y complejos a partir de esos mismos datos, es aquí donde aparece el campo del Deep Learning.

El Deep Learning o aprendizaje profundo es un subconjunto dentro del Machine Learning que tiene como objetivo generar modelos que sean capaces de representar conceptos abstractos y complejos a partir de otros más sencillos. En este caso, el modelo es capaz de crear automáticamente una jerarquía de conceptos, empezando por los más sencillos y creando otros más complejos a partir de la combinación de los conceptos más simples.

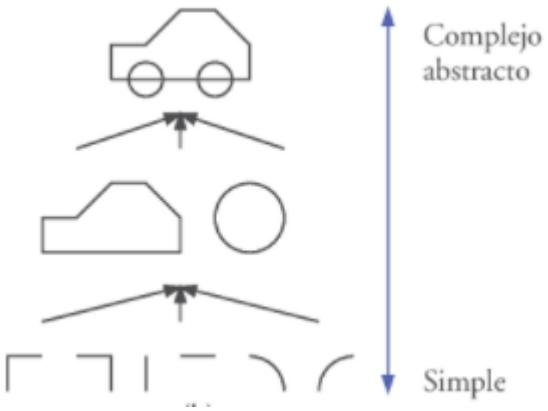

Figura 14. Ejemplo de generación de conceptos abstractos a partir de conceptos simples.

Fuente: Ilustración del libro "Deep Learning principios y fundamentos" [9]

2.4.2- La convolución.

La convolución es una operación ampliamente utilizada en las áreas de la estadística, procesamiento de señal, ingeniería y visión por computador entre otras, es una operación matemática sobre dos funciones que produce una tercera función que expresa como la forma de una es modificada por la otra.

Para el caso de las redes neuronales convolucionales la operación de convolución se expresa de la siguiente forma:

$$s(t) = (f * g)(t) \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau)$$

Donde f corresponde a la entrada o input, g corresponde al filtro o "kernel" y s corresponde al mapa de características o "feature map".

A la hora de trabajar con imágenes, lo más habitual es que estas se compongan de matrices de dos dimensiones, para el caso de una imagen de dos dimensiones, la operación de convolución se expresa de la siguiente forma:

$$S(i,j) = (I * K)(i,j) \sum_{m} \sum_{n} I(i-m, j-n)K(m,n)$$

Donde I corresponde a la matriz de la imagen de entrada y K corresponde a la matriz filtro o kernel, en esta expresión anterior, se ha aplicado la propiedad conmutativa, para que esta propiedad se cumpla, se debe realizar un flipping del kernel.

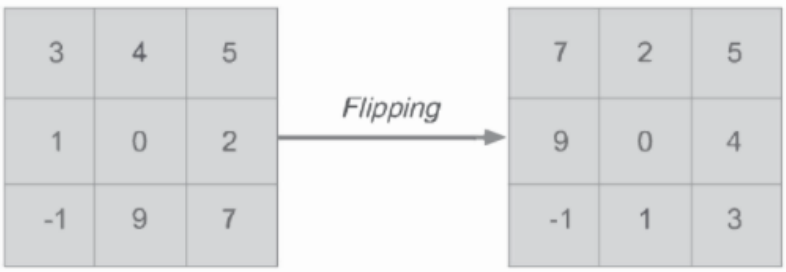

Figura 15. Ejemplo de flipping de un kernel.

Fuente: Ilustración del libro "Deep Learning principios y fundamentos" [9]

En la gran mayoría de aplicaciones de Deep Learning no se realiza el flipping, ya que no es realmente necesario para la implementación de redes neuronales, y al no realizarlo se mejora la eficiencia. Por lo tanto, la gran mayoría de librerías de redes neuronales no realizan realmente la operación de convolución, en su lugar realizan una operación denominada cross-correlation que se expresa de la siguiente forma:

$$S(i,j) = (I * K) \sum_m \sum_n I(i+m, j+n) K(m,n)$$

A pesar de que esta operación es en realidad una cross-correlation, por convención, en el área de Deep Learning se llama a esta operación convolución.

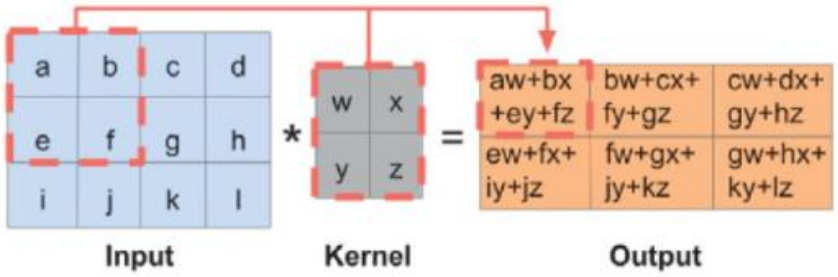

Figura 16. Ejemplo de una convolución 2D sin flipping del kernel.

Fuente: Ilustración del libro "Deep Learning principios y fundamentos" [9]

Dentro de una red neuronal de Deep Learning, las neuronas de una capa se conectan sólo a un subconjunto de las neuronas de la capa siguiente, esto se denomina interacciones dispersas y se debe a que el tamaño del kernel utilizado en la convolución es mucho más pequeño que la entrada. Esto supone una mejora frente a las redes neuronales tradicionales, en las que cada neurona se encuentra conectada a todas las neuronas de las capas anteriores y posteriores, (fully connected), lo que da lugar a redes con un gran número de parámetros que requieren de más espacio en memoria y más tiempo de computación.

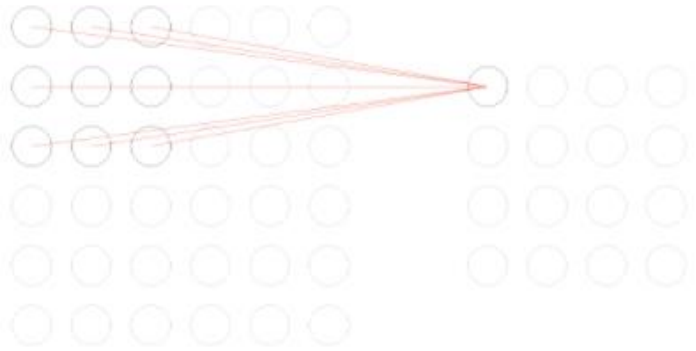

Figura 17. Ejemplo de interacciones dispersas de una red neuronal convolucional.

Fuente: Ilustración del libro "Deep Learning principios y fundamentos" [9]

La estructura y tamaño del conjunto de neuronas de la capa i que se conectan con una neurona de la capa i + 1 forma lo que se conoce como el campo receptivo local, (local receptive fields).

A diferencia de lo visto anteriormente para redes neuronales, en este tipo de redes se usan los mismos pesos y sesgos, (bias) para la todas las neuronas ocultas de una misma capa. Teniendo en cuenta esto, la salida para una neurona es de la siguiente forma:

$$\sigma\left(b + \sum_{l=0}^{n-1}\sum_{m=0}^{n-1} \omega_{l,m}.\, a_{j+l,k+m}\right)$$

Donde σ representa la función de activación, b es el valor compartido de sesgo, $\omega_{l,m}$ es una matriz de nxn que contiene los pesos compartidos de las neuronas de una misma capa y $a_{x,y}$ es el valor de entrada de la posición x,y.

Esto significa que todas las neuronas de la capa oculta detectan exactamente la misma característica, solo que la detectan en las distintas ubicaciones de la imagen.

Al conjunto de neuronas que comparten un mismo filtro detector de características o kernel, se le denomina mapa de activaciones. En una red neuronal Deep Learning se utilizan múltiples mapas de activaciones para que la red pueda detectar diversas características.

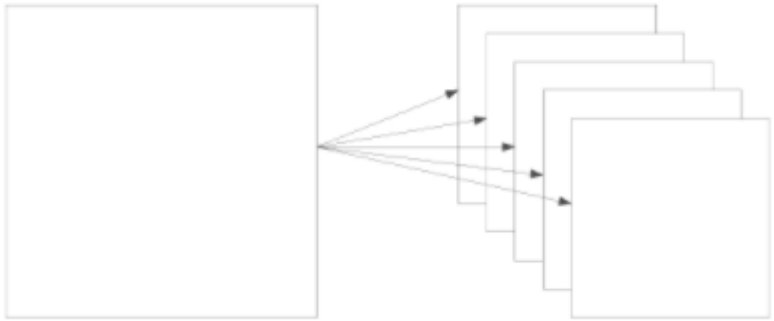

Figura 18. Ejemplo de estructura con cinco mapas de características en la capa oculta.

Fuente: Ilustración del libro "Deep Learning principios y fundamentos" [9]

Una gran ventaja de compartir pesos y sesgos es que se reduce el número de parámetros a ajustar, lo que permite que los modelos de red neuronal ocupen menos espacio y requieran un tiempo menor de computación, esto también se traduce en que se pueden crear modelos con más capas de profundidad para que la red sea capaz de aprender a identificar conceptos más complejos.

2.4.3- Componentes de una red neuronal convolucional.

Dentro de una red neuronal convolucional podemos encontrar distintos tipos de componentes con distintas funcionalidades, en este apartado se explicará el funcionamiento de algunos de estos componentes.

Para comenzar, se va a explicar el funcionamiento de las capas de convolución de la red, estas capas son las que contienen los filtros o kernel que se aplicarán para extraer características de las imágenes.

Los kernel son las matrices de pesos que implementan la convolución en toda la matriz de entrada de la capa, son matrices de un tamaño generalmente muy inferior al de las matrices de entrada. La entrada a estas capas puede ser una imagen en escala de grises, (un solo canal) una imagen a color, (tres canales) o una imagen con más canales, es por esto que los kernel deben tener el mismo número de canales que la entrada.

Al aplicar la convolución con los kernel, la salida sufre una reducción del tamaño respecto a la imagen de entrada, (excepto en el caso de utilizar un kernel de tamaño 1x1). Para evitar la reducción del tamaño de la salida, se puede aplicar una técnica denominada "zero padding", que consiste en añadir píxeles con valor 0 en los bordes de la imagen de entrada tal y como se muestra en la Figura 19.

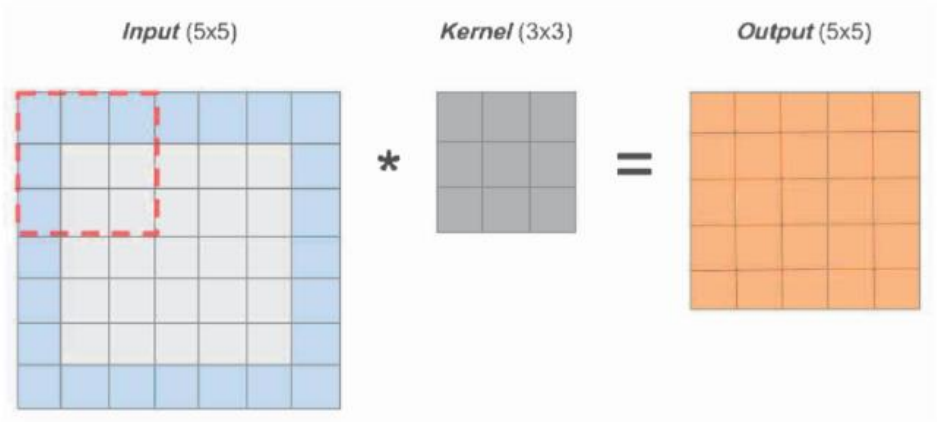

Figura 19. Ejemplo de la aplicación del "zero padding" para mantener la dimensión de la salida.

Fuente: Ilustración del libro "Deep Learning principios y fundamentos" [9]

En el caso de que se desee reducir las dimensiones de la salida para aumentar la eficiencia de la red, se puede aplicar otra técnica denominada convoluciones por pasos o “strided convolutions”, esta técnica consiste en aplicar la convolución solamente en algunas de las celdas de la imagen de entrada, saltándose un número de celdas determinado por el parámetro del paso o “stride”. Esta técnica permite aumentar la eficiencia a costa de perder algunas de las características en la salida. Podemos observar el funcionamiento de la convolución por pasos para el caso “stride”=2 en la Figura 20 .

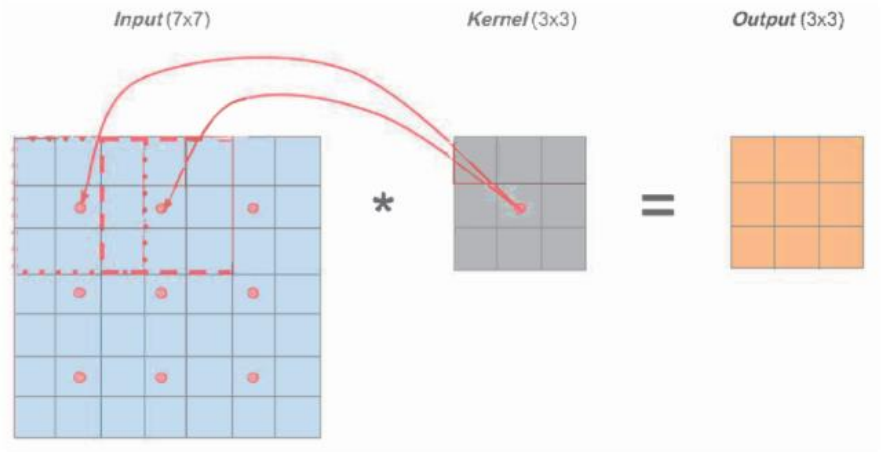

Figura 20. Ejemplo de una convolución usando un “stride” de 2 y un “padding” de 0.

Fuente: Ilustración del libro “Deep Learning principios y fundamentos” [9]

Otras de las capas más comunes empleadas además de las de convolución son las capas de agrupamiento o “pooling”, estas capas se emplean para hacer que algunas de las características obtenidas sean más robustas y eficientes.

Existen varios tipos de “pooling”, siendo el “max pooling” el más utilizado, en este agrupamiento, se obtiene el mayor valor del conjunto de valores de entrada. El “max pooling” se aplica mediante matrices de unas determinadas dimensiones y con un valor de paso como veíamos en las convoluciones por pasos. Podemos ver un ejemplo de “max pooling” con un filtro de 2x2 y paso 2 en la Figura 21 .

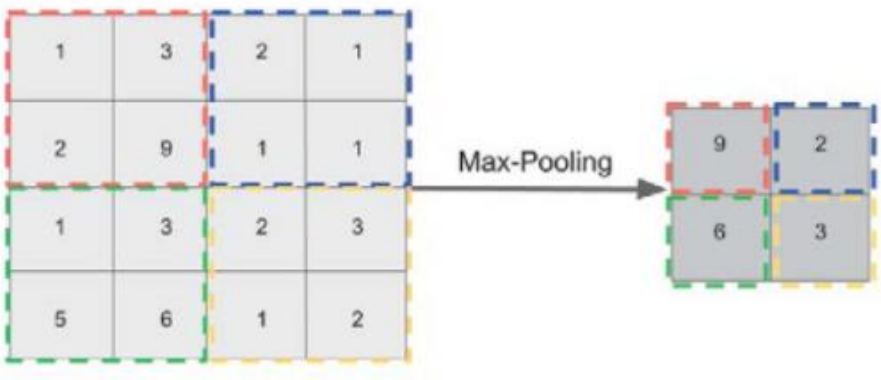

Figura 21. Ejemplo de agrupamiento basado en “max pooling”.

Fuente: Ilustración del libro “Deep Learning principios y fundamentos” [9]

La aplicación del max pooling sirve para obtener una representación de las zonas de la imagen con las características más frecuentes y relevantes. Otra propiedad importante de este tipo de capas es que únicamente tiene dos hiperparámetros que no se necesitan entrenar, se fijan antes del entrenamiento y corresponden al tamaño de la matriz a aplicar y el tamaño de paso.

Como vimos en apartados anteriores, para crear redes de mayor profundidad, es necesario introducir funciones no lineales en la red neuronal, es aquí donde resultan de utilidad las capas ReLU, (Rectified Linear Unit). El propósito de estas capas es introducir una no linealidad en la red, anteriormente se utilizaban otras funciones como la tangente hiperbólica o la sigmoide, pero las investigaciones recientes comprobaron que el funcionamiento y rendimiento es mejor utilizando funciones ReLU, además ayudan a aliviar el problema de la desaparición del gradiente, (problema que provoca que el entrenamiento sea más lento en las primeras capas de la red que en las últimas debido al decrecimiento exponencial del gradiente a través de las capas).

La capa ReLu aplica la función f(x)=max(0, x) a todos los valores de la entrada, es decir, cambia los valores negativos de entrada por 0, incrementando las propiedades no lineales del modelo y de toda la red, sin que esto afecte a los campos receptivos de la capa de convolución[9].

Otras de las capas utilizadas habitualmente son las capas totalmente conectadas o “fully connected layers”, estas son capas en las que las entradas se encuentran conectadas a todas las salidas de la capa anterior y dan como salida un vector de N-dimensiones. Estas capas se pueden utilizar para crear redes neuronales de clasificación de imágenes, al colocar una capa fully connected al final de la red, el número de N-dimensiones será igual al número de clases que el modelo tiene que escoger, cada una de estas dimensiones representará la probabilidad de que un objeto pertenezca a cierta clase; para este tipo de tareas es frecuente añadirle a la capa una función de activación softmax, que da como resultado un valor entre 0 y 1 para cada clase que representa la probabilidad de que la imagen pertenezca a esa clase.

Las capas fully connected se suelen colocar al final de la red convolucional con la función de determinar cuáles son las características de alto nivel más relevantes que correlacionan con una clase en particular.

2.4.4 Arquitecturas de redes neuronales convolucionales.

En la actualidad existen diversas arquitecturas de redes neuronales ya establecidas, en este apartado se va a explicar la estructura y el funcionamiento de las arquitecturas VGG-16 y Resnet-50

La arquitectura VGG-16 [10]es una arquitectura que destaca por su simplicidad, para todas sus capas de convolución utiliza filtros de tamaño 3x3 y un tamaño de paso o stride de 1, también utiliza capas de agrupamiento del tipo max pooling con filtros de tamaño 2x2 y stride 2.

Como se puede observar en la Figura 22, la red se compone de 13 capas de convolución, 5 capas de max pooling y 3 capas totalmente conectadas, además las capas de convolución y las totalmente conectadas utilizan la función de activación ReLu.

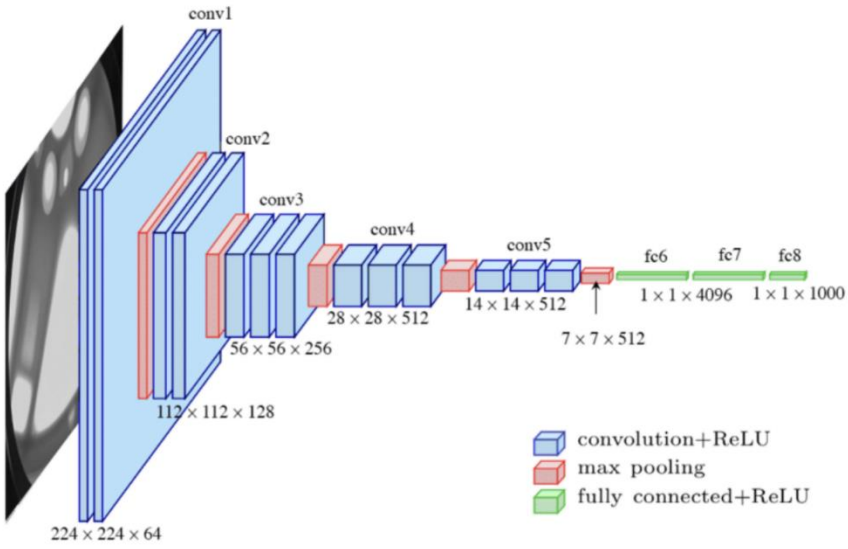

Figura 22. Representación de la arquitectura de la red neuronal convolucional VGG-16.

Fuente: https://www.researchgate.net/figure/Fig-A1-The-standard-VGG-16-network-architecture-as-proposed-in-32-Note-that-only_fig3_322512435

Esta arquitectura se empleó para clasificar el dataset ImageNet en el año 2014, recibiendo el primer puesto en la competición ILSVR (ImageNet Large Scale Visual Recognition Challenge 2014). VGG16 es una red bastante profunda que tiene alrededor de 138 millones de parámetros entrenables, esto es una gran cantidad de parámetros y es la principal desventaja de esta red.

Otro tipo de arquitectura de redes neuronales convolucionales es la arquitectura de “residual networks” o ResNet[10], este tipo de arquitectura se planteó con el objetivo de puentear el progreso de aprendizaje en algunas de las capas mediante la implementación de la función identidad. En la realidad, la implementación directa de la función identidad no es posible debido al problema de desaparición y explosión del gradiente, (vanishing/exploding gradient) es por esto que las redes residuales implementan la identidad a partir del cálculo del residuo de entrada y salida de algunas capas (o subredes), como se muestra en la Figura 23 .

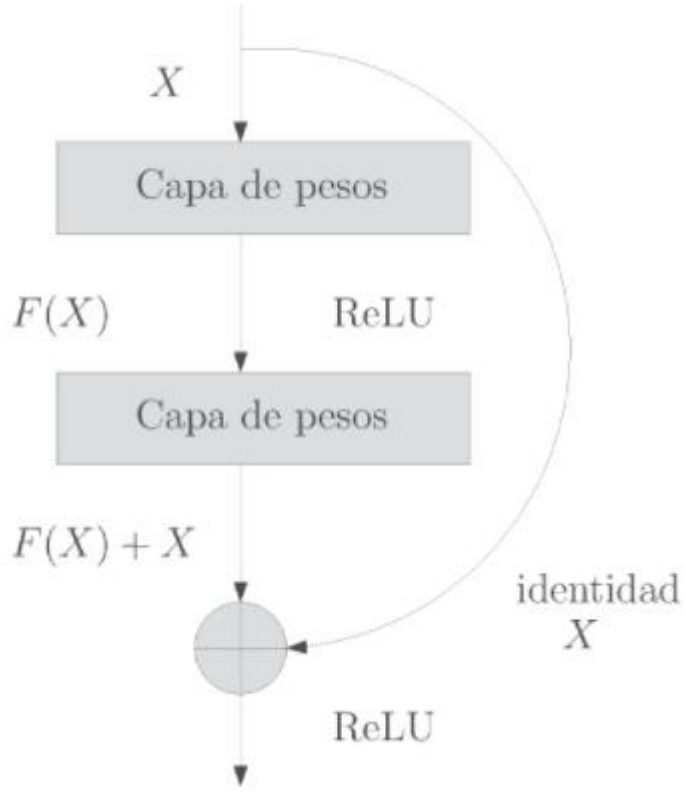

Figura 23. Residual ResNet.

Fuente: Ilustración del libro “Deep Learning principios y fundamentos” [9]

Considerando X como la entrada de la subred y H(X) al verdadero resultado de salida de la subred, el residual es la diferencia entre ellos: F(X) = H(X)-X, nos interesa obtener el valor del resultado de la subred, por lo que reorganizamos la anterior expresión de la siguiente forma: H(X)=F(X)+X.

La obtención del resultado a partir del residual supone una ventaja sobre los otros tipos de redes en los que se obtiene directamente el resultado, al utilizar el residual, se le da a la red la opción de omitir subredes, haciendo F(X)=0, de esta forma tenemos que H(X)=X, esto quiere decir que la salida de una subred en particular es solo la salida de la última subred. La utilización del residual también aporta una buena propiedad durante la retropropagación o backpropagation del gradiente, ya que permite a la red ignorar el gradiente de algunas de las subredes y simplemente reenviar el gradiente desde las capas superiores a las capas inferiores sin ninguna modificación, esto permite evitar realizar cálculos innecesarios.

Dentro de este tipo de arquitectura, encontramos la arquitectura ResNet-50 que podemos observar en la Figura 24, esta arquitectura consiste en 5 etapas compuestas cada una por un bloque de convolución y un bloque de identidad.

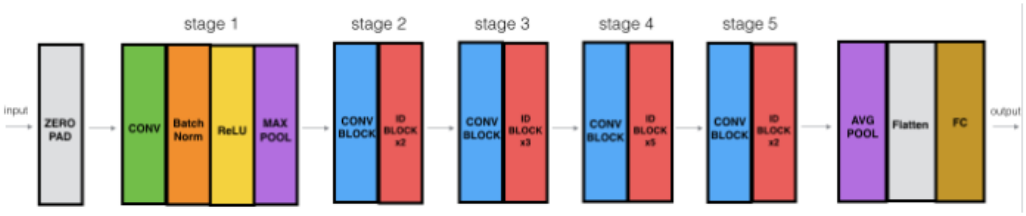

Figura 24. Representación por bloques de la arquitectura ResNet-50.

Fuente: https://github.com/priya-dwivedi/Deep-Learning/blob/master/resnet_keras/Residual_Networks_yourself.ipynb

El bloque de convolución tiene 3 capas de convolución y el bloque de identidad otras 3 capas, podemos ver la estructura de estos bloques en la Figura 25 y la Figura 26.

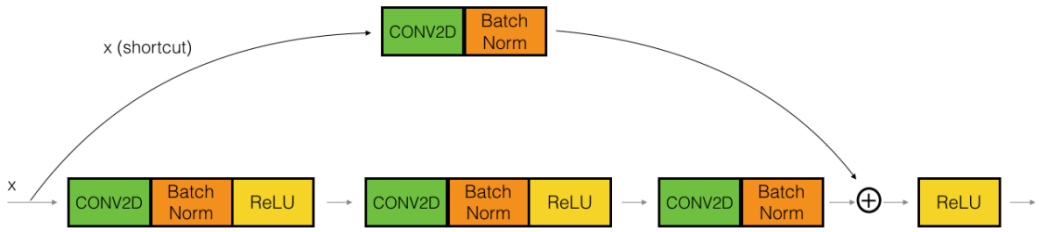

Figura 25. Bloque de convolución de Resnet-50

Fuente: https://github.com/priya-dwivedi/Deep-Learning/blob/master/resnet_keras/Residual_Networks_yourself.ipynb

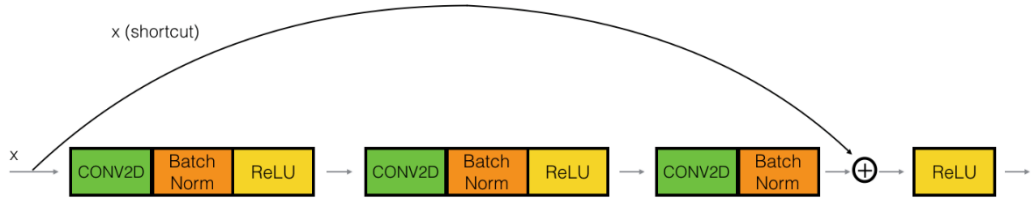

Figura 26. Bloque de identidad de ResNet-50.

Fuente: https://github.com/priya-dwivedi/Deep-Learning/blob/master/resnet_keras/Residual_Networks_yourself.ipynb

La arquitectura ResNet-50 tiene un total de 50 capas y en torno a 23 millones de parámetros entrenables, es una arquitectura que permite obtener mayor profundidad con un número mucho menor de parámetros que otras arquitecturas como por ejemplo VGG-16.

2.4.5- Redes neuronales convolucionales de segmentación.

En este apartado se van a exponer el funcionamiento y características de las redes neuronales convolucionales de segmentación, estas son las redes que se utilizarán para desarrollar la aplicación objetivo de este proyecto.

Una de las formas de realizar una red neuronal de segmentación consiste en modificar la arquitectura VGG-16, reemplazando las capas totalmente conectadas del final por la convolución, a este tipo de red se la denomina Fully Convolutional Network, ya que solamente utiliza convoluciones.

Este tipo de arquitectura es capaz de realizar la clasificación de los píxeles de una imagen, es decir, es capaz de segmentar elementos dentro de una imagen. Debido a la reducción de la resolución al realizar a las convoluciones y agrupaciones, el tamaño de la imagen de salida es más pequeño que la imagen de entrada. Para recuperar el tamaño de entrada, se pueden aplicar interpolaciones, pero lo más probable es que la segmentación no abarque correctamente las zonas que debería, debido a la pérdida de información producida al reducir la resolución. El uso de este tipo de redes se describe en el papel *Fully Convolutional Networks for Semantic Segmentation[11]*.

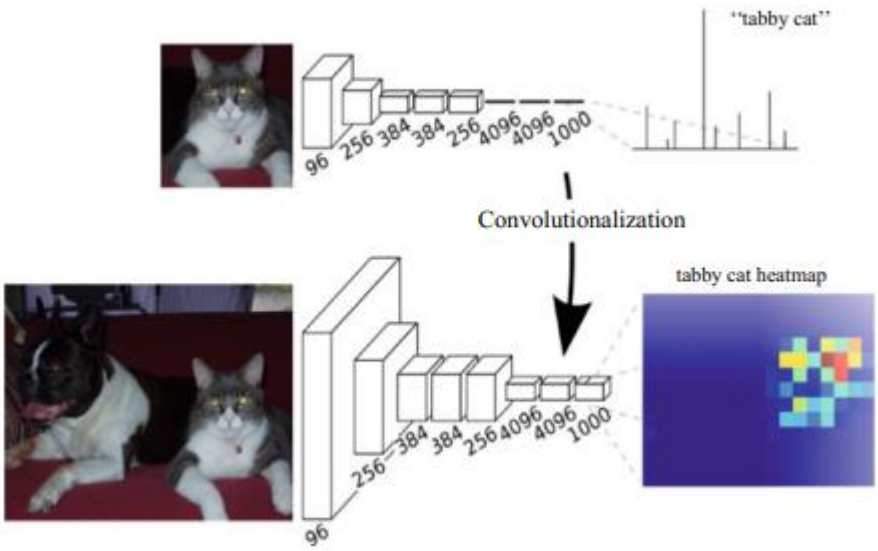

Figura 27. Ejemplo de Fully Convolutional Network para segmentación de imágenes.

Fuente: Ilustración del libro "Handbook of Deep Learning applications" [24]

Para evitar tener que realizar una interpolación para obtener segmentaciones del mismo tamaño que la entrada surgen otro tipo de arquitecturas como las arquitecturas codificador-decodificador, (encoder-decoder). Estas arquitecturas tienen una primera etapa codificadora en la que se extraen las características y se va disminuyendo la resolución a medida que se avanza en profundidad, y una segunda etapa decodificadora en la que se recupera gradualmente el tamaño original.

Un ejemplo de arquitectura codificador decodificador es la arquitectura U-Net[12], que fue desarrollada por el departamento de ciencias de la computación de la universidad de Friburgo para realizar segmentaciones de imágenes biomédicas.

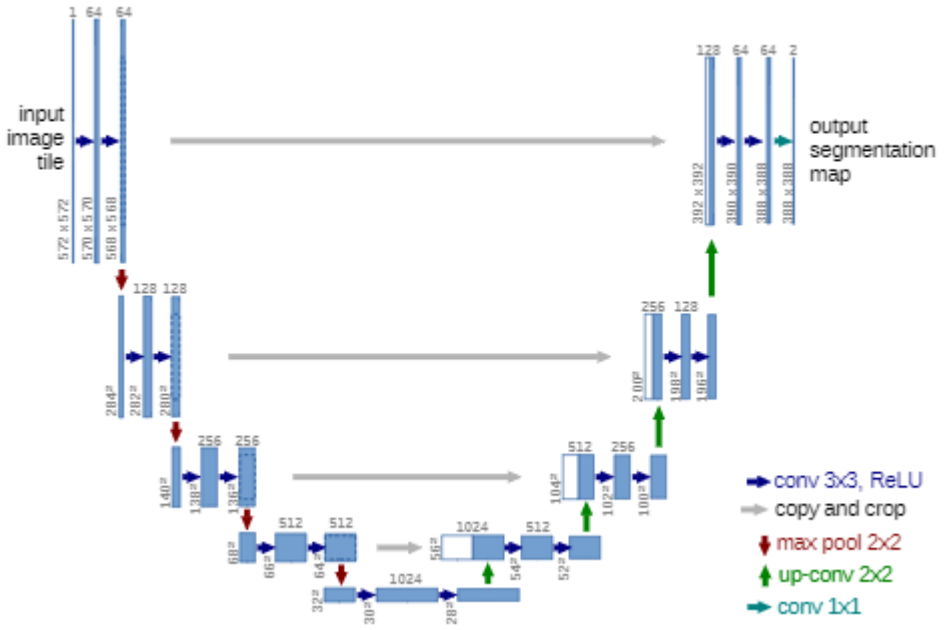

Figura 28. Representación de la arquitectura U-Net.

Fuente: https://towardsdatascience.com/paper-summary-u-net-convolutional-networks-for-biomedical-image-segmentation-13f4851ccc5e

Como podemos observar en la Figura 28, la etapa codificadora de la red está formada por bloques con max pooling que reducen la resolución, la etapa decodificadora utiliza bloques de convolución transpuesta que invierten el proceso. Lo especial de esta arquitectura es que concatena la entrada de cada bloque de la etapa codificadora, con la salida de los bloques de la etapa decodificadora, consiguiendo mantener la información espacial que de otra manera se habría perdido. A pesar de que se desarrolló para usarse con imágenes biomédicas, la arquitectura de u-net es capaz de resolver todo tipo de problemas de segmentación.

Una de las modificaciones que se pueden realizar a la arquitectura es la de modificar la etapa codificadora por una de las redes convolucionales preestablecidas como VGG-16 o ResNet, para ello se eliminan las capas totalmente conectadas del final y se utilizan únicamente las capas de extracción de características. Esta modificación permite por ejemplo inicializar los pesos de la parte codificadora con pesos ya entrenados en datasets como imagenet, lo que habitualmente mejora el funcionamiento del modelo[13].

Para el caso de datos de entrada volumétricos, como las secuencias de imágenes de resonancia magnética o tomografía computarizada, también se puede adaptar la arquitectura U-Net para segmentar datos volumétricos, esta arquitectura modificada se llama 3D U-Net[14]. Esta variante emplea kernels de forma cúbica, (3 dimensiones por canal) en las convoluciones, es decir aplica la convolución de un mismo kernel a varias capas de imágenes 2D al mismo tiempo. Para entrenar este tipo de redes, es necesario que el volumen etiquetado esté compuesto por capas de superficies 2D etiquetadas.

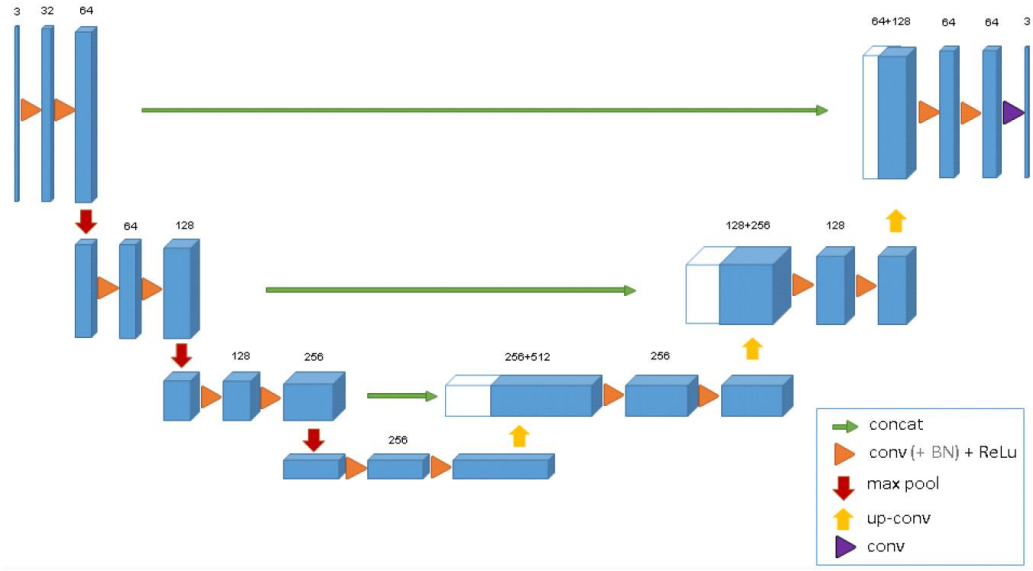

Figura 29. Representación de la arquitectura 3D U-Net.

Fuente: https://towardsdatascience.com/review-3d-u-net-volumetric-segmentation-medical-image-segmentation-8b592560fac1

3- Desarrollo del proyecto

3.1- Recursos utilizados.

3.1.1- Dataset OASIS-1 Brains.

Para la realización de este proyecto se necesita una cantidad grande de imágenes de resonancia magnética del cerebro, tanto para la parte de entrenamiento de la red neuronal, como para la parte de prueba de funcionamiento de la red.

La gran mayoría de estas imágenes no se encuentran disponibles al público, su disponibilidad está reservada a centros médicos o centros de investigación, por lo que resulta complicado encontrar grandes cantidades de estas imágenes disponibles para su uso público. En el proceso de búsqueda de estos recursos, se encontró el dataset OASIS-1 Brains[15]

El dataset OASIS-1 Brains, (Open Access Series of Imaging Studies) es un dataset con disponibilidad de uso público para investigación y estudio que contiene secuencias de imágenes de resonancia magnética cerebral tipo T1 de 416 pacientes con edades comprendidas entre los 18 y 96 años. Los pacientes del dataset son tanto hombres como mujeres, 100 de estos pacientes fueron diagnosticados con Alzheimer y para cada paciente se realizaron varias adquisiciones de secuencias. Para este proyecto lo más interesante de este dataset es la gran cantidad y variedad de secuencias disponibles.

El dataset se puede descargar libremente desde su página oficial, el dataset completo se divide en 12 discos que contienen los archivos de todos los pacientes, por lo tanto, para descargarlo en su totalidad, se deben descargar uno a uno estos discos.

Para la adquisición de las secuencias IRM, se inmovilizó la cabeza de los pacientes con una máscara facial hecha con un material termoplástico y se colocó una cápsula de vitamina E, sobre la parte izquierda de la frente, esta cápsula se puede ver en las secuencias de imágenes resaltada en color blanco y sirve como marca de referencia.

Como se ha visto anteriormente, las secuencias de imágenes de resonancia magnética permiten obtener una representación tridimensional de los tejidos blandos del cuerpo humano, en una IRM de la cabeza, además de los tejidos intracraneales, también se pueden observar sin problema los tejidos que forman la cara del paciente, por lo que sus rasgos faciales pueden ser reconocibles. Para mantener el anonimato de los pacientes, en este dataset, se empleó un algoritmo que elimina las partes de la secuencia de imágenes correspondientes a la zona de la nariz y boca, haciendo que los rasgos faciales del paciente sean irreconocibles.

El dataset se encuentra estructurado en carpetas numeradas que contienen la información y secuencias de los pacientes, dentro de cada una de estas carpetas, existen otras tres carpetas y un documento de texto que contiene información sobre el paciente. La estructura de carpetas para cada paciente es la siguiente:

- Carpeta RAW: Contiene las secuencias IRM originales sin ningún procesamiento, estas secuencias se encuentran almacenadas en archivos tipo .hdr, .img y .gif.
- Carpeta PROCESSED: Contiene secuencias IRM a las que se le ha aplicado un procesamiento para mejorar el contraste entre tejidos y se han adaptado las dimensiones de las imágenes para que se adapten a sistema espacial del atlas de Talairach y Tournoux.
- Carpeta FSL_SEG: Contiene una secuencia IRM en la que se ha aplicado una segmentación de los distintos tipos de tejido del cerebro.

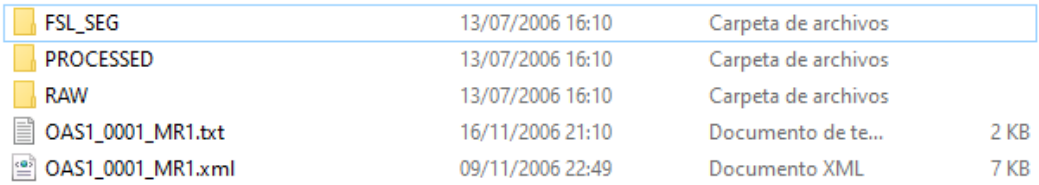

Figura 30. Estructura de carpetas para cada paciente del dataset OASIS-1.

Fuente: elaboración propia

Para este proyecto, las secuencias más adecuadas serán las que se encuentran en la carpeta "PROCESSED", ya que presentan un contraste y calidad mayor y esto facilitará el proceso de etiquetado y el aprendizaje de la red neuronal.

3.1.2- Entorno de programación Google colaboratory.

Google colaboratory es un entorno de programación online gratuito que permite crear programas en código Python y ejecutarlos remotamente en un computador de Google, el único requisito para utilizar este entorno es poseer una cuenta de Google.

Los programas que se crean desde este entorno se denominan Python notebooks o cuadernos de Colab, consisten en celdas de texto en las que podemos escribir código Python; estas celdas se pueden ejecutar individualmente para dividir la ejecución del programa completo en secciones más cortas.

Uno de los principales atractivos de este entorno es la posibilidad de utilizar recursos hardware como GPUs para la ejecución del código. El uso de GPUs es de gran utilidad para ejecutar algoritmos de redes neuronales convolucionales, la ejecución con GPU permite mejores entrenamientos que la ejecución convencional en la que se usa únicamente la CPU del computador.

Otra de las posibilidades de este entorno es la de compartir de forma sencilla el código generado con otras personas, los programas generados se guardan dentro del espacio de almacenamiento de la nube de Google Drive, por lo que se pueden compartir con cualquier persona utilizando un enlace. Además de programas, también se pueden compartir otros archivos como modelos ya entrenados y datasets, lo que permite que cualquier persona con acceso a un programa que requiera de estos archivos pueda probar el funcionamiento del programa de forma sencilla.

El uso de Google Colab también aporta las siguientes ventajas:

- Simplicidad a la hora de instalar nuevas librerías.
- Elevada velocidad para bajar y subir archivos.
- Posibilidad de emplear distintas versiones de Python.
- Posibilidad de crear interfaces gráficas para introducir datos a los programas.
- Visualización directa de imágenes y gráficas.
- Posibilidad de añadir cuadros de texto enriquecido en el mismo documento.

Teniendo en cuenta estas funcionalidades, se consideró que este entorno era la mejor opción para desarrollar las aplicaciones de este proyecto.

3.1.3- 3DSlicer

El proceso de etiquetado en este proyecto se ha realizado mediante el software 3DSlicer[16], (https://www.slicer.org) es un software gratuito y de código abierto que dispone de diversas herramientas para etiquetar y visualizar imágenes médicas.

3.2- Proceso de etiquetado de imágenes con 3DSlicer.

Para entrenar la red neuronal artificial necesitamos “etiquetar” varias secuencias de imágenes, el proceso de etiquetado consiste en marcar los píxeles de la imagen que componen el objeto o elemento buscado. Una vez se han marcado estas zonas, se genera una nueva imagen en la que los píxeles de las zonas marcadas presentan un valor determinado, las zonas que no han sido marcadas se señalizan en la imagen con el valor de píxel 0.

En los siguientes apartados, se describe el proceso completo para etiquetar imágenes empleando este software.

3.2.1- Cargar la secuencia de imágenes IRM de un paciente

Antes de poder etiquetar las imágenes, necesitamos cargar la secuencia de imágenes de resonancia magnética del paciente en el programa, para hacer esto, hacemos clic sobre el botón “load data” y seleccionamos el directorio del paciente que contiene la secuencia de imágenes, para que la secuencia se cargue correctamente, es necesario que las imágenes estén correctamente numeradas.

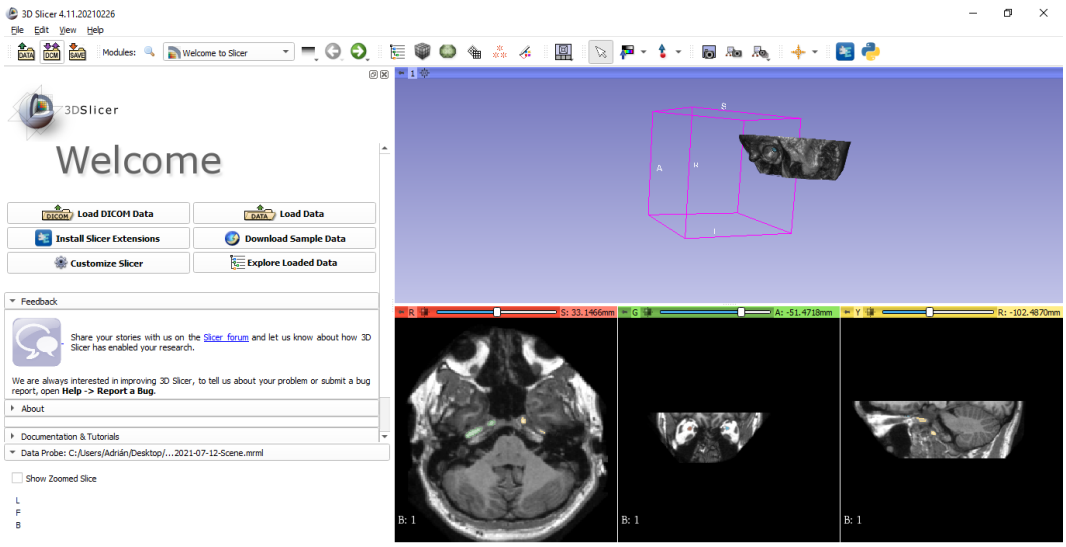

Figura 31. Vista general del programa con una secuencia de imágenes cargada.

Fuente: elaboración propia.

Una vez cargadas, se podrán visualizar en las ventanas del programa tres representaciones del volumen formado por las secuencias, (coronal, axial y sagital) y una representación en tres dimensiones. Empleando los deslizadores situados encima de cada ventana o la ruleta del ratón podemos desplazarnos por las capas que forman el volumen.

3.2.2- Crear las etiquetas

Para este proyecto necesitamos señalizar las zonas de la imagen que contengan las arterias carótidas y los nervios ópticos, para hacer esto, seleccionamos la pestaña “segment editor” del programa y hacemos clic en add para añadir una nueva etiqueta. A la hora de crear las etiquetas, conviene considerar el orden de creación de estas, en este proyecto el orden seguido es el siguiente:

1. Etiqueta de la carótida derecha
2. Etiqueta de la carótida izquierda
3. Etiqueta del nervio óptico derecho
4. Etiqueta del nervio óptico izquierdo

Este orden se debe respetar siempre, si no se hace, los valores de las máscaras de las etiquetas serán distintos y su interpretación será errónea. También se debe tener en cuenta que las ventanas de visualización del programa de etiquetado muestran la secuencia de imágenes vista “desde abajo” por lo que la carótida derecha se ve en el lado izquierdo y viceversa.

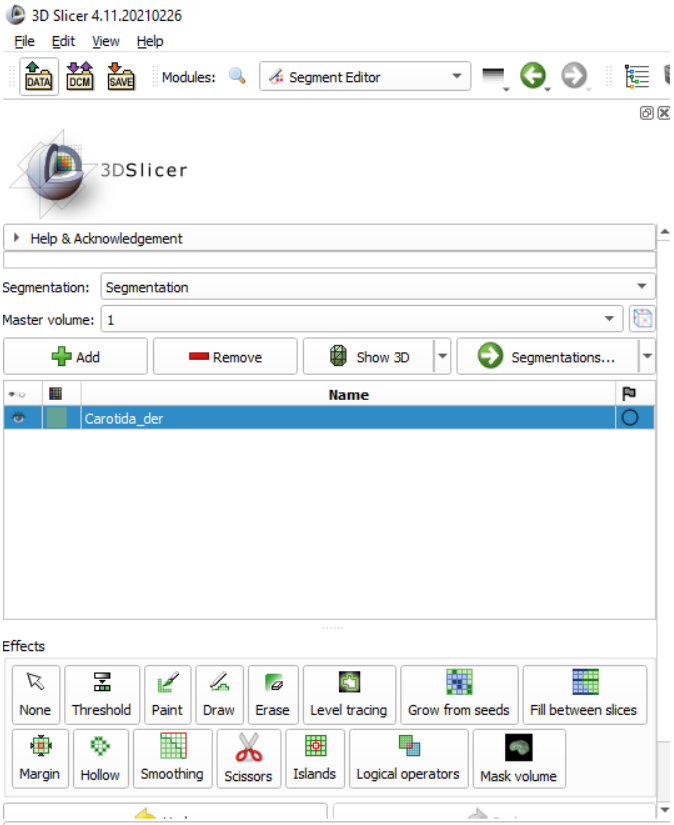

Figura 32. Pestaña de creación de etiquetas.

Fuente: elaboración propia.

Una vez creada la etiqueta debemos seleccionar las partes de la imagen que nos interesan, para esto disponemos de varias herramientas, en este trabajo se ha empleado la siguiente técnica:

1. Seleccionamos la opción “threshold”, (umbral) del apartado “effects”, al hacer esto se mostrará en las ventanas de las imágenes las áreas de la imagen con un valor de píxel dentro del rango de umbral seleccionado; ajustamos el rango para que contenga las partes que deseamos seleccionar y hacemos clic sobre el botón use for masking, de esta forma sólo se podrá pintar sobre estas zonas cuando usemos la herramienta Paint.
2. Pintamos las zonas a seleccionar utilizando la herramienta “Paint” en todas las imágenes de la secuencia, en el caso de no poder pintar una zona podemos volver a ajustar el rango del umbral.
3. Una vez hemos terminado de etiquetar la secuencia de imágenes, debemos exportar las etiquetas creadas, para hacer esto, seleccionamos la pestaña “Segmentations” y buscamos el apartado “export to files”, en este apartado deberemos seleccionar el directorio en el que guardar el archivo, después seleccionaremos el formato de archivo, (“file format”) como formato NIFTI y pulsaremos el botón export. Siguiendo este proceso obtendremos un archivo que contiene las etiquetas de toda nuestra secuencia de imágenes.

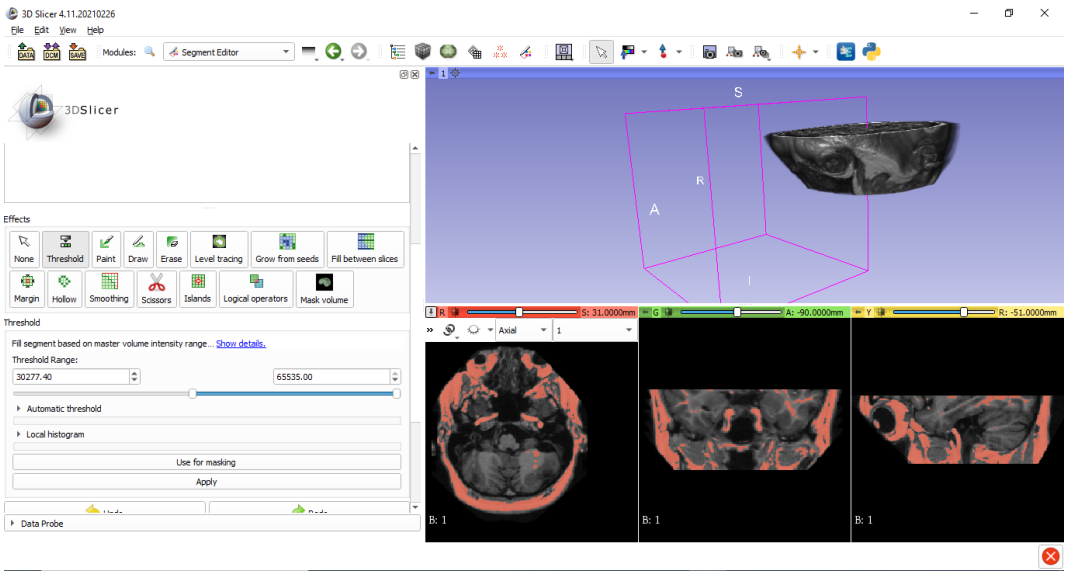

Figura 33. Utilización de la herramienta “threshold”.

Fuente: elaboración propia.

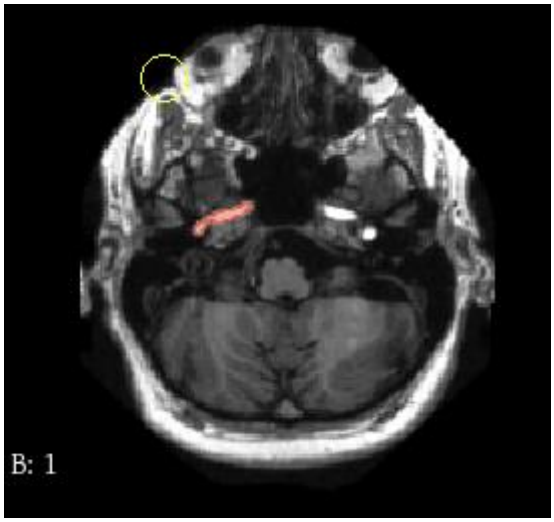

Figura 34. Etiquetado de la arteria carótida derecha

Fuente: elaboración propia.

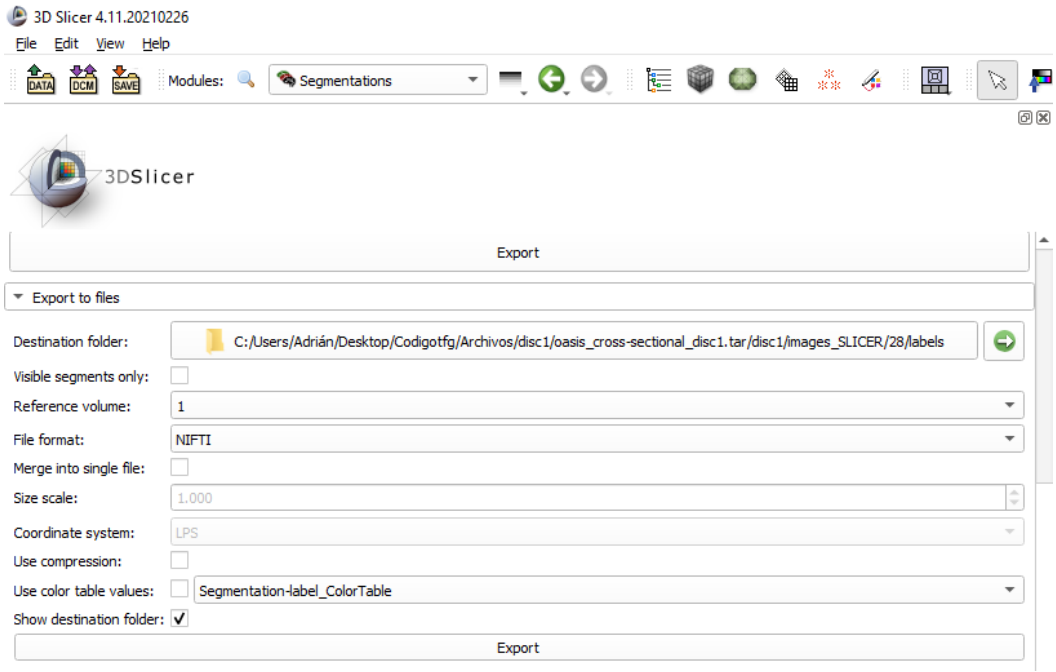

Figura 35. Apartado "Export to files" de la pestaña "Segmentations"

Fuente: elaboración propia.

Siguiendo este proceso podemos ir etiquetando las secuencias de imágenes paciente a paciente para crear el dataset que emplearemos posteriormente para entrenar y validar la red neuronal de segmentación.

3.2.3- Creación del dataset. Estructura de directorios.

Con los datos ya etiquetados, debemos organizar los datos generados en una estructura de directorios o carpetas para cargarlos correctamente en el programa de Deep learning. La estructura es la siguiente:

Para cada paciente se crea una carpeta con el nombre original correspondiente del dataset Oasis-1, dentro de cada una de estas carpetas deben existir dos archivos, uno llamado volume.tif que corresponde a la secuencia de imágenes mri y otro archivo llamado Segmentation.nii que corresponde a las etiquetas creadas.

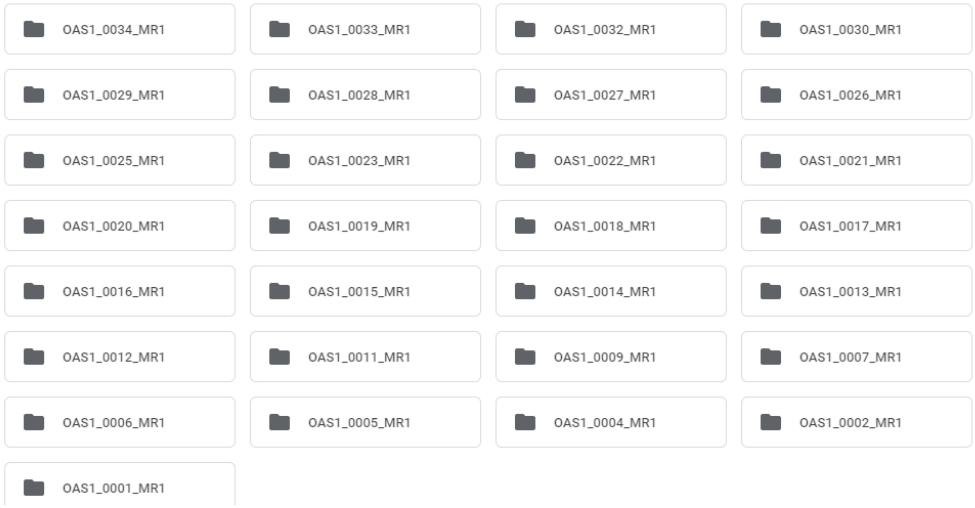

Figura 36. Estructura de directorios del dataset.

Fuente: elaboración propia.

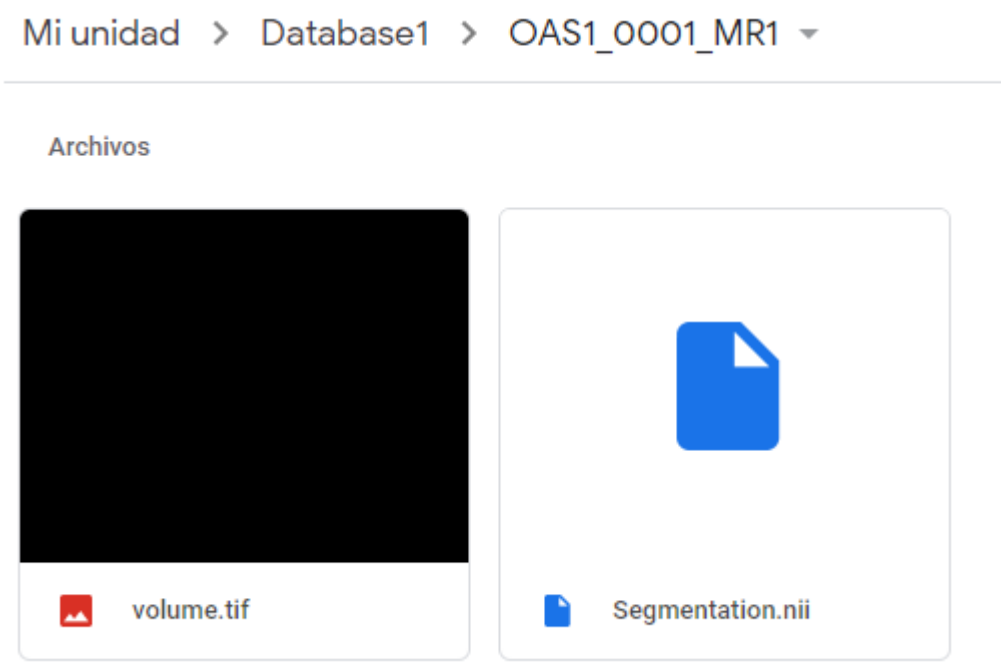

Figura 37. Archivos de secuencia de imágenes y etiquetas que se encuentran dentro de la carpeta de uno de los pacientes del dataset.

Fuente: elaboración propia.

Una vez creadas las carpetas con los archivos dentro, introducimos todas las carpetas en otra carpeta con el nombre que le queramos dar al dataset generado, con esto ya estaría listo para poder usarse en la aplicación de entrenamiento.

3.3- Aplicación de entrenamiento de modelos.

La aplicación de entrenamiento de modelos es la primera aplicación desarrollada en este proyecto, tiene la función principal de generar modelos entrenados de redes neuronales convolucionales de segmentación que se puedan utilizar posteriormente por la aplicación de inferencia.

Se consideró que la aplicación debía tener las siguientes funcionalidades:

- Capacidad para entrenar y guardar modelos de segmentación.
- Posibilidad de cargar datasets etiquetados.
- Posibilidad de modificar parámetros de entrenamiento de forma sencilla.
- Visualización de información sobre el entrenamiento para facilitar la validación de los modelos.

En los siguientes apartados se procederá a explicar el funcionamiento y justificación de las distintas partes de la aplicación.

3.3.1- Instalación e importación de librerías.

La primera parte del código de la aplicación es la de instalación e importación de librerías, al ejecutar esta parte del código, se descargarán, instalaran e importaran las librerías necesarias para el funcionamiento del código posterior. A continuación, se muestra el código correspondiente:

```
#@title Instalar librerias necesarias
!pip install nibabel
!pip install classification-models-3D
!pip install efficientnet-3D
!pip install segmentation-models-3D
!pip install h5py==2.10.0
!pip install tensorflow==1.15.0
!pip install keras==2.3.1
!pip install --upgrade --no-cache-dir gdown
```

La primera librería instalada es la librería nibabel[17], es una librería que permite trabajar con diversos formatos de imágenes médicas, en este proyecto nos servirá para leer los archivos formato NIFTI que contienen las etiquetas del dataset.

Las librería segmentation-models-3D[18] es la siguiente librería importante a instalar, las dos librerías que se instalan previamente son necesarias para su funcionamiento, esta librería contiene diversos modelos para realizar segmentación de secuencias tridimensionales de imágenes, esta librería se utilizará posteriormente para construir el modelo de la red neuronal.

A continuación, se instalan las librerías h5py, tensorflow[19] y keras[20], para estas librerías es necesario seleccionar una versión concreta para que la librería de segmentation-models-3D funcione correctamente, estas librerías son las que nos permitirán trabajar con modelos de redes neuronales.

La última instalación se realiza para evitar un problema a la hora de descargar el dataset etiquetado, si no se realiza este paso, se produce un error con los permisos de descarga al intentar descargar el archivo.

```
#@title Importar librerias necesarias
import tensorflow as tf
import os
from skimage import io
from skimage.transform import resize
from matplotlib import pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from os import path
import cv2
import keras
import segmentation_models_3D as sm
import glob
import shutil
import nibabel as nib
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split

print(tf.__version__)
print(keras.__version__)

device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

Una vez instaladas las librerías, se procede a importar dichas librerías, además de algunas otras que ya se encuentran instaladas de forma predeterminada en los equipos de Google, en esta parte del código también se muestra por pantalla las versiones importadas de tensorflow y keras y se comprueba que existe una GPU disponible para el procesamiento.

3.3.2- Proceso de carga del dataset etiquetado.

Para entrenar los modelos de redes neuronales necesitamos un cargar un dataset que contenga las secuencias de imágenes con sus respectivas etiquetas, en esta parte del programa se descargará el dataset que se etiquetó previamente de la forma descrita en el apartado anterior. En la interfaz gráfica se permite seleccionar si se desea utilizar el dataset con los nervios ópticos etiquetados o el dataset en el que sólo se encuentran etiquetados las arterias carótidas.

```
Dataset_con_nervios_opticos = False #@param {type:"boolean"}
!gdown --id "1C4_Wy_r0O1-42KFMi53mzDGrbFsbeko6" #dataset con
nervios opt
!gdown --id "16BrOmbOKoyP9P19UGdpvwCoVKs-zWfn0" #dataset sin
nervios opt

if(Dataset_con_nervios_opticos):
  os.mkdir("dataset1")
  shutil.unpack_archive("dataset1.zip","dataset1/")
else:
  os.mkdir("dataset")
  shutil.unpack_archive("dataset.zip","dataset/")
```

En esta parte del código se descarga desde el almacenamiento en la nube un archivo comprimido que corresponde al dataset etiquetado previamente, una vez descargado se descomprime en un directorio llamado dataset que se encuentra dentro del espacio de almacenamiento del computador de Google. Podemos visualizar la carpeta del dataset con su contenido si hacemos clic en la pestaña archivos, (icono en forma de carpeta), desde esta pestaña también podemos descargar el dataset a nuestro ordenador personal haciendo clic derecho sobre el archivo .zip del dataset.

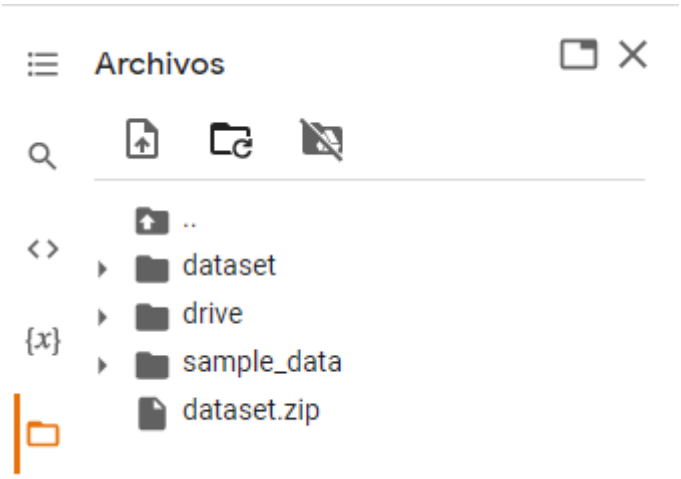

Figura 38. Carpeta dataset generada en el espacio de almacenamiento del computador de Google.

Fuente: elaboración propia.

```
#@title Cargar imágenes y etiquetas del dataset
model_img_size = 128 #@param {type:"integer"}
image_final_size = 256 #@param {type:"integer"}
MRI_num_layers = 64 #@param {type:"integer"}
labeled_dataset_path = "dataset/" #@param {type:"string"}

#---------------Cargar imágenes y etiquetas del dataset----------
-------------------
size = model_img_size
final_size = image_final_size
layers = MRI_num_layers
labels =
['carotida_der','carotida_izq','nervio_opt_der','nervio_opt_izq',
'background']
dataset_path = labeled_dataset_path
```

Una vez descargado el dataset, procedemos a cargar las imágenes y etiquetas en la memoria del ordenador para utilizarlas posteriormente, esta parte del programa está integrada en una interfaz gráfica sencilla desde la que introducimos ciertos parámetros que se tendrán en cuenta durante la carga de las imágenes y etiquetas.

Figura 39. Interfaz gráfica del apartado "cargar imágenes y etiquetas del dataset" de la aplicación de entrenamiento de modelos.

Fuente: elaboración propia.

El primer parámetro que introducimos, (*image_final_size*) corresponde al tamaño en píxeles, (altura y anchura) que tendrán las imágenes una vez cargadas, el tamaño original de las imágenes del dataset es de 176 píxeles de ancho por 208 de alto, pero debido a requerimientos de la red neuronal es necesario reescalar las imágenes para que tengan la misma anchura y altura, también es necesario que el tamaño sea divisible entre 8.

El parámetro *model_img_size* corresponde al tamaño de entrada en píxeles para la imagen que se introducirá en el modelo, este parámetro se utiliza para hacer un recorte cuadrado dentro de la imagen original, de tal forma que quede una imagen más reducida de la zona de búsqueda de las arterias carótidas y los nervios ópticos, esto se hace para aprovechar mejor los recursos de memoria de la tarjeta gráfica, ya que cuanto más grande sea la imagen introducida en el modelo de red neuronal, más memoria de la tarjeta gráfica consumirá.

El parámetro *MRI_num_layers*, tiene la función de indicar cuantas imágenes de la secuencia original deseamos cargar, las secuencias originales del dataset están compuestas por 176 imágenes, para este proyecto no necesitamos utilizar todas las imágenes, por lo tanto, introduciendo este parámetro, se pueden cargar por ejemplo las 64 primeras imágenes de la secuencia original y así ahorrar espacio en la memoria GPU.

El último parámetro introducido, (*labeled_dataset_path*) sirve para seleccionar el directorio en el que se encuentra el dataset etiquetado, en el caso de que se desee utilizar un nuevo dataset, este se deberá subir primero al espacio de almacenamiento del ordenador asignado en la sesión o al espacio de almacenamiento personal en la nube de Google Drive; una vez subido se introducirá la ruta del directorio en la caja de texto correspondiente para asignar este parámetro.

```
dir = glob.glob(dataset_path+"*/")
n_img = len(dir)
input_img = np.zeros((n_img,layers,size,size))
input_mask = np.zeros((n_img,layers,size,size))
j=0
for item in dir:
  vol_path = item+'/volume.tif'

  if os.path.isfile(vol_path):
    image = io.imread(vol_path)
    mask = nib.load(item+'Segmentation.nii').get_fdata()
    for k in range (layers):
      img_aux =image[:,:,k]
      mask_aux = mask[:,:,k]
      mask_aux = cv2.transpose(mask_aux)
      img_res = cv2.resize(img_aux,(final_size,final_size))
      mask_res =
cv2.resize(mask_aux,(final_size,final_size),interpolation =
cv2.INTER_NEAREST)

      col1= (image_final_size/2)-(model_img_size/2)
      row1= (image_final_size/2)-(model_img_size/2)
      col2= (image_final_size/2)+(model_img_size/2)
      row2= (image_final_size/2)+(model_img_size/2)
      np.int32(col1)
      np.int32(row1)
      np.int32(col2)
      np.int32(row2)
      input_img[j,k,:,:] =
img_res[np.int32(col1):np.int32(col2),np.int32(row1):np.int32(row
2)]
      input_mask[j,k,:,:] =
mask_res[np.int32(col1):np.int32(col2),np.int32(row1):np.int32(ro
w2)]
    j=j+1
```

Una vez que se han introducido los parámetros, el programa obtiene una lista de los pacientes del dataset y ejecuta un bucle en el que carga paciente a paciente las secuencias de imágenes y las etiquetas en tensores o arrays multidimensionales.

El proceso de carga es el siguiente:

1. Carga de la secuencia de imágenes y etiquetas en un tensor tridimensional: se cargan en un tensor en el que las dos primeras dimensiones contienen los píxeles de la imagen y la tercera dimensión corresponde al número de la imagen en la secuencia.
2. Procesamiento de las imágenes: se modifica la orientación y el tamaño de las imágenes y etiquetas para ajustarlas a los parámetros seleccionados previamente, este proceso se hace para cada imagen dentro de la secuencia de cada paciente. Una consideración importante es que a la hora de reescalar el tamaño de las etiquetas, se debe seleccionar el modo de interpolación de píxel más cercano, para que los nuevos píxeles que se generen mantengan el mismo valor que los píxeles originales, si seleccionamos otro tipo de interpolación, los nuevos píxeles podrían tener un valor distinto y las etiquetas ya no nos servirían.
3. Carga en tensores: una vez procesadas, las imágenes y etiquetas se almacenan en tensores de 4 dimensiones, de tal forma que la primera dimensión corresponde al número de paciente, la segunda dimensión corresponde al número de secuencia y las dos últimas dimensiones contienen los píxeles de las imágenes; esto se hace para cumplir con el formato de entrada necesario del modelo de red neuronal.

```
#@title Visualizar imágenes y etiquetas del dataset { run:"auto"}
num_paciente = 0 #@param {type:"integer"}
slice = 0 #@param {type:"slider", min:0, max:63, step:1}

num_img = num_paciente
# slice = 20

plt.figure(figsize =(12,8))
plt.subplot(231)
plt.title('Imagen')
plt.imshow(input_img[num_img,slice,:,:],cmap='gray')
plt.subplot(232)
plt.title('Etiquetas')
plt.imshow(input_mask[num_img,slice,:,:],cmap='gray')
```

La siguiente parte del código permite visualizar las imágenes cargadas, dispone de una interfaz gráfica con una caja de texto en la que se puede seleccionar el número de paciente y un control deslizable que permite navegar entre las imágenes y etiquetas de la secuencia, de esta forma podemos comprobar que el dataset se ha cargado correctamente.

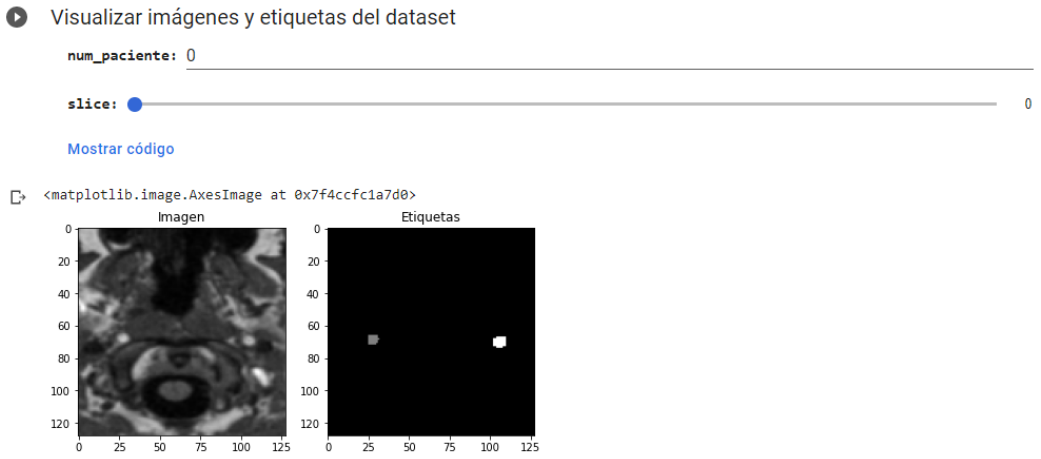

Figura 40. Interfaz gráfica del apartado "Visualizar imágenes y etiquetas del dataset" de la aplicación de entrenamiento de modelos.

Fuente: elaboración propia.

```
#@title Creación de datasets entrenamiento y validación
porcentaje_validacion = 10 #@param {type:"number"}
#------------------------------
nervios_opticos = Dataset_con_nervios_opticos
test_size = porcentaje_validacion/100

train_img = np.stack((input_img,)*3,axis=-1)
train_mask =  np.expand_dims(input_mask, axis=4)
train_mask_cat  = to_categorical(train_mask-1, num_classes=5)
if nervios_opticos==False:
  train_mask_cat = np.delete(train_mask_cat,3,4)
  train_mask_cat = np.delete(train_mask_cat,2,4)
  labels = ['carotida_der','carotida_izq','background']

X_train, X_test, y_train, y_test = train_test_split(train_img,
train_mask_cat, test_size = test_size, random_state = 0)
```

La última parte del proceso de carga es la división del dataset en dos nuevos datasets, uno para el entrenamiento y otro para la validación. En esta parte del código también se vuelven a ajustar las dimensiones de los tensores del dataset para que coincidan con la forma requerida por la red. Desde la interfaz gráfica se permite indicar si se desea utilizar las etiquetas de los nervios ópticos, en el caso de no utilizarse estas etiquetas, la red neuronal entrenaría únicamente con las etiquetas de las arterias carótidas.

En la interfaz gráfica también se permite indicar el porcentaje de secuencias de imágenes y etiquetas que se desea emplear para validar el modelo de red neuronal.

3.3.3- Definición de la red y preprocesamiento del dataset.

En esta parte del código se define la arquitectura de red para el modelo que vamos a entrenar, la interfaz gráfica le permite seleccionar al usuario varios parámetros:

- La red preentrenada para usar como etapa codificadora de la 3D U-Net.
- La opción de utilizar pesos preentrenados con el dataset Imagenet.
- La tasa de aprendizaje o learning rate para el entrenamiento.
- La opción para que el programa muestre por pantalla un esquema de la estructura resultante.

```
#@title Definición de los parámetros del modelo y preprocesado
dataset
red_preentrenada = "resnet50" #@param ["resnet50", "vgg16",
"efficientnetb7"]
Pesos_preentrenados = True #@param {type:"boolean"}
learning_rate = 0.0001 #@param {type:"number"}
mostrar_estructura_red = True #@param {type:"boolean"}
#-----------------------------------------
if Pesos_preentrenados: encoder_weights = 'imagenet'
else:encoder_weights =None
BACKBONE = red_preentrenada
activation = 'softmax'
n_classes = len(labels)
channels=3
LR = learning_rate
optim = tf.keras.optimizers.Adam(LR)

total_loss = sm.losses.categorical_focal_dice_loss
metrics = [sm.metrics.IOUScore(threshold=0.5)]

preprocess_input = sm.get_preprocessing(BACKBONE)
X_train_prep = preprocess_input(X_train)
X_test_prep = preprocess_input(X_test)

model = sm.Unet(BACKBONE, classes=n_classes, input_shape=(layers,
size, size, channels),
encoder_weights=encoder_weights,activation='softmax')

model.compile(optimizer =
tf.keras.optimizers.Adam(learning_rate=LR), loss=total_loss,
metrics=metrics)
if mostrar_estructura_red: print(model.summary())
```

Lo primero que se hace en este apartado del código es establecer algunos parámetros como el tipo de red preentrenada a utilizar, el tipo de función de activación que se utilizará, el número de clases, el número de canales de las imágenes de entrada, la tasa de aprendizaje, el optimizador que se utilizará para el aprendizaje, las medidas o métricas de aprendizaje que se mostrarán, y la función de error que se utilizará para calcular como de correctas son las predicciones durante el entrenamiento.

El preprocesamiento de las imágenes dependerá de la red preentrenada o "backbone" seleccionada, ya que cada red tiene un preprocesamiento específico que debe realizarse para que funcione correctamente. Utilizando la función sm.get_preprocessing se pueden obtener los parámetros de preprocesamiento necesarios para cada red preentrenada, para aplicar el preprocesamiento, se utiliza la función preprocess_input con las imágenes de entrenamiento y validación.

Por último, se crea la estructura de la red mediante la función sm.Unet que genera una red de arquitectura U-Net con los parámetros establecidos previamente, posteriormente se compila la red, dejándola lista para su entrenamiento.

Algunos de los parámetros establecidos son los siguientes:

- Optimizador Adam[21]: es un tipo de método de descenso de gradiente estocástico, se ha decidido utilizar este método porque generalmente proporciona mejores resultados que otros métodos.
- Función de error Focal Dice Loss[22]: la función de error Dice Loss es una función de error inspirada en el coeficiente de Sørensen-Dice o Dice coefficient ampliamente utilizada en redes de segmentación de imágenes médicas, ya que permite abordar el problema de desbalance de datos entre las etiquetas y el fondo de la imagen, (el fondo suele ocupar una superficie mayor que las etiquetas en las imágenes). Además en este caso utilizamos la variante focal, que es capaz de reducir el desbalance entre muestras difíciles y fáciles, reduciendo la influencia de las muestras fáciles que suelen ser más numerosas en los datasets.
- Metrics IOUScore: estas son medidas que se mostrarán durante el entrenamiento y que nos permitirán evaluar el modelo, el IOUScore, (Intersection Over Union score o índice de Jaccard)[23] es una medida que en segmentación se interpreta como el porcentaje de píxeles de la máscara de salida que coinciden con los de la máscara de entrada.

3.3.4- Entrenamiento de la red.

Esta parte del código es la encargada de entrenar el modelo creado anteriormente, durante este proceso se irán ajustando los pesos de las neuronas de la red hasta que se haya completado el número de iteraciones seleccionado por el usuario.

```
#@title Entrenar modelo
batch_size = 4 #@param {type:"integer"}
epochs =  200#@param {type:"integer"}
ruta_guardado_modelo = "modelo.pb" #@param {type:"string"}
#@markdown (Conviene guardar el modelo dentro de una carpeta
drive)
history=model.fit(X_train_prep,
          y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test_prep,
y_test),callbacks=[tf.keras.callbacks.ReduceLROnPlateau(monitor='
val_loss', factor=0.1,
                              patience=5)])

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

acc = history.history['iou_score']
val_acc = history.history['val_iou_score']

plt.plot(epochs, acc, 'y', label='Training IOU')
plt.plot(epochs, val_acc, 'r', label='Validation IOU')
plt.title('Training and validation IOU')
plt.xlabel('Epochs')
plt.ylabel('IOU')
plt.legend()
plt.show()

model.save(ruta_guardado_modelo)
```

El modelo generado anteriormente dispone de una función que podemos llamar para comenzar su entrenamiento, esta es la función fit. La función fit tiene varios parámetros de entrada:

- Input data: en este parámetro se introducen los datos de entrada para el entrenamiento, en este caso corresponde al tensor de imágenes de entrenamiento generado anteriormente.
- Target: en este parámetro se introducen los datos objetivo, es decir, los datos a los que se debe aproximar la salida de la red neuronal, en este caso corresponde al tensor de etiquetas de entrenamiento generado anteriormente.
- batch size, (tamaño de lote): corresponde al número de muestras que se pasarán por la red antes de actualizar los pesos de la red neuronal, la influencia de este parámetro en el entrenamiento se verá más adelante
- epoch, (épocas): es un parámetro que indica el número de veces que se introducirá el dataset de entrenamiento completo en la red neuronal.
- Verbose: este parámetro se utiliza para indicar cada cuantas épocas se va a mostrar por pantalla el estado del entrenamiento y los valores de las medidas del error.
- Validation_data: corresponde a la entrada de datos de validación, en este caso se introduce el tensor de imágenes de validación y el tensor de etiquetas de validación.
- Callbacks: es un parámetro en el que podemos indicar un conjunto de funciones o callbacks a las que llamar durante el entrenamiento para realizar alguna tarea como reducir el learning rate o guardar copias de seguridad del modelo.

La ejecución de la función fit requiere de bastante tiempo, dependiendo de los parámetros introducidos puede tardar desde unos minutos a varias horas.

Una vez ejecutada la función fit, las siguientes líneas de código sirven para mostrar por pantalla la representación gráfica de los datos del entrenamiento, en la primera gráfica se representan los valores del error en cada época para los datasets de entrenamiento y validación, y en la segunda se muestra el valor de la medida del IOU, (Intersection Over Union) para los dos datasets.

Por último, al final del segmento de código se guarda el modelo entrenado en la ruta especificada por el usuario a través de la interfaz gráfica, de forma predeterminada, el modelo se almacena en el espacio de almacenamiento del ordenador de Google en el que se está ejecutando el código, este almacenamiento se borra cada vez que se abandona la sesión de ejecución de código, por lo que sólo se podrá utilizar este modelo hasta que finalice la sesión. En el caso de querer guardar el modelo para utilizarlo posteriormente, podemos descargar el archivo del modelo generado en nuestro ordenador personal haciendo clic derecho sobre el archivo que se encuentra en la pestaña archivos, o bien podemos introducir una ruta de guardado dentro del espacio de almacenamiento personal de Google Drive. Esta forma es más conveniente, ya que nos permitirá cargar fácilmente el modelo para usarlo en otras aplicaciones desarrolladas en el entorno de Google Colab.

3.3.5- Visualización de inferencia con datos de validación.

La última parte de la aplicación de entrenamiento permite probar el funcionamiento del modelo con el dataset de validación, al ejecutar esta parte del código, el usuario puede visualizar las etiquetas generadas por la red neuronal y comparar su aspecto con el de las originales.

```
#@title Probar modelo imagen validacion
num_img_test = 2 #@param {type:"number"}
test_img_number = num_img_test
test_img = X_test[test_img_number]
ground_truth=y_test[test_img_number]
test_img_input=np.expand_dims(test_img, 0)
test_img_input1 = preprocess_input(test_img_input)

test_pred1 = model.predict(test_img_input1)
test_prediction1 = np.argmax(test_pred1, axis=4)[0,:,:,:]

ground_truth_argmax = np.argmax(ground_truth, axis=3)

#@title Visualizar inferencia imagen validacion { run: "auto" }
slice = 20 #@param {type:"slider", min:0, max:63, step:1}

plt.figure(figsize=(12, 8))
plt.subplot(231)
plt.title('Testing Image')
plt.imshow(test_img[slice,:,:,0], cmap='gray')
plt.subplot(232)
plt.title('Testing Label')
plt.imshow(ground_truth_argmax[slice,:,:],cmap='gray')
plt.subplot(233)
plt.title('Prediction on test image')
plt.imshow(test_prediction1[slice,:,:],cmap='gray')
plt.show()
```

El funcionamiento de esta parte del código es bastante similar a algunas de las partes explicadas anteriormente. Primero se carga la imagen del dataset de validación seleccionada por el usuario, se preprocesa y se introduce en la red neuronal para obtener el resultado. Una vez obtenido, el usuario puede utilizar el control deslizante de la interfaz gráfica para visualizar las imágenes y etiquetas de cada capa. Al ejecutar este código, se muestran en tres figuras, la imagen original, las etiquetas originales y por último las etiquetas generadas por la red neuronal.

3.4- Aplicación de inferencia de modelos.

La aplicación de inferencia de modelos es la segunda aplicación desarrollada, esta aplicación tiene la función principal de servir como entorno de prueba para los modelos entrenados por la aplicación de entrenamiento.

Se consideró que la aplicación debía tener las siguientes funcionalidades:

- Capacidad para cargar modelos entrenados en la aplicación de entrenamiento.
- Visualización sencilla de los resultados de inferencia
- Posibilidad de descargar directamente imágenes nuevas del dataset OASIS para probar la inferencia.

En los siguientes apartados se procederá a explicar el funcionamiento y justificación de las distintas partes de la aplicación.

3.4.1- Instalación e importación de librerías.

Al igual que en la anterior aplicación, hay algunas librerías que necesitan ser instaladas e importadas para que todo funcione correctamente.

```
#@title Descarga de librerías necesarias
!pip install nibabel
!pip install classification-models-3D
!pip install efficientnet-3D
!pip install segmentation-models-3D
!pip install h5py==2.10.0
!pip install tensorflow==1.15.0
!pip install keras==2.3.1
!pip install nilearn
!pip install --upgrade --no-cache-dir gdown
```

Todas las librerías a instalar excepto la última, son las mismas que se instalan para la aplicación de entrenamiento. La última librería a instalar es la librería nilearn, esta es una librería que contiene funciones útiles a la hora de trabajar con imágenes de resonancia magnética del cerebro, en esta aplicación se utilizará más adelante para extraer la secuencia de imágenes de resonancia magnética de los archivos .img del dataset OASIS.

```
#@title Importar librerías necesarias

import tensorflow as tf
import os
from skimage import io
from skimage.transform import resize
from matplotlib import pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
import os.path
from os import path
import cv2
import keras
import segmentation_models_3D as sm
import glob
import time
import nibabel as nib
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import nilearn
from tifffile import imsave
import shutil
from nilearn.image import get_data

print(tf.__version__)
print(keras.__version__)

device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

Al igual que en la aplicación anterior se importan las librerías instaladas y se muestra por pantalla las versiones de Tensorflow y Keras utilizadas y el mensaje de comprobación de que existe una tarjeta gráfica disponible.

3.4.2- Descarga del modelo entrenado

Para realizar la inferencia en esta aplicación es necesario descargar un modelo entrenado previamente en la aplicación de entrenamiento, el código que se muestra a continuación sirve para descargar el archivo de un modelo ya entrenado que se encuentra guardado en un espacio de almacenamiento de Google Drive. Una vez descargado, el modelo se carga en la memoria GPU utilizando la función load_model importada de tensorflow.keras, es importante establecer el valor del parámetro compile de esta función como False, debido a que no es necesario volver a compilar el modelo para realizar inferencias.

```
#@title Descarga el modelo entrenado
Modelo_seleccionado = "Modelo_ResNet50_ICAs" #@param
["Modelo_VGG16", "Modelo_ResNet50", "Modelo_ResNet50_ICAs"]
#Descarga de los modelos preentrenados desde Google Drive

!gdown --id "1-1JtiWQaQVzjaJq1m9CrQiJuvd_L2NCZ"
!gdown --id "138GvGOZ-zevKYjLVLdwpHQEeUKUvWcnT"
!gdown --id "1QAAhInt9SACit4tZIylKJrjuY7_C2Xhn"
from tensorflow.keras.models import load_model
labels
=['carotida_der','carotida_izq','nervio_opt_der','nervio_opt_izq'
,'background']
#Carga del modelo preentrenado seleccionado en memoria
if(Modelo_seleccionado=='Modelo_VGG16'):
model=load_model('modelo_ICAyNO_vgg16_100Epoch_BS2.pb',compile=Fa
lse)
  size = 128
  final_size = 128
  n_labels=5
  BACKBONE = 'vgg16'
elif(Modelo_seleccionado=='Modelo_ResNet50'):
model=load_model('modelo_ICAyNO_resnet50_100Epoch_BS2.pb',compile
=False)
  size = 128
  final_size = 128
  n_labels=5
  BACKBONE = 'resnet50'
else:
  model=load_model('modelo_resnet-
50_200Epoch_BS4_1.pb',compile=False)
  size = 128
  final_size = 256
  n_labels=3
  BACKBONE = 'resnet50'
```

Este apartado de código dispone de un selector en la interfaz gráfica que permite seleccionar entre tres modelos preentrenados distintos. Dependiendo del modelo seleccionado, se ajustan los parámetros de tamaño de imagen, número de etiquetas y tipo de red utilizada en la etapa decodificadora para cada caso.

3.4.3- Descarga de nuevas imágenes del dataset OASIS.

Para probar la inferencia se podrían cargar a mano nuevas secuencias de imágenes de resonancia magnética, pero en esta aplicación se decidió simplificar este proceso descargando directamente los archivos de la página original del dataset.

```
#@title Descarga de discos dataset OASIS1 Brains
disc = "disc3" #@param ["disc1", "disc2", "disc3", "disc4",
"disc5", "disc6", "disc7", "disc8", "disc9", "disc10", "disc11",
"disc12"]
generar_zip = False #@param {type:"boolean"}
loaded = []
#---Exportación del dataset OASIS a imagenes tif----
nom_disc='0'
#Selección del disco del dataset, el valor debe estar entre 1 y
12
if(disc=="disc1"):
  !wget https://download.nrg.wustl.edu/data/oasis_cross-
sectional_disc1.tar.gz
  nom_disc='1'
elif(disc=="disc2"):
  !wget https://download.nrg.wustl.edu/data/oasis_cross-
sectional_disc2.tar.gz
  nom_disc='2'
elif(disc=="disc3"):
  !wget https://download.nrg.wustl.edu/data/oasis_cross-
sectional_disc3.tar.gz
  nom_disc='3'
elif(disc=="disc4"):
  !wget https://download.nrg.wustl.edu/data/oasis_cross-
sectional_disc4.tar.gz
  nom_disc='4'
elif(disc=="disc5"):
  !wget https://download.nrg.wustl.edu/data/oasis_cross-
sectional_disc5.tar.gz
  nom_disc='5'
elif(disc=="disc6"):
  !wget https://download.nrg.wustl.edu/data/oasis_cross-
sectional_disc6.tar.gz
  nom_disc='6'
```

```
elif(disc=="disc7"):
  !wget https://download.nrg.wustl.edu/data/oasis_cross-
sectional_disc7.tar.gz
  nom_disc='7'
elif(disc=="disc8"):
  !wget https://download.nrg.wustl.edu/data/oasis_cross-
sectional_disc8.tar.gz
  nom_disc='8'
elif(disc=="disc9"):
  !wget https://download.nrg.wustl.edu/data/oasis_cross-
sectional_disc9.tar.gz
  nom_disc='9'
elif(disc=="disc10"):
  !wget https://download.nrg.wustl.edu/data/oasis_cross-
sectional_disc10.tar.gz
  nom_disc='10'
elif(disc=="disc11"):
  !wget https://download.nrg.wustl.edu/data/oasis_cross-
sectional_disc11.tar.gz
  nom_disc='11'
elif(disc=="disc12"):
  !wget https://download.nrg.wustl.edu/data/oasis_cross-
sectional_disc12.tar.gz
  nom_disc='12'


shutil.unpack_archive('/content/oasis_cross-
sectional_disc'+nom_disc+'.tar.gz',"/content/")
```

En la primera parte del código, se descarga el archivo comprimido de uno de los discos que compone el dataset OASIS, el disco a descargar será el que haya seleccionado el usuario a través de la interfaz gráfica. La descarga del archivo se realiza mediante el comando "!wget", con el enlace de descarga de la página a modo de argumento, de esta forma, la aplicación descargará el archivo comprimido en el espacio de almacenamiento del ordenador en el que se esté ejecutando.

Una vez descargado, el archivo se descomprime, generando una carpeta que contiene el disco del dataset con todos los archivos. Un dato importante a tener en cuenta es que los modelos entrenados en este proyecto utilizan las imágenes del disco 1, por lo que si se prueba la inferencia con estas imágenes los resultados serán probablemente mejores que los obtenidos con las imágenes del resto de discos, para probar correctamente el modelo, lo adecuado es utilizar las imágenes del resto de discos.

```
      #Directorio de guardado del dataset
export_dir='/content/Export_OASIS_Disc'+nom_disc+'/'

#Comprobacion de que no existe el directorio de guardado
if(os.path.isdir(export_dir)==False):
  os.mkdir(export_dir)

dataset_dirs = glob.glob("/content/disc"+nom_disc+"/*/")
n_samples = len(dataset_dirs)
i=1
#Carga y adaptación de las imágenes al formato adecuado
img_export = np.zeros((208,176,176))
for dir in dataset_dirs:
  file_dir = dir +"/PROCESSED/MPRAGE/T88_111/"
  patient = dir.split(os.sep)
  file = glob.glob(file_dir+'*_anon_111_t88_gfc.img')
  if os.path.isfile(file[0]):
    data = get_data(file[0])
    img=data
    img = np.rot90(img)
    img_export = img[:,:,:,0]
    exp_dir=export_dir+'/'+patient[-2]


    if(os.path.isdir(exp_dir)==False):
      os.mkdir(exp_dir)


    exp_dir= exp_dir+'/volume.tif'
    imsave(exp_dir,img_export)


loaded.append(nom_disc)
if generar_zip:
  shutil.make_archive('/content/Export_OASIS_Dataset', 'zip',
export_dir)
```

El dataset descargado, contiene una gran cantidad de archivos con información de los pacientes y diversas secuencias de resonancia magnética, para esta aplicación, únicamente necesitamos obtener la secuencia de imágenes que contiene uno de los archivos .img del dataset.

Al ejecutar el código, se genera una nueva carpeta que contendrá las secuencias de imágenes de resonancia magnética de cada paciente con el mismo formato que tenían las imágenes de entrenamiento. El proceso seguido es el siguiente:

1. Se obtiene el nombre del archivo .img.
2. Se convierte el archivo .img en un tensor de imágenes utilizando la función get_data de la librería nilearn.
3. Se aplican las transformaciones necesarias al tensor de imágenes para que tenga el mismo formato que los tensores utilizados en la aplicación de entrenamiento.
4. Se crea un nuevo directorio con el nombre del paciente y se guarda el tensor de imágenes como archivo .tif en su interior.

Este proceso se realiza para todas las carpetas de pacientes del dataset, una vez finalizado, la nueva carpeta está lista para usarse. La carpeta generada se puede descargar para etiquetar las secuencias y generar nuevos datasets de entrenamiento, para esto, existe una opción que se puede marcar en la interfaz gráfica para generar un archivo zip de la nueva carpeta y así facilitar su descarga.

Para poder introducir las secuencias de imágenes a la red neuronal, es necesario cargarlas previamente en memoria con el formato adecuado a la red, al igual que se hacía en la aplicación de entrenamiento. Para cargar las imágenes se ejecuta el siguiente fragmento del código:

```
#@title Cargar secuencias de imágenes
nom_disc = "3" #@param ["1", "2", "3", "4", "5", "6", "7", "8",
"9", "10", "11", "12"]
export_dir='/content/Export_OASIS_Disc'+nom_disc+'/'
#---------------Cargar imágenes y etiquetas del dataset----------
layers = 64
test_path = export_dir
dir = glob.glob(test_path+"*/")
n_img = len(dir)
layers = 64
input_img = np.zeros((n_img,layers,size,size))
reduced_input = np.zeros((5,layers,size,size))
img_orig=np.zeros((n_img,layers,final_size,final_size))
j=0
for item in dir:
  vol_path = item+'volume.tif'

  if os.path.isfile(vol_path):

    image = io.imread(vol_path)

    for k in range (layers):
      img_aux =image[:,:,k]
      img_res = cv2.resize(img_aux,(final_size,final_size))
      col1= (final_size/2)-(size/2)
      row1= (final_size/2)-(size/2)
      col2= (final_size/2)+(size/2)
      row2= (final_size/2)+(size/2)

      input_img[j,k,:,:] =
img_res[np.int32(col1):np.int32(col2),np.int32(row1):np.int32(row
2)]
      img_orig[j,k,:,:] = img_res[:,:]

    j=j+1
if j==0:
  print("Dataset no descargado")
else:
  print("Num secuencias MRI: ",j)
```

El proceso de carga de las secuencias de imágenes es muy similar al de la aplicación de entrenamiento, se cargan las secuencias de imágenes en tensores tridimensionales y posteriormente se transforma estos tensores teniendo en cuenta los parámetros de tamaño establecidos previamente para que cumplan con el formato de entrada de la red neuronal.

3.4.4- Inferencia del modelo.

Esta parte del código es la encargada de realizar el proceso de inferencia con las secuencias de imágenes cargadas, proporcionando un tensor que contiene las etiquetas generadas por el modelo para cada una de las secuencias.

El primer paso del proceso consiste en preprocesar las secuencias cargadas de la forma que se preprocesaron las secuencias con las que se entrenó el modelo, si se realiza un preprocesamiento distinto de las secuencias, es posible que el modelo no funcione correctamente.

```
#@title Preprocesamiento de imágenes
test_img = np.stack((input_img,)*3,axis=-1)

preprocess_input = sm.get_preprocessing(BACKBONE)
X_test = preprocess_input(test_img)
```

Con las secuencias ya preprocesadas, se procede a realizar la inferencia de las secuencias cargadas de una en una y se van almacenando los resultados obtenidos en un nuevo tensor. Para realizar la inferencia, se llama a la función predict del modelo cargado y se introduce el dato a inferir. En este apartado de código también se calcula el tiempo promedio de inferencia por secuencia de imagen, una vez que se termina de inferir todos los datos, se muestra por pantalla el valor del tiempo promedio.

```
#@title Obtener predicciones
preds = np.zeros((j,layers,size,size,n_labels))
test = np.zeros((1,layers,size,size,3))
t_infer=0
for i in range(j):

  test[0,:,:,:,:] = X_test[i,:,:,:,:]
  t0= time.time()
  pred = model.predict(test)
  t_infer = t_infer+(time.time()-t0);

  preds[i,:,:,:,:]=pred


print("Tiempo promedio de inferencia: ",t_infer/j)
```

3.4.5- Visualización de resultados.

La última parte de la aplicación permite visualizar las etiquetas generadas en el proceso de inferencia, esta parte dispone de una interfaz gráfica desde la que el usuario puede seleccionar el número del paciente y la capa de la secuencia de imágenes que desea visualizar. Se puede observar un ejemplo de visualización en la Figura 41. Figura 41

```
#@title Visualizador de resultados { run: "auto" }
Slice = 30 #@param {type:"slider", min:0, max:63, step:1}
Paciente =  35#@param {type:"integer"}
image_n =Paciente
test_prediction1 = np.argmax(preds, axis=4)[image_n,:,:,:]
if(final_size==256):
  A = np.zeros((64,final_size,final_size))
  A.fill(2)
  A[:,64:192,64:192]=test_prediction1
else:
  A = np.zeros((64,final_size,final_size))
  A[:,:,:]=test_prediction1
plt.figure(figsize=(20, 20))
plt.subplot(231)
plt.title('Testing Image')
plt.imshow(img_orig[image_n,Slice,:,:], cmap='gray')
plt.subplot(232)
plt.title('Prediction on test image')
plt.imshow(A[Slice,:,:],cmap='gray')
plt.show()
print(dir[Paciente])
```

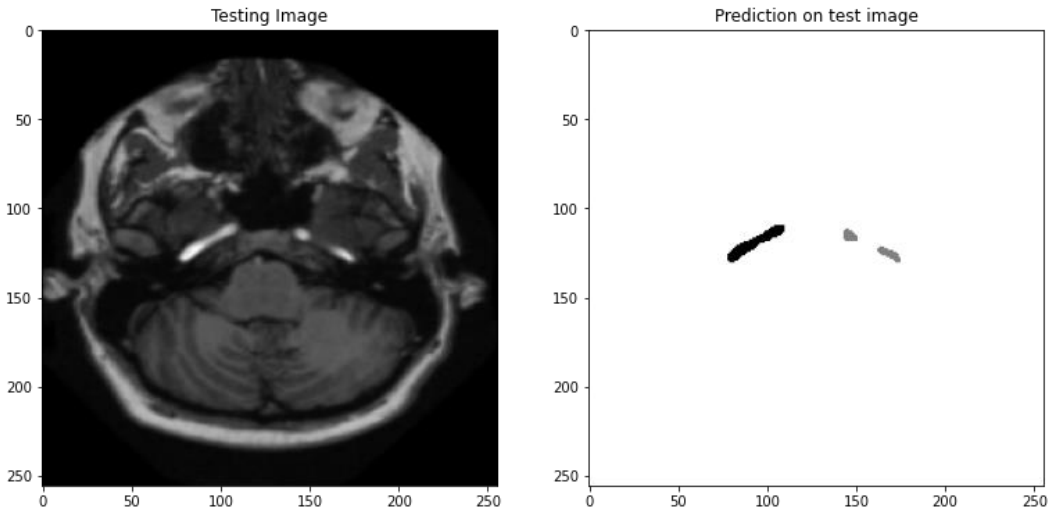

Figura 41. Ejemplo de visualización de resultados de inferencia.

En el lado derecho se muestra el resultado de la segmentación de las arterias carótidas, la arteria carótida derecha se pinta en negro y la arteria carótida derecha en gris.

Fuente: elaboración propia.

4- Resultados obtenidos

En este apartado del trabajo se mostrarán los resultados obtenidos con las aplicaciones desarrolladas. Primero se utilizará la aplicación de entrenamiento para generar los modelos de redes neuronales y posteriormente se probará su funcionamiento con la aplicación de inferencia.

4.1- Entrenamiento de modelos

Los primeros entrenamientos se realizaron utilizando los parámetros que se indican en la Figura 42 y en la Figura 43, el tamaño de las imágenes se tuvo que reducir a 128x128 para evitar que se produjera un problema de desbordamiento de la memoria RAM del computador de Google Colab. Otra consideración es que se utilizan 21 secuencias de imágenes para el entrenamiento y 3 para la validación.

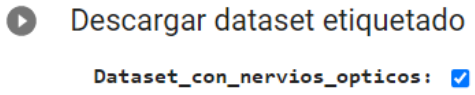

Figura 42. Selección de dataset con nervios ópticos en la interfaz gráfica de la aplicación de entrenamiento de modelos.

Fuente: elaboración propia.

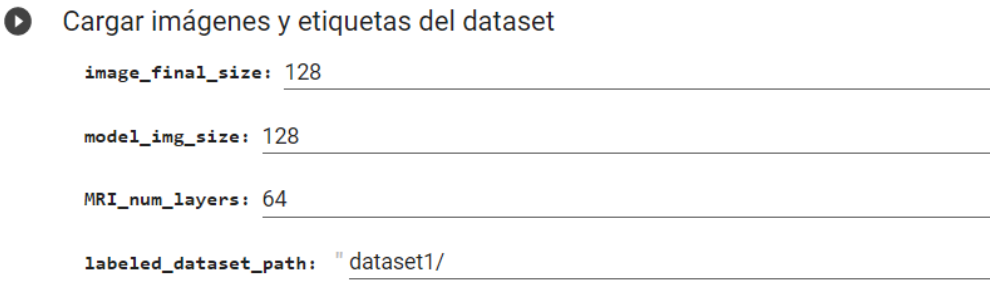

Figura 43. Ajuste de parámetros en la interfaz gráfica del apartado "Cargar imágenes y etiquetas del dataset" de la aplicación de entrenamiento de modelos para el entrenamiento con el dataset con nervios ópticos.

Fuente: elaboración propia.

Para el primer entrenamiento utilizamos la red preentrenada ResNet-50 con los pesos preentrenados de ImageNet, el resto de los parámetros como el learning rate y el batch size se ajustaron en entrenamientos de prueba previos comprobando que no se producía desbordamiento de memoria GPU ni inestabilidad en el entrenamiento. También se escogió un número de épocas suficientemente grande como para que en el entrenamiento se alcanzara un error mínimo. La introducción de estos parámetros se puede ver en la Figura 44 y en la Figura 45 .

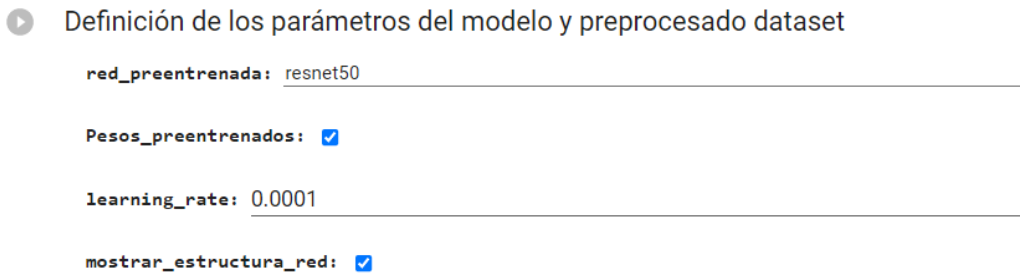

Figura 44. Ajuste de parámetros en la interfaz gráfica del apartado " Definición de los parámetros del modelo y preprocesado del dataset" en la aplicación de entrenamiento de modelos. Modelo con etapa decodificadora ResNet-50.

Fuente: elaboración propia.

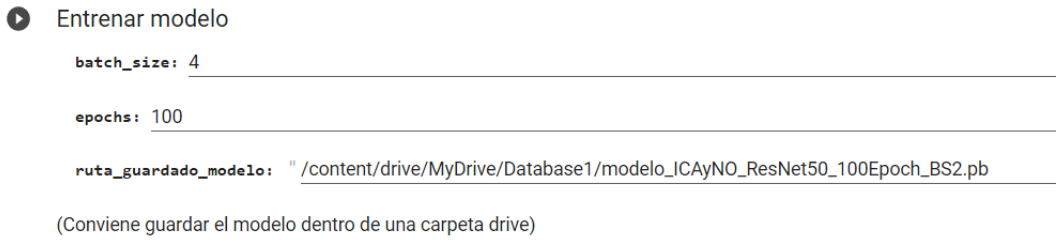

Figura 45. Ajuste de parámetros en la interfaz gráfica del apartado " Entrenar modelo" en la aplicación de entrenamiento de modelos.

Fuente: elaboración propia.

Con los ajustes descritos anteriormente, ejecutamos el proceso de entrenamiento, que tarda aproximadamente 54 minutos en completarse. En la Figura 46 podemos ver la reducción del valor del error de entrenamiento y el valor del error de validación. El error de entrenamiento alcanza el valor de 0.0832 en la época 100, el error de validación es algo mayor: 0.2512, esto quiere decir que el modelo se ha adaptado correctamente a los datos de entrenamiento, pero no es capaz de funcionar con tanta precisión con los datos de validación. En la Figura 47podemos visualizar el aumento de los valores del IOU, (Intersection Over Union) el IOU de entrenamiento alcanza el valor de 0.8801, mientras que el de validación solamente llega a 0.6381.

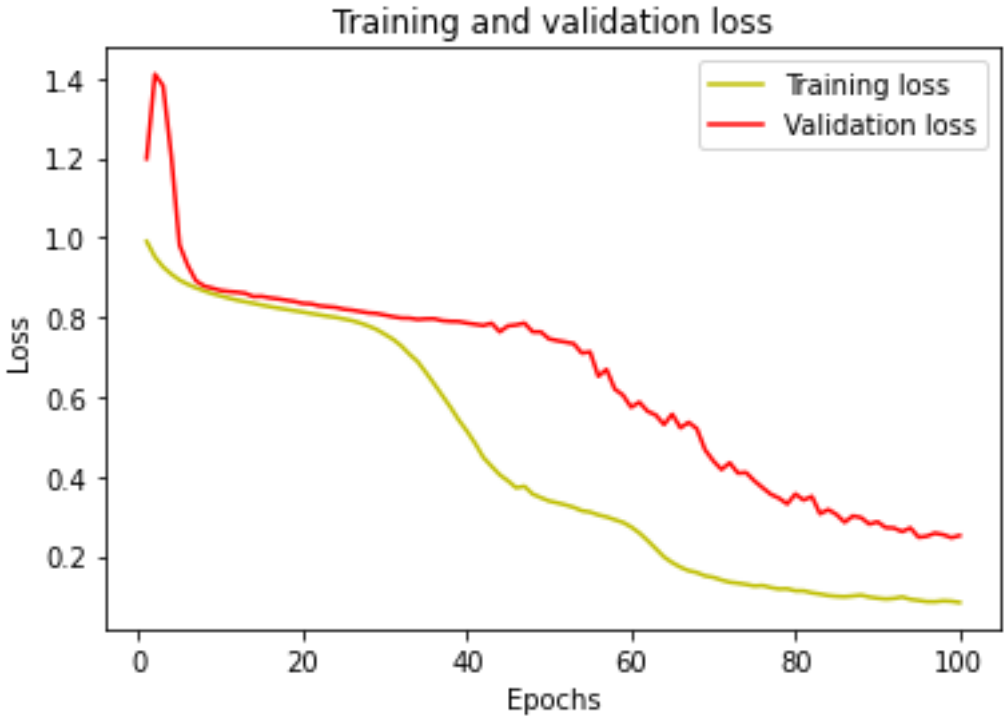

Figura 46. Representación gráfica del valor del error de entrenamiento y el error de validación por épocas. Modelo con etapa decodificadora ResNet-50

Fuente: elaboración propia.

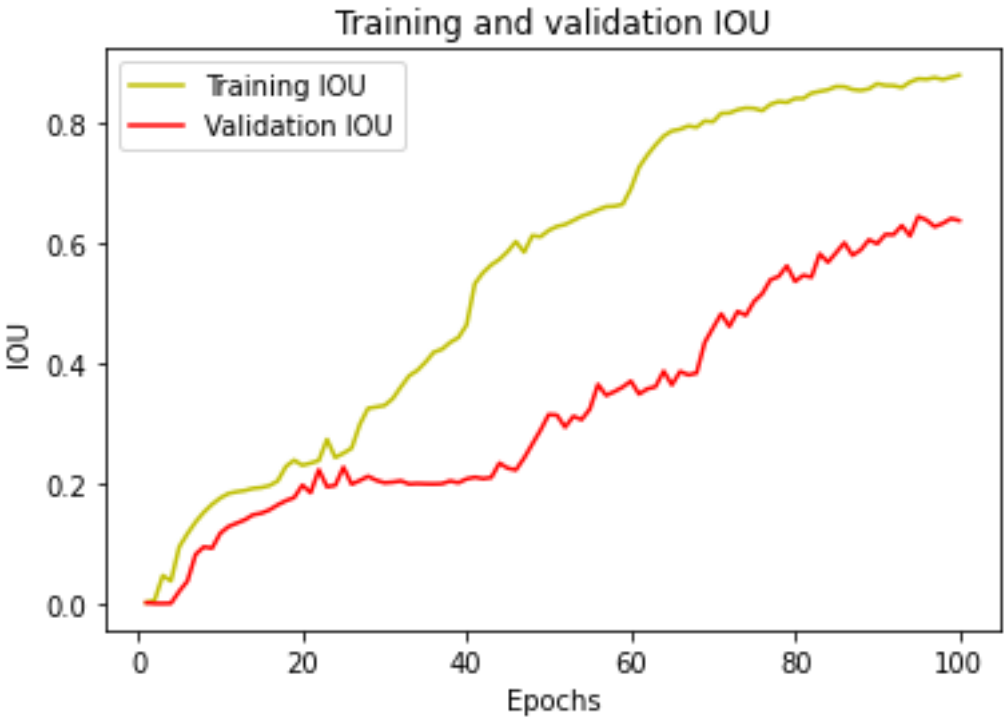

Figura 47. Representación gráfica de la medida de "IoU" de entrenamiento y validación por épocas. Modelo con etapa decodificadora ResNet-50.

Fuente: elaboración propia.

Para visualizar mejor los resultados del entrenamiento, realizamos una prueba de inferencia con las secuencias del dataset de validación, estos resultados se pueden ver en las figuras Figura 48, Figura 49 y Figura 50, En estos resultados podemos observar que la forma de las etiquetas generadas por la red neuronal es bastante similar a la de las etiquetadas manualmente, se diferencian en que algunas de las etiquetas generadas por la red cubren una superficie menor que las etiquetas originales.

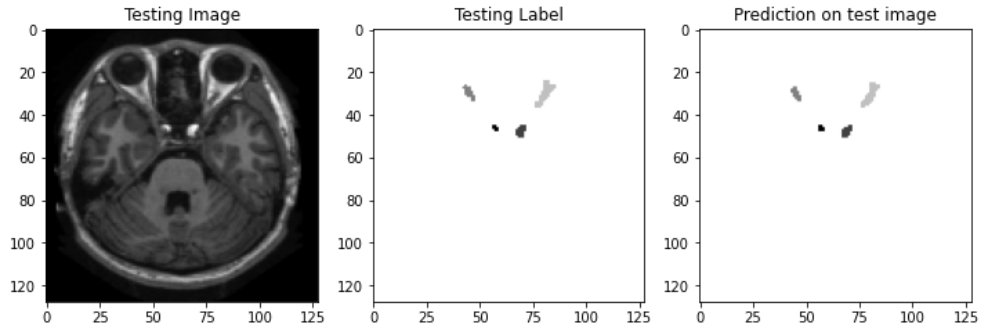

Figura 48. Resultado de validación para la primera secuencia de imágenes del dataset de validación. Modelo con etapa decodificadora ResNet-50.

Fuente: elaboración propia.

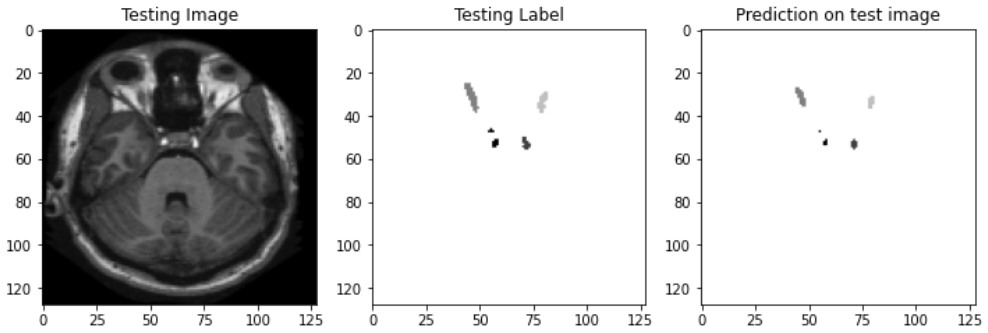

Figura 49. Resultado de validación para la segunda secuencia de imágenes del dataset de validación. Modelo con etapa decodificadora ResNet-50.

Fuente: elaboración propia.

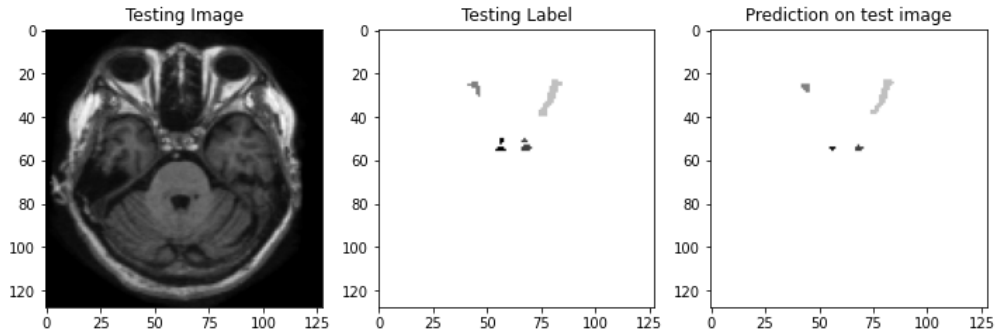

Figura 50. Resultado de validación para la segunda secuencia de imágenes del dataset de validación. Modelo con etapa decodificadora ResNet-50.

Fuente: elaboración propia.

La segunda prueba de entrenamiento que realizamos difiere de la primera en que se utiliza la red preentrenada VGG-16 como etapa decodificadora de la 3D-UNet en vez de ResNet-50, este cambio se muestra en la Figura 51.

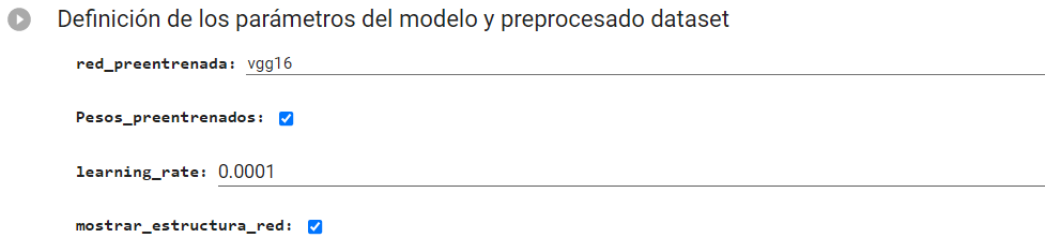

Figura 51. Ajuste de parámetros en la interfaz gráfica del apartado "Definición de los parámetros del modelo y preprocesado del dataset" en la aplicación de entrenamiento de modelos. Modelo con etapa decodificadora VGG-16

Fuente: elaboración propia.

Al utilizar VGG-16 como etapa decodificadora, una de primeras diferencias que podemos observar es que el tiempo de entrenamiento utilizando la misma tarjeta gráfica es mayor que el del caso anterior, para completar el proceso de entrenamiento fueron necesarias 2 horas aproximadamente. Este incremento en el tiempo de entrenamiento se debe a que VGG-16 no dispone de capas con conexiones residuales a diferencia de ResNet-50 que si las tiene.

Los resultados obtenidos utilizando VGG-16 como etapa decodificadora son peores que los obtenidos utilizando ResNet-50, en la Figura 52 podemos observar la disminución de los valores del error, el valor del error del entrenamiento sólo se reduce hasta 0.4960 en la época 100, y el del error de validación hasta 0.5061. En la Figura 53, también podemos ver que los valores del IOU son peores que los obtenidos en el entrenamiento anterior.

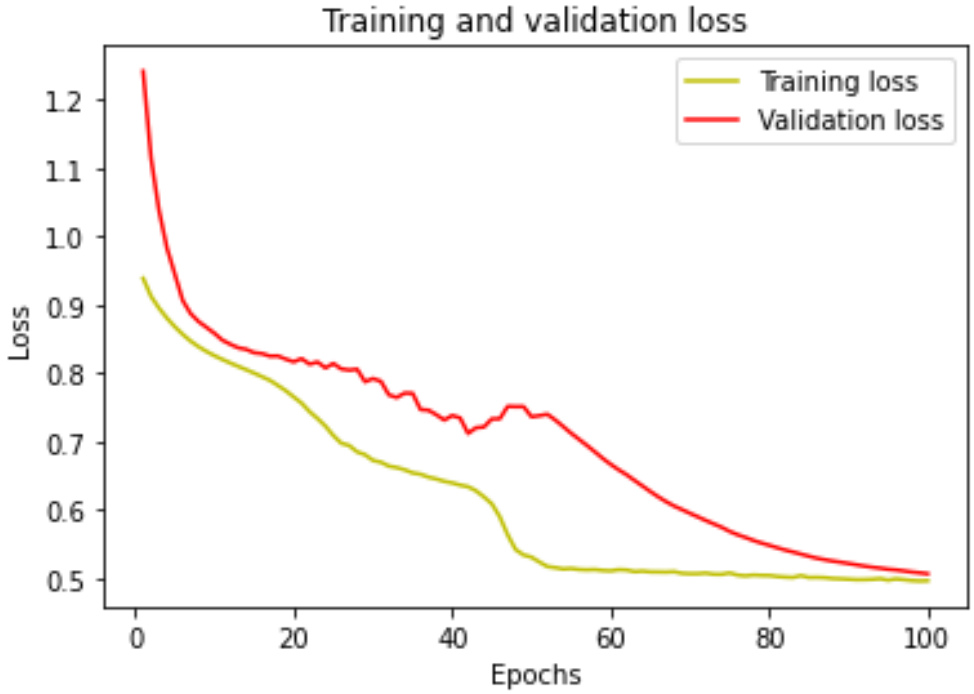

Figura 52. Representación gráfica del valor del error de entrenamiento y el error de validación por épocas. Modelo con etapa decodificadora VGG-16.

Fuente: elaboración propia.

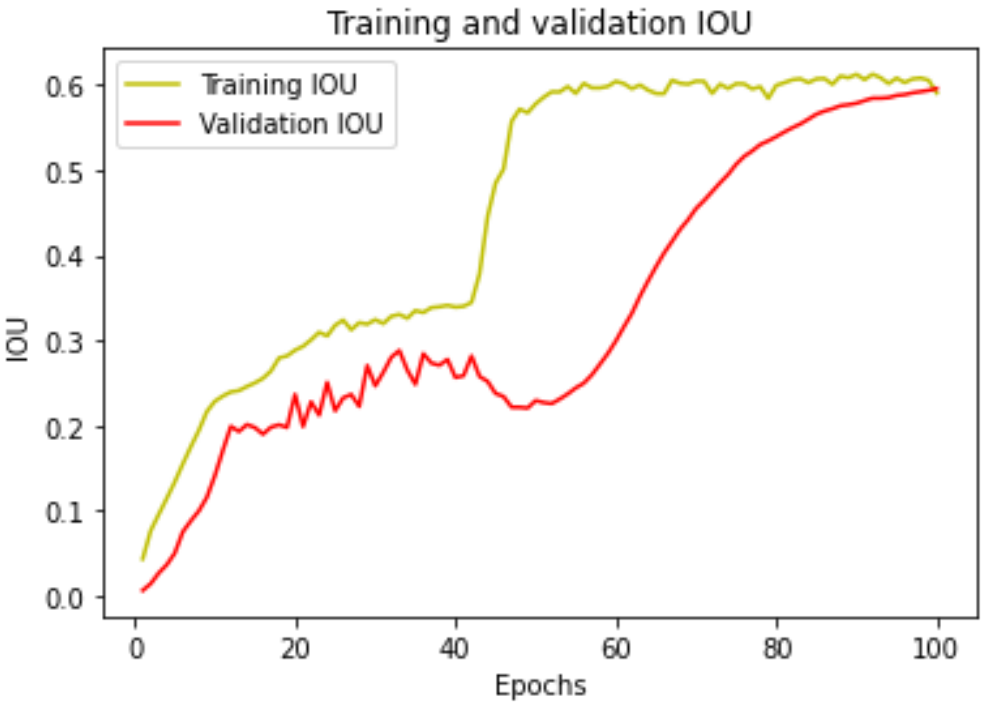

Figura 53. Representación gráfica de la medida de "IoU" de entrenamiento y validación por épocas. Modelo con etapa decodificadora VGG-16.

Fuente: elaboración propia.

Las pruebas de inferencia con los datos de validación, (Figura 54, Figura 55 y Figura 56) también nos muestran que los resultados son algo peores que los obtenidos con el anterior modelo entrenado, podemos ver que también aparecen pequeños puntos en los que se ha etiquetado una zona de nervio óptico como si fuera una arteria carótida. En la Figura 56 también se puede apreciar que no se ha detectado correctamente la arteria carótida derecha.

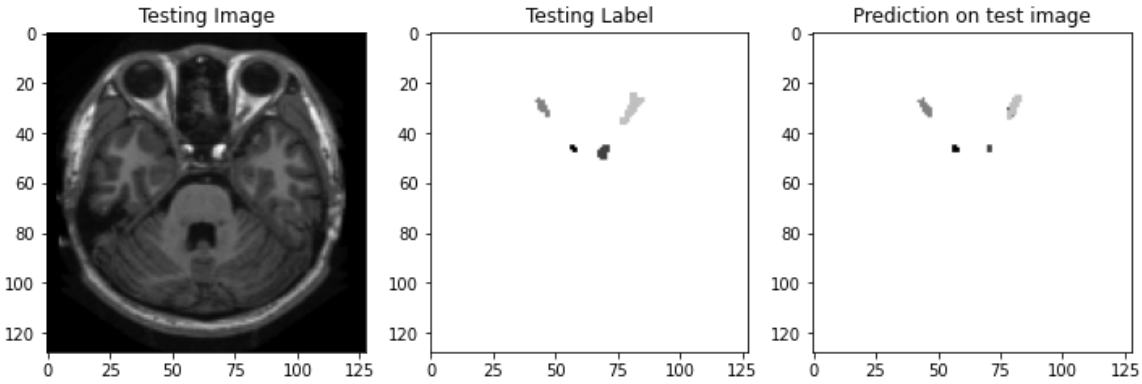

Figura 54. Resultado de validación para la primera secuencia de imágenes del dataset de validación. Modelo con etapa decodificadora VGG-16.

Fuente: elaboración propia.

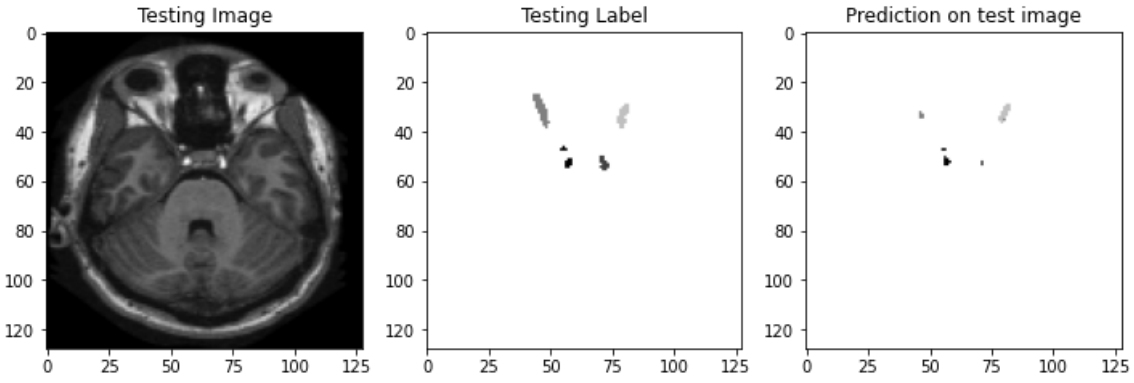

Figura 55. Resultado de validación para la segunda secuencia de imágenes del dataset de validación. Modelo con etapa decodificadora VGG-16.

Fuente: elaboración propia.

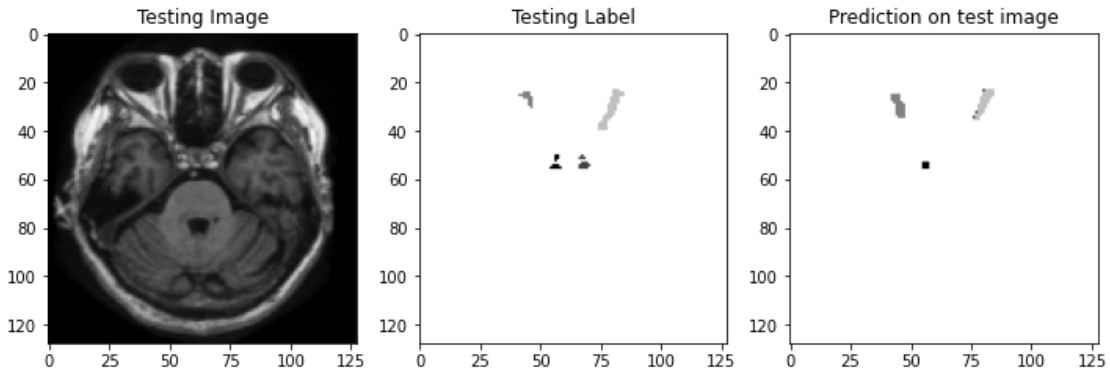

Figura 56. Resultado de validación para la tercera secuencia de imágenes del dataset de validación. Modelo con etapa decodificadora VGG-16.

Fuente: elaboración propia.

4.2- Inferencia de los modelos entrenados.

Con el entrenamiento de los modelos completados, procedemos a probar su funcionamiento con nuevas secuencias de imágenes utilizando la aplicación de inferencia desarrollada.

Para las pruebas realizadas en este apartado, descargamos el disco 3 del dataset OASIS, tal y como se indica en la Figura 57 y en la Figura 58.

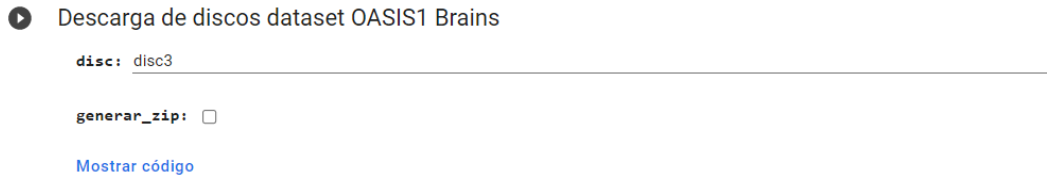

Figura 57. Selección de la descarga del disco 3 del dataset OASIS1 Brains en la interfaz gráfica del apartado "Descarga de discos dataset OASIS1 Brains" en la aplicación de inferencia de modelos.

Fuente: elaboración propia.

Figura 58. Selección de la carga del disco 3 del dataset OASIS1 Brains en la interfaz gráfica del apartado "Cargar secuencias de imágenes" en la aplicación de inferencia de modelos.

Fuente: elaboración propia.

El primer modelo que probamos es el de la red con ResNet-50, para hacer esto seleccionamos el modelo correspondiente en la interfaz gráfica, tal y como se indica en la Figura 59 .

Figura 59. Selección de la descarga del modelo preentrenado con etapa decodificadora ResNet-50 en la interfaz gráfica del apartado "Descarga el modelo entrenado" de la aplicación de inferencia de modelos.

Fuente: elaboración propia.

Con el modelo y las secuencias de imágenes cargadas, ejecutamos los siguientes segmentos de código para poder visualizar los resultados obtenidos para cada paciente. Los resultados obtenidos se pueden observar en las siguientes figuras, también se ha añadido una representación 3D de las etiquetas obtenidas utilizando un script de Matlab.

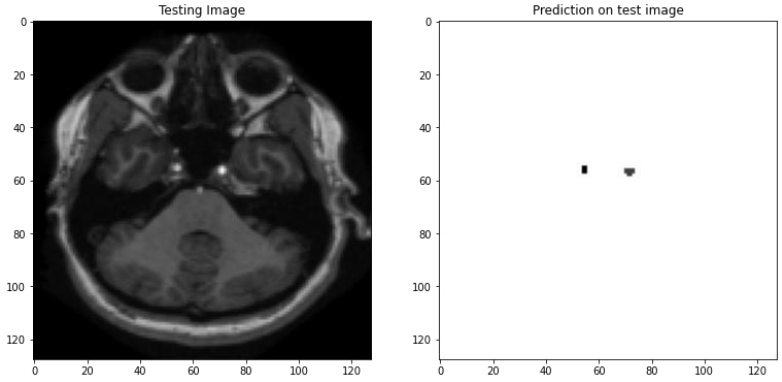

Figura 60. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0081_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

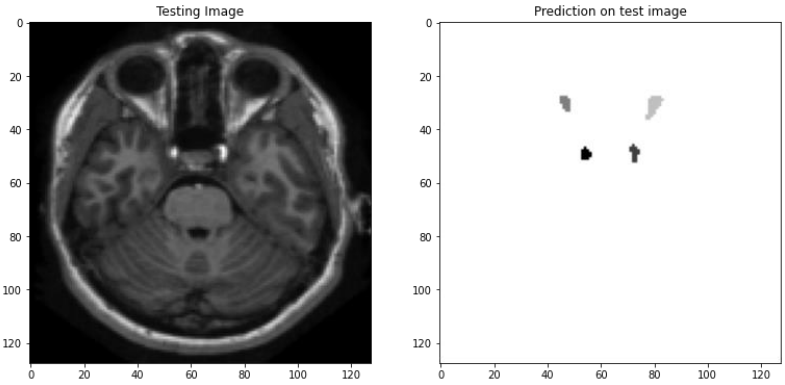

Figura 61. Resultado de la segmentación de las arterias carótidas y los nervios ópticos para el paciente OAS1_0081_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

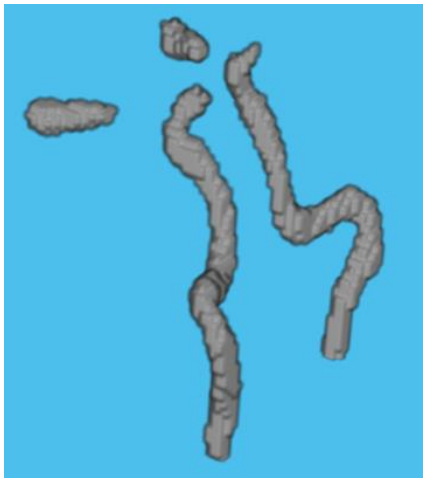

Figura 62. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0081_MR1 utilizando ResNet-50.

Las estructuras horizontales de menor tamaño corresponden a los nervios ópticos, mientras que las verticales corresponden a las arterias carótidas.

Fuente: elaboración propia.

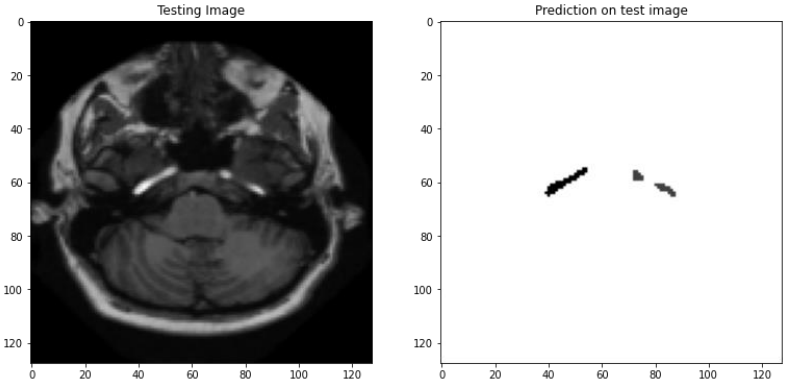

Figura 63. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0086_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

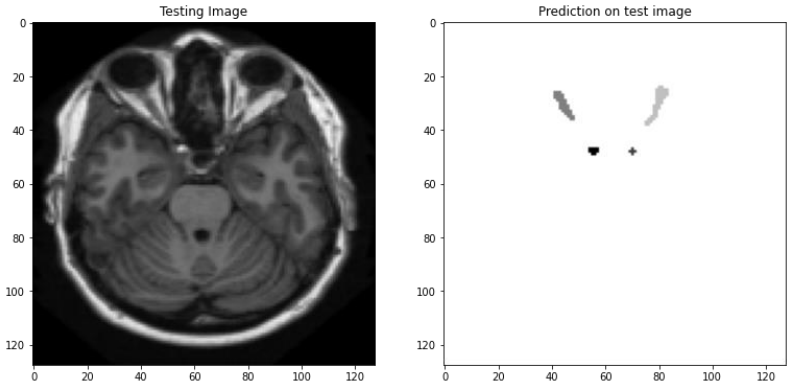

Figura 64. Resultado de la segmentación de las arterias carótidas y los nervios ópticos para el paciente OAS1_0086_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

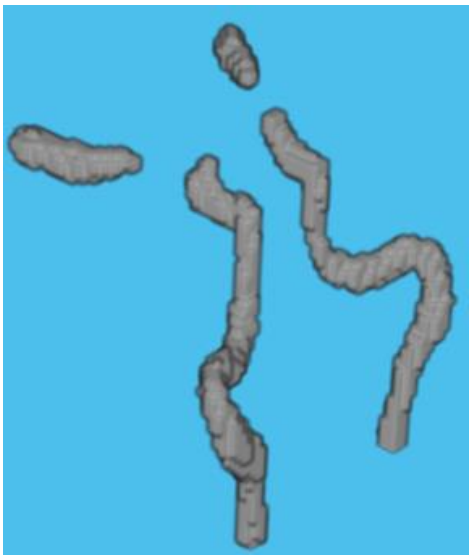

Figura 65. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0086_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

En las pruebas realizadas, podemos observar casos como el mostrado en las figuras Figura 60, Figura 61 y Figura 62 en el que las arterias carótidas y los nervios ópticos se segmentan de forma aparentemente precisa, ya que las formas que podemos ver en la representación tridimensional de las etiquetas generadas se asemejan bastante a las formas anatómicas reales. En este caso la secuencia de entrada presenta un buen contraste en las zonas de las arterias carótidas y los nervios ópticos respecto del resto de zonas de alrededor, lo que facilita su segmentación automática por parte de la red neuronal.

En las figuras Figura 63, Figura 64 y Figura 65 podemos observar otro caso similar en el que la segmentación es bastante precisa.

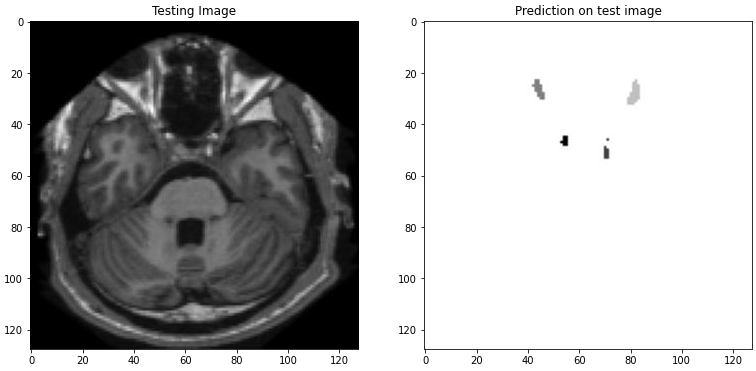

Figura 66. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0092_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

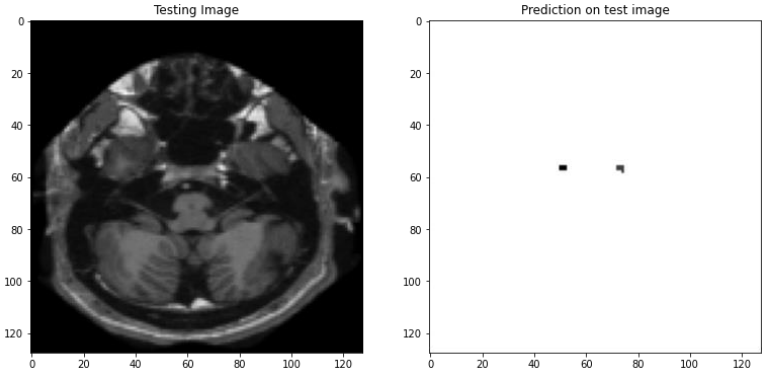

Figura 67. Resultado de la segmentación de las arterias carótidas y los nervios ópticos para el paciente OAS1_0092_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

Figura 68. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0092_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

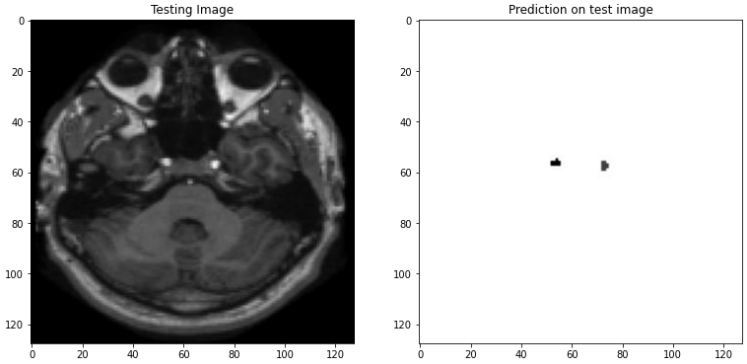

Figura 69. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0095_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

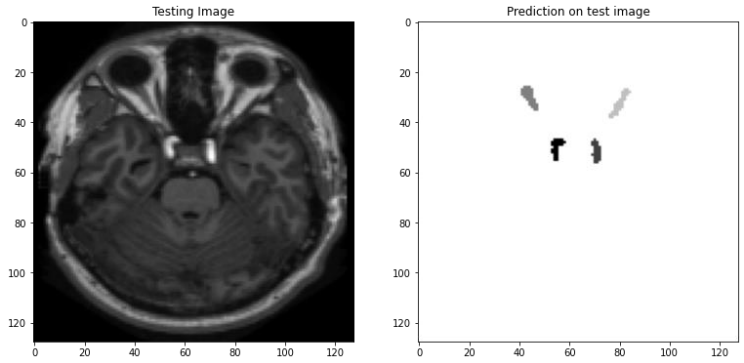

Figura 70. Resultado de la segmentación de las arterias carótidas y los nervios ópticos para el paciente OAS1_0095_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

Figura 71. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0095_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

También podemos encontrar casos como los mostrados en las figuras Figura 66, Figura 67, Figura 68, Figura 69, Figura 70 y Figura 71 en los que la segmentación es bastante precisa, pero se segmentan partes externas que no pertenecen a las arterias carótidas ni a los nervios ópticos. Estos son casos en los que la secuencia de imágenes presenta alguna zona que no pertenece ni a las arterias carótidas ni a los nervios ópticos, pero que la red segmenta como si perteneciera a alguna de estas zonas.

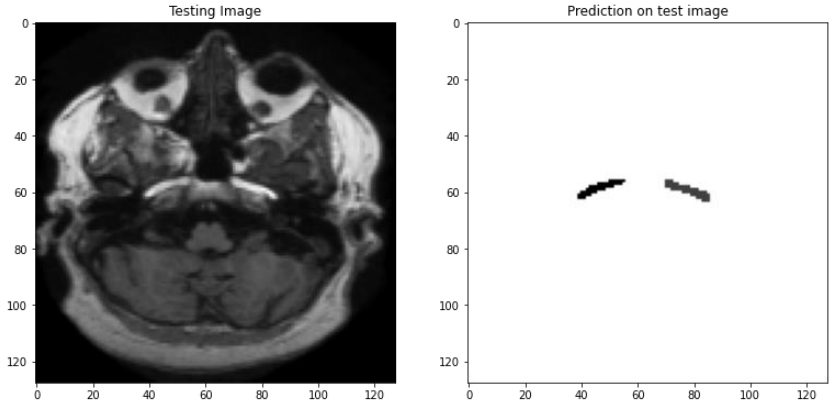

Figura 72. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0114_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

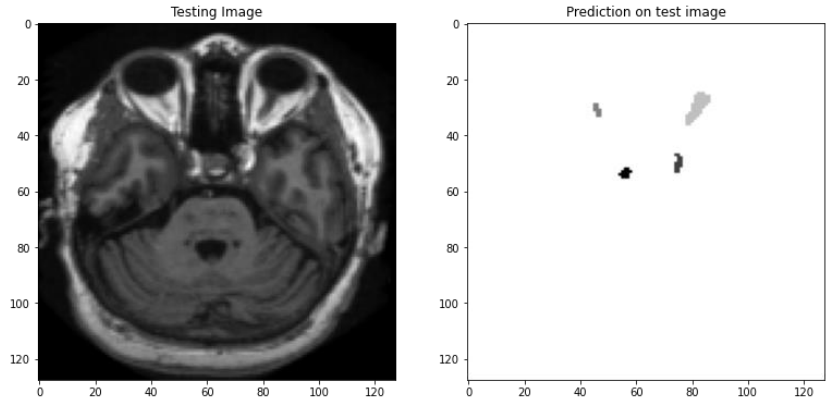

Figura 73. Resultado de la segmentación de las arterias carótidas y los nervios ópticos para el paciente OAS1_0114_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

Figura 74. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0114_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

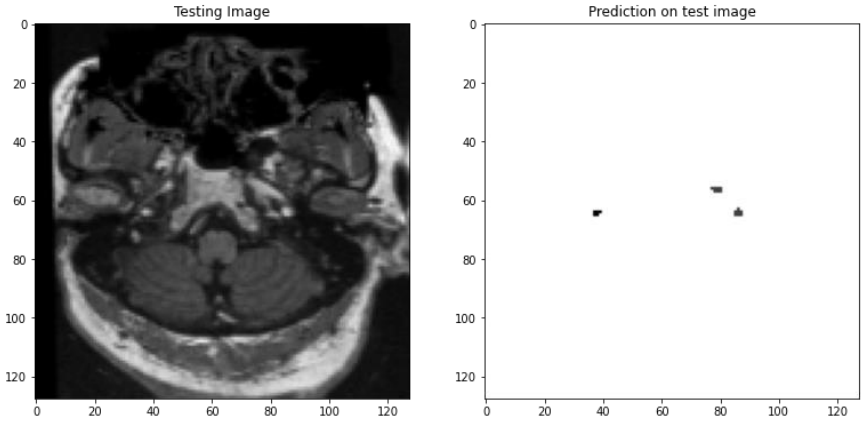

Figura 75. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0082_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

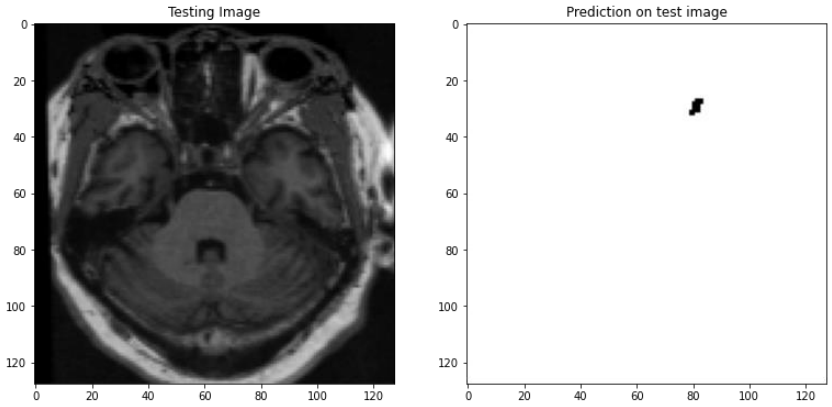

Figura 76. Resultado de la segmentación de las arterias carótidas y los nervios ópticos para el paciente OAS1_0082_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

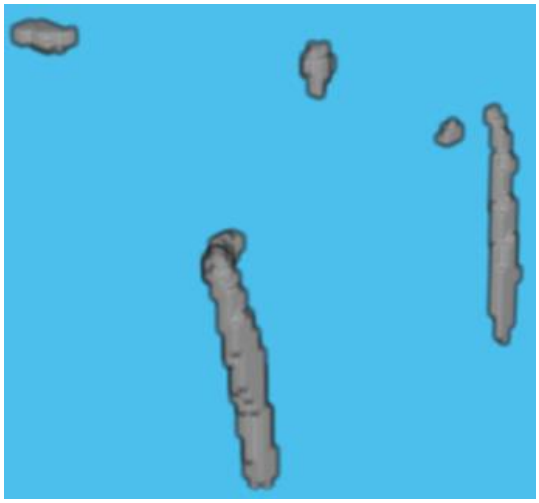

Figura 77. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0082_MR1 utilizando ResNet-50.

Fuente: elaboración propia.

En las figuras Figura 72, Figura 73, Figura 74, Figura 75, Figura 76 y Figura 77 podemos observar dos casos en los que la segmentación no se ha podido realizar correctamente, ya que las zonas etiquetadas presentan cortes y hay zonas que no han quedado etiquetadas. Estos son casos en los que las secuencias presentan poco contraste y poca visibilizad de las zonas de las arterias carótidas y los nervios ópticos, por lo que la red no es capaz de segmentar dichas zonas en algunas de las capas de la secuencia.

A continuación, realizamos las mismas pruebas, pero utilizando la red entrenada con VGG-16 como etapa codificadora:

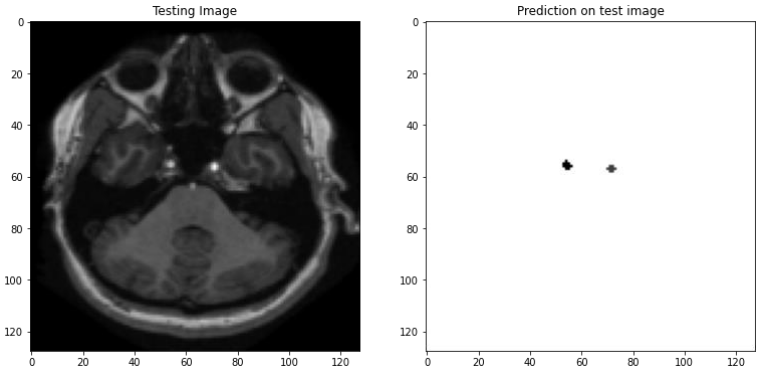

Figura 78. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0081_MR1 utilizando VGG-16.

Fuente: elaboración propia.

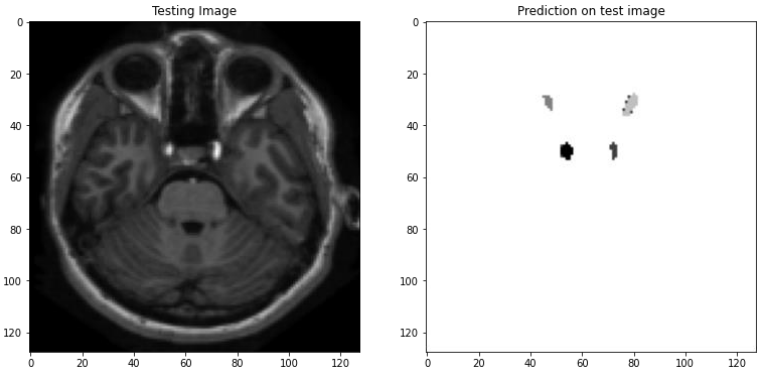

Figura 79. Resultado de la segmentación de las arterias carótidas y los nervios ópticos para el paciente OAS1_0081_MR1 utilizando VGG-16.

Fuente: elaboración propia.

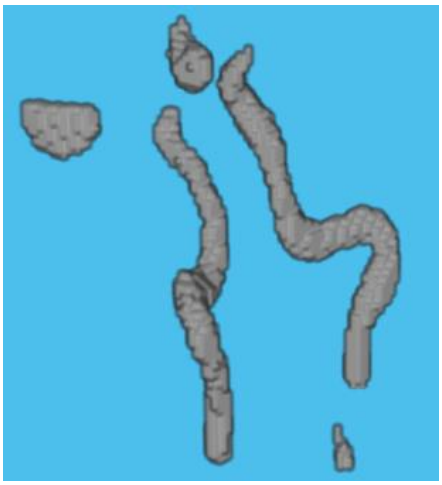

Figura 80. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0081_MR1 utilizando VGG-16.

Fuente: elaboración propia.

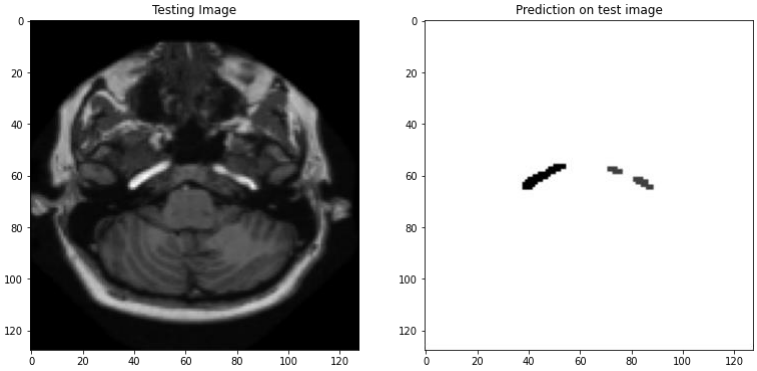

Figura 81. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0086_MR1 utilizando VGG-16.

Fuente: elaboración propia.

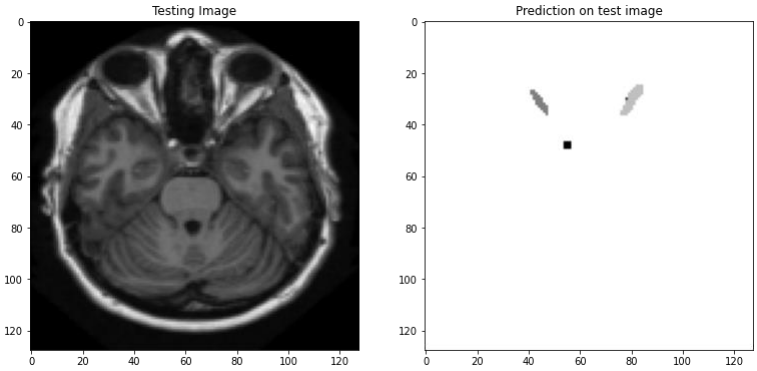

Figura 82. Resultado de la segmentación de las arterias carótidas y los nervios ópticos para el paciente OAS1_0086_MR1 utilizando VGG-16.

Fuente: elaboración propia.

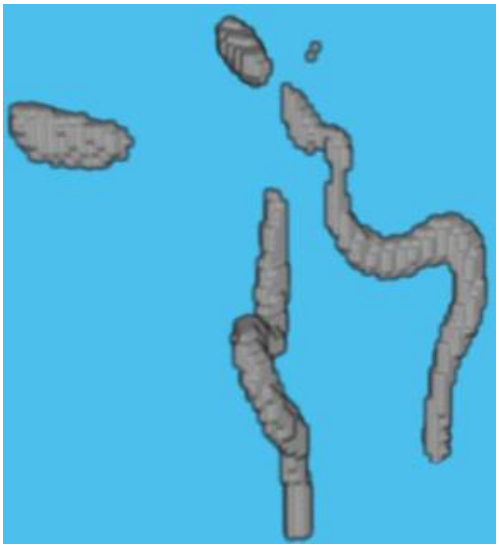

Figura 83. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0086_MR1 utilizando VGG-16.

Fuente: elaboración propia.

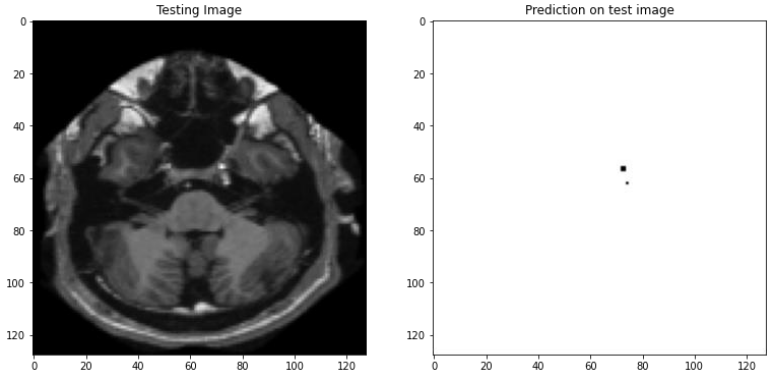

Figura 84. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0092_MR1 utilizando VGG-16.

Fuente: elaboración propia.

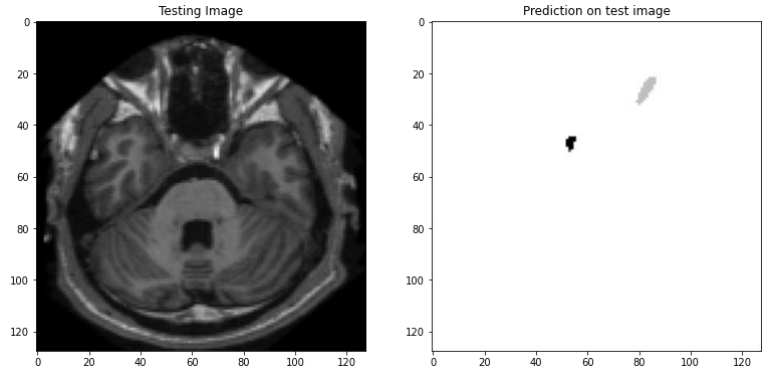

Figura 85. Resultado de la segmentación de las arterias carótidas y los nervios ópticos para el paciente OAS1_0092_MR1 utilizando VGG-16.

Fuente: elaboración propia.

Figura 86. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0092_MR1 utilizando VGG-16.

Fuente: elaboración propia.

En general, podemos observar que los resultados generados por la red con VGG-16 son peores que los obtenidos con ResNet-50, por ejemplo, en la Figura 80 podemos apreciar que la segmentación es menos precisa y que presenta más zonas segmentadas incorrectamente que el resultado proporcionado en la Figura 62.

En cuanto al tiempo que tardan en realizar la inferencia los modelos, el modelo con ResNet-50 tarda de media 0,48 segundos por cada secuencia de datos introducida, mientras que el modelo VGG-16 tarda 1 segundo aproximadamente.

4.3- Entrenamiento del modelo de segmentación exclusiva de las arterias carótidas

Para mejorar los resultados de la segmentación de las arterias carótidas se programó una variante del entrenamiento que emplea únicamente las etiquetas de las arterias carótidas. A las secuencias de imágenes del dataset se les realiza un recorte cuadrado de 128x128 píxeles que contiene la zona en la que se suelen encontrar las arterias carótidas, de esta forma se reduce el tamaño de las imágenes de entrada a la red sin necesidad de reescalarlas.

En este caso se utilizará el dataset en el que sólo se han etiquetado las arterias carótidas, para seleccionar este dataset se debe desmarcar la casilla Dataset_con_nervios_opticos de la interfaz gráfica como se muestra en la Figura 87.

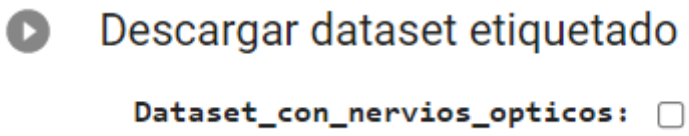

Figura 87. Selección de dataset sin nervios ópticos en la interfaz gráfica de la aplicación de entrenamiento de modelos.

Fuente: elaboración propia.

Para realizar el recorte también se deben ajustar los parámetros en la interfaz gráfica del apartado "*Cargar imágenes y etiquetas del dataset*" tal y como se indica en la Figura 88.

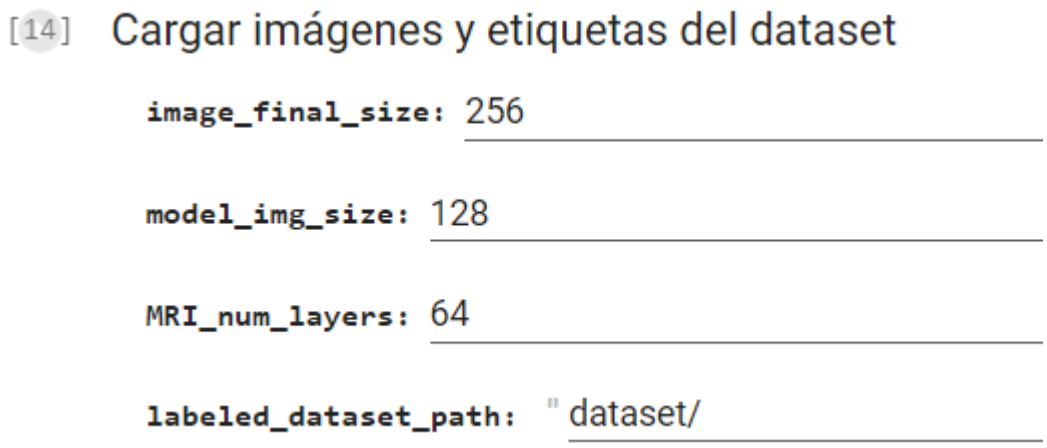

Figura 88. Ajuste de parámetros en la interfaz gráfica del apartado "Cargar imágenes y etiquetas del dataset" de la aplicación de entrenamiento de modelos para el entrenamiento con el dataset sin nervios ópticos.

Fuente: elaboración propia.

El siguiente apartado de código se puede utilizar para visualizar que las secuencias de imágenes de entrada de la red presentan el recorte de la zona de búsqueda de las arterias carótidas, tal y como se puede ver en la Figura 89 .

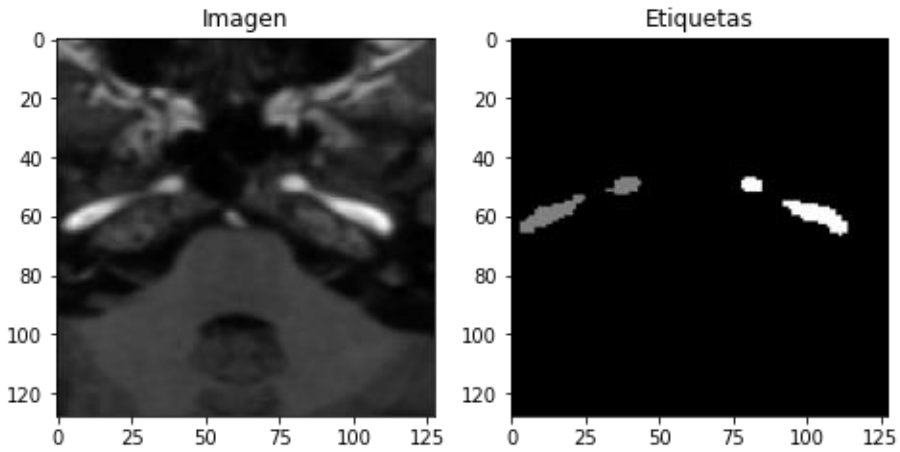

Figura 89. Visualización del recorte de la zona de búsqueda de las arterias carótidas.

Fuente: elaboración propia.

Los parámetros del modelo se establecieron como se indica en la Figura 90, se utilizó la red preentrenada resnet-50 como etapa decodificadora, ya que como se pudo comprobar en el apartado anterior, proporciona mejores resultados que vgg-16.

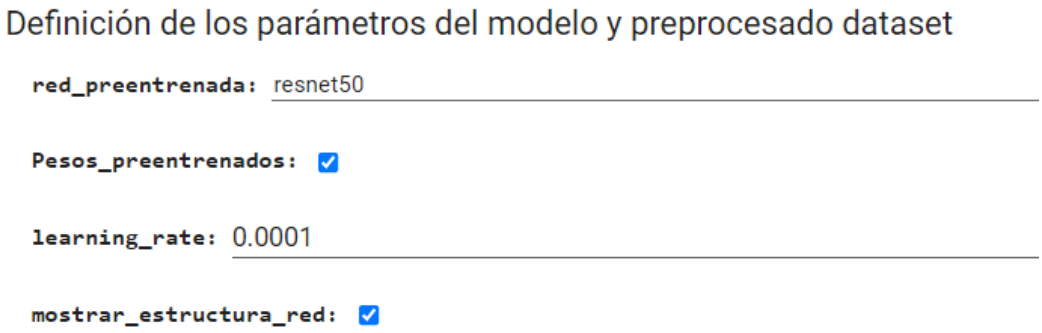

Figura 90. Ajuste de parámetros en la interfaz gráfica del apartado " Definición de los parámetros del modelo y preprocesado del dataset" en la aplicación de entrenamiento de modelos. Modelo de segmentación exclusiva de las arterias carótidas.

Fuente: elaboración propia.

Para el entrenamiento se empleó un 10% de las secuencias de imágenes del dataset para realizar la validación, lo que supone que se entrenó con 26 secuencias de imágenes y se validó con 3. Se estableció un número de 200 épocas para el entrenamiento de la red, valor que se consideró lo suficientemente grande como para que la red alcanzara su valor de error mínimo.

Con los ajustes descritos anteriormente, ejecutamos el proceso de entrenamiento, que tarda aproximadamente 125 minutos en completarse. En la figura 90 podemos ver la reducción del valor del error de entrenamiento y el valor del error de validación. El error de entrenamiento alcanza el valor de 0.0570 en la época 200, el error de validación es algo mayor: 0.1141. Observando la Figura 91 podemos apreciar que el valor de los errores se estanca aproximadamente en la época 75, por lo que en realidad se podría haber parado el entrenamiento en esta época.

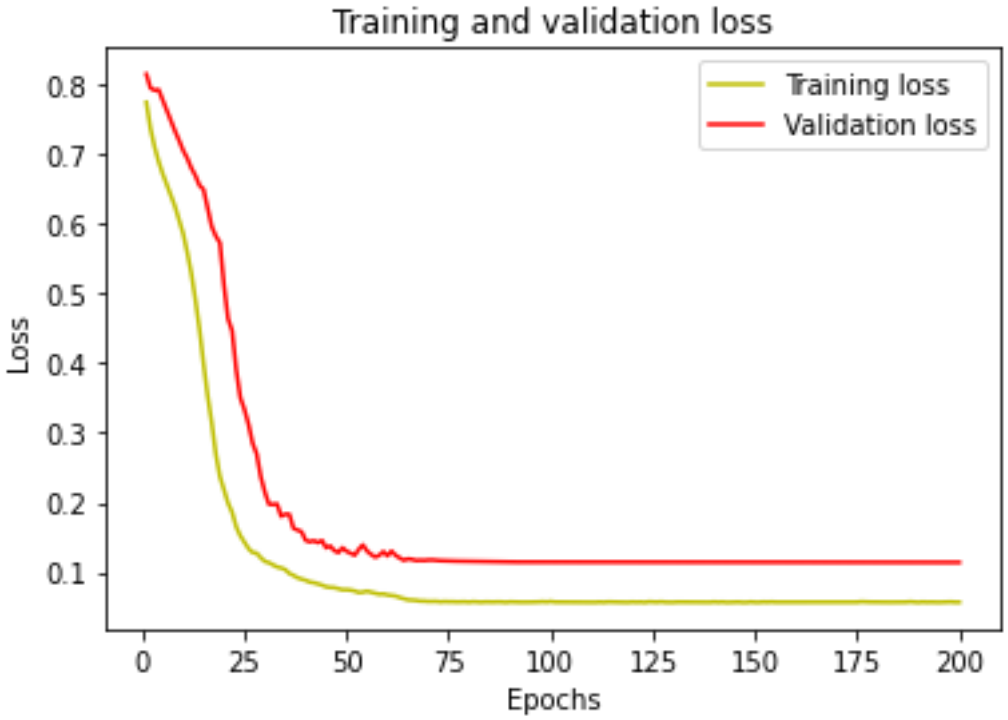

Figura 91. Representación gráfica del valor del error de entrenamiento y el error de validación por épocas. Modelo de segmentación exclusiva de las arterias carótidas.

Fuente: elaboración propia.

En la Figura 92 podemos visualizar el aumento de los valores del IOU, el IOU de entrenamiento alcanza el valor de 0.9219, mientras que el de validación solamente llega a 0.8172.

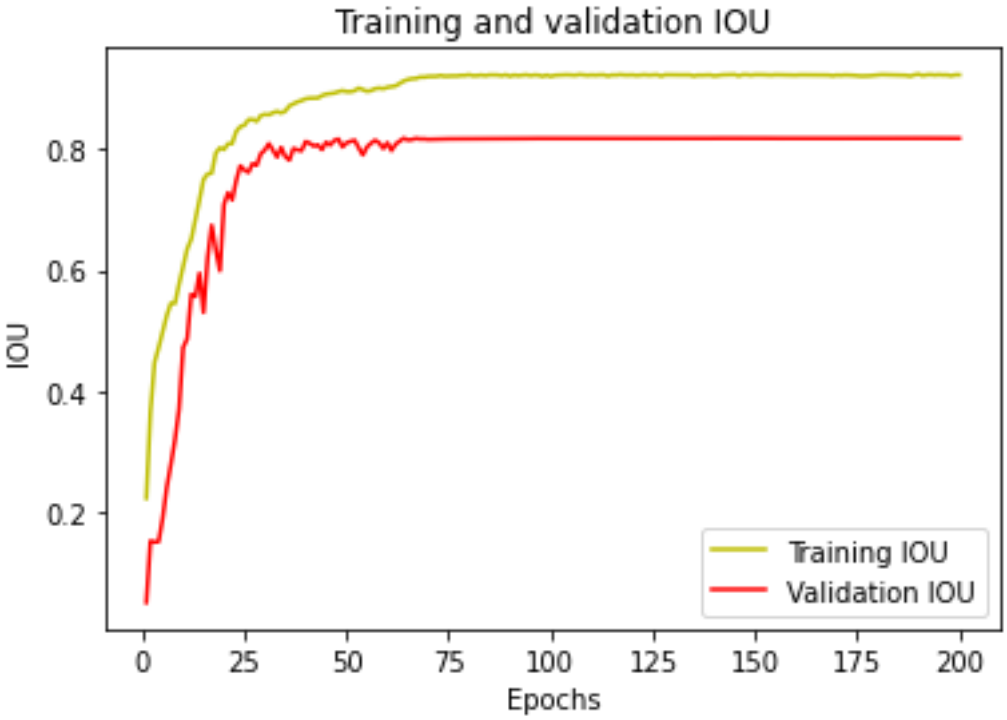

Figura 92. Representación gráfica de la medida de "IoU" de entrenamiento y validación por épocas. Modelo de segmentación exclusiva de las arterias carótidas.

Fuente: elaboración propia.

Al utilizar únicamente las etiquetas de las arterias carótidas y recortar la zona de búsqueda, se ha conseguido reducir el error de entrenamiento y validación, el valor del IoU también ha aumentado considerablemente. Para comprobar esta mejora de funcionamiento se realizarán las pruebas de inferencia en el siguiente apartado.

4.4- Inferencia del modelo de segmentación exclusiva de las arterias carótidas.

Para realizar las pruebas de inferencia se debe seleccionar el modelo "Modelo_ResNet50_ICAs" en la interfaz gráfica del apartado "Descarga del modelo entrenado" de la aplicación de inferencia, tal y como se indica en la Figura 93.

Figura 93. Selección de la descarga del modelo preentrenado de segmentación exclusiva de las arterias carótidas en la interfaz gráfica del apartado "Descarga el modelo entrenado" de la aplicación de inferencia de modelos .

Fuente: elaboración propia.

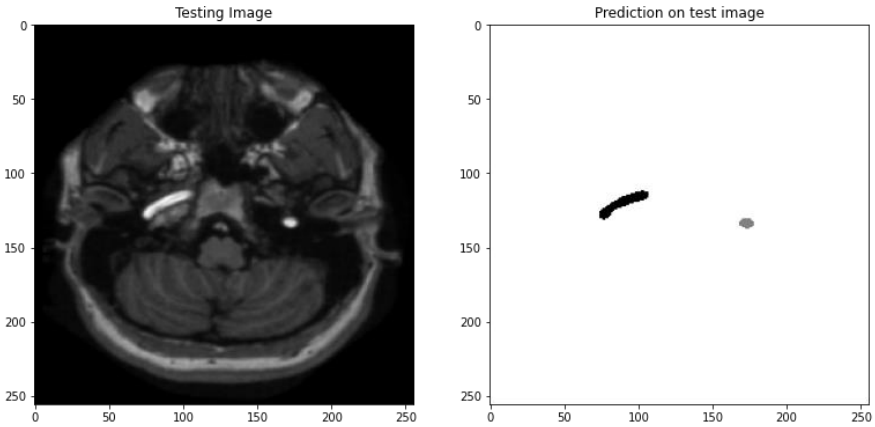

Figura 94. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0081_MR1 utilizando el modelo de segmentación exclusiva de arterias carótidas.

Fuente: elaboración propia.

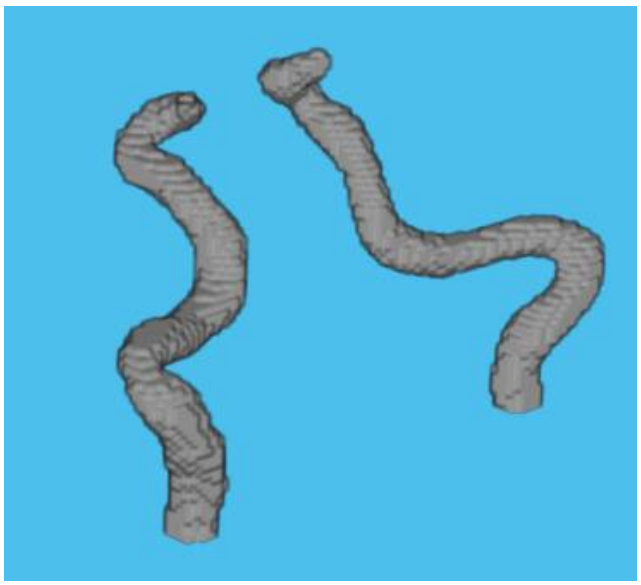

Figura 95. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0081_MR1 utilizando el modelo de segmentación exclusiva de arterias carótidas.

Fuente: elaboración propia.

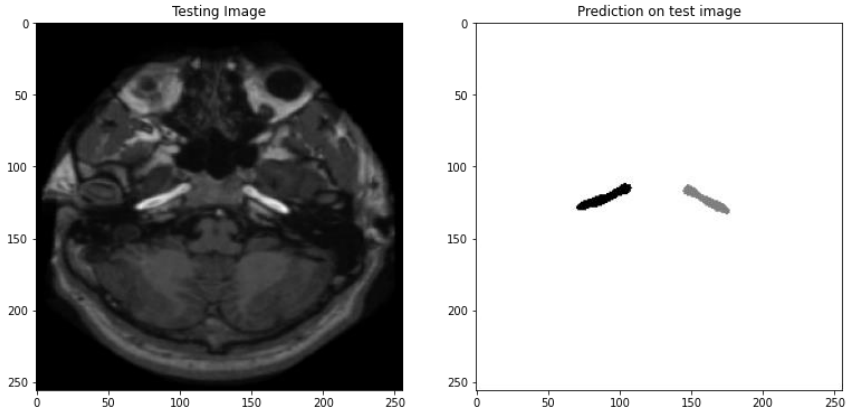

Figura 96. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0095_MR1 utilizando el modelo de segmentación exclusiva de arterias carótidas.

Fuente: elaboración propia.

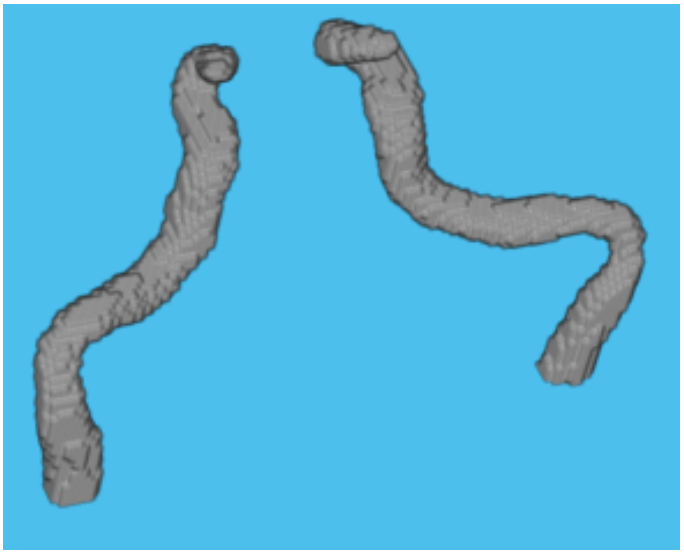

Figura 97. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0095_MR1 utilizando el modelo de segmentación exclusiva de arterias carótidas.

Fuente: elaboración propia.

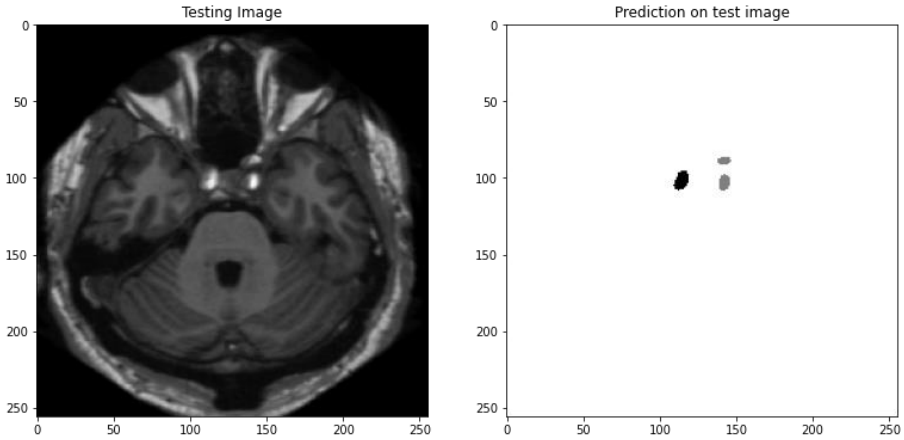

Figura 98. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0114_MR1 utilizando el modelo de segmentación exclusiva de arterias carótidas.

Fuente: elaboración propia.

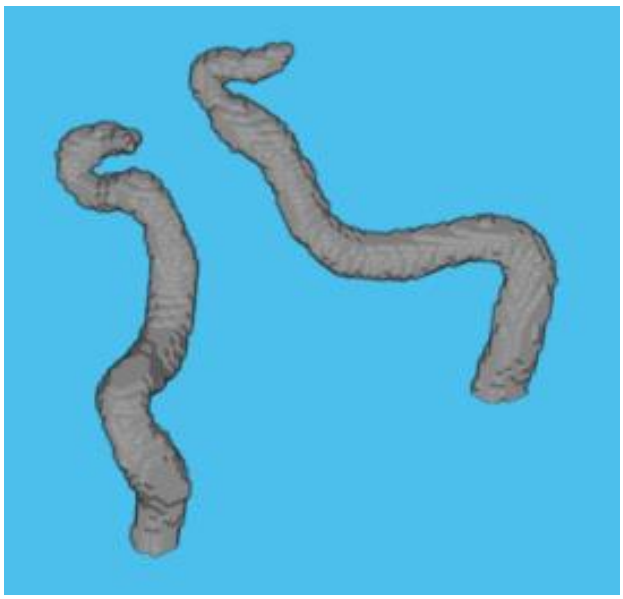

Figura 99. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0114_MR1 utilizando el modelo de segmentación exclusiva de arterias carótidas.

Fuente: elaboración propia.

En las pruebas realizadas, podemos observar que en los casos en los que las secuencias tienen una buena calidad como las de los pacientes OAS1_0081_MR1, OAS1_0095_MR1 y OAS1_0114_MR1, las arterias carótidas se segmentan de forma bastante precisa y tienen mucha más definición que las segmentadas con el modelo entrenado con etiquetas de nervios ópticos y arterias carótidas, esto se puede observar en las figuras Figura 95, Figura 97 y Figura 99.

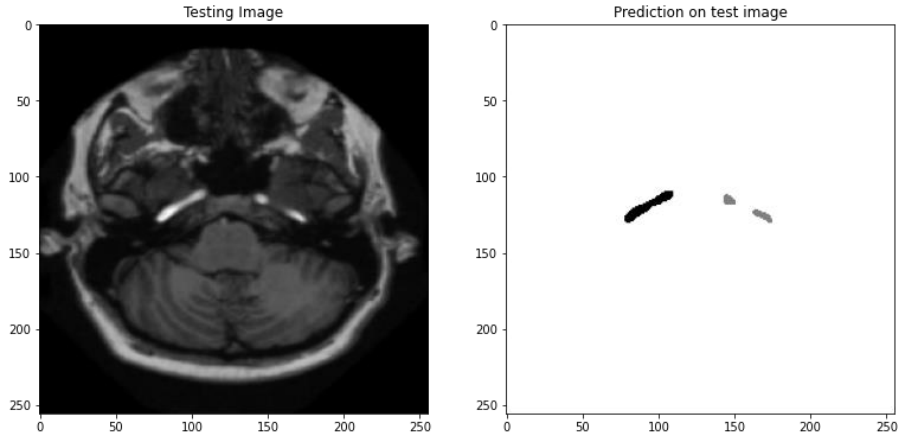

Figura 100. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0086_MR1 utilizando el modelo de segmentación exclusiva de arterias carótidas.

Fuente: elaboración propia.

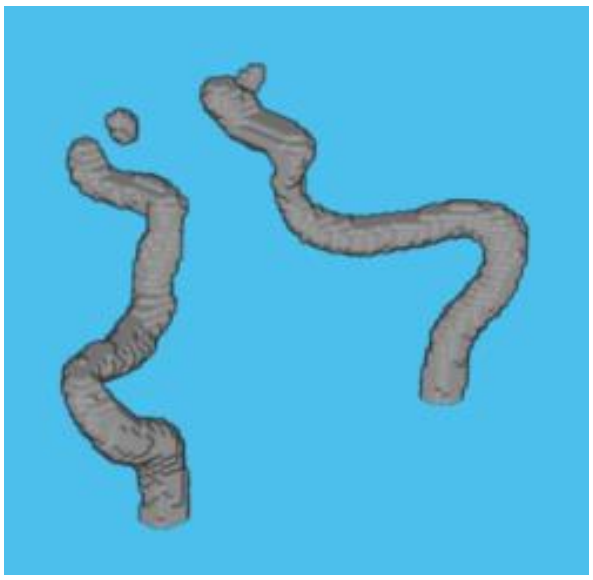

Figura 101. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0086_MR1 utilizando el modelo de segmentación exclusiva de arterias carótidas.

Fuente: elaboración propia.

Para el caso del paciente OAS1_0086_MR1, con el nuevo modelo se han segmentado partes de las arterias carótidas que con el modelo anterior se habían omitido, esto se puede observar en la Figura 97.

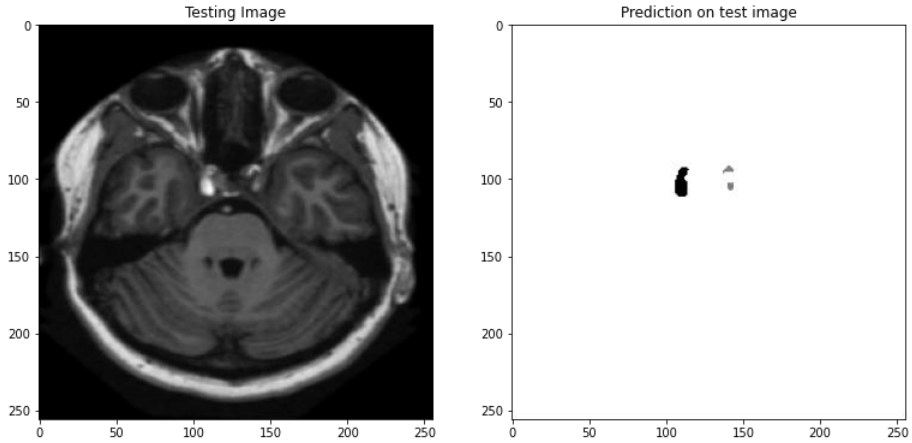

Figura 102. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0092_MR1 utilizando el modelo de segmentación exclusiva de arterias carótidas.

Fuente: elaboración propia.

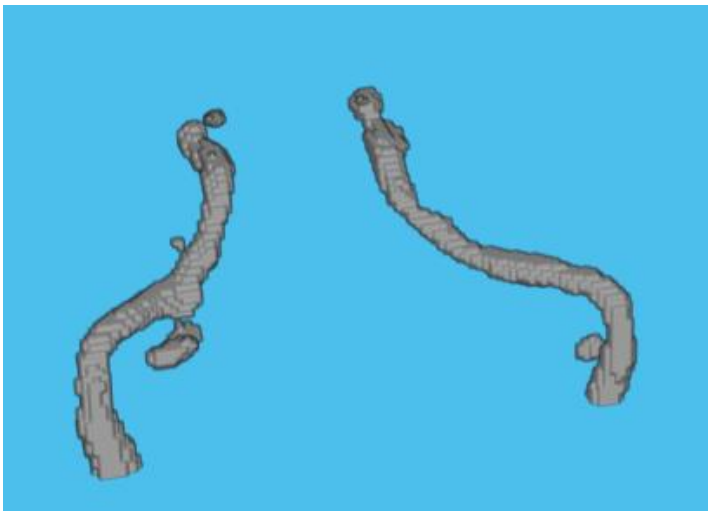

Figura 103. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0092_MR1 utilizando el modelo de segmentación exclusiva de arterias carótidas.

Fuente: elaboración propia.

En la segmentación de secuencia del paciente OAS1_0092_MR1 de la Figura 103, se puede apreciar que se siguen segmentando partes externas a las arterias carótidas al igual que sucedía con el modelo anterior.

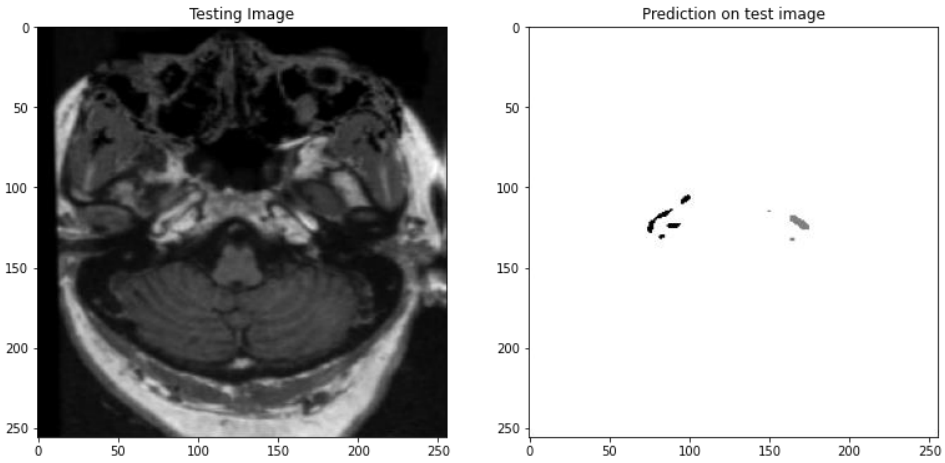

Figura 104. Resultado de la segmentación de las arterias carótidas para el paciente OAS1_0082_MR1 utilizando el modelo de segmentación exclusiva de arterias carótidas.

Fuente: elaboración propia.

Figura 105. Representación 3D de las etiquetas generadas por la red neuronal para el paciente OAS1_0082_MR1 utilizando el modelo de segmentación exclusiva de arterias carótidas.

Fuente: elaboración propia.

En el caso del paciente OAS1_0082_MR1, las arterias carótidas presentan una visibilidad bastante mala o nula sobre todo en las capas superiores de la secuencia de imágenes, es por esto que la segmentación no se realiza correctamente como se puede observar en la Figura 105.

4.5- Resumen de resultados.

En este apartado se va a realizar un breve resumen de los resultados obtenidos, en la tabla de la Figura 106 se muestra el tiempo promedio de inferencia y la precisión en el dataset de validación de cada uno de los modelos entrenados.

Modelo	Tiempo de inferencia (seg)	Precisión en dataset validación (%IoU)
Etapa decodificadora con ResNet-50	0,48	63,81
Etapa decodificadora con VGG-16	1	59,59
Segmentación exclusiva arterias carótidas	0,55	81,72

Figura 106. Tabla resumen de resultados.

Fuente: Elaboración propia.

Para la tarea de segmentación de las arterias carótidas y los nervios ópticos el modelo que proporciona mejores resultados es el de la etapa decodificadora con ResNet-50, con este modelo se consigue una segmentación con una similitud del 63,81% a la segmentación realizada manualmente.

En el caso de la segmentación exclusiva de las arterias carótidas, el modelo entrenado proporciona resultados de segmentación con una similitud del 81,72% a la segmentación realizada manualmente.

En cuanto al tiempo que tardan los modelos en segmentar las secuencias de imágenes, el tiempo de todos los modelos entrenados es muy inferior al tiempo que requiere realizar la segmentación manualmente, para segmentar una secuencia manualmente se requiere. aproximadamente 20 minutos. El tiempo máximo que tarda uno de los modelos entrenados en segmentar una secuencia de imágenes es de 1 segundo.

5- Conclusiones y líneas futuras

La segmentación o etiquetado de imágenes de resonancia magnética del cerebro no es una tarea sencilla, debido en gran parte a que en muchas ocasiones estas imágenes tienen una resolución y un contraste bajos. El proceso de etiquetado manual es largo y tedioso, por lo que es susceptible al error humano.

En este trabajo se ha desarrollado una aplicación con redes neuronales convolucionales para realizar la segmentación de estas imágenes de forma automática.

La obtención de datasets de imágenes de resonancia magnética del cerebro no es sencilla, generalmente son imágenes que no se encuentran a disposición del público. También resulta complejo encontrar imágenes ya etiquetadas, en el caso de este trabajo no fue posible encontrarlas. Para poder entrenar las redes neuronales de la aplicación fue necesario etiquetar manualmente algunas imágenes. El proceso de etiquetado se realizó con sumo cuidado para que el aprendizaje de las redes neuronales fuera lo más correcto posible.

En la aplicación se han implantado dos posibles estructuras de red neuronal que emplean un "*backbone*" o etapa decodificadora distinto. En el trabajo se han entrenado ambas estructuras y se han contrastado los resultados que proporcionan.

Los resultados obtenidos en la segmentación utilizando la aplicación desarrollada son muy buenos, las etiquetas generadas automáticamente por la aplicación se ajustan correctamente a las zonas deseadas. Los resultados son más deficientes solo en las imágenes que presentan un bajo contraste o mala calidad. Adicionalmente la segmentación con la aplicación es mucho más rápida que la segmentación manual.

Con los resultados obtenidos se considera que se han cumplido satisfactoriamente con los objetivos del trabajo. Se considera también que se ha demostrado la utilidad de las redes neuronales convolucionales para la tarea de segmentación de imágenes médicas.

Adicionalmente, la aplicación se ha desarrollado de tal forma que se pueda emplear de una forma accesible para su utilización en futuras líneas de investigación. También se ha definido una metodología para generar los datos de entrenamiento y para evaluar el funcionamiento de los modelos generados.

Lógicamente, aunque los resultados son muy prometedores, la aplicación necesitaría todavía mucha investigación para ser incorporada a los quirófanos. A continuación, se presentan las líneas de trabajo futuras que se sugieren:

- Introducción de más secuencias etiquetadas al dataset para el entrenamiento y la validación de los modelos
- Etiquetado del dataset por parte de un especialista: durante el proceso de etiquetado del dataset surgieron complicaciones sobre todo a la hora de etiquetar los nervios ópticos debido a dudas con la anatomía de estos, para evitar estas dudas, sería adecuado que el proceso de etiquetado lo realizara una persona con buenos conocimientos de anatomía e imágenes de resonancia magnética.
- Obtención de otros datasets distintos de OASIS1: en el momento en el que se desarrolló este trabajo no se encontraron más datasets de imágenes de resonancia magnética cerebral, sería beneficioso para generar modelos más robustos añadir secuencias de otros datasets al entrenamiento.
- Optimización del uso de memoria GPU para el entrenamiento: se podría reducir el número de imágenes de las secuencias para emplear únicamente las que se encuentren a la altura del seno esfenoidal, que es la zona más relevante para la cirugía, esto supondría una reducción del uso de memoria GPU utilizada y se podrían entrenar modelos con imágenes de mayor resolución.
- Desarrollo de método alternativo de carga de secuencia de imágenes durante el entrenamiento: una de las limitaciones actuales de la aplicación es que las imágenes se cargan en la memoria RAM para su uso en el entrenamiento, cuando la cantidad de secuencias de imágenes es muy grande, se produce un desbordamiento de la memoria, por lo que sería adecuado desarrollar un método alternativo para que las imágenes se carguen desde el disco durante el entrenamiento.

6- Bibliografía

[1] N. Sanno *et al.*, “Pathology of pituitary tumors,” *Neurosurgery Clinics of North America*, vol. 14, no. 1. pp. 25–39, Jan. 2003, doi: 10.1016/S1042-3680(02)00035-9.

[2] Wormald PJ, *Cirugía endoscópica nasosinusal. Anatomía, reconstrucción tridimensional y técnica quirúrgica.* 2019.

[3] P. Ajler *et al.*, “Cirugía transnasal endoscópica para tumores de hipófisis,” *Surg. Neurol. Int.*, vol. 3, no. 7, p. 389, 2012, doi: 10.4103/2152-7806.104403.

[4] D. L. Pham, C. Xu, and J. L. Prince, “CURRENT METHODS IN MEDICAL IMAGE SEGMENTATION 1,” 2000, Accessed: May. 05, 2022. [Online]. Available: www.annualreviews.org.

[5] H. S. Song, H. S. Yoon, S. Lee, C. K. Hong, and B. J. Yi, “Surgical navigation system for transsphenoidal pituitary surgery applying U-net-based automatic segmentation and bendable devices,” *Appl. Sci.*, vol. 9, no. 24, 2019, doi: 10.3390/app9245540.

[6] T. Henry *et al.*, “Brain tumor segmentation with self-ensembled, deeply-supervised 3D U-net neural networks: a BraTS 2020 challenge solution,” Oct. 2020, Accessed: Apr. 26, 2022. [Online]. Available: http://arxiv.org/abs/2011.01045.

[7] J. T. Hathcock and R. L. Stickle, “Principles and concepts of computed tomography.,” *Vet. Clin. North Am. Small Anim. Pract.*, vol. 23, no. 2, pp. 399–415, 1993, doi: 10.1016/S0195-5616(93)50034-7.

[8] R. F. López and J. M. F. Fernández, *Las Redes Neuronales Artificiales*. 2008.

[9] A. Bosch Rué, *Deep learning : principios y fundamentos*, Primera ed. Barcelona: Editorial UOC, 2019.

[10] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition.” [Online]. Available: http://image-net.org/challenges/LSVRC/2015/.

[11] J. Long, E. Shelhamer, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” Nov. 2014, Accessed: Mar. 22, 2022. [Online]. Available: http://arxiv.org/abs/1411.4038.

[12] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation”, Accessed: May. 05, 2022. [Online]. Available: http://lmb.informatik.uni-freiburg.de/.

[13] M. A. Wani, M. Kantardzic, and M. Sayed-Mouchaweh, *Deep Learning Applications*, 1st ed. 20. Singapore: Springer Singapore, 2020.

[14] A. Annotation¨ozgün, C. ¸ Içek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, “3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation¨Ozgün.”, Accessed: May. 05, 2022. [Online]. Available: http://lmb.informatik.uni-freiburg.de/resources/opensource/unet.en.html.

[15] D. S. Marcus, T. H. Wang, J. Parker, J. G. Csernansky, J. C. Morris, and R. L. Buckner, “Open Access Series of Imaging Studies (OASIS): Cross-sectional MRI Data in Young, Middle Aged, Nondemented, and Demented Older Adults,” *J. Cogn. Neurosci.*, vol. 19, no. 9, pp. 1498–1507, Sep. 2007, doi: 10.1162/JOCN.2007.19.9.1498.

[16] A. Fedorov *et al.*, “3D Slicer as an Image Computing Platform for the Quantitative Imaging Network,” *Magn Reson Imaging*, vol. 30, no. 9, pp. 1323–1341, 2012, doi: 10.1016/j.mri.2012.05.001.

[17] M. Brett *et al.*, “nipy/nibabel: 3.2.1.” Nov. 28, 2020, doi: 10.5281/ZENODO.4295521.

[18] R. Solovyev, A. A. Kalinin, and T. Gabruseva, “3D convolutional neural networks for stalled brain capillary detection,” *Comput. Biol. Med.*, vol. 141, p. 105089, Feb. 2022, doi: 10.1016/J.COMPBIOMED.2021.105089.

[19] M. Abadi *et al.*, “TensorFlow: A system for large-scale machine learning,” Accessed: Mar. 22, 2022. [Online]. Available: https://tensorflow.org.

[20] F. Chollet and others, “Keras.” 2015.

[21] D. P. Kingma and J. Lei Ba, “15iclr-ADAM,” *Iclr*, pp. 1–15, 2015, Accessed: Apr. 27, 2022. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf %22 entire document.

[22] R. Zhao *et al.*, “Rethinking Dice Loss for Medical Image Segmentation,” in *2020 IEEE International Conference on Data Mining (ICDM)*, 2020, pp. 851–860, doi: 10.1109/ICDM50108.2020.00094.

[23] P. Jaccard, “HEW PHYTOIiOGIST. THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.’ Professor at the Federal Polyteehnic, Ziirich.”

[24] V. E. Balas, S. S. Roy, D. Sharma, and P. Samui, *Handbook of Deep Learning Applications*, 1st ed. 2019. Cham: Springer International Publishing, 2019.

7- Anexos

7.1- Script Matlab para visualizar etiquetas de nervios ópticos y arterias carótidas en tres dimensiones.

```
clear
file = 'labels.tif'
for i=1:63
    A(:,:,i) = imread(file,i);

end
B = A <4;
volumeViewer(B);
```

7.2- Script Matlab para visualizar etiquetas de arterias carótidas en tres dimensiones.

```
clear
file = 'labels.tif'

for i=1:63
    A(:,:,i) = imread(file,i);

end

B = A <2;

volumeViewer(B);
```

7.3- Enlace a aplicación de entrenamiento de modelos.

https://colab.research.google.com/drive/1x6wVn7KFvtWFpPQjf2geEF_xFDgNpHDm?usp=sharing

7.4- Enlace a aplicación de inferencia de modelos.

https://colab.research.google.com/drive/1h5hYWxyigw6VAquV4d5mZZaSbt2PaIJM?usp=sharing