UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

CÁLCULO DE CURVA DE PERFUSIÓN MEDIANTE APLICACIÓN Y SEGMENTACIÓN AUTOMÁTICA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

CAMILO ALEJANDRO CARRASCO BECERRA

PROFESOR GUÍA:
ÁNGEL JIMENEZ MOLINA

MIEMBROS DE LA COMISIÓN:
ANDRÉS CABA RUTTE
CRISTOBAL RAMOS GOMEZ

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: CAMILO ALEJANDRO CARRASCO BECERRA
FECHA: 2019
PROF. GUÍA: ÁNGEL JIMENEZ MOLINA

CÁLCULO DE CURVA DE PERFUSIÓN MEDIANTE APLICACIÓN Y SEGMENTACIÓN AUTOMÁTICA

El presente trabajo muestra el diseño e implementación de una aplicación capás de analizar imágenes de perfusión miocárdica utilizando segmentación automática y análisis de intensidad de señales. La segmentación automática logra detectar la zona de sangre y el miocardio de la imagen.

Se busca como objetivo dar las herramientas para realizar análisis visual sobre las imágenes, permitiendo visualización en paralelo entre imágenes de *Stress* y *Rest*, además de calcular de las imágenes la segmentación automática, se calculan y muestran el área de la curva de perfusión, su máximo y la pendiente inicial.

Basado en una clase controladora de la aplicación y un modelo de datos eficiente, se desarrolla un algoritmo que logra separar por la mitad el miocardio, entre epicardio y endocardio. Además, de un algoritmo capas de separar las zonas detectadas en 4 divisiones, según el input del usuario. Estas secciones se pueden analizar por separado, con valores y visualización.

Para el desarrollo visual, se utiliza Tkinter, librería de Python, la cual da las herramientas para el control de imágenes, interacción con el usuario y muestra de valores.

Como resultado, se obtiene una aplicación en fase uno, capaz de ingresar paquetes de imágenes, realizar la segmentación de miocardio y mostrar los resultados esperados según lo indicado en los objetivos.

Aún siendo primera iteración, la aplicación cumple con los objetivos y logra ser funcional, sin descartar arreglos y mejoras.

Para mi familia y amigos,
en especial a mi madre
muchas gracias por todo

Agradecimientos

Muchas gracias a mi familia, que es muy grande y muy unida. Gracias por estar en momentos de penas y alegrías y apoyar en todo momento, ya sea con un abrazo, un consejo o una sonrisa.

A mis padres, que se han esforzado tanto para que esto. Esto también es parte de ustedes.

A mis amigos, que han estado en muchas alegrías y han estado en momentos duros sin esperar nada a cambio. A queue, por ser los más tiernos y gracias por compartir por tantos años juntos. A todas las personas maravillosas que conocí en la Universidad.

A RR, Pablo, Germán, Chino, Esteban y todas las personas de eléctrica, que hicieron más a meno un viaje de un poco más de 4 años. Gracias por ser parte de uno de los viajes más importantes de mi vida.

Gracias a profesores, ayudantes y compañeros. No solo de la Universidad, sino de toda mi vida. Todos han aportado un pequeño grano de arena que se agradece.

A Dominique, Franco y Sergio, gracias por estar en momentos duros, escuchar y ayudar. Les debo más de alguna y se los agradeceré por siempre.

De todo corazón muchas gracias.

Tabla de Contenido

Índice de Ilustraciones

Introducción

La mayor causa de muerte en el mundo es la isquemia cardiaca y los infartos[1], y dentro de las causas de estas enfermedades, está el mal cuidado personal por falta de ejercicio y mala alimentación. Es por esto que por parte de las autoridades se intenta incentivar el cuidado personal y por otro lado, la ciencia intenta dar soluciones para encontrar mejores tratamientos y/o para una detección temprana.

Existen varios tipos de exámenes que permiten la detección del infarto o la isquemia, pero algunos llegan a ser invasivos y otros llegan a tomar mucho tiempo.

Uno de estos exámenes poco invasivos, es el examen de perfusión miocárdica, el cual mediante resonancia magnética, logra obtener imágenes del corazón, las cuales posteriormente son analizadas para detectar problemas en el sistema coronario. Es este análisis el que toma mucho tiempo por lo que, aunque da muy buenos resultados, a veces se evita.

Dentro de los procesos que hacen de este examen costoso en tiempo, es la detección del miocardio del resto de la imagen, ya que este músculo es el que se busca analizar. Además de esto, el cálculo de su comportamiento también es costoso y engorroso con las herramientas existentes.

En el año 2018, un equipo de la Universidad de Chile desarrolló un algoritmo de segmentación del miocardio para imágenes de perfusión, con grandes resultados de detección. Esto hace que el análisis de la perfusión sea más rápida y da la oportunidad de hacer más asequible este examen.

Es por esto, que dentro de este proyecto se propone el desarrollo de una plataforma capaz de utilizar el algoritmo desarrollado para la utilización dentro del ámbito médico. Se propone entonces una aplicación que tome imágenes de perfusión, las procese y dé los resultados necesarios de estas para un diagnóstico más eficiente y rápido.

[1]Fuente OMS: https://www.who.int/es/news-room/fact-sheets/detail/the-top-10-causes-of-death

Mediante este presente, se procede a dar un pequeño marco teórico y estado del arte para explicar el proceso de diseño e implementación de esta aplicación, además de explicar los conceptos desarrollados, cómo se implementaron, el trabajo desarrollado, y el trabajo futuro, con el fin de mejorar la aplicación, según los requerimientos necesitados.

Capítulo 1

Marco Teórico

Con el fin de entender cada punto que se tratará en este trabajo, se procederá inicialmente a dar un repaso general a los temas más importantes que se tocarán en el desarrollo de esta memoria.

1.1. Redes Neuronales

Antes de aclarar que es una red neuronal o una red U-Net, se verá que es inteligencia artificial o inteligencia computacional. Aunque no haya definición clara de este concepto, si se sabe que busca resolver problemáticas que son posibles para un humano o un animal, y que además necesiten cierto análisis o procesamiento antes de ser resueltos, lo que puede llamarse inteligencia [5] [6].

Aunque se nota la cierta ambigüedad, hay características comunes que ayudan a caracterizar de alguna forma el concepto de inteligencia computacional [7].

- Para representar conocimiento, la información numérica es básica
- El procesamiento de la información está basado en el cálculo numérico
- El procesamiento no está expresado de forma explícita

Aunque estos puntos se puedan declarar comunes, no necesariamente todos los métodos están representados por éstos, en base debido, a la ambigüedad de cuál es la definición correcta.

Actualmente, la inteligencia computacional está en muchas interacciones del día a día, y sin percatarnos de ello. Desde el reconocimiento de voz y el desbloqueo con huella, iris o rostro en los teléfonos inteligentes, pasando por vehículos autónomos hasta el análisis de información para tomar decisiones a nivel empresa, lo que se debe a la gran utilidad, eficiencia y asertividad que entregan sus resultados.

La experiencia que se ha tenido con este tipo de tecnología ha sido de tanta utilidad, que no hay duda de que se siga investigando y aplicando en otras áreas de desarrollo.

Durante los años de investigación y aplicación de la inteligencia computacional, se han desarrollado una gran cantidad de métodos que cumplan con las exigencias requeridas por esta ciencia, pero sin lugar a dudas el método más popular y conocido es el de redes neuronales o también llamado de deep learning, debido a su capacidad, facilidad de uso y de entendimiento.

1.1.1. Redes Neuronales

Las redes neuronales, como se comentó con anterioridad, son el método más popular y usado en la actualidad. Este modelo se basa en el cómo funcionan nuestras neuronas, con un núcleo y sus conexiones.

Como se puede inferir, la unidad básica de esta red es la neurona. Un conjunto de ellas, según la forma y las condiciones en las que se dispongan, pueden generar variados tipos de funcionamiento. La más simple de todas las neuronas es llamada perceptrón, la cual se compone de varias entradas y una salida, como se puede observar en la figura 1.1.

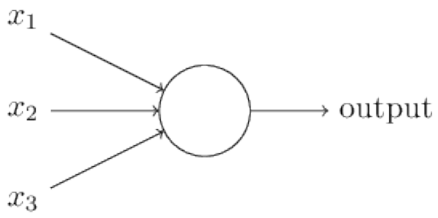

Figura 1.1: Modelo del perceptrón

Las entradas de un perceptrón, x_i según la figura 1.1, son computadas con una cierta proporción w_i, llamada peso, la cual define la importancia de esta entrada en el cálculo final. El cálculo del valor de salida en su forma más básica está determinada por la ecuación 1.1, la cual indica que dependiendo de las entradas y sus pesos, si la suma de estos valores supera cierto límite, su salida será 1, de lo contrario 0.

$$\text{output} = \begin{cases} 0 & \text{si } \sum w_i x_i \leq \text{threshold} \\ 1 & \text{si } \sum w_i x_i > \text{threshold} \end{cases} \tag{1.1}$$

Las entradas del perceptrón pueden ser tanto las entradas del sistema, como las salidas de otros perceptrones. Juntando varias de estas unidades, uniendo salidas de unas con entradas de otras, se puede crear una red, como la que se observa en la figura 1.2, la cual es capaz resolver problemas, según el valor de los pesos de cada unidad.

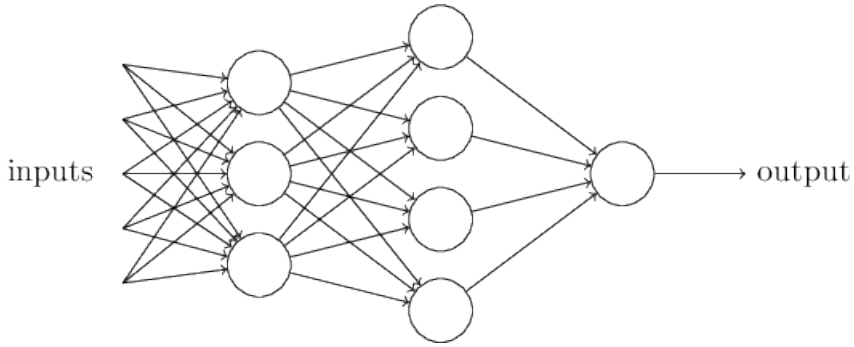

Figura 1.2: Red de neuronal básica

La red de la 1.2 está formada por capas o niveles, las cuales se pueden entender como el conjunto de neuronas que están alineadas de forma vertical en la figura. Las redes como la de la figura, donde cada una de las salidas de una capa son las entradas de la siguiente, se les llama redes completamente conectada (fully connected).

El threshold de la ecuación 1.1 es el que determina finalmente la salida de la neurona. A este valor, por simplicidad se le llama "bias" b, donde threshold = -b, lo que redefine lo anterior a la ecuación 1.2.

$$\text{output} = \begin{cases} 0 & \text{si } \sum w_i x_i + b \leq 0 \\ 1 & \text{si } \sum w_i x_i + b > 0 \end{cases} \tag{1.2}$$

Como se puede intuir, el que tiene la responsabilidad del funcionamiento correcto de la red, es el *bias* de cada neurona. Pero por el tipo *outout* que tiene el perceptrón, un pequeño cambio en el valor del bias puede generar una gran diferencia en su salida, debido a que es binario, 0 ó 1.

Es por esta razón que existen otros tipos de neurona con otras funciones de salida, como la neurona sigmoide. Ésta tiene la misma estructura que la neurona perceptrón, sin embargo

calcula la salida de la unidad según la función sigmoide, que se puede observar en la ecuación 1.3.

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{1.3}$$

Por consiguiente, se tiene que la salida de la neurona sigmoide queda determinada por la ecuación 1.4, evitando así el cambio drástico de la salida por un leve cambio del valor del bias.

$$\text{output} = \sigma(\mathbf{w} \cdot \mathbf{x} + b) \tag{1.4}$$

Con la estructura definida, ahora se deben ajustar los parámetros correctos para cada una de las neuronas, y así el sistema podrá entregar el valor correcto deseado según la situación.

Para realizar este ajuste, existen varios métodos, llamados de "entrenamiento", de los cuales, el más conocido y usado es el de *back propagation*. Éste, como a modo general, realiza los cambios comparando la salida deseada del sistema con la que entrega la red, para luego modificar desde la salida hacia la entrada los valores de cada una de las neuronas, hasta encontrar el balance correcto.

Existen varios tipos de redes neuronales, las que se basan en el mismo concepto de neurona explicado anteriormente. A continuación se procede a exponer dos de los más conocidos.

Redes neuronales convolucionales *CNN*

Las redes neuronales convolucionales siguen los mismos principios de funcionamiento que la red neuronal explicada anteriormente. La gran diferencia entre éstas, es que al inicio del proceso se tiene una capa de extracción de características, la cual es muy usada en áreas como el análisis de imágenes.

Su arquitectura está compuesta, a grandes rasgos, de tres etapas [14], como se puede observar en el esquema CNN que muestra la figura 1.3.

Las tres etapas de capa que componen la red CNN son las siguientes.

- **Capa convolutiva**: A través del uso de filtros(kernel), analiza los datos de entrada

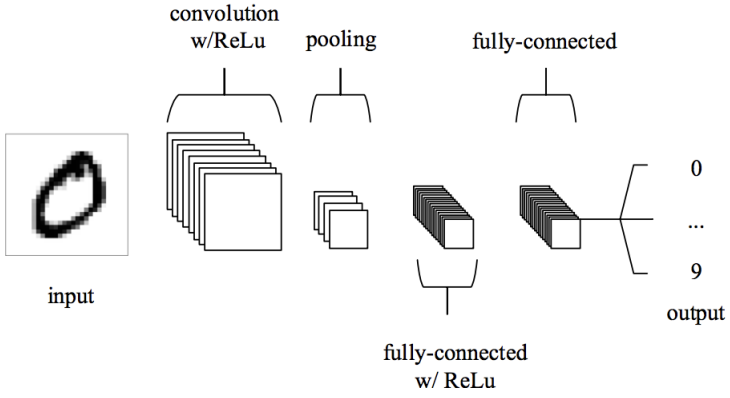

Figura 1.3: Red de Neuronal Convolucional

para realizar la extracción de características. Los kernel son matrices de menor dimensionalidad que la matriz de entrada, que se encargan de recorrer ésta ejecutando alguna acción activa que filtre los datos. La convolución ayuda a detectar de gran manera bordes, cambios de color y otros rasgos. Cabe destacar que según como se recorra este kernel en la matriz de entrada y que tipo de acción genere sobre él, puede producir cambios de dimensionalidad en capas futuras.

- **Capa de Agrupamiento o Pooling**: Tiene la misión de reducir la dimensionalidad de la entrada. Esto generalmente se hace recorriendo completamente la matriz usando alguna función sobre una sección de ésta. Una de estas funciones es la llamada *max*, la cual toma los valores de la sección donde se aplicará y deja el valor mayor de ésta.
- **Capa completamente conectada (fully connected)**: Realiza el proceso común de una red neuronal, con sus pesos y funciones de activación. Se encarga de realizar finalmente la clasificación de los datos.

Redes Neuronales Recurrentes

Las redes neuronales recurrentes, tienen la característica diferenciadora de recordar su estado anterior, guardando un vector escondido llamado h_t[10].

De esta forma, el valor de una neurona en el momento t, estará determinado por la entrada en el momento t y el estado de la misma en el momento $t-1$, como se observa en la ecuación 1.5.

$$\mathbf{h}_t = \begin{cases} 0 & t = 0 \\ f(\mathbf{h}_{t-1}, \mathbf{x}_t) & \text{otro caso} \end{cases} \tag{1.5}$$

Un esquema de como funciona este tipo de red se puede encontrar en al figura 1.4.

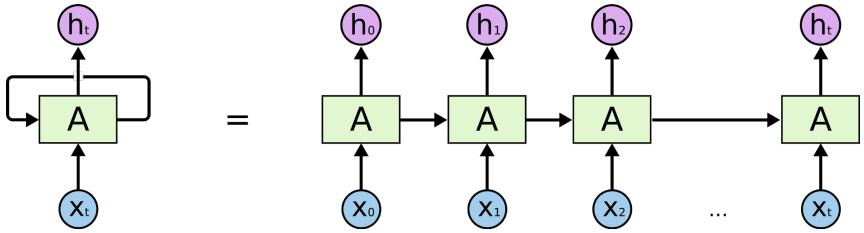

Figura 1.4: Arquitectura de modelo RNN

Este tipo de redes se ha hecho muy popular debido a su gran utilidad en áreas como sistemas de predicción de siguiente palabra, reconocimiento de discursos y análisis sentimental.

1.2. Redes U-Net

Uno de los grandes problemas que tienen los métodos de Deep Learning, es el hecho de que para obtener un buen entrenamiento de la red y, por ende, un buen resultado de salida, se necesita una gran cantidad de datos de entrenamiento y validación. Esta necesidad de datos conlleva a que en sistemas, donde la obtención de éstos es muy complicada, no sea viable la automatización de ciertos procesos con *machine learning*.

En casos como la biología, la clasificación por zonas o segmentación es muy común. Normalmente, se trabaja la clasificación de un píxel dentro de una imagen o el análisis por ventanas de estos píxeles[4], por ende el problema antes mencionado sobre la gran cantidad de datos, hace más complejo el uso de estos métodos en situaciones médicas, sin considerar el hecho de que para clasificar todos los píxeles de una imagen, tomará aún más tiempo, y generalmente se analiza más de una imagen por examen.

Dentro de este contexto, Olaf Ronneberger propone, en una publicación del año 2015, el desarrollo de un nuevo tipo de red convolucional, a la cual determina "U-Net" [15]. A rasgos generales, es una red *fully connected*, pero con una estructura modificada, la que permite un gran resultado en usos médicos.

Las redes U-Net llevan ese nombre debido a la forma en la que se puede caracterizar su uso,

como se puede observar en la figura 1.5, tiene la forma de la letra “U”. Estas redes tienen dos grandes características que la diferencian de las redes neuronales profundas, una de ellas es que se agregan nuevas capas de convolución a la red, lo que hará que suba la dimensionalidad de la red, esto implica que la salida será mayor y por ende con más datos.

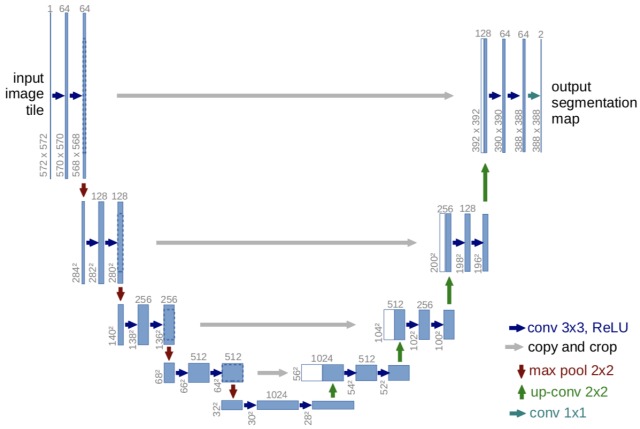

Figura 1.5: Estructura presentada de U-Net[15]

Finalmente, la otra gran diferencia, es el uso de un alto número de canales de características, las cuales aumentan por cada paso de convolución. Este detalle permite que el contexto de la imagen pueda ser traspasado capa a capa de forma más eficiente, lo que implica que se rescata más información, y entonces una mejor clasificación de los píxeles de la imagen.

En el caso de la red entrenada en dicho paper, se usa un método que aumenta los datos de entrenamiento, llamado *Data Augmentation*, el que es muy usado en el procesamiento de imágenes. Este método consigue aumentar los datos cambiado detalles de la imagen, como la orientación, la escala, entre otros. Además, se utiliza la función de pérdida como base del proceso de entrenamiento.

Lo que este tipo de redes logró, según los resultados del paper presentado[15], es que se obtienen grandes resultados en aplicaciones de segmentación biomédicas, usando pocas imágenes y en tiempos de entrenamiento razonables. Si este tipo de redes son usadas en diferentes contextos, y con los ajustes necesarios, se lograrán varios avances en diversas áreas de la medicina.

1.3. Perfusión Miocárdica

En el área de la medicina, hay una variedad de exámenes cuyo elemento final es una imagen, de la que se obtienen conclusiones para detectar o descartar problemas o enfermedades. Dentro de estos exámenes, está el de perfusión miocárdica, el cual es un examen que se hace al corazón para conocer el estado del sistema coronario.

A continuación se expondrán ciertos temas de importancia en el desarrollo de este trabajo.

1.3.1. Corazón

El corazón es un órgano ubicado entre ambos pulmones, en una zona denominada mediastino. Compuesto básicamente por 4 recámaras, dos llamadas aurículas y las otras dos ventrículos. Estás cámaras funcionan de a pares independientes, separadas un grupo a la izquierda y el otro a la derecha, aunque están conectadas por válvulas entre ellas[9].

El corazón tiene la importante función de trabajar como una gran bomba, que hace circular la sangre a través de todo nuestro cuerpo, transportando nutrientes y desechos. Encontramos un esquema del corazón en la figura 1.6.

El ciclo de funcionamiento de este órgano se resume de la siguiente forma:

- La sangre del sistema venoso llega por la vena cava hacia la aurícula derecha, la que mediante la válvula tricúspide, llega al ventrículo derecho, después de haber recorrido todo el cuerpo.
- Durante la sístole[1],, la sangre es expulsada por la artería pulmonar, hacia el sistema pulmonar, donde la sangre se oxigena y limpia.
- La sangre oxigenada llega a la aurícula izquierda a través de las venas pulmonares, y luego por la válvula mitral es llevada al ventrículo izquierdo.
- Durante la sístole, la sangre es enviada de vuelta al sistema circulatorio, mediante la arteria aorta.
- Desde esta arteria, salen dos ramas, las llamadas arterias coronarias, las cuales son las encargadas de alimentar al corazón. Posterior a esto, las venas coronarias llegan al ceno coronario, para luego entrar directamente a la aurícula derecha.

[1]Sístole: proceso de contracción del corazón, que produce la salida de sangre por los vasos sanguíneos

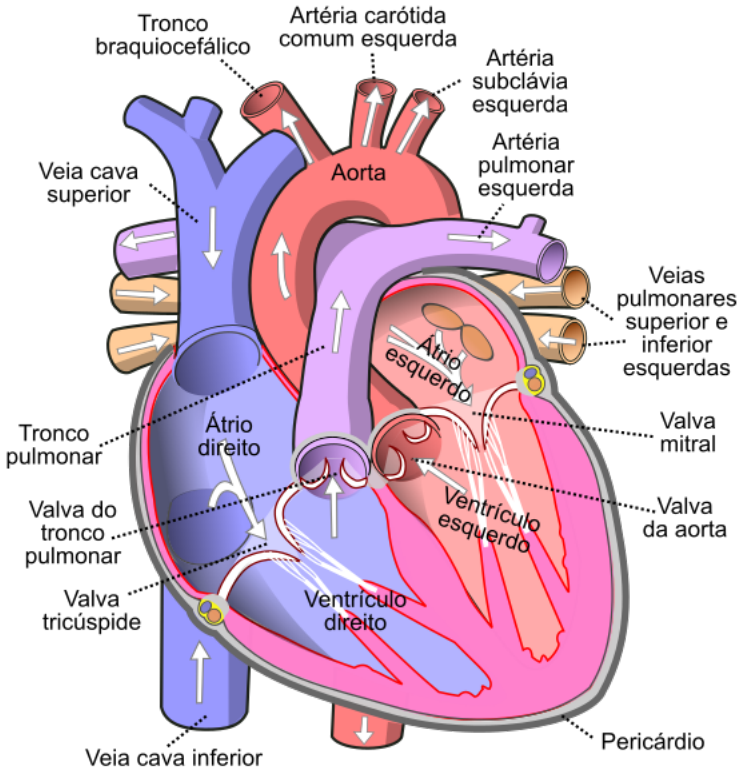

Figura 1.6: Esquema representativo de nuestro corazón[1]

Dentro de los músculos internos del corazón, se encuentra miocardio, es el encargado de bombear la sangre en el corazón y está ubicado rodeando preferentemente a los ventrículos, especialmente el izquierdo, que es el encargado de mover la sangre a todo el resto del cuerpo. Dentro de la imagen de la figura 1.6, se observa de color rosa, dentro del pericardio.

Las células del miocardio, como toda célula muscular, pueden contraerse y relajarse, provocando el movimiento. Además de esto, son capaces de generar un campo potencial propio, lo que le permite contraerse involuntariamente. Estas células son alimentadas por el sistema coronario. Gracias al sistema nervioso autónomo simpático(tensa las células) y parasimpático(relaja), el miocardio se contrae entre 60 y 100 veces por minuto en estado de reposo.

El miocardio cumple una importante función para el trabajo del corazón, por lo que es de vital importancia que esté sano y no genere algún tipo de enfermedad, como la isquemia miocárdica.

La isquemia miocárdica es el mayor causante de muertes en el mundo, se produce por la obstrucción de las arterias coronarias debido a grasa o plaquetas de grasa en los conductos de éstas, como se puede observar en la figura 1.7. Esto lleva a que el corazón no reciba la sangre suficiente y se produzca un ataque cardiaco. Por esto hay muerte celular, lo que es

una consecuencia irreversible.

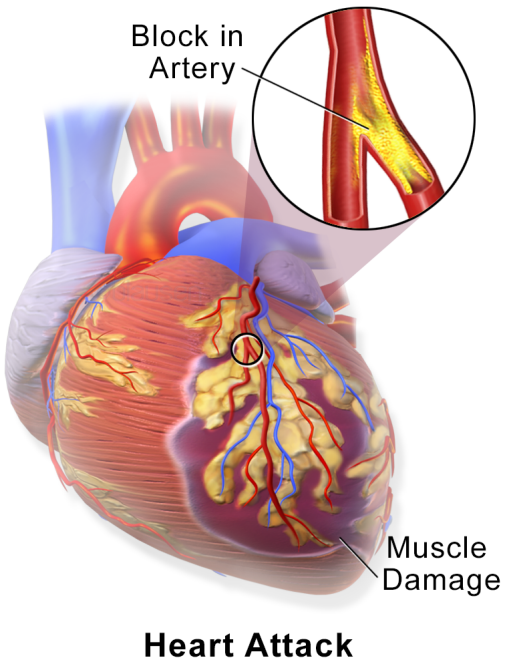

Figura 1.7: Infarto por obstrucción de arteria coronaria[3]

Está caracterizado por un fuerte dolor y opresión en el pecho, aveces acompañado de dolor en brazos, cuello, espalda, entre otros. Además, hay mareos, sudor, cansancio, nauseas entre otros síntomas.

Para detectar la existencia de una isquemia, se pueden hacer varios exámenes, como un electrocardiograma, muy efectivo para confirmar la presencia de obstrucciones, o una resonancia magnética, el cual solo se realiza en casos dudoso de esta enfermedad.

La resonancia magnética no solo permite detectar cuando la enfermedad está en proceso, sino que en ciertas condiciones puede servir como un examen preventivo, el que puede dar grandes resultados al analizar las imágeness.

1.3.2. Resonancia Magnética Cardiaca y Perfusión Miocárdica

Una imagen de resonancia magnética, es una técnica no invasiva, que utilizando el principio de la resonancia magnética nuclear, logra obtener imágenes de la estructura y composición de las partes del cuerpo humano que se quieran analizar estudiando la reacción de frecuencia de los núcleos de los átomos de nuestro cuerpo al ser expuestos a un campo magnético.

El equipo utilizado para realizar la resonancia magnética es de gran tamaño, ya que usa grandes imanes, para generar el campo magnético de gran intensidad requerido para el examen; el sistema puede ser costoso, como se puede ver en la figura 1.8, aunque sin duda, los resultados que se entregan son de alta verasidad.

Figura 1.8: Equipo de Resonancia Magnética [8]

Para la aplicación de las resonancias magnéticas, es muy común que se utilice un agente de contraste, basado en el gadolinio, inyectando este producto en el cuerpo, se logra que las zonas que se quieren revisar, se puedan observar de mejor forma en el examen. Cabe destacar que este agente es utilizado con mucho cuidado y con un análisis preciso del paciente, ya que podría causar problemas renales.

Como se puede extraer de este método, los resultados se basan en el análisis de imágenes. De éstas se pueden extraer varios resultados de interés, que según las investigaciones desarrolladas durante los últimos años, muestran que aun siendo una técnica muy poco invasiva, da resultados comparados con las que sí lo son.

Algunos de los análisis que se obtienen de las imágenes, son los cálculos de la función global ventricular, que es básicamente estudiar el volumen, masa y distribución dinámica de estos, analizar el flujo de corriente de sangre del corazón, y analizar marcas en el miocardio [2].

Los resultados de estos análisis, al compararlos con los estándares médicos obtenidos con métodos invasivos, según el trabajo de RJ van der Geest et al. [2], son parecidos, aún siendo poco invasivos, por lo cual es recomendable trabajar con ellos.

Además de los métodos antes mencionados, se nombra el método de Perfusión miocárdica. Inicialmente, la definición de perfusión según la RAE es la acción de introducir un líquido

lenta y continuamente a través de la sangre o conductos sobre algún órgano[2].

Dentro del contexto de la medicina y el análisis cardiaco, la perfusión miocárdica consiste en la aplicación de un contrastante, como el gadolinio, para poder observar el miocardio y su trabajo a través de una resonancia magnética. Este examen ha sido muy aceptado por ser poco invasivo, obtener imágenes con muy buena resolución, tener un amplio espectro de estudios, entre otras características [12].

Para obtener resultados de perfusión miocárdica, se debe pasar por un proceso de doble toma de datos. Inicialmente, el paciente al cual se le hace el examen debe estar en reposo. El segundo, consiste en aplicar un agente activante que haga trabajar al corazón a su mayor capacidad.

En cada uno de estos casos, se extraen imágenes en varias posiciones del corazón, llamados cortes o "slice". Para ambos exámenes, se deben tomar la misma cantidad de imágenes y en los mismos cortes, teniendo entonces material para comparación y estudio.

Gracias a estos exámenes, se pueden obtener varios tipos de características que son de interés para detectar enfermedades y problemas. En general, existen tres tipos de análisis que se pueden realizar sobre las imágenes, una vez que se haya detectado y separado el miocardio de la imágen, como se observa en la figura 1.9

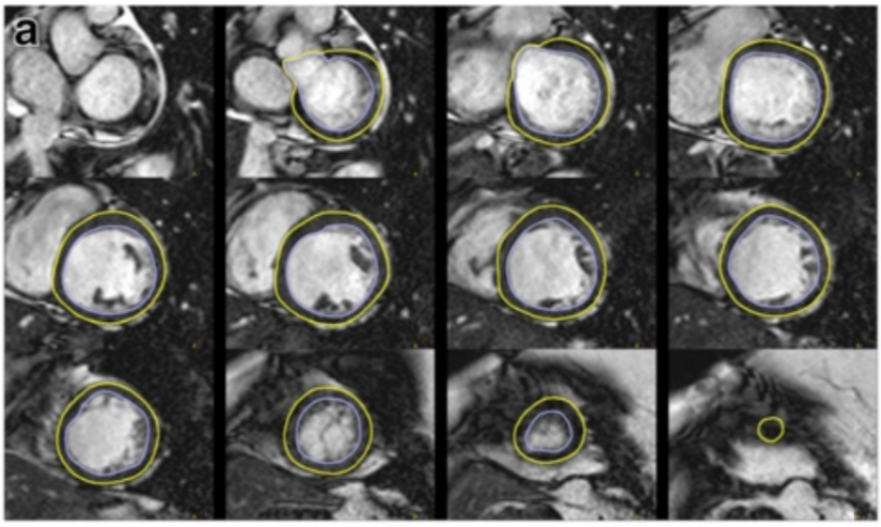

Figura 1.9: Imágenes de Resonancia Magnética al corazón, donde se identifica la zona del miocardio, para luego su posterior análisis [16]

El primer tipo es el llamado análisis cuantitativo. Este consiste en una análisis completo de

[2]Definición según RAE de perfusión: `https://dle.rae.es/?id=SbTNFIc`

volumen, flujos de sangre y sus diferencias, tanto en sístole como diástole[3], como por ejemplo el cálculo de volúmen que se extrae de la figura 1.9. Gracias a estos resultados, se pueden comparar con los comportamientos normales, y así, ver la existencia de algún problema.

El otro tipo de análisis es el cualitativo, esto implica que es un análisis visual. Teniendo ambos exámenes, tanto en reposo como en estrés, se busca comparar entre éstos diferencias o indicios de que existe un posible defecto en la zona [16], como se observa en los defectos encontrados en la imagen 1.10.

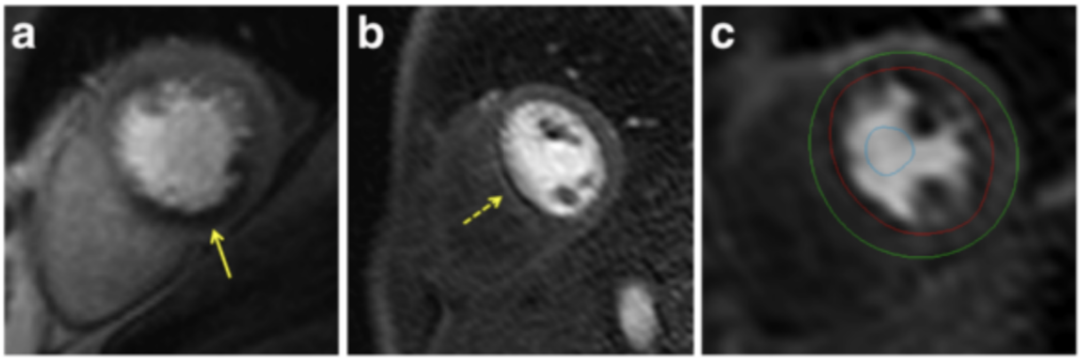

Figura 1.10: Detección de defectos sobre las imágenes de miocardio [16]

El último proceso es el semi cuantitativo. Durante este análisis, se procede a calcular de la imagen valores específicos que junto a la observación de la imagen, se pueden inferir de estos la existencia de alguna enfermedad o deficiencia.

Los datos calculados en este último análisis, son las curvas de intensidad versus tiempo, llamadas curvas de perfusión, y que se obtienen calculando el promedio de las intensidades de píxel en la zona de interés, que puede ser el miocardio, separado entre epicarpio y endocardio, y la zona de sangre interna de este, el "pool" de sangre, en el caso del ventrículo izquierdo.

Además, es útil de calcular la pendiente inicial de esta curva, el área bajo ésta y el máximo de intensidad que se registra en el examen. Junto a esto, es necesario el coeficiente entre las pendientes en estado de reposo y estrés.

Gracias a estos valores, se pueden hacer diagnósticos buscando defectos, sobre esfuerzos, retardos en el proceso de trabajo, entre otros problemas. Estos son los datos que se buscan analizar en el presente, y que se calcularán en la aplicación.

[3]Diástole: Periodo de relajación del ciclo cardiaco, donde baja la presión arterial

Capítulo 2

Estado del Arte

Al momento de realizar los estudios de curvas de perfusión, generalmente el trabajo de la selección de áreas y el cálculo de las variables de interés son realizadas por un experto de forma manual. Esto implica que una persona debe sentarse en un computador a analizar cada una de las imágenes por separado, y así de esta forma poder obtener el resultado esperado.

Existen varias aplicaciones dentro del área que se encargan de la visualización de los resultados de exámenes con imágenes, tanto pagadas como gratuitas. Éstas se caracterizan en general por entregar herramientas de cálculo no automático de utilidad, como selección de zonas, partición de estas y cálculos de áreas, entre otros.

Una de estás aplicaciones, que tiene funcionalidades para visualización de imágenes, es el proyecto "Horos"[1]. Este programa es gratuito para computadores de la marca Apple; y es *Open Source*, lo que implica que permite ser descargado sin ningún cobro y además acceder libremente a su código programación, con el fin de que la comunidad pueda implementar mejoras y arreglos.

Figura 2.1: Aplicación Horos

[1]Pagina de referencia de Horos: `https://horosproject.org/`

Según la misma fundación, su misión es entregar una herramienta de visualización de imágenes clínicas, con poder de procesamiento y apoyado por la comunidad. Sus ingresos están dados por clases y entrenamiento a personas además de donaciones.

Horos permite la visualización de las imágenes médicas, su análisis y cálculo correspondiente. Tiene muchas opciones para la visualización correcta de las imágenes, pero aún así, no contempla el uso de segmentación automática, aunque si existen otros complementos que pueden ser añadidos a la aplicación. En la figura 2.2 se puede observar la vista de la aplicación con una imagen en procesamiento.

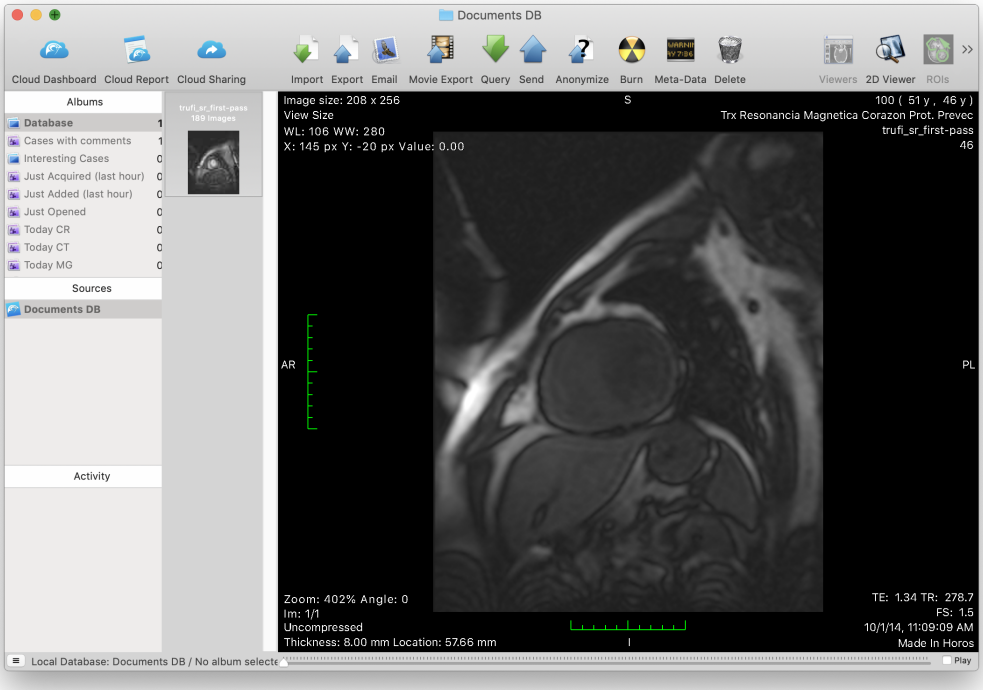

Figura 2.2: Vista de aplicación Horos en funcionamiento

Las imágenes con las que trabaja Horos, son un tipo de archivo estándar en el área, llamada DICOM(Digital Imaging and Communication On Medicine). Este tipo de formato es el protocolo usado en gran parte de los dispositivos y equipamiento que producen imágenes médicas.

Éstos archivos se caracterizan, no solo por guardar la imagen obtenida en el examen, sino que también es capaz de almacenar información extra que es de utilidad tanto para la visualización como para obtener información importante del paciente, el examen, el médico a cargo, entre otros datos que pueden llegar a ser importantes.

A continuación, se procederá a nombrar algunas de las características que para el presente trabajo son de gran utilidad. Cabe destacar que los nombres entregados en cada ítem son los reconocidos por la librería de python *pydicom*.

- "Pixel Array": Este parámetro entrega la intensidad de señal en el píxel indicado. Contiene la información del examen contenida en formato imagen. Puede tener valores tanto de 8 bits, como una imagen RGB estándar, como de 16 bits.
- "Slice Location" y "Image Position": Son parámetros que entregan el lugar donde la máquina en cuestión está realizando el examen. En el presente trabajo se utiliza el primero de ellos, para detectar cual corte transversal del examen de perfusión es al cual pertenece cada imagen.
- "Acquisition Time": Como su nombre lo indica, es el momento en el que se obtuvo la imagen. Éste es de utilidad para realizar los cálculos de curvas de perfusión, las cuales son de Intensidad de Señal versus tiempo.
- "Patient ID": Contiene el ID del paciente durante el examen. A parte de este dato, se pueden obtener datos personales como el nombre completo, rut, entre otros.
- "Window Width" (WW) y "Window Level/Center" (WL): Estos parámetros se pueden encontrar dentro de la información guardada en la imagen, y tienen la ofrecen parámetros de visualización de las imágenes. Mientras la primera de ellas modifica la intensidad de luz de la imagen, la segunda define el punto medio entre estas de la luz. De esta forma al modificar estos valores, se puede mejorar la luminosidad y contraste de la imagen [13].

  Su característica más importante, es el hecho de que, aunque se modifiquen estos valores de visualización, la información obtenida en el examen, guardada por el "Pixel Array" sigue intacta.

Aunque sea un estándar médico, no hay aplicaciones abiertas que trabajen con segmentación automática para perfusión miocárdica, y que, posterior a esto, realicen los respectivos cálculos. Los existentes son cerrados y sin acceso a otros recintos. Es por esto que urge la necesidad de tener un sistema capaz de realizar este proceso, y así mejorar el diagnóstico preventivo de la isquemia cardiaca.

Dentro de este contexto, es que un grupo de alumnos de la Universidad de Chile, durante el año 2018, trabajó en un algoritmo basado en las redes U-Net, capaz de hacer segmentación automática de miocardio, trabajando con el lenguaje de programación Python, y librerías como Numpy, Scipy y Scikit-learn.

Este código fue entrenado con imágenes de perfusión miocárdica sobre el ventrículo izquierdo, las cuales fueron segmentadas a mano para proceder al desarrollo del algoritmo de detección. Las imágenes son en estar DICOM, por lo que se puede proceder a su utilización sin mayores esfuerzos.

El algoritmo desarrollado es capaz de segmentar dos partes en específicas de las imágenes, como se puede observar en la figura 2.4. Una de ellas es el miocardio, mientras que la otra es la zona de sangre que envuelve el músculo. Estas dos zonas son de interés al momento de hacer los análisis pertinentes y nombrados con anterioridad.

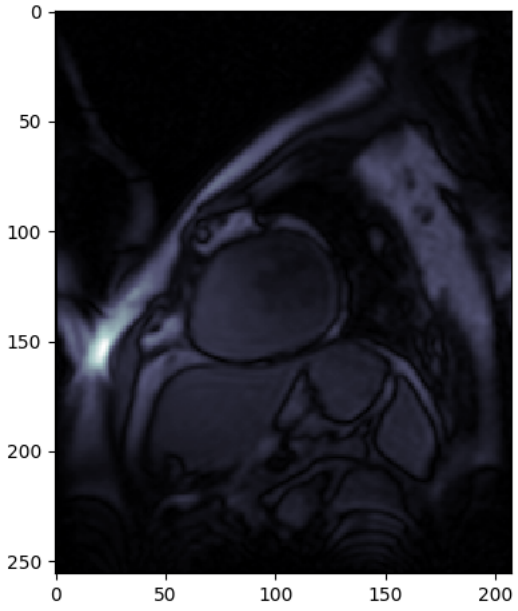

Figura 2.3: Imagen de resonancia magnética, utilizada por algoritmo de segmentación automática

Su funcionamiento se basa, como se dijo anteriormente, en el tipo de redes U-Net, las cuales son muy utilizadas en el área médica, debido a la poca necesidad de datos de entrenamiento. Es por esto, que el sistema fue probado y entrenado con un total de 180 exámenes, cada uno con una cantidad de variable de imágenes y cortes.

El procedimiento interno del algoritmo toma las imágenes de un examen completo, las divide por localización de corte para luego ordenarlas en tiempo, para posteriormente realizar la detección del miocardio y su interior imagen por imagen revisando píxel a píxel. De esta forma, cada imagen del examen es trabajada por separado, pero teniendo en consideración las imágenes adyacentes.

El código a utilizar es el empaquetamiento del trabajo, de esta forma, en ningún momento

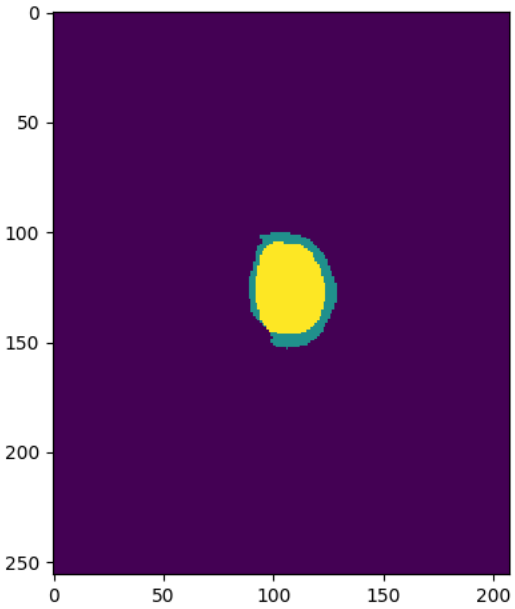

Figura 2.4: Predicción obtenida por algoritmo de segmentación de imagen de la figura 2.3

se tocará el modelo con sus variables durante el desarrollo de la aplicación a realizar. Es por esto, que se considerará el algoritmo como una caja negra.

Las entradas y salidas de esta caja negra tienen un formato de un arreglo de datos de 4 dimensiones (Z, T, X, Y), donde cada parámetro corresponde a:

- Z: Referencia a la locación del examen a realizar, el llamado "Slice Location". De esta forma, si es que el examen consta de 6 cortes distintos, la entrada debe contener en la posición Z valores entre 0 y 5, cada valor dando referencia a uno de estos cortes, los cuales están de forma ascendente según el valor por el archivo DICOM.
- T: Referencia al tiempo de adquisición de cada imagen, el llamado "Acquisition Time". Al igual que en el formato anterior, si un corte en específico tiene 30 imágenes, el valor T tendrá valores de 0 a 29, de forma ordenada según el valor de tiempo obtenido en el DICOM.
- X e Y: Estos valores corresponden a la imagen del examen, el "Pixel Array". Para el caso de la entrada del sistema, el valor que se encontrará en estos parámetros corresponde a los datos obtenidos en el DICOM, mientras que en la salida, será un arreglo con 3 valores, los cuales pueden ser:
  - 0: El píxel no fue clasificado.
  - 1: El píxel fue clasificado como miocardio.
  - 2: El píxel fue clasificado como el pozo de sangre del miocardio.

De esta forma, cualquier uso del algoritmo debe considerar estas entradas y salidas para obtener un resultado correcto.

En su fase de prueba, el algoritmo logró tener buenos resultados, teniendo en cuenta además de que la cantidad de imágenes utilizadas para entrenamiento y test son mucho menores que en otros métodos. En la tabla de 2.5, se logra ver que el promedio, tanto de validación como de prueba, tiene un valor sobre el 90 % para el modelo generado. Mientras que en la figura 2.6, se puede observar un ejemplo de la predicción realizada por el algoritmo sobre la zona del miocardio.

Conjunto de Validación (2 pacientes RIS-PACS HCUCH)			
Dice-coeff. Endocardio (LV D)	Dice-coeff. Pericardio (Myo D)	Dice-coeff. Miocardio	Mean (LV D, Myo D)
0,9402	0,9682	0,8621	0,9542
Conjunto de Test (6 pacientes RIS-PACS HCUCH)			
Dice-coeff. Endocardio (LV D)	Dice-coef. Pericardio (Myo D)	Dice-coeff. Miocardio	Mean (LV D, Myo D)
0,9143	0,9417	0,8001	0,9280

Figura 2.5: Coeficientes de validación DICE sobre el modelo U-Net desarrollado

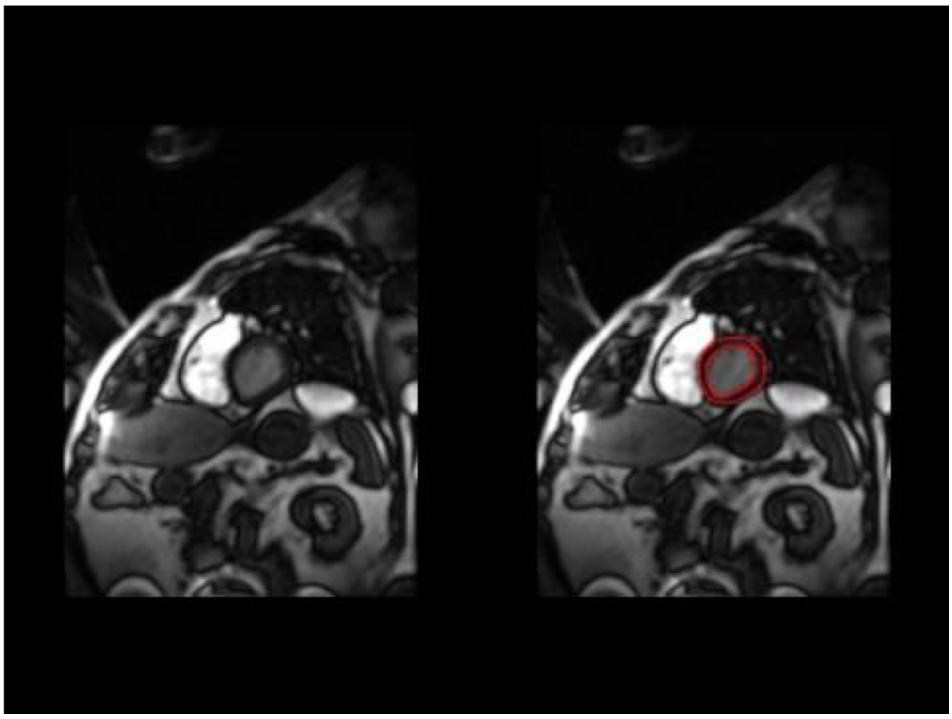

Figura 2.6: Predicción realizada por el algoritmo. A la izquierda la imagen original, a la derecha el borde de la predicción en rojo

Para el desarrollo de la aplicación, de manera inicial, se necesita una interfaz gráfica capaz de mostrar los resultados.

Python contiene una GUI nativa, llamada Tkinter[2]. Viene instalada en el paquete inicial de Python y contiene varios elementos que son de utilidad para el desarrollo de la presente aplicación.

Su funcionamiento básico consiste en la creación de un objeto llamado *Window*, el cual es la ventana de la aplicación y el que contiene la información visible de ésta. Es la interfaz principal, por lo que debe iterar en todo momento, mientras la aplicación esté en funcionamiento.

Dentro de esta ventana se deben integrar los elementos visibles de la interfaz. Para esto existen dos métodos, el primero es agregar los elementos directamente a la ventana, y la segunda consiste en utilizar un complemento especial, llamado Frame. Este consiste en una sección de ventana, la cual puede contener otros elementos. De esta forma, la ventana principal puede ser dividida en secciones, lo que es muy útil en aplicaciones complejas y que conllevan una gran cantidad de elementos.

Tkinter cuenta con varios elementos o widgets básicos que son de utilidad al momento de crear una interfaz, como botones, secciones de texto, vistas de imágenes llamadas Canvas, instrumentos de ingreso de variables, como cajas de opciones o de ingreso de texto, entre otras muchas.

Para la creación de una interfaz, solo queda el uso correcto de los widgets y la distribución de ellos, y de esta forma se puede observar una aplicación ordenada, donde el usuario pueda interactuar con ella libremente.

[2]Documentación Tkinter: `https://docs.python.org/3.7/library/tkinter.html`

Capítulo 3

Metodología

Para especificar cual será la metodología que se utilizará en este trabajo, primero se debe dejar claro cual es el objetivo que se busca. En este trabajo se busca el desarrollo de una aplicación, que usando segmentación automática de miocardio, dé herramientas para realizar el análisis de los exámenes de perfusión miocárdica de forma visual y semi cuantitativa.

Para realizar el sistema de segmentar el miocardio automáticamente, se utilizará el algoritmo basado en redes U-Net, nombrado con anterioridad durante el estado del arte. Este código será levemente modificado, para que el código pueda ser insertado en la aplicación de forma correcta.

3.1. Herramientas

Para crear la aplicación, a continuación se procederán a nombrar las herramientas que se utilizarán. Como punto de inicio, se debe elegir el lenguaje de programación en el que se trabajará.

Como el código de segmentación está trabajado completamente en python, se procede a usar Python 3.7 para la implementación completa del programa. De esta forma, se mantiene tanto el algoritmo de segmentación como el código del programa bajo el mismo lenguaje. Esto permite una rápida comunicación entre ambos sistemas, y de esta forma, tambien evitar problemas de traducción entre lenguajes.

Para la interfaz, se utiliza la librería nativa de python, Tkinter. Esta librería, como se

mencionó con anterioridad, ofrece nativamente todos los elementos que se buscan desplegar en pantalla, tanto botones, imágenes, cajas de texto, entre otros.

Además de Tkinter, al trabajar con listas de datos e imágenes, se utilizan librerías conocidas como Numpy, Pillow y OpenCV. Éstas son muy utilizadas en el área de imágenes y datos, y serán de mucha utilidad al momento de hacer cálculos u otras operaciones.

3.2. Objetivos

Teniendo las herramientas claras, se enuncia el objetivo general del trabajo, el cual es el desarrollo y diseño de una aplicación de ordenador, capaz de aplicar segmentación automática sobre imágenes de resonancia magnética en perfusión miocárdica, para luego calcular y entregar sus curvas de perfusión, junto con poder visualizar los resultados.

Entones, ahora se procede a enumerar los objetivos específicos, que ayudarán a completar el objetivo general.

- Mostrar imágenes ingresadas, tanto en pre visualización como en los resultados, junto a la segmentación realizada.
- Muestra en paralelo de las imágenes de estrés y relajo, para un análisis completo de este proceso.
- Sistema de partición de las zonas de interés en 4 secciones diferentes.
- Cálculo de las curvas de perfusión y parámetros de utilidad (Área bajo la curva de intensidad, máximo, pendiente inicial, coeficiente de pendientes), cuyos valores deben ser entregados al usuario.

Cada uno de estos objetivos específicos, dan pequeños pasos para completar el objetivo general planteado en este trabajo.

3.3. Desafíos

Para poder cumplir el objetivo impuesto durante este trabajo, existen diversos desafíos que deben ser abordados, los que llevan a completar la aplicación a cabalidad. A continuación, se procede a explicar cada uno de ellos, comentando su razonamiento e implementación.

3.3.1. Ajuste de algoritmo de segmentación

Una de las características importantes de esta aplicación, es el uso del algoritmo de segmentación automática desarrollada por el equipo de alumnos de la Universidad de Chile. Algoritmo que debe ser ingresado de forma eficiente al código de la aplicación.

Un elemento importante del código recibido, es el hecho de que al momento de crear el modelo de predicción, no es necesario entrenar la red con datos, ya que éste tiene la capacidad de guardar un estado anterior, el cual corresponde al último entrenamiento hecho a la red. Cabe destacar que este estado está ingresado en la aplicación, y puede ser reemplazado por uno mejor solo cambiando los archivos necesarios.

De esta forma, y como se comentó durante secciones pasadas, el código de segmentación será utilizado como caja negra, por lo que la aplicación debe preocuparse de cumplir los requerimientos de entrada y de salida de ésta.

La parte inicial del análisis debe ser capaz de leer los archivos DICOM, extraer los datos de tiempo, localización y la imagen para posteriormente realizar la conversión de los datos al arreglo de cuatro dimensiones (Z, T, X, Y), las cuales fueron detalladas en la sección de estado del arte.

De esta forma, el tratamiento inicial de los datos en formato DICOM es el siguiente:

1. Se lee cada uno de los archivos DICOM del examen, generando el vector de cuatro dimensiones. Se extrae la información de las imágenes, ordenando estas por tiempo, y dejando la cuarta dimensión sin datos, el cual sería la posición según el corte.

2. A continuación, se procede a asignar a cada imagen la posición de localización según el "Slice Location" de cada archivo DICOM, asigando a cada uno un número ascendente. De esta forma, cada imagen contiene el tiempo, su posición y sus datos de intensidad de señal.

3. Se procede a reordenar el arreglo según tiempo, configurándolos con número ascendentes según cada corte.

4. Se reordena el arreglo, de forma tal de cumplir el orden (Z, T, X, Y).

En este punto, se puede proceder a ingresar los datos a la caja negra del modelo, esperando la predicción de salida, la cual tendrá el mismo formato de cuatro dimensiones, solo que cambiarán los datos del array de la imagen.

Dentro del proceso de predicción, el código de segmentación realiza operaciones sobre el arreglo de entrada, los cuales se deben tener en consideración al momento de guardar las predicciones junto a su imagen correspondiente.

Una de estas modificación, es el hecho de que el código elimina las primeras imágenes de algunos *Slice*, en caso de que tenga más que el resto. De forma práctica, si es que un examen contiene 6 *Slice*, de los cuales dos de ellos contienen una imagen más que el resto, la primera imagen de estos cortes, según tiempo, son eliminados del arreglo, de forma que cada corte tenga la misma cantidad de datos.

Esto implica que al momento de guardar las predicción en el modelo de datos de la aplicación, hay imágenes que deben ser eliminadas de ésta, ya que no contienen resultado del algoritmo. De esta forma, se debe analizar cuales son los slice que contienen imágenes extra, y antes de guardar el resultado del código de segmentación, los datos sobrantes deben ser ignorados.

No está demás recalcar que los archivos DICOM están ordenados por tiempo según orden alfabético. Dentro de este orden, los archivos de los exámenes, se guardan sacando una foto a cada corte, para luego repetir el ciclo. Es por esta razón, que el código prefiere eliminar los primeros datos, y así no interferir en dicho ciclo.

El otro proceso importante que realiza el modelo de predicción, es hacer un relleno de ceros a la imagen, para que quede de forma cuadrada y de tamaño 256×256. Cabe destacar que el modelo no funciona con imágenes mayores a esta, por lo que se debe tener precaución con la imagen ingresada, aunque no se debiese dar el caso de una imagen con alguna dimensión mayor a 256.

El relleno de ceros es parejo hacia los lados de la imagen. Esto quiere decir que se agrega la misma cantidad de ceros a la derecha que a la izquierda de la imagen original. Lo mismo con la parte superior e inferior, aunque en las imágenes de muestra y prueba que se utilizaron en el proceso, nunca tuvieron un alto menor a 256.

Es por esta razón, que al momento de agregar las predicciones a las imágenes correspondientes, se deben eliminar los ejes extra agregados, ya que estos pueden interferir en los cálculos de las curvas de perfusión como también en la visualización de los resultados. Se debe tener en cuenta la condición antes explicada para hacer el proceso inverso.

3.3.2. Detección de epicardio y endocardio

Una vez la predicción esté agregada, se debe tener en consideración que los datos deben ser útiles para los resultados que se quieren obtener. Por ende uno de los trabajos que se deben hacer, es la separación de la zona de la sangre interna en una nueva variable de la imagen, de forma tal de poder ser asequible de forma rápida para futuros trabajos con esta.

Además, uno de los objetivos que se buscan en este trabajo es el de analizar el epicardio y el endocardio de forma independiente, por lo que la predicción realizada por el algoritmo está incompleta en este sentido, ya que solo detecta el miocardio completo. Es por esto, que se debe separar la esta zona en dos.

Lo que se busca es que dentro del "anillo" que es el miocardio, la parte interna sea clasificada como endocardio y la parte externa como epicardio. Es por esto que se debe aplicar un método sencillo, que no use tantos recursos y que pueda hacer la clasificación debida.

Para esto, se elige el procedimiento de analizar los píxeles aledaños de uno detectado como miocardio, buscando zonas que no sean parte de éste. La forma elegida para buscar los valores aledaños, es el de recorrer la zona en forma de anillo, como se puede observar en la figura 3.1.

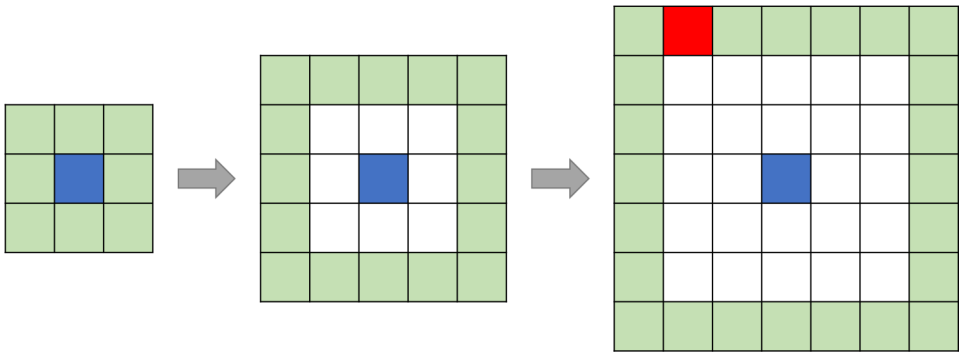

Figura 3.1: Lógica de búsqueda de píxeles cercanos

La lógica usada en la detección, se basa en ir buscando a los alrededores del píxel, agrandando el anillo hasta encontrar un valor que no sea 1 (ya que este número indica que es miocardio). Al encontrarlo, entonces:

- Si se encuentra un 2: está más cerca del "pool" de sangre, y por ende es parte del endocardio
- Si en encuentra un 0: está más cerca de la zona exterior, por lo que es parte del epicardio

Entonces, al momento de procesar la imágen de la predicción, basta con recorrer los elementos de esta y detectar su valor.

- Si es un 1: se realiza la división epicardio/endocardio, y se agrega donde corresponde
- Si es un 2: se agrega como valor en el "pool" de sangre
- Si es un 0: no hace nada

Se generan estos valores, teniendo en consideración mantener las dimensiones de la imagen original, con la detección en el punto correspondiente, de esta forma, los datos obtenidos en esta sección pueden ser utilizados para mostrarlos en pantalla.

3.3.3. Cálculos de Opciones de "WW" y "WL"

Como se mencionó con anterioridad, el arreglo de píxeles que entrega el formato DICOM, es un arreglo invariante, por lo que los valores que entrega, no están en relación a algún formato de imagen conocido.

Para esto, existen los valores de *Window Width (WW)* y *Window Level (WL)*, los cuales, como se refirió en el estado del arte, permiten modificar niveles de luminosidad y contraste en la vista de la imagen.

Para realizar la conversión del arreglo de datos a una imagen en escala de grises, con valores entre 0 y 255, el estándar de 8 bits, se procedió a utilizar de referencia la transformación para Tkinter de los archivos DICOM, encontrado en su página web [11].

De esta referencia, se debe tener en consideración, inicialmente que *Window Width (WW)* debe ser siempre mayor a 1, por lo que se debe hacer el alcance en el código.

La referencia, además, indica como se deben escalar los píxeles de la imagen, según los valores de WW y WL, con el siguiente recorrido.

1. Detectar el valor mínimo y máximo dentro del estándar de la imagen original, de forma que:

$$min\_val = WL - 0{,}5 - \frac{WW - 1{,}0}{2{,}0} \tag{3.1}$$

$$max\_val = WL - 0{,}5 + \frac{WW - 1{,}0}{2{,}0} \tag{3.2}$$

2. Escalar los valores. Se analiza cada valor del arreglo, si el valor del pixel es menor al min_val, entonces su valor es 0; si es mayor a max_val, entonces su valor es 255, y los valores intermedios, se escalan con la ecuación 3.3

$$valor\_final = \frac{valor\_original - (WL - 0{,}5)}{WW - 1{,}0} * (max\_val - min\_val) + min\_val \tag{3.3}$$

De esta forma, se obtiene la imagen en los valores de una escala de grises. Además, el traspaso, junto a la impresión de la imagen en la aplicación, se deben diseñar como función, para aplicar el cambio de valores manual por el usuario de forma rápida y poco engorrosa. Además, los valores de WW y WL son guardados en la clase "Slice", así que son las mismas para todas las imágenes de este conjunto.

Por otro lado, se debe imprimir sobre esta imagen la predicción realizada por el algoritmo. Esta se imprime como una imagen RGBA. Se recorre el predict, y se le asigna un color específico al lugar detectado, mientras que se asigna transparencia al resto. Para diferenciar, se elige un color por zona detectada:

- Zona de sangre: Rojo
- Zona de Epicardio: Verde
- Zona de Endocardio: Azul

3.3.4. Detección de Toque para división en sub sectores

Uno de los requerimientos, y por ende objetivo, de este trabajo es poder realizar la división del miocardio en 4 partes iguales, separadas equidistantes, para poder estudiar con más detalles las zonas de interés.

El sistema debe dar la posibilidad al usuario de elegir un punto en cada imagen, definido estratégicamente, para poder realizar la división. Este punto es extrapolado al resto de imágenes del mismo paquete, por ende, es deber del médico encargado elegir el punto de forma correcta.

Ahora, desde que se confirma la posición de los puntos, uno para cada imagen, se procede al cálculo de la división. Para realizarla, se debe calcular otro valor, el centro del *pool* de sangre. Para esto, se calcula el promedio de las posiciones, tanto en el eje X como en el Y y se trunca al valor entero. De esta forma, se obtiene el centro de la figura.

Teniendo estos dos puntos, se procede a dividir la zona, calculando la pendiente que se genera entre estos dos puntos, su extensión y la perpendicular que sale desde el radio, como se puede observar en la figura 3.2

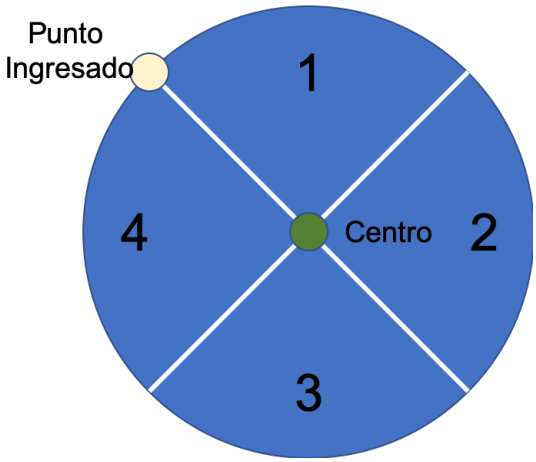

Figura 3.2: Esquema de separación de las zonas, dependiendo del punto de toque

De esta figura, se puede observar que la recta que pasa por el punto tocado y el centro, junto a la perpendicular, generan 4 zonas, con el mismo ángulo de enfoque. De este modo, se separan las cuatro secciones requeridas.

Ahora, se deben comparar todos los puntos pertenecientes a las zonas de predicción, para saber a que sección de estos pertenece. Para esto, se piensa, desde el centro calculado, como un sistema de coordenadas, donde existen cuatro cuadrantes, como se puede ver en la figura 3.3.

Siguiendo este concepto, se procede a comparar cada punto, según su cuadrante y su pendiente con el centro de la figura. Por ejemplo, en el caso de que el punto ingresado esté en el segundo cuadrante, como en la figura 3.3, se hacen las siguientes comparaciones.

- Si el punto está en el cuadrante I:
  - Si la pendiente es menor a la pendiente de la perpendicular, entonces pertenece a la sección 2, según figura 3.2
  - En caso contrario, pertenece a la sección 1
- Si el punto está en el cuadrante II:
  - Si la pendiente es menor a la pendiente del punto, entonces pertenece a la sección 1.
  - En caso contrario, pertenece a la sección 4
- Si el punto está en el cuadrante III:

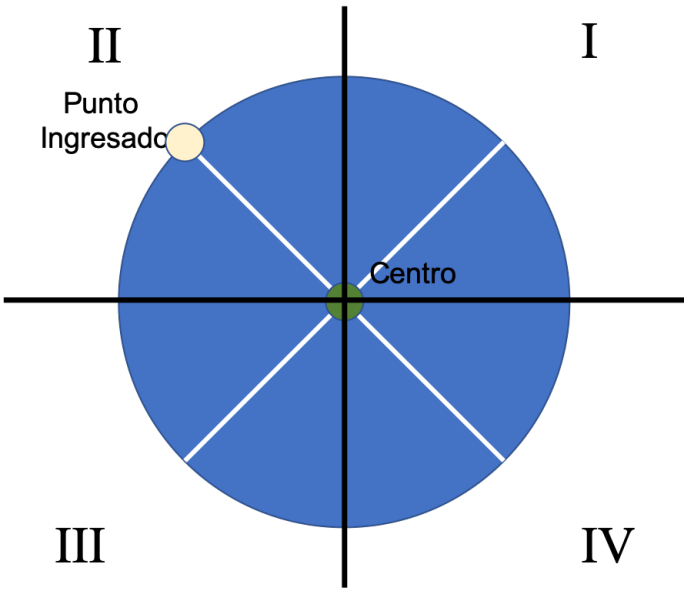

Figura 3.3: Esquema de división, según cuadrante

  - Si la pendiente es menor a la pendiente de la perpendicular, entonces pertenece a la sección 4.
  - En caso contrario, pertenece a la sección 3
- Si el punto está en el cuadrante IV:
  - Si la pendiente es menor a la pendiente del punto, entonces pertenece a la sección 3.
  - En caso contrario, pertenece a la sección 2

De esta forma, y extrapolando a las otras opciones del punto inicial, se puede separar la figura en sus secciones

3.3.5. Cálculo de Valores relevantes

Una de las ventajas de la forma en que se guarda el modelo de datos, es que en cada una de las clases se pueden almacenar los elementos que se necesiten a futuro, además de hacer funciones de cálculo rápidas en ellos. Gracias a esto, es fácil calcular el promedio de los valores de intensidad de cada imagen, poder guardarlos en una lista, y compararlos con la lista de tiempo de cada imagen.

La curva de perfusión, usa las dos listas nombradas con anterioridad, solo con la precaución de elegir la zona correcta de análisis, junto a los datos correctos que se requiere observar.

Con estas listas, con la de promedios y tiempo, se debe calcular los valores de interés que están definidos en los objetivos y en la zona de resultados. El formato de despliegue, se puede observar en la figura 3.4

	Rest	Stress
Área		
Máximo		
Pendiente		
Coeficiente		

Figura 3.4: Esquema de tabla que muestra resultados

Esta tabla sería la que está en la esquina inferior derecha de la pantalla de resultados, en la figura 4.3. Los valores se despliegan y cambian, dependiendo de la sección que esté viendo el usuario. Los valores son calculados como se expresa a continuación.

- Área bajo la curva: Se utiliza la función de Numpy llamada trapz, la cual recibe el arreglo de valores y de tiempo y entrega el área debajo de esta, haciendo una aproximación de las zonas discretas
- Máximo: usando la función de scipy, find_peaks_cmt, el cual encuentra los peaks de la función, utilizando cierto porcentaje de error esperado.
- Pendiente: utilizando el máximo anterior, la pendiente inicial de todo *Rise Time*, se calcula entre el periodo en el que pasa de estar en un 10 % al 90 % del máximo.
- Coeficiente: Se entrega el coeficiente obtenido al dividir la pendiente de la curva en *Stress* por la pendiente de la curva en *Rest*

3.3.6. Posición dentro de la Pantalla de Widgets

Este es un desafío que nace de la naturaleza de Tkinter. Los tamaños de los elementos que van en la pantalla no son fijos a las dimensiones que se les dan. Por ejemplo, el elemento "Label", el cual sirve para escribir texto en la pantalla o poner una imagen fácilmente, sus dimensiones dependen del contenido, variando entre píxeles si es que es una imagen, o tamaño texto, si es algo escrito.

Es por esto, que cada elemento ingresado en pantalla debe estar en un widget invariante, llamado "Frame". Se debe crear uno de estos por cada elemento en pantalla, pero al usarlo, y al definir que el objeto que va en este Frame se ajusta a los bordes de este, se puede asegurar que las dimensiones dadas para cada parte de la pantalla estén del tamaño adecuado.

Dentro de este mismo contexto, para asegurar la posición del elemento donde se quiere, se debe utilizar un posicionamiento llamado "grid", lo que implica que la pantalla se ve como una plantilla, donde se rellenan datos.

Para asegurar los posicionamientos adecuados, la grilla se adecua a su contenido, por lo que las dimensiones elegidas en cada caso deben ser precisas, además de tener que usar elementos vacíos para generar espacios.

Aunque un poco engorroso, es la solución elegida, ya que logra mantener el contenido en su posición sin variar al agregar o modificar elementos.

Capítulo 4

Diseño

Uno de los puntos más importantes en este trabajo, es el diseño que tendrá la aplicación. Con esto no solo se está refiriendo al diseño visual de la ésta, sino que también al diseño de las clases del código, las cuales son las bases de funcionamiento de la aplicación.

A continuación, se procederá a explicar ambos pasos de diseño, tanto visuales como estructurales, los cuales están pensados para ser atractivos al usuario y eficientes en el procesamiento de datos.

4.1. Diseño visual

Es muy importante para todo tipo de aplicación que su interfaz sea clara y completa, debido a que ésta es la comunicación directa del funcionamiento, en este caso el algoritmo de segmentación y las curvas de perfusión, con el usuario final.

Debido a esto, en cada momento de diseño, se debe tener en consideración los objetivos planteados para el desarrollo de las pantallas. Es por esto que inicialmente, se procede a elegir el diseño general que tendrá el sistema.

Basado sen el proyecto Horos, mencionado con anterioridad, se elige un diseño que consta de tres secciones, que funcionan como bases de la aplicación, y por ende de las pantallas del sistema. Estas secciones son pensadas según los requerimientos de cada aplicación. Las tres partes se pueden observar en la imagen 4.1.

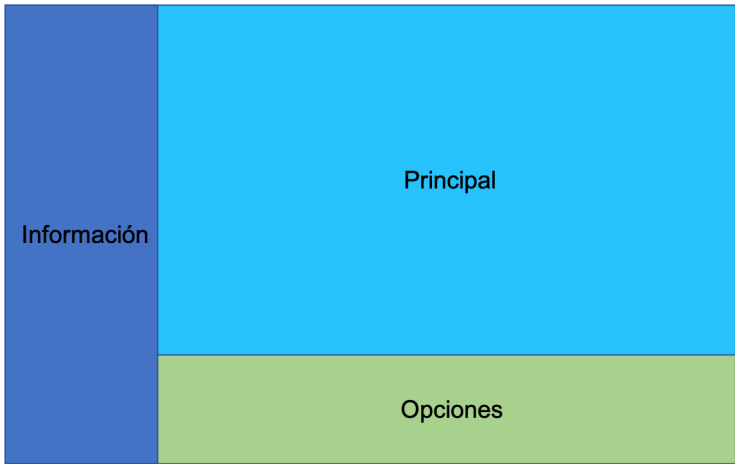

Figura 4.1: Diseño general de la aplicación, con las 3 secciones

En la imagen, de color azul, a la izquierda, se puede observar la sección “Información". Ésta está diseñada para desplegar información relevante al usuario, como las curvas a calcular, o los datos del paciente del examen que se está estudiando.

En segundo lugar, al lado inferior y de color verde se encuentra la zona de “Opciones", donde a grandes rasgos, se presentan opciones de acción que el usuario puede hacer con el general de la aplicación. Se entiende como un panel de administración.

Por último, de forma central y de color celeste, está la zona “Principal". En este lugar se presentarán tablas, imágenes y otras opciones de las imágenes. Al almacenar más información que el resto de las secciones, es más grande que los otros.

Teniendo el diseño general, se procede a especificarlo, para saber cuales serán las pantallas distintas que tendrá la aplicación. Para esto se analizan los objetivos, y así detectar grupos entre ellos.

En general, al analizar los requerimientos que se buscan, se detectan dos bloques de trabajo. El primero de ellos, consta en la toma de las imágenes del usuario, junto a la predicción de segmentación a realizar. Y por otro lado, se tiene el despliegue de la información obtenida desde las imágenes ingresadas. Es por esto, que se decide la creación de dos pantallas, las cuales se detallarán a continuación.

4.1.1. Pantalla de Bienvenida

Esta pantalla es la primera que verá el usuario al abrir la aplicación. El objetivo de ésta es la obtención de las imágenes del paciente para luego poder procesarlas mediante el algoritmo de segmentación. La distribución elegida para esta pantalla se observa en la figura 4.2.

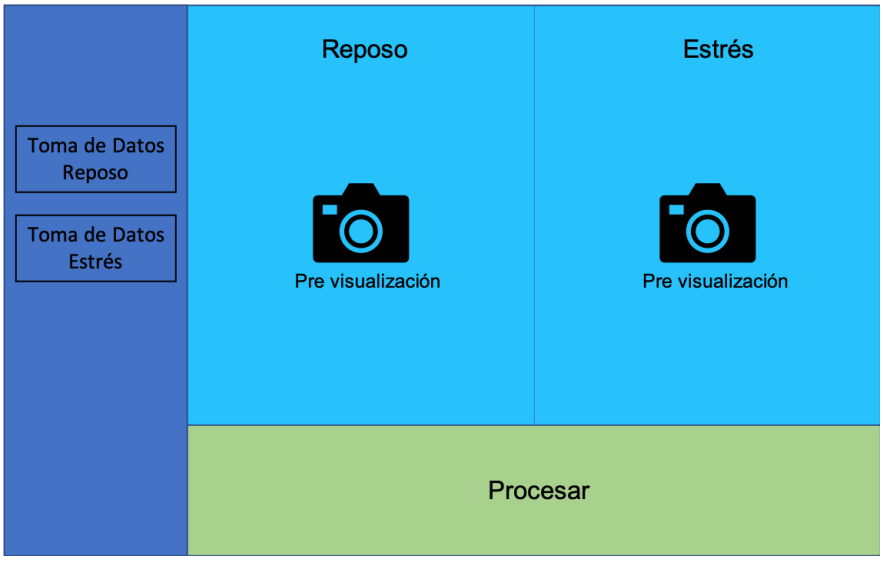

Figura 4.2: Diseño pantalla general de bienvenida

Por una parte, en la sección de “Información", se deja el lugar para que el usuario pueda ingresar las imágenes. Esta sección consta de dos botones, los cuales dan las opciones para ingresar la carpeta donde se encuentra tanto las fotos del tipo *Stress*(Estrés), como de tipo *Rest*(Relajo).

En la sección “Principal"se muestra la pre-visualización de las imágenes, una vez ingresadas por el usuario. Del total de imágenes por tipo, visualiza la primera de ellas, una por cada lado, como se ve en la figura 4.2. Las imágenes aparecen independientes al momento de ingresar el paquete correspondiente al tipo de examen.

Y por último, en la sección de “Opciones", se sitúa un botón con la opción de procesar las imágenes ingresadas. Esta opción solo aparece una vez que el usuario haya ingresado ambos paquetes. Al apretar el botón, el sistema comienza a procesar las imágenes, utilizando el algoritmo de segmentación, como los procesos internos para la muestra futura de datos.

4.1.2. Pantalla de Resultados

Una vez que se hayan procesado las imágenes, la pantalla mostrada cambia, para poder exponer los resultados obtenidos. Ésta muestra todos los resultados que se buscan del exámen, como las opciones necesarias para analizar los diversos matices. El diseño de ésta, se puede observar en la figura 4.3.

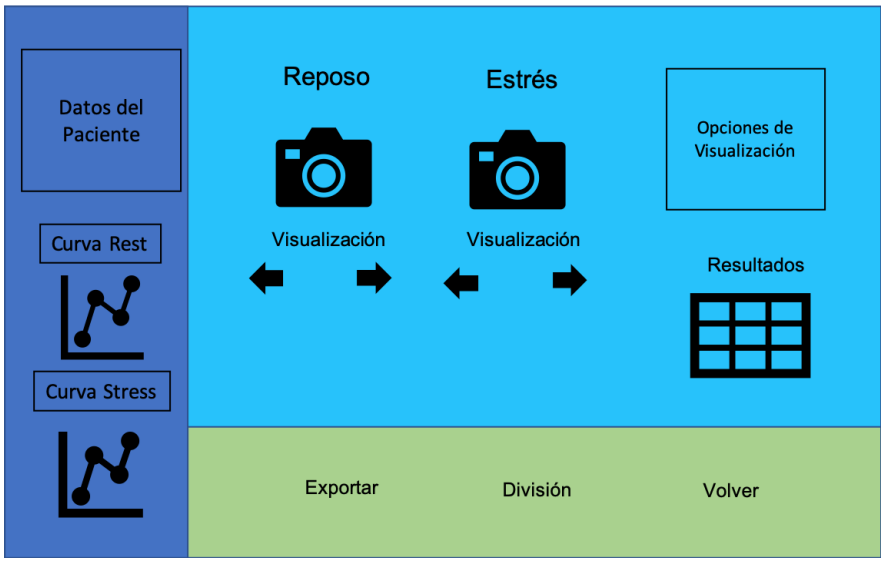

Figura 4.3: Diseño de pantalla de resultados

En esta pantalla hay variada información que debe ser analizada. Partiendo con la sección de “Información". En ésta, se procede a mostrar información relevante del paciente al que se le realiza el examen. Estos datos no son necesarios que sea agregada al sistema por el usuario, puesto que se puede obtener de los mismos archivos de las imágenes, ya que este tipo de información es parte del estándar DICOM. La información a desplegar es la siguiente:

- Nombre del Paciente
- Descripción del Estudio
- ID de la serie
- Descripción de la Serie

Estos datos fueron seleccionados según la información relevante destacada por el proyecto Horos, antes mencionado.

Bajo la información, se procede a mostrar las curvas de perfusión, tanto del examen en reposo como en estrés. Estas curvas, van debajo de su título respectivo y representan la curva Intensidad vs Tiempo, que es un requerimiento y objetivo de este trabajo. Además de las

curvas, en esta zona hay botones que permiten cambiar el lugar de análisis, entre la zona de sangre del miocardio, el epicardio y el endocardio.

En la sección “Principal", se deben mostrar las dos imágenes en paralelo, por un lado la de *Stress* y por otro la de *Rest*, junto a la segmentación encontrada por el algoritmo. Debajo de ellas existen opciones para avanzar en las imágenes en tiempo, para poder observar el proceso de avance. Ambos sistemas, son requisitos y objetivos de este trabajo.

A la derecha de las imágenes están las opciones de visualización, para modificar lo que se esté observando. Hay opciones visuales como el *Window Width* y *Window Level*, como tambien opciones para cambiar el corte de la imagen que se está observando, lo cual modifica tambien los resultados y las curvas que se están mostrando, para mantener sentido entre lo que se observa y lo que se calcula.

Por último en esta zona está la tabla de resultados, la cual mostrará los cálculos especificados en el marco teórico, y que son necesarios para un mejor diagnóstico. Las variables son:

- Área bajo la curva
- Máximo de intensidad alcanzado
- Pendiente inicial de la curva
- Coeficiente de pendientes Stress/Rest

Finalmente, en la sección de “Opciones”, se tienen tres botones de opción. A la izquierda un botón de exportar, el que exportará a PDF los datos obtenidos, lo cual no es parte del actual trabajo, pero si de una futura iteración.

A la derecha está el botón “Volver”, que permite, como dice la palabra, volver a la pantalla de bienvenida, para poder ingresar un nuevo usuario al sistema. Finalmente, al medio está el botón “División”.

Este botón permite la interacción del usuario con las imágenes, buscando que se seleccione un punto en específico, que permita la división de la zona segmentada en 4 partes, para poder ser analizadas por separado.

Para este proceso, los botones de abajo cambian de función, para confirmar, borrar los puntos o salir de la división. Además, a las opciones de visualización se le agrega la zona de división que se quiere observar en la imagen y que se desplieguen sus resultados. Con esto, se tiene el último objetivo buscado y cierra el ciclo de trabajo de esta aplicación.

4.2. Diseño de clases

El diseño interno es un punto muy importante en el desarrollo de cualquier aplicación. Según como se implemente el programa, éste tiene mejores opciones para realizar mejoras, realizar cálculos de variables necesarias, además de que sea legible para otro programador.

Es por esto que uno de los primero desafíos que a los que se debe enfrentar al iniciar una aplicación, es definir la estructura de ésta y el modelo de datos necesarios.

Dentro de este contexto, se decide que la aplicación esté mediada por una clase[1] principal. Esta clase es el controlador de la aplicación, y contiene todos los recursos que se necesitan para el funcionamiento, esto quiere decir, pantallas, modelo de datos, recursos externos, entre otros, como se puede ejemplificar en la figura 4.4.

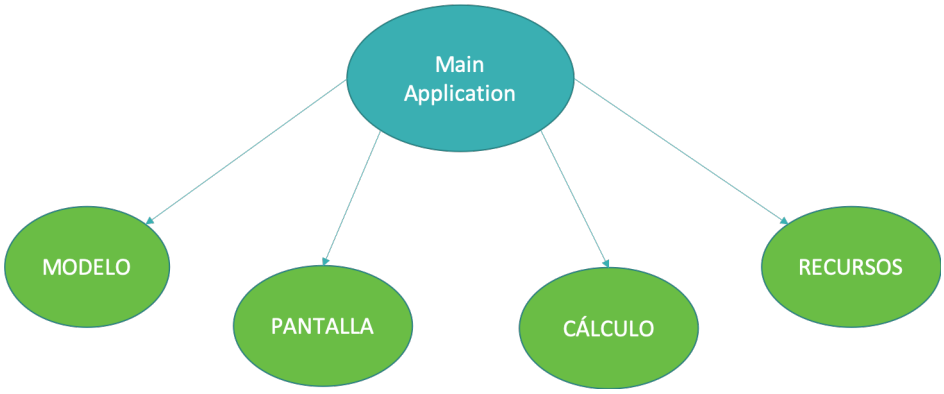

Figura 4.4: Estructura de Clase principal

Además de esto, la clase controladora, tiene la misión no solo de guardar los datos procesados, sino que también controlar el flujo de las pantallas y las secciones de ésta. En unos momentos se abordará más sobre estos temas.

Teniendo claro el sistema controlador, también se debe definir la estructura del modelo de datos. Para esto, es de mucha utilidad hacer un análisis general de las imágenes en un examen individual.

En el examen de perfusión se realizan dos mediciones distintas, una correspondiente a *Rest* y la otra a *Stress*. Por ende se tienen dos paquetes de imágenes relacionados a partes distintas del examen, pero con el mismo tipo de contenido.

Estos paquetes están compuestos por las imágenes DICOM de la resonancia magnética

[1]Clase: en informática, se refiere a una plantilla creadora de objetos informáticos, que representa un comportamiento o entidad

realizada. Además, el examen es realizado en varias secciones del corazón, lo que implica que las imágenes pertenecen a grupos según el corte del corazón al que pertenzcan. De esta forma, entonces se pueden agrupar estas imágenes en conjuntos llamados “Slice".

Si se hace un resumen del análisis anterior, se obtiene el esquema de la figura 4.5. De esta forma, la aplicación principal contiene paquetes de imágenes, las cuales están agrupados según su localización en el corazón.

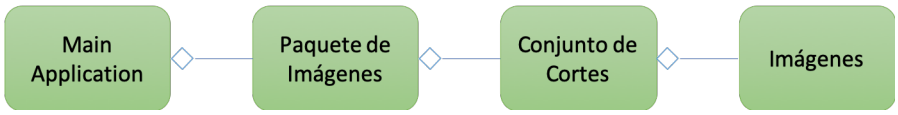

Figura 4.5: Esquema del modelo de datos

Lo importante de hacer esta separación, es que cada clase de este modelo tiene sus propias funciones, y por ende sus propios datos. Esto hace que modificar el código o agregar una nueva función sea más sencillo, ya que solo se debe analizar el grupo de interés del modelo.

A continuación, se procede a explicar cada uno de las clases del modelo de datos y de la vista de la aplicación, con sus funciones y datos internos.

4.2.1. Image_model

Como se subentiende de su nombre, es la clase que contiene la información de una imagen. Es la unidad básica dentro del modelo, y solo está preocupado de su propia información.

El conjunto de datos iniciales que contiene esta clase son los siguientes:

- Pixel Array: Es el arreglo de la imagen con las intensidades de señal del examen realziado.
- Location: Ubicación de la imagen dentro del examen realizado. Es el valor propio del Slice al que pertenece.
- WW y WL: Valores propios de WW y WL que contiene la imagen como datos internos del archivo DICOM.
- Image Acq Time: Tiempo de adquisición de la imagen correspondiente, en el formato interno del protocolo DICOM

Con estos datos se puede dar inicio a la clase en el momento de iniciarla. Ahora, hay otros datos internos que se van generando al momento de trabajar con las imágenes, las cuales se

pueden observar a continuación.

- Radio: es el radio de la imagen, calculado como el promedio de las posiciones del pozo de sangre.
- Predict: Arreglo que contiene la predicción original entregada por el algoritmo de segmentación automática.
- Endocardio: Arreglo que contiene la zona de endocardio detectada por el algoritmo de detección creado en este trabajo.
- Epicardio: Arreglo que contiene la zona de epicardio, detectada por el algormitmo de detección.
- Sangre: Arreglo que contiene la zona del pozo de sangre entregado por la segmentación automática.
- M: Diccionario que contiene las pendientes calculas, según el punto ingresado por el usuario. Contiene las pendientes de cada uno de los cuadrantes en los que se divide el miocardio, además de señalar el cuadrante original del toque.
- End_part: Diccionario que contiene arreglos de endocardio con cada una de las 4 zonas separadas según el toque ingresado.
- Epi_part: Diccionario con el contenido del epicardio, al momento de separar cada una de las zonas.
- San_part: Diccionario que contiene las 4 partes separadas por el toque del pozo de sangre.

Estos valores, aunque al crear la imagen estén vacíos, al momento de ir trabajando sobre éstas, se van a ir completando donde corresponda. Para esto, existen métodos internos que procesan el contenido y van calculando según corresponda. Los métodos más importantes a desarrollar se explican a continuación.

- arreglar_pad: Este método toma los valores de los arreglos de predict, epicardio, endocardio y sangre, para luego, buscando la diferencia con el pixel_array original, quitar los ceros que agrega el proceso de predicción.
- separacion_miocardio: Este método toma el predict agregado del sistema, y analiza sus píxeles, buscando si pertenecen al epicardio, endocardio o al pozo de sangre, para luego generar los arreglos correspondientes de estos. Para separar el miocardio entre epicardio y endocardio, se utiliza el método "mas_cercano".
- mas_cercano: Entregándole la posición del punto original y una cierta distancia a su alrededor, entrega si es que tiene cerca el pool de sangre o el exterior del miocardio.

- get_prom: Dependiendo de la zona de interés, recorre el arreglo solicitado y entrega el promedio de las intensidades de señal.
- dividir_miocardio: Se le entrega el punto ingresado por el usuario, para luego calcular su pendiente con el centro, calcular el resto de pendientes de los otros cuadrantes, y luego según el punto en cuestión, calcular a que partición corresponde cada punto del epicardio, endocardio y el pozo de sangre. Cada uno de los pasos intermedios son calculados con sus propias funciones, y finalmente, este método es un recopilación de todos los métodos utilizados.

Como se nombró, estás son las funciones generales del sistema. El detalle de cada uno de los métodos de esta clase, además de los campos internos, se puede encontrar en la imagen 4.6.

4.2.2. Slice_model

La clase que contiene imágenes, como se vio en la figura 4.5. Esta, como su nombre lo advierte, contiene las imágenes de un corte en específico del examen.

El conjunto de datos que contiene está clase es el siguiente:

- slice_loc: Valor de la localización del slice, según el valor del archivo DICOM.
- imgs: Arreglo que contiene objetos de la clase image_model. Esto implica que éste contiene las imágenes correspondientes al corte.
- pos_actual: Contiene la posición actual de la visualización del sistema de imágenes. Sirve para localizar cuál imagen es la que el usuario está viendo.
- data_varios: En total son 4 (sangre, epicardio, endocardio y miocardio). Cada uno de ellos es un diccionario que contiene los promedios de las intensidades de señal de todas las imágenes. Se puede encontrar de la zona completa, como también de las subdivisiones, luego de que el usuario haya ingresado el punto en la imagen.
- data_tiempo: Arreglo que contiene el arreglo en tiempo de las imágenes.

Al igual que en la clase anterior, hay varios de estos valores que inicialmente se encuentran vacios, pero que según se vaya trabajando con las imágenes, se van compilando valores. El único valor inicial es el slice_loc.

Los métodos más importantes de esta clase se explican a continuación.

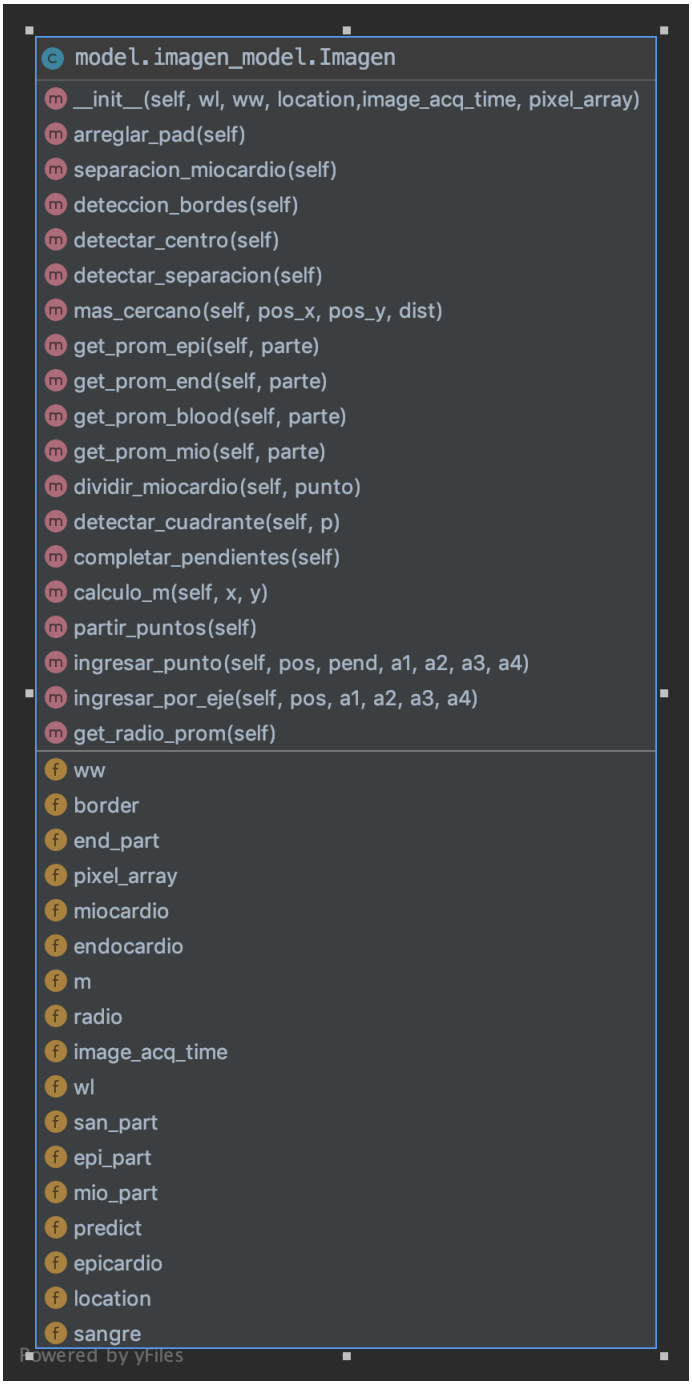

Figura 4.6: UML de la clase encargada de las imágenes

- agregar_img_slice: Agrega una imagen al arreglo. Además, ordena la lista según el tiempo de adquisición.
- cantidad_imgs: Entrega la cantidad de imágenes que se encuentran en el slice.
- agregar_predict_img: Entregada la predicción, se guarda en la imagen correspondiente, para luego aplicar sobre ésta la separación del miocardio (separacion_miocardio) y el arreglo de las dimensiones (arreglar_pad), métodos que pertenecen a la clase Ima-

ge_model.

- current_img: Al entregar las dimensiones requeridas de salida, toma la imagen actual, la transforma a una imagen en formato PIL, necesaria para poder ser mostrada en el canvas de la vista, con las dimensiones entregadas. Además, entrega la posición actual dentro del slice, junto con dos variables lógicas, que determinan si es que hay o no imágenes a la izquierda o la derecha de la actual, lo que sirve para activar o desactivar botones en la vista.
- aumentar_pos y disminuir_pos: Como su nombre lo indica, bajan o aumentan la posición actual, sin dejar que pasen los límites del slice.
- calcular_division_miocardio: Dado el punto de el usuario, aplica el método dividir_miocardio en cada una de las imágenes que contiene. Luego de esto, calcula los promedios de las zonas y los almacena.

El completo de métodos y variables internas de esta clase, se pueden encontrar en la imagen 4.7

4.2.3. PaqImgModel

Dentro de las clases del modelo de datos, es el más general. Contiene un conjunto de slice, que con estos se completan todas las imágenes correspondientes a esta parte del examen.

Los valores internos que maneja el sistema son los siguientes:

- tipo: String que sirve de referencia para el App Controller. Puede ser "Rest" o "Stress". Unico valor necesario para crear la clase.
- contenido: Arreglo que contiene todos los slice de este paquete del examen.
- actual_slice: Número del slice que está observando el usuario. Sirve de referencia para acceder al corte y la imagen en especifica.

Dentro de esta clase, se pueden encontrar los siguientes métodos:

- agregar_img: Entregada una imagen, según el valor de su localización, busca el slice al que pertenece y lo agrega usando agregar_img_slice. Si es que no encuentra el correspondiente slice, genera uno nuevo con la localización de la imagen ingresada.
- reiniciar_paq: Vacía el contenido de este.
- sort_slice: Ordena el contenido según el valor de la localización de los slice.

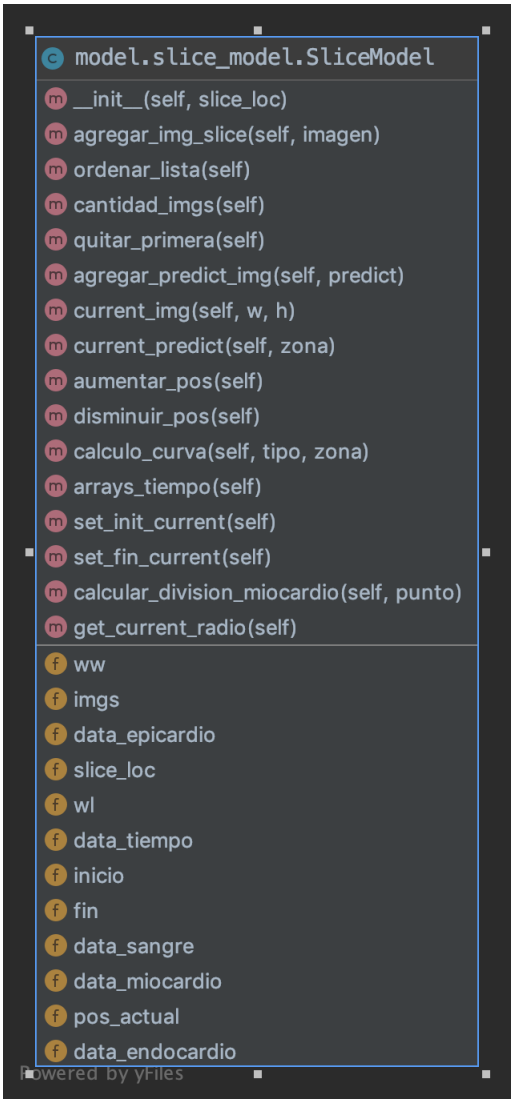

Figura 4.7: UML de la clase slice_model

- borrar_frames: busca los slice que tengan imágenes extra con respecto al resto, y borra las primeras para que todas tengan la misma cantidad.

- agregar_predict: borra los frames extra de ser necesario, y luego agrega cada paquete de predict al slice correspondiente, teniendo encuenta la salida del algoritmo de segmentación.

- calculo_division: Dado el punto entregado de división para el paquete, se entrega este punto a cada uno de los slice para realizar la división de 4 partes.

El esquema completo con todas las variables y métodos de esta clase, se pueden observar en la figura 4.8.

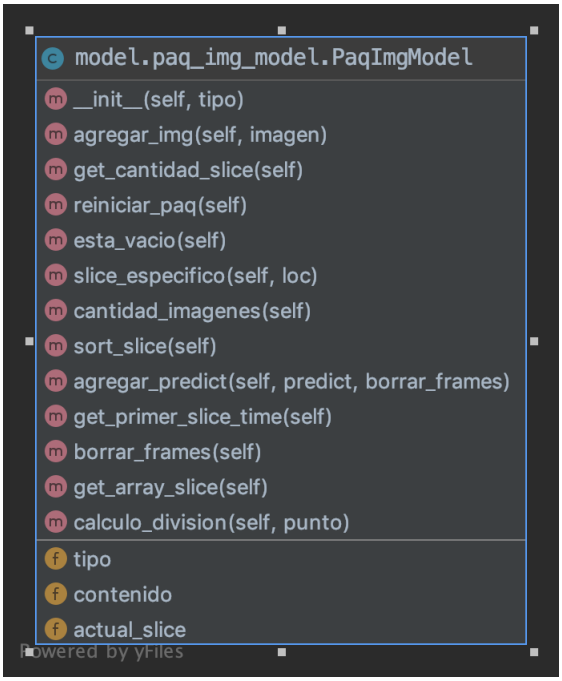

Figura 4.8: UML de clase PaqImgModel

4.2.4. AppController

Clase general y controladora de la aplicación. Está encargada de manejar el flujo entre pantallas, además de controlar el flujo de los datos que se utilizan durante ésta.

Dentro de los elementos internos que maneja esta clase, los más importantes son las siguientes:

- img_rest: Paquete de reposo del examen. Es el objeto de la clase PaqImgModel.
- img_stress: Paquete en estrés del examen. Es el objeto de la clase PaqImgModel.
- pantalla: es la ventana de la aplicación, donde todos los elementos visuales van integrados.
- frame_datos: Widget Frame que contiene los datos principales en la aplicación, como se observó en la figura 4.1.
- frame_img: Widget Frame principal, ubicado en el centro, como se observa en la figura 4.1.
- fram_opt Widget Frame de opciones, ubicado en la parte baja, como se observa en la figura 4.1.

Dentro de este controlador, existen varios métodos que son interesantes de destacar, que se explican a continuación.

- start: método inicial, que inicia la ventana de la interfaz. Se debe llamar inmediatamente si se quiere hacer funcionar la aplicación. Está inicia la pantalla de inicio.
- all_img: entrega un valor lógico que determina si es que se han ingresado ambos paquetes de imágenes al sistema correctamente. Sirve para la activación del botón de procesamiento de imágenes.
- predict_img: dados los paquetes, llama al modelo del algoritmo de segmentación automática, transforma los datos de las imágenes al formato (Z, T, X, Y) necesarios para el proceso y ejecuta la segmentación. Posterior a esto, agrega los resultados a cada paquete por separado. Al finalizar, inicia la pantalla de resultados.
- clean_pantalla: Borra los elementos de cada uno de los frames. De esta forma se pueden ingresar nuevos parámetros. Útil para cambiar de pantalla.
- nuevo_paciente: Borra los datos de anteriores y muestra la pantalla de inicio de la aplicación.

Para más detalles de este controlador, se puede observar su UML en la imagen 4.9.

4.2.5. UploadScreen

Esta clase es la encargada de mostrar al usuario los elementos correspondientes a la pantalla inicial de la aplicación. No se entrará en detalles en los elementos internos de la aplicación, ya que estos son los mismo que se pueden observar en el diseño visual de esta ventana.

Aún así, esta clase contiene métodos que ayudar al funcionamiento correcto de la aplicación, además de interactuar con los datos del sistema. A continuación, se proceden a explicar los más importantes:

- start: este método inicializa los parámetros visibles de la ventana, definiendo su ubicación en ésta, además de sus características.
- subir_img: Dos posibles, rest y stress. Guardan los datos en los paquetes correspondientes, en caso de que los archivos subidos sean los correctos. Luego revisa si es que el botón de procesamiento se puede mostrar, además de mostrar una previsualización de las imágenes subidas.
- check_button: llama al método all_img del controlador, para saber si mostrar o no el botón de procesamiento.
- print_img: Muestra la imagen previsualizada donde corresponda, para cada paquete.

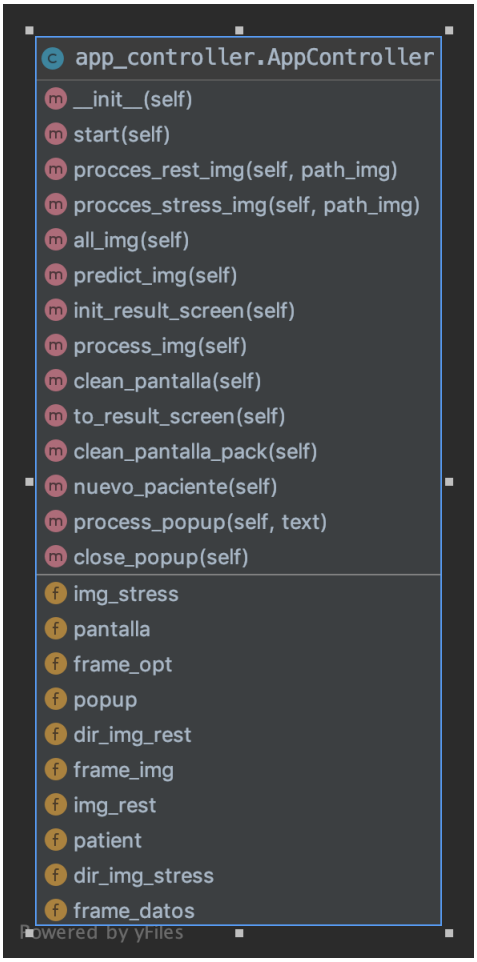

Figura 4.9: UML de Controlador de la aplicación

- agregar_paciente: Guarda los datos del paciente para mostrarlos posteriormente en el frame de datos.

Todos los métodos que contiene esta clase, se pueden observar en la figura 4.10

4.2.6. ResultScreen

Segunda pantalla de la aplicación. Esta clase está encargada de manejar y mostrar los datos y elementos de la ventana de resultados.

De la misma forma que la clase anterior, no se darán detalles de los elementos internos de la clase, ya que en su mayoría son los elementos que se pueden observar en el diseño visual de la aplicación. Aún así, se procederá a detallar los métodos más importantes de esta clase.

- start: junto con los métodos iniciar_lateral, iniciar_principal y iniciar_opciones, se

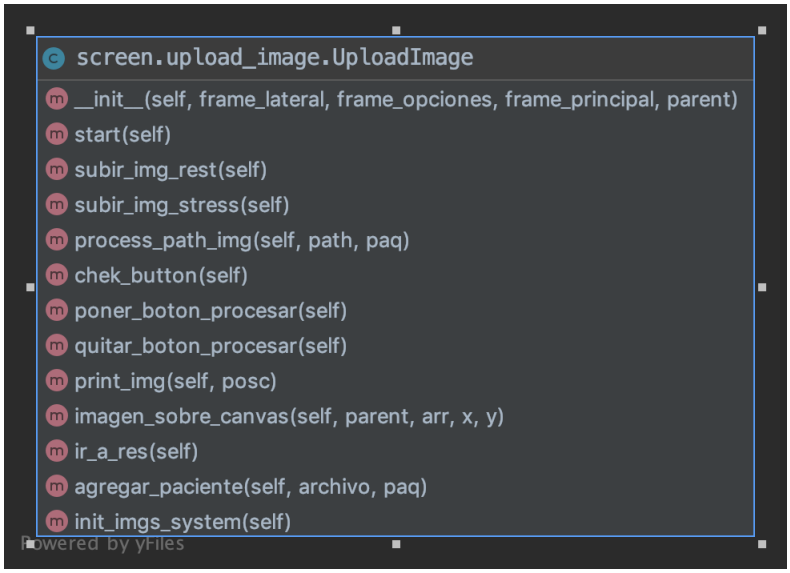

Figura 4.10: UML de clase UploadScreen

encargan de iniciar los elementos visuales de la pantalla. Además de esto, llama al método iniciar data, el que se encarga de entregar los datos iniciales de la pantalla.

- iniciar_data: Carga los datos del paciente, además de la primera imagen a mostrar, las curvas y los valores de la tabla.
- pressed_rest y pressed_stress: Detectan el toque sobre el canvas de las imágenes. En el caso de que la aplicación esté esperando el punto, este se dibuja sobre el canvas, de manera que el usuario pueda observar donde tocó.
- recarga_img: Recarga la imagen al momento de cambiar los valores de WW y WL entregados por el usuario.
- mov_img: hace funcionar las flechas de las imágenes, para observar la siguiente o anterior imagen sobre la que se está mostrando.
- imprimir_imagenes: Muestra la imagen correspondiente en el Canvas correspondiente. Se llama en cada momento que se necesita un cambio de vista.
- print_img_prediccion: imprime sobre la imagen del canvas la imagen de la predicción, mostrando la zona detectada por el algoritmo.
- curva_print: Muestra ambas curvas, del slice correspondiente.
- rellenar_tabla: Calcula y muestra los resultados de las curvas de perfusión.

El detalle completo de los métodos de esta clase, se pueden observar en UML de la figura 4.11.

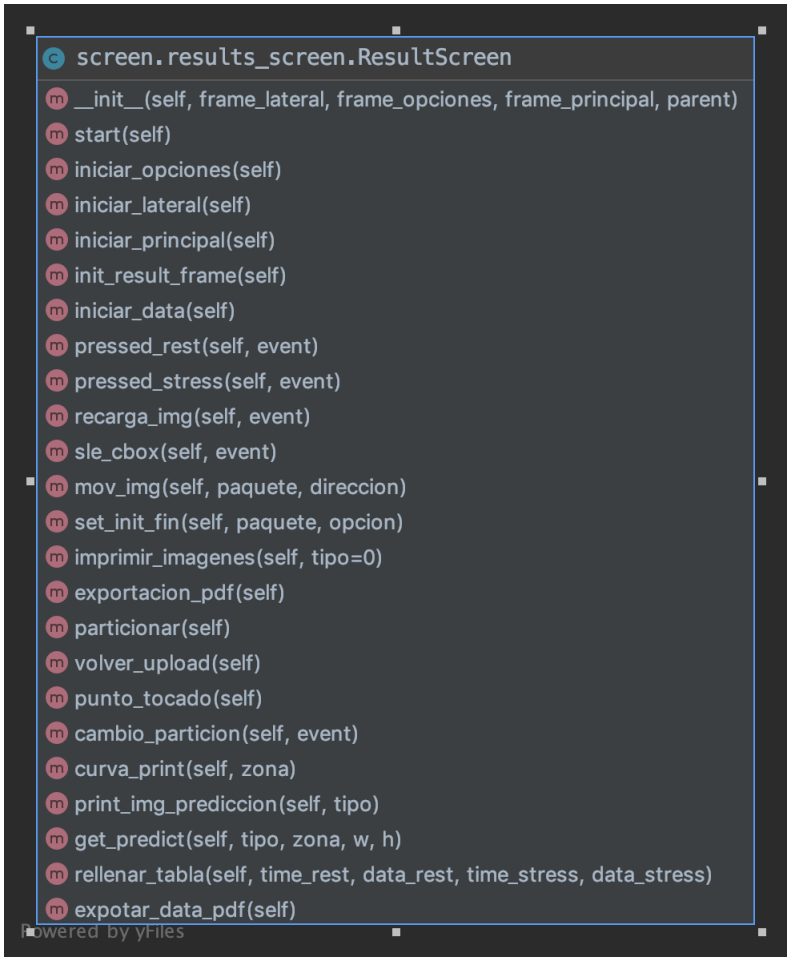

Figura 4.11: UML de clase ResultScreen

Capítulo 5

Resultados

A continuación, se presentan los resultados obtenidos en el desarrollo e implementación de la aplicación, según los detalles y desafíos mencionados en la metodología.

5.1. Desarrollo de las pantallas

5.1.1. Pantalla de Bienvenida

Al momento de iniciar la aplicación, se observa la primera pantalla, la cual se encuentra en la figura 5.1.

Ésta es la versión sin datos de la figura 4.2. Es por esto que las imágenes pre-visualizadas y la acción de preprocesar no se encuentran. Si se puede encontrar un nuevo botón, llamado "Usar últimos datos". Al interactuar con éste se ingresa directamente a la pantalla de resultados, usando el último examen ingresado.

Al momento de interactuar con alguno de los botones de la sección de "Información" para subir imágenes, lleva a la pantalla predeterminada para seleccionar la ubicación de los paquetes necesarios. Esta ventana de diálogo se puede observar en la figura 5.2.

Posterior a que se agreguen ambos paquetes de imágenes, la pantalla de bienvenida luce como en la figura 5.3.

Como se observa, ahora se encuentran la pre-visualización y el botón que permite comenzar

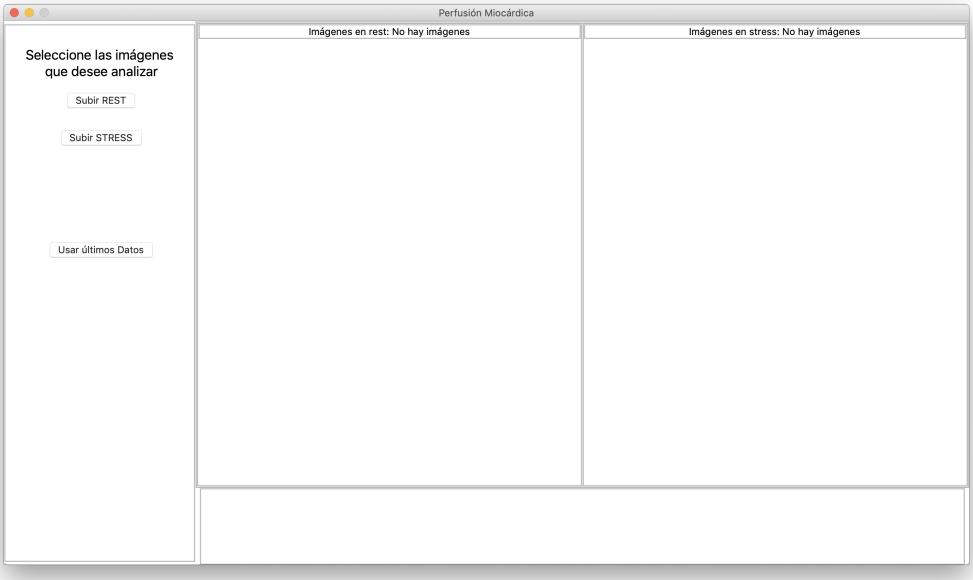

Figura 5.1: Pantalla inicial de la aplicación

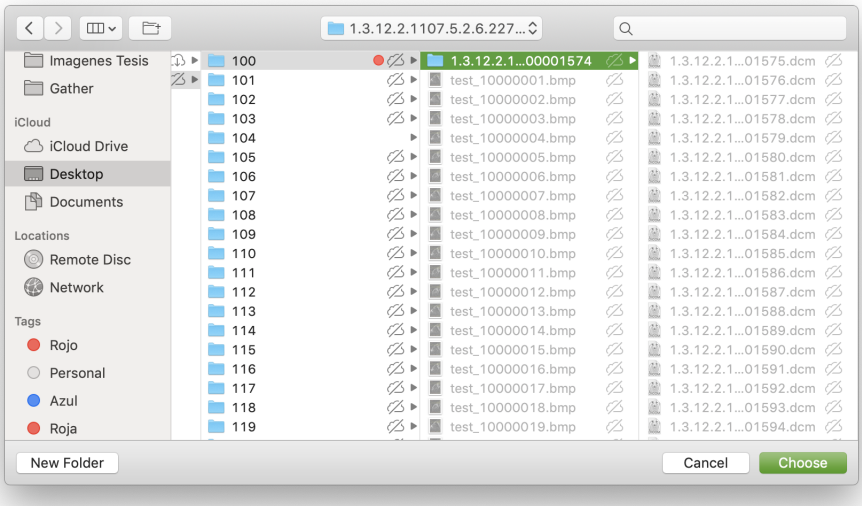

Figura 5.2: Ventana de dialogo de selección de imágenes

el procesamiento. Con esto, se cumple el diseño especificado en la figura 4.2 y por ende el objetivo específico de pre-visualizar la imagen antes de ser procesada.

Como complemento, se ha agregado un detalle al lado del título de la imagen. Se puede observar en la figura 5.4 que la cantidad total de imágenes iniciales encontradas en el paquete se encuentra en este lugar. En caso de no obtener imágenes, despliega el mensaje “No hay Imágenes”.

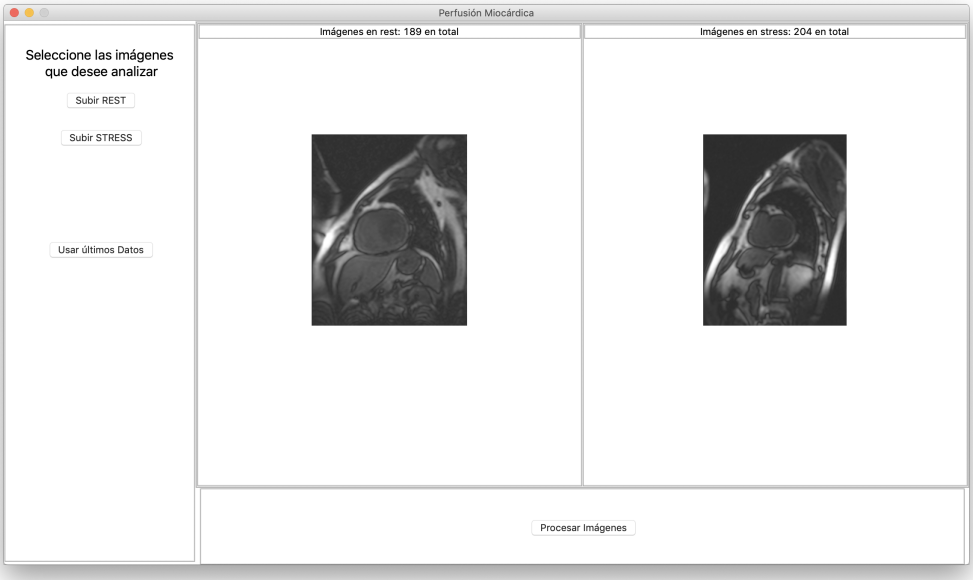

Figura 5.3: Pantalla de bienvenida, con los datos ingresados

Figura 5.4: Título de pre-visualización en pantalla de bienvenida

5.1.2. Pantalla de Resultados

Luego de procesar las imágenes, se procede a desplegar la pantalla con los resultados, la cual se presenta en la figura 5.5. Esta pantalla cumple los requerimientos establecidos por la figura 4.3 durante la metodología.

Analizando la pantalla por zonas específicas, en la sección de información, se encuentra la información del paciente, que se puede ver con más detalles en la figura 5.6. Se despliegan los valores nombrados en la metodología, que en este caso están ocultos, para proteger los datos del examen.

Debajo de la información del paciente, se encuentran las curvas de perfusión calculadas, mostradas en la figura 5.7. Las primeras en ser desplegadas son las correspondientes a la zona de sangre de las imágenes que pertenecen al primer corte del examen.

Por sobre las curvas hay tres botones, cada uno de ellos indica la sección que se analiza en las curvas. Al apretar uno de éstos, se actualizan los valores de las curvas según la sección

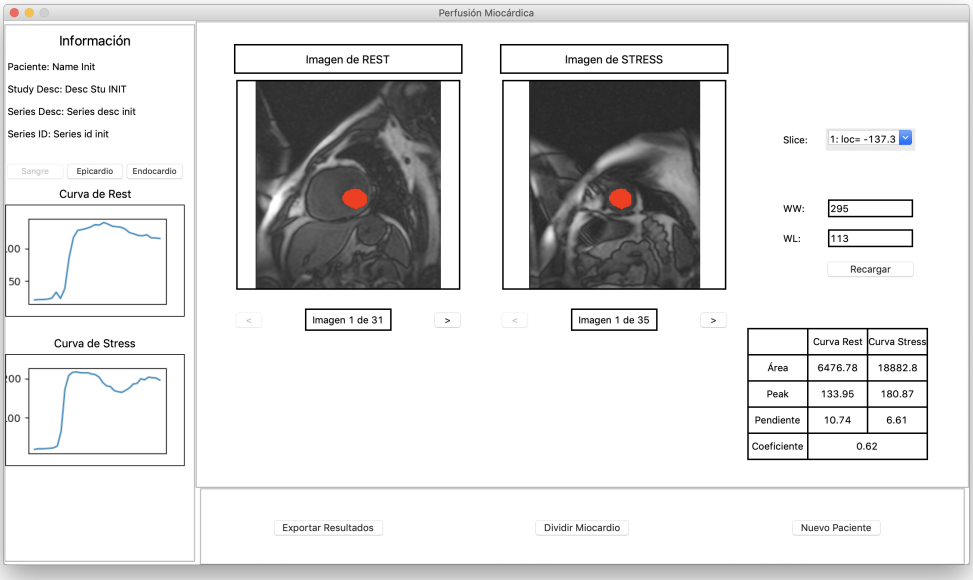

Figura 5.5: Pantalla de resultados de la aplicación

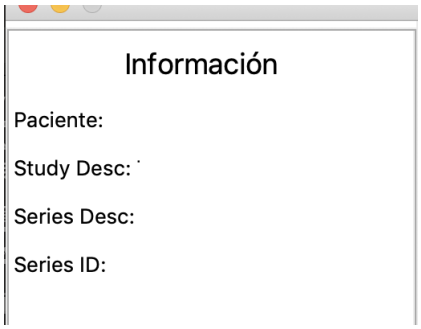

Figura 5.6: Sección de información del paciente

seleccionada, además de los parámetros de la tabla de resultados y la predicción desplegada en las imágenes, como se verá posteriormente.

Cabe destacar, que el modo seleccionado queda desactivado hasta activar otro, como se puede ver con la opción de "Sangre", que es la inicial. Con esto se cumple otro objetivo, que es el de mostrar las curvas de perfusión extraídas de las imágenes.

En la sección de "Opciones", se pueden encontrar los 3 botones especificados en la metodología, determinados en la figura 4.3. Los detalles de los botones se pueden observar en la figura 5.8. Además, se debe destacar que los botones cumplen las funciones que fueron explicadas en la sección de metodología.

En la sección "Principal", se observa la tabla con los valores calculados, la cual se pue-

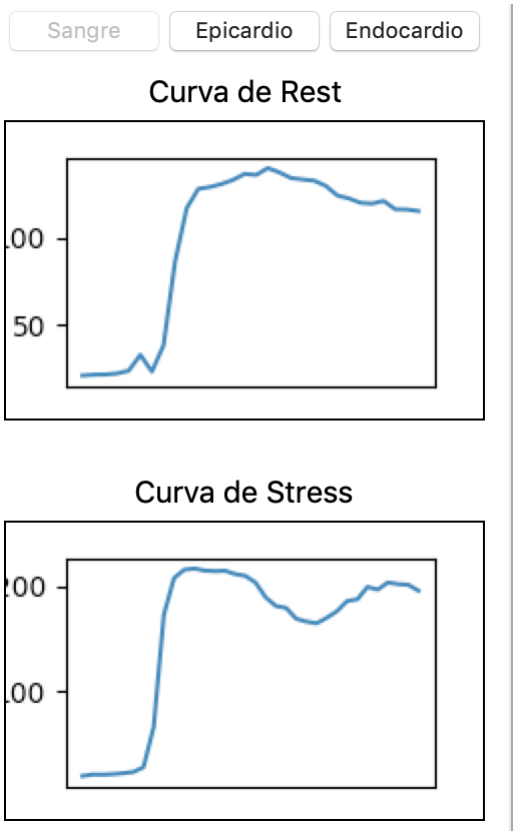

Figura 5.7: Curvas de perfusión calculadas en la aplicación

Figura 5.8: Botones iniciales de sección "Opciones" en la pantalla de resultados

de observar con más detalle en la figura 5.9. Como se dijo anteriormente, los datos están relacionados a las curvas mostradas en la sección "Información".

	Curva Rest	Curva Stress
Área	3330.66	6757.62
Peak	87.27	103.75
Pendiente	3.61	6.82
Coeficiente	1.89	

Figura 5.9: Tabla de Resultados mostrada al usuario

En esta sección, también se encuentra la visualización de las imágenes en paralelo. Ésta

tiene dos imágenes sobrepuestas, una de ellas corresponde a la indicada a mostrar según la posición; la otra, es la predicción según la zona que se esté analizando , que también está controlada por los botones de las curvas. En la figura 5.10 se ve la visualización en caso de mostrar la zona de sangre, en la figura 5.11 la zona de epicardio y en la figura 5.12 la zona de endocardio.

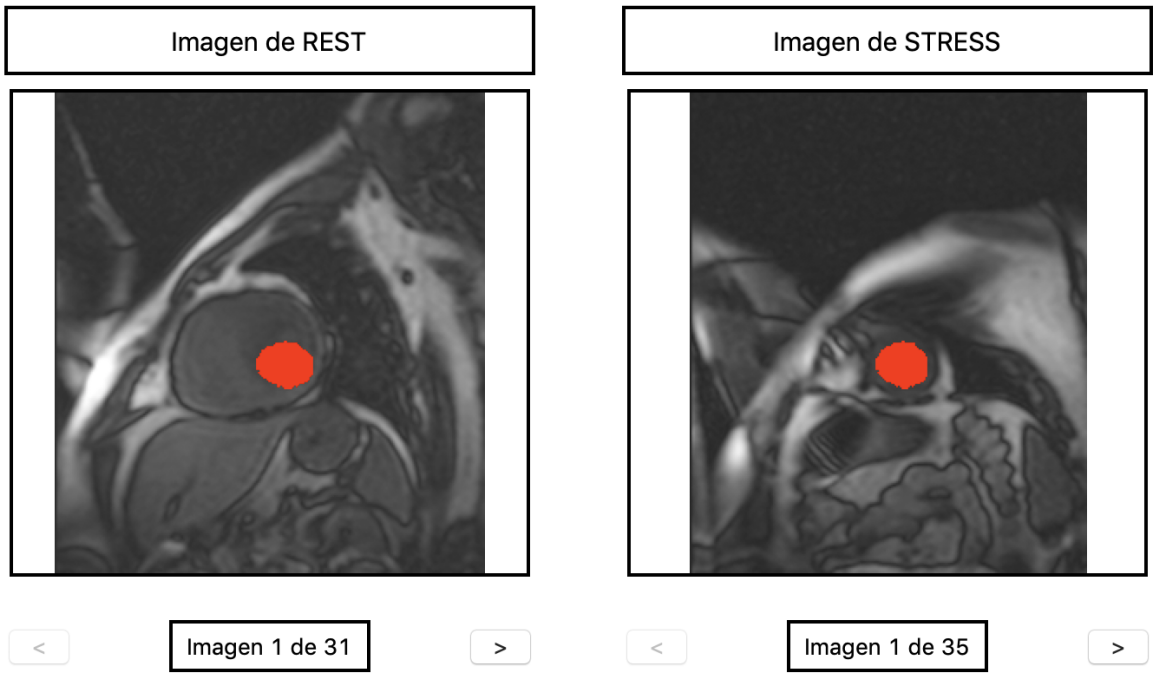

Figura 5.10: Imagen visualizada, con la sección de sangre mostrada

Debajo de cada imagen, se informa cual es la imagen que se muestra del conjunto completo. Además, a los lados hay dos botones, los cuales permiten desplazarse por el conjunto, lo que modifica la imagen que se está mostrando, además de la predicción y el número de la imagen actual.

Esto cumple otro objetivo, el de mostrar la imagen junto a su predicción, además de poder desplazarse por las imágenes para realizar el análisis visual correspondiente.

Al lado de esto, se tienen las opciones de visualización de las imágenes, que se puede observar más específico en la figura 5.13. En ésta, se puede observar el seleccionador de "Slice", el cual al ser cambiado modifica la tabla de resultados, las curvas y las imágenes, mostrando los datos correspondientes.

Además, se encuentra el input de WW y WL. Cambiando los valores por números enteros y luego apretando el botón recargar, la imagen se actualiza, permitiendo cambiar contraste y luminosidad.

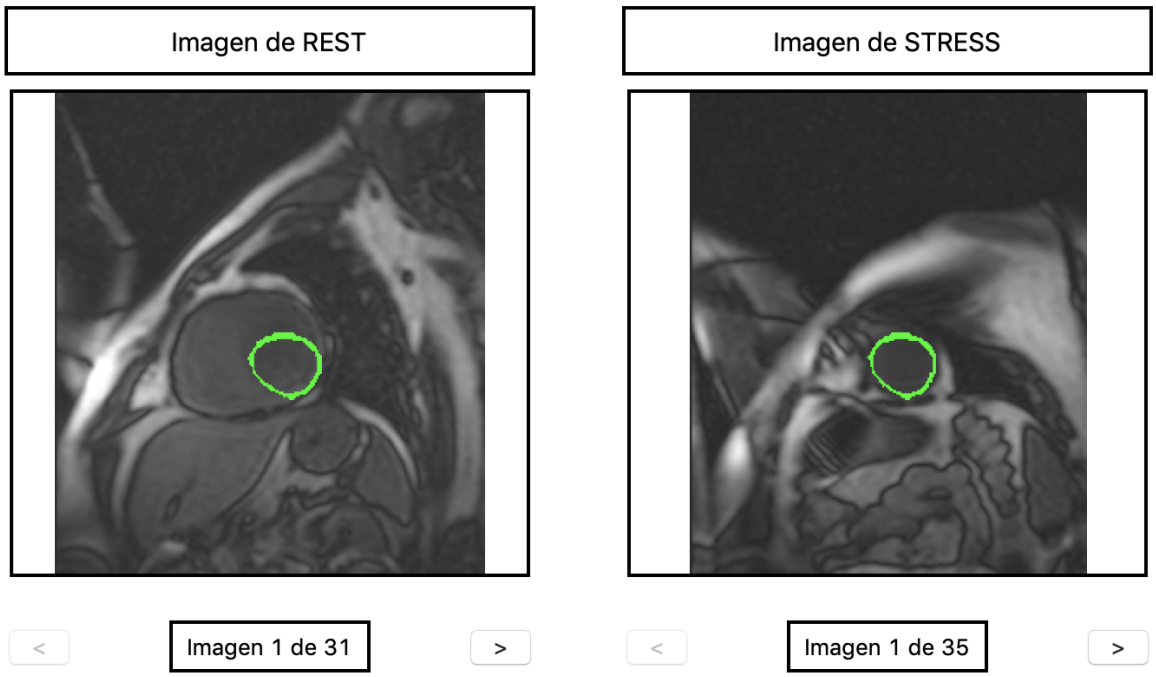

Figura 5.11: Imagen visualizada, con la sección de epicardio mostrada

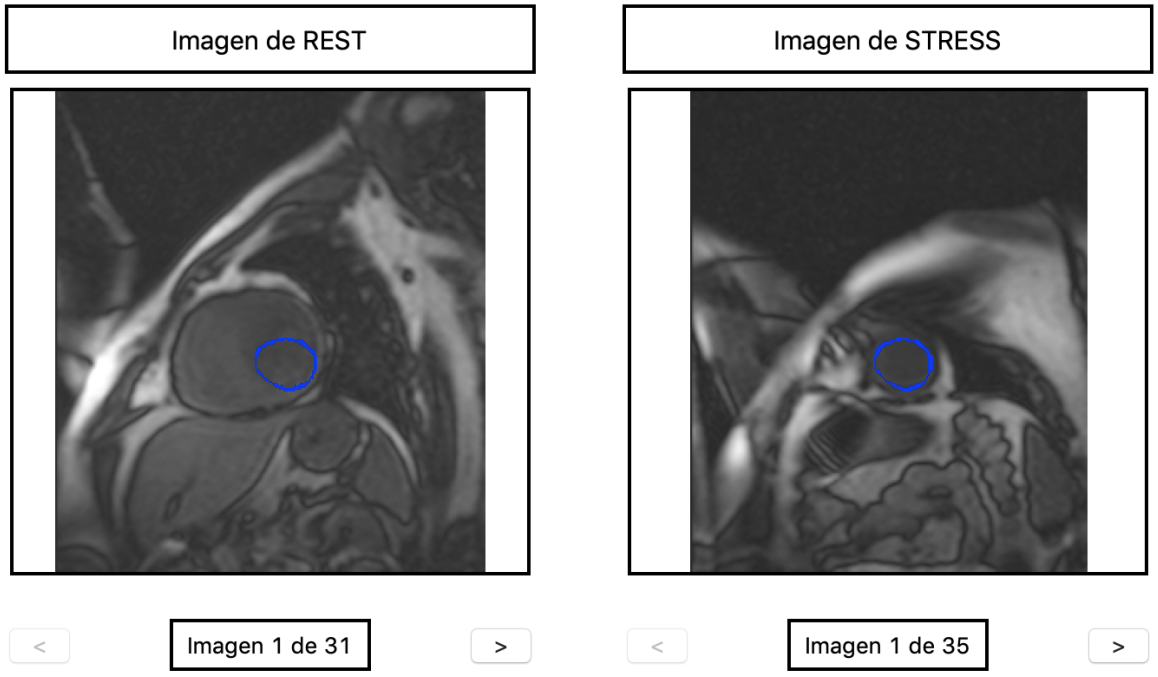

Figura 5.12: Imagen visualizada, con la sección de endocardio mostrada

Como se mencionó en la sección de metodología, al momento de apretar el botón para dividir el miocardio en cuatro partes, cambian ciertas funciones y mensajes. En este estado, en la zona de "Opciones" se modifican las acciones y mensajes, que se puede observar en la

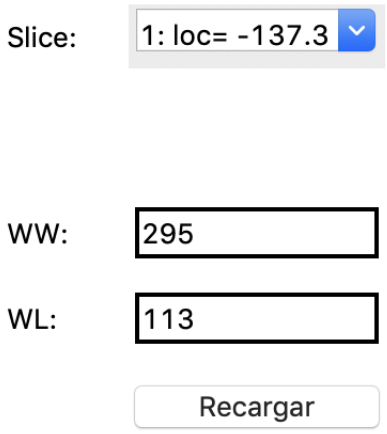

Figura 5.13: Zona de opciones de visualización

figura 5.14.

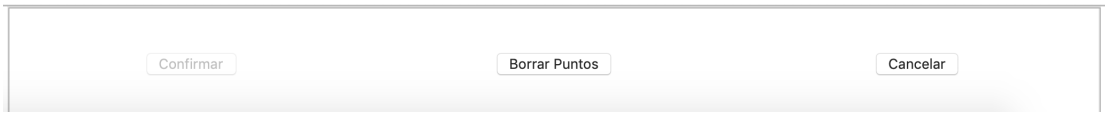

Figura 5.14: Vista de las opciones en caso de dividir el miocardio

Este modo permite al usuario ingresar los puntos sobre las imágenes, como se observa en la figura 5.15. Al haber seleccionado ambos puntos, se da la opción de poder completar la división, lo que puede tardar un par de minutos.

Cuando la división se ha realizado por completo, cambian ciertas funciones del sistema. Primero, en las opciones de visualización, se puede observar que se agrega un parámetro, llamado partición, como se observa en la figura 5.16. Éste permite elegir cuál zona, de las divididas es la que se quiere analizar y observar. El valor elegido en esta selección, modifica las curvas, la tabla de resultados y la vista de la imagen.

Además, en la zona de visualización, la predicción mostrada solo corresponde a la división seleccionada, como se puede observar en la figura 5.17, 5.18, 5.19 y 5.20 en el caso de la zona de sangre, en la figura 5.25, 5.26, 5.27 y 5.28 en el caso de la zona de epicardio, y en la figura 5.21, 5.22, 5.23 y 5.24 en el caso de la zona de endocardio.

Cabe destacar también que la tabla de resultados y las curvas de perfusión están referidas a la división seleccionada.

Finalmente, en las opciones, al entrar en este modo, aparece la opción de volver al estado

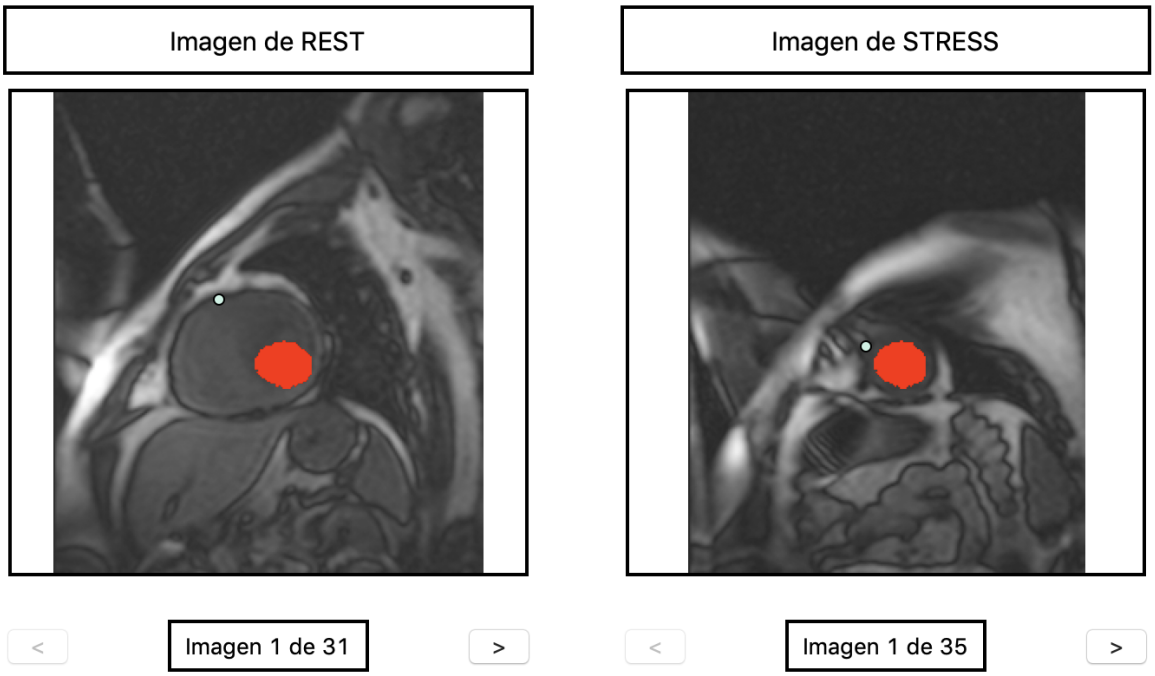

Figura 5.15: Muestra de puntos elegidos por el usuario

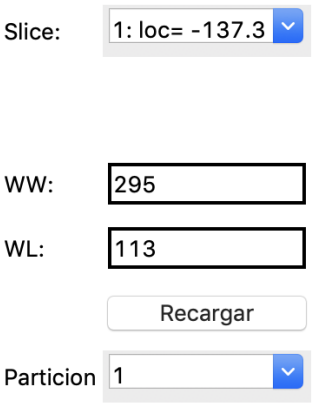

Figura 5.16: Nuevo parámetro en la zona de Opciones de Visualización

sin división, como se aprecia en la figura 5.29. Apretando este botón se vuelve al estado inicial de la pantalla de imágenes, con la sección completa.

Este proceso completa el último objetivo planteado, donde se permite la división del miocardio en cuatro secciones de mismo tamaño, se pueden calcular los valores correspondientes a cada división y se puede observar la predicción con la imagen correspondiente.

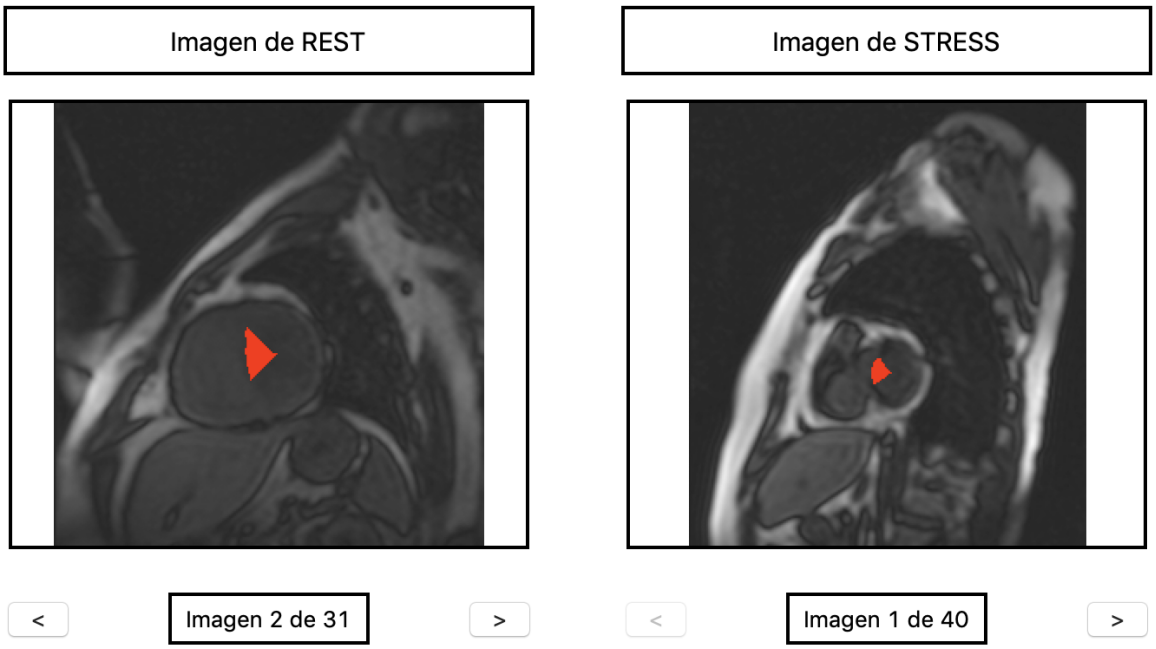

Figura 5.17: División de sangre, subdivisión 1

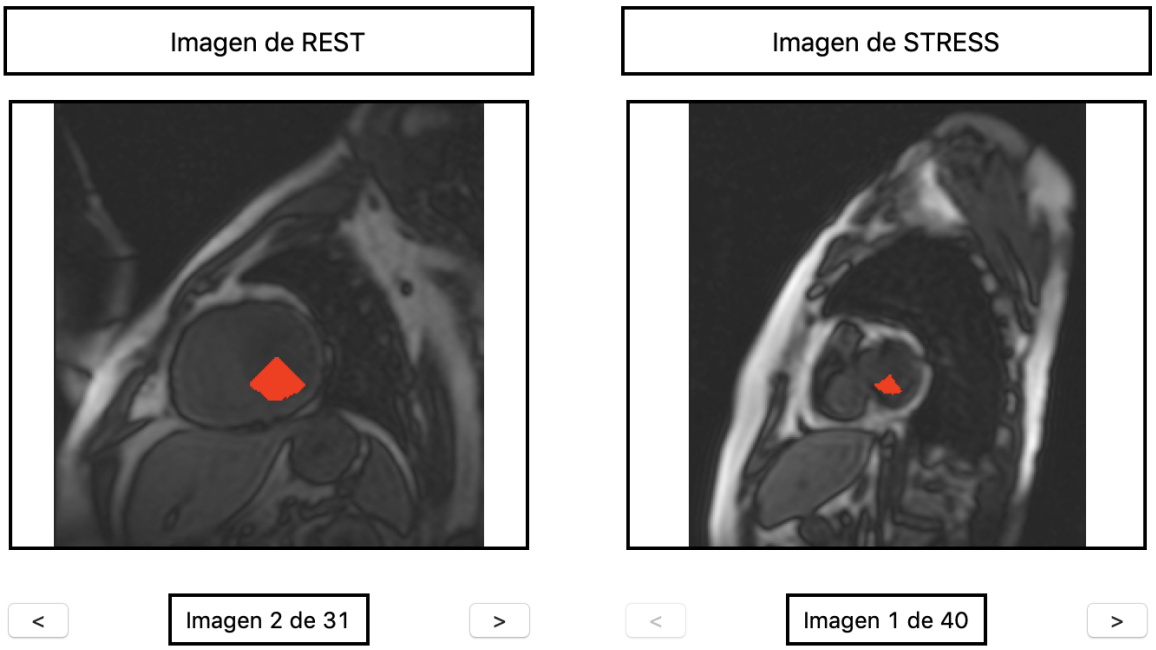

Figura 5.18: División de sangre, subdivisión 2

5.2. Extras

Cabe destacar que todos los botones e información relevante al usuario trae consigo una característica llamada *Tooltip*, lo que implica que al dejar el puntero del mouse sobre el

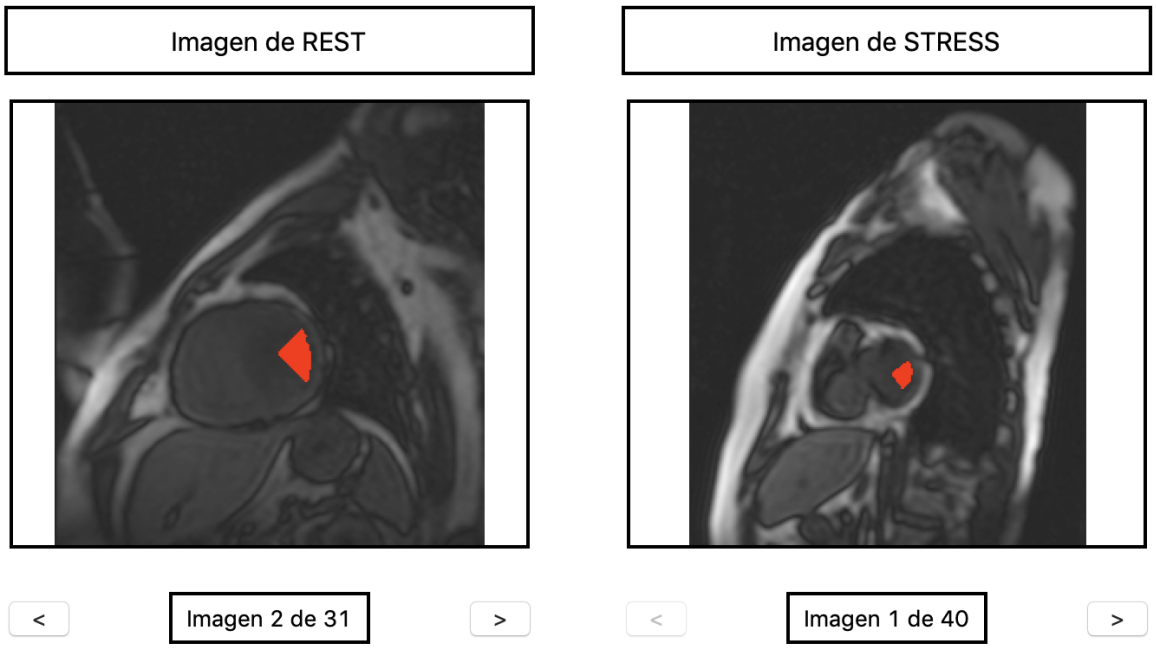

Figura 5.19: División de sangre, subdivisión 3

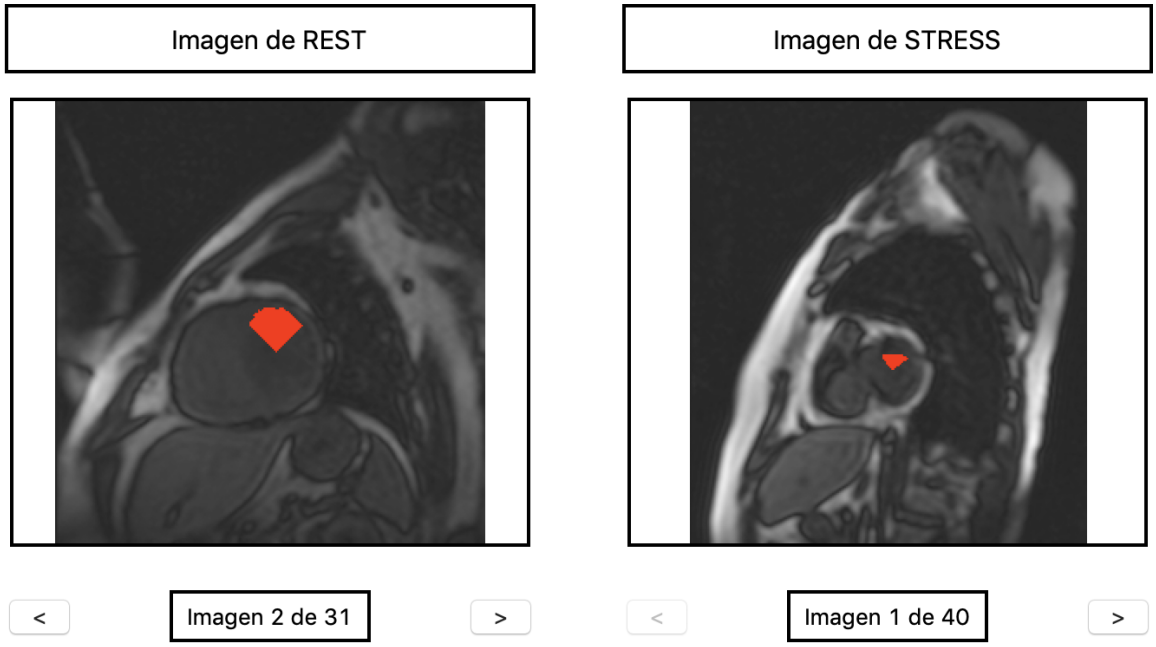

Figura 5.20: División de sangre, subdivisión 4

elemento, se despliega un mensaje explicativo de éste, para ayudar a entender las funciones de la aplicación.

Un ejemplo de esta función, se puede observar en la figura 5.30, donde al colocar el mouse sobre el título “Pendiente”, explica como fue calculado este valor.

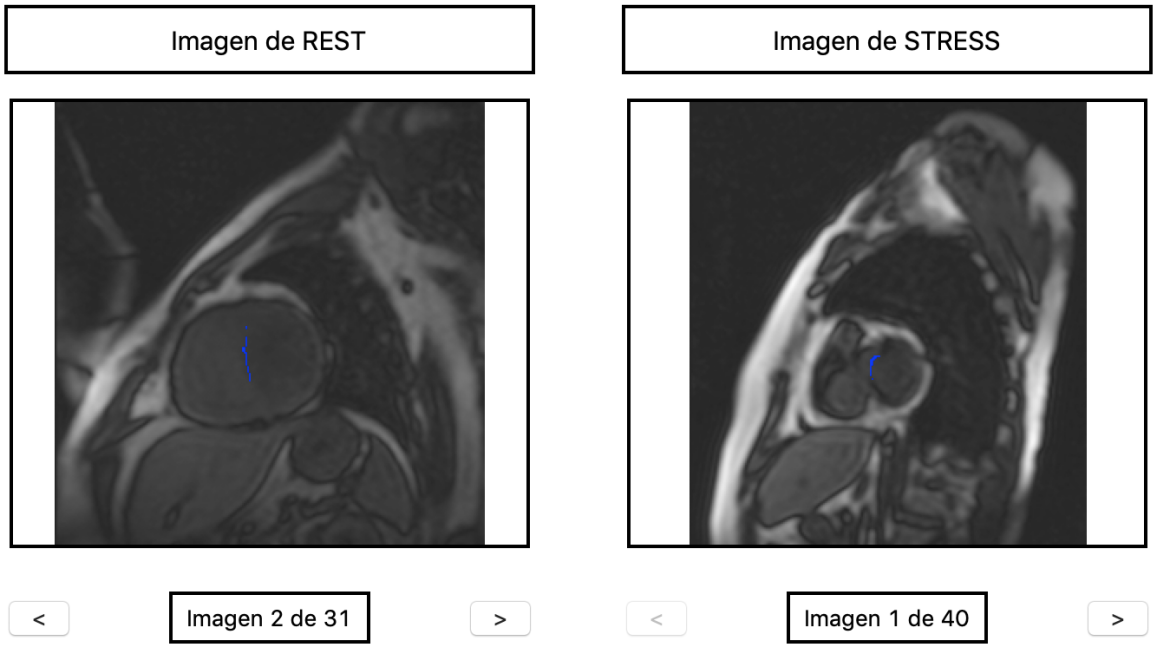

Figura 5.21: División de Endocardio, subdivisión 1

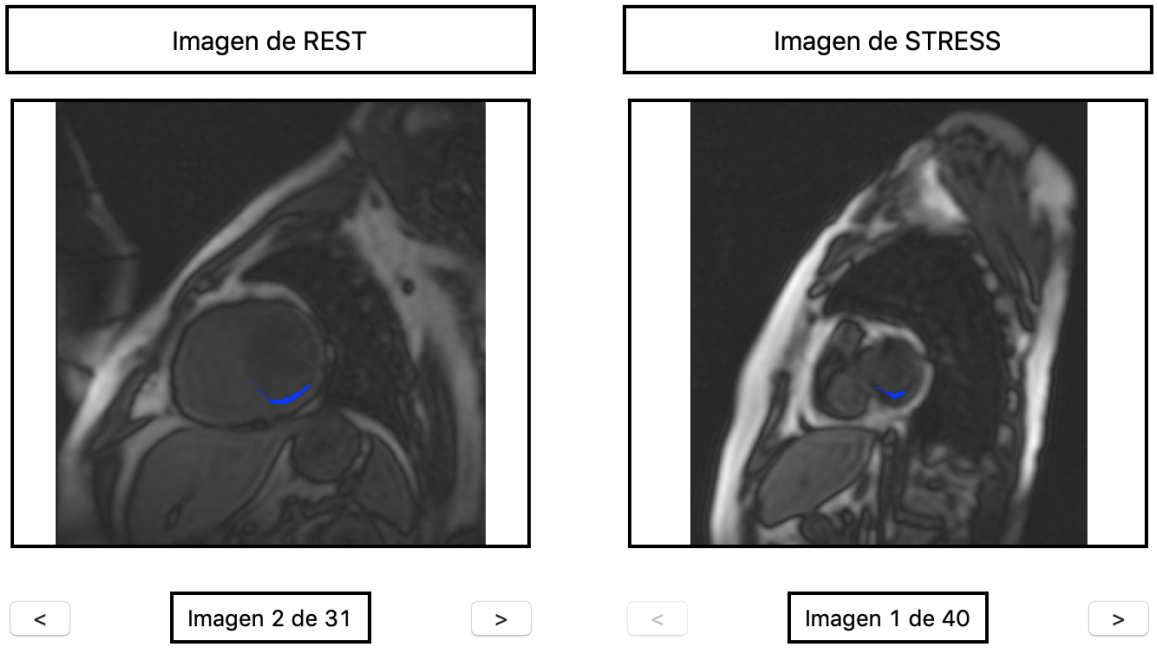

Figura 5.22: División de Endocardia, subdivisión 2

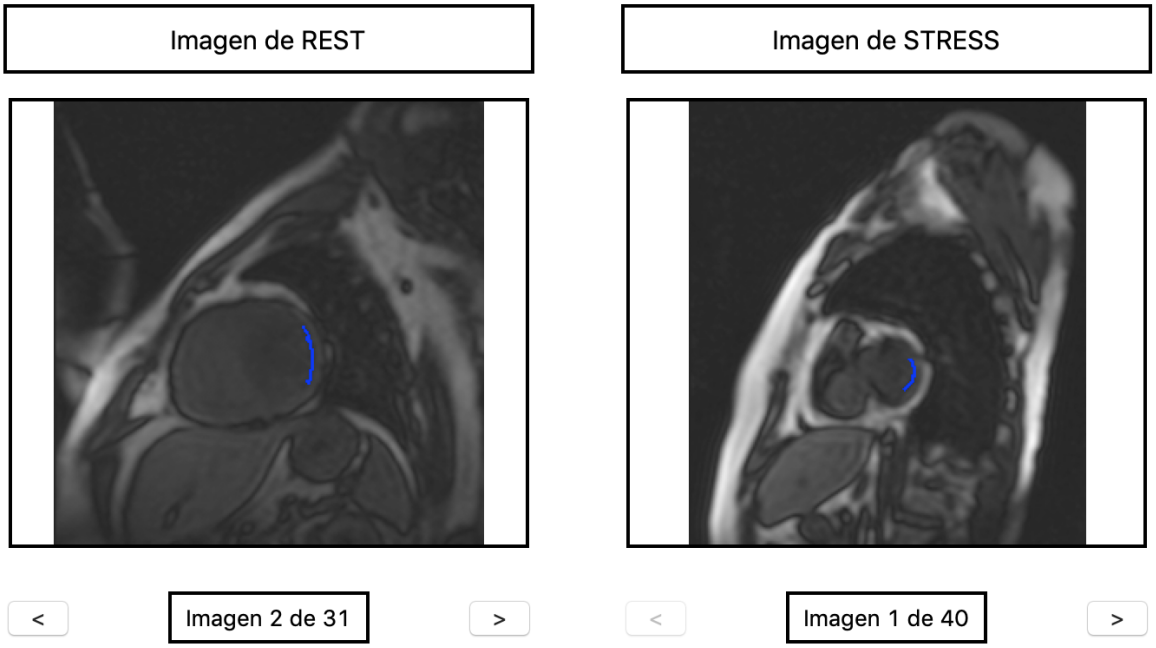

Figura 5.23: División de Endocardio, subdivisión 3

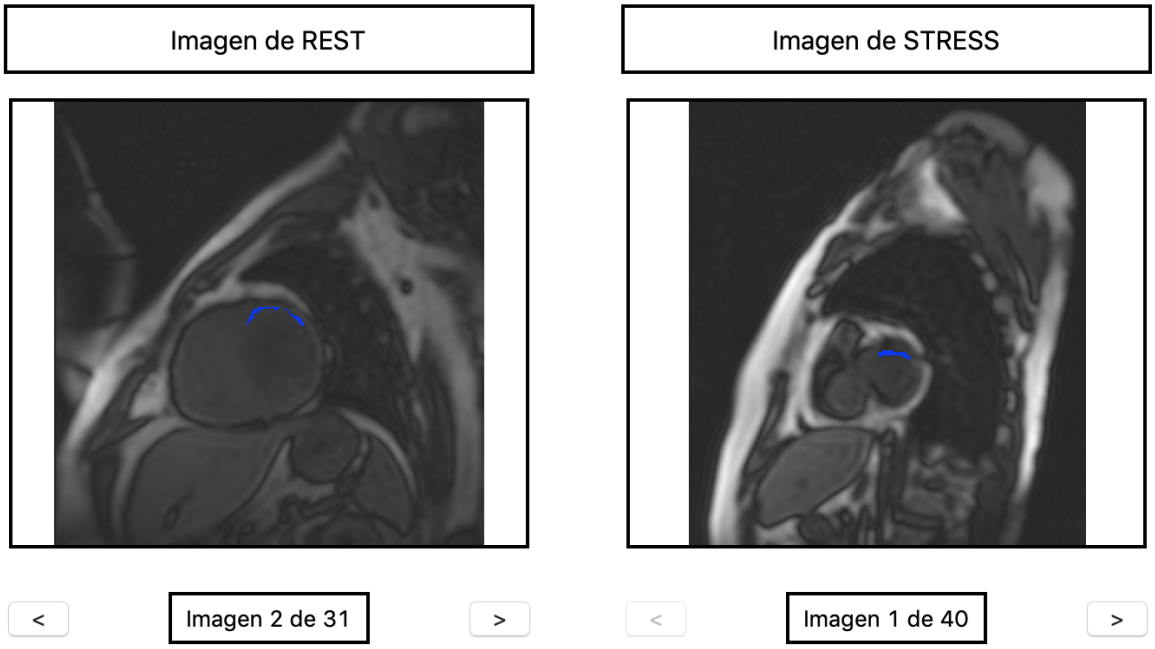

Figura 5.24: División de Endocardio, subdivisión 4

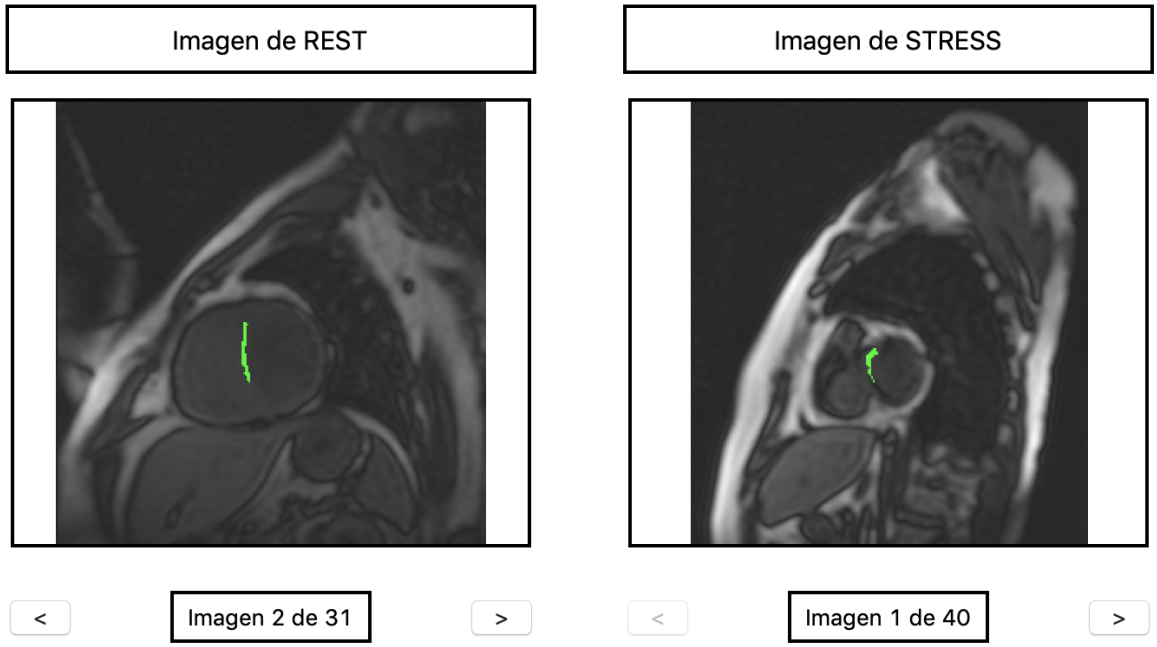

Figura 5.25: División de Epicardio, subdivisión 1

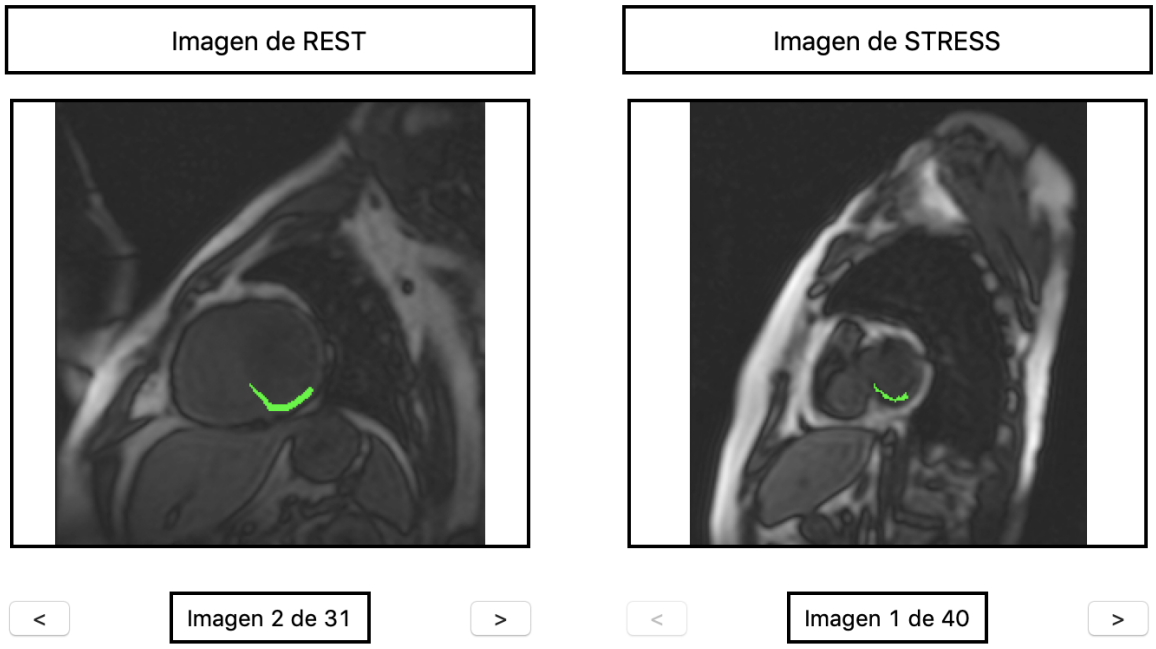

Figura 5.26: División de Epicardio, subdivisión 2

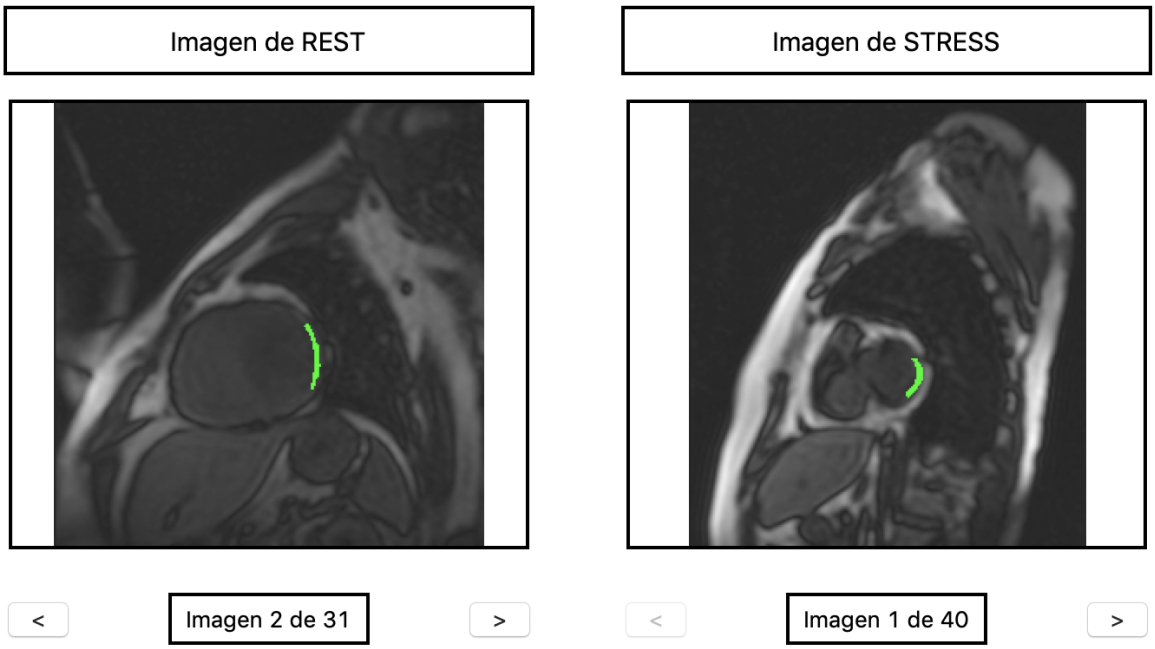

Figura 5.27: División de Epicardio, subdivisión 3

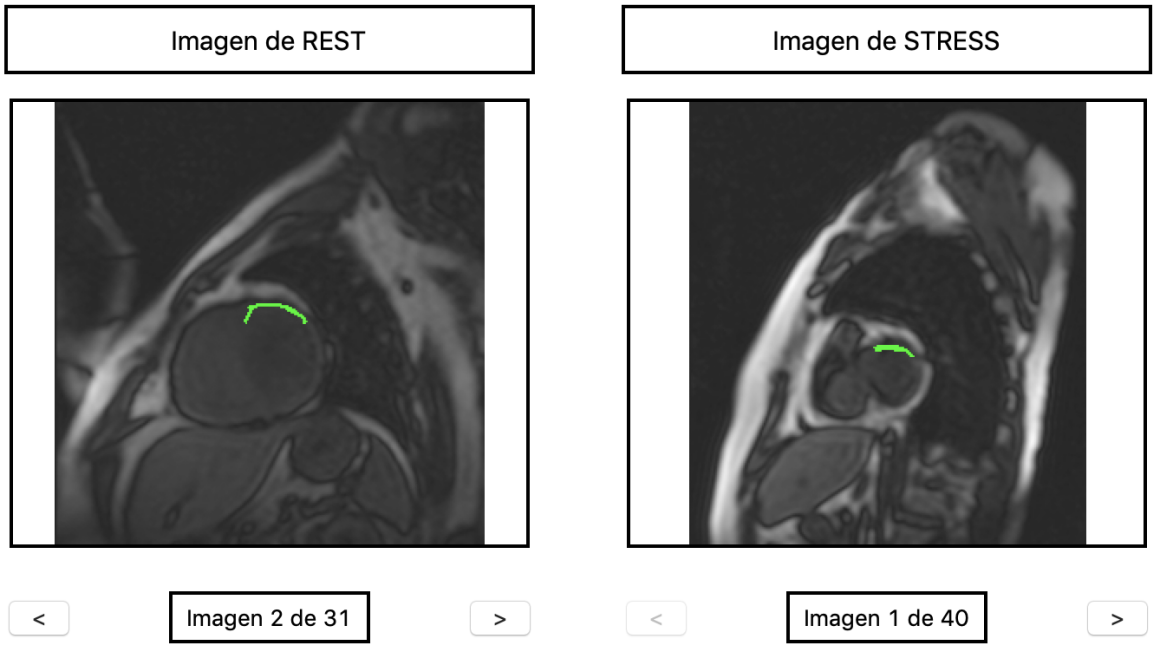

Figura 5.28: División de Epicardio, subdivisión 4

Figura 5.29: Opciones posibles en modo de miocardio dividido

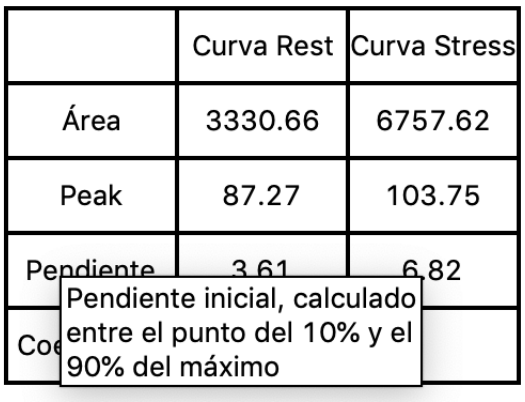

Figura 5.30: Ejemplo de Tooltip de ayuda al Usuario

Capítulo 6

Conclusiones

Durante el presente estudio, se pudo trabajar con imágenes reales de exámenes de perfusión miocárdica, diseñando e implementando una aplicación de utilidad para el área de la salud, ya que se logra mejorar el tiempo de pronóstico de una enfermedad tan mortal como la isquemia al miocardio y el infarto.

La aplicación es capaz de leer imágenes de resonancia magnética, aplicarles un algoritmo de segmentación automática para la zona del miocardio, y luego entregar los resultados obtenidos, mediante análisis visual y semi cuantitativo, que era el objetivo general planteado. Por esto, se puede concluir que éste fue cumplido a cabalidad.

Se logró la visualización de las imágenes ingresadas, tanto en el proceso de pre visualización 5.3, como en el proceso de resultados, junto a los resultados de la segmentación automática 5.10. Por esto se puede decir que se cumplió el primer objetivo específico.

Además, al momento de mostrar las imágenes, es posible observar ambos exámenes de forma paralela con sus respectivos resultados, para permitir el análisis visual del examen. De esto se advierte el cumplimiento del segundo objetivo específico.

Junto a esto, el sistema es capaz de dividir la zona de interés en las cuatro secciones necesarias, según la interacción del usuario. Esto permite un análisis más minucioso y preciso de los resultados. Lo anterior cumple el tercer objetivo específico.

Por último, se calculan los parámetros necesarios para poder hacer un diagnóstico preventivo de los posibles problemas cardiacos, tantos de las secciones divididas, como en las zonas completas del epicardio, endocardio y "pool' de sangre. Con esto se concluye que se

cumplieron los cuatro objetivos específicos planteados al inicio del trabajo.

La aplicación desarrollada está en su primera fase, cumple su función, pero aun tiene muchos puntos en los que se puede trabajar, para obtener más y mejores funcionalidades que el usuario pueda necesitar.

Dentro de los trabajos futuros planteados, se encuentra el de poder exportar los resultados, tanto en formato PDF como en formato DICOM, de modo que se pueda traspasar toda la información del examen y de la segmentación a las propias imágenes.

Además, se plantea realizar la visualización automática en paralelo de ambos exámenes, que puede ser mucho más sencillo para el usuario. Junto a esto, agregar más opciones de visualización, para acomodar al usuario a sus propias necesidades.

Para finalizar, se recomienda también empaquetar la aplicación, para que pueda ser utilizada fuera de los ambientes donde fue creado, con las herramientas nombradas al inicio de este informe.

Gracias a esta memoria, se pudo trabajar en un área que aveces como ingeniero, pocas veces se entra en contacto, como el área de la salud, aún cuando ambas trabajan muy bien juntas. Se pueden complementar muy bien, y de esta forma pueden obtener resultados beneficios tanto para las instituciones como para los pacientes.

Como persona, nunca se debe centrar el conocimiento en un solo tema. El trabajo con otras personas y de otras áreas entrega más sabiduría y conocimientos, que pueden ser útiles en cualquier momento, tanto profesional como personalmente.

Se logró trabajar en un ambiente nuevo, como es Tkinter, la librería de Python para aplicaciones, lo que también es un nuevo conocimiento que puede ser aplicado. Además, se pudo aprender de un órgano tan importante como es el corazón, las fallas que puede tener, su funcionamiento y las posibles complicaciones.

Finalmente, el aprendizaje obtenido gracias a este estudio se materializa en nuevos conocimientos, nuevas áreas de interés, la experiencia del diseño de un sistema para solucionar problemas y la aplicación de conocimientos a otras áreas no comunes, como el análisis de señales y de machine learning en el área médica.

Bibliografía

[1] Diagram of the human heart. "https://commons.wikimedia.org/wiki/File:Diagram_of_the_human_heart_(cropped).svg", 2006.

[2] Anil K Attili, Andreas Schuster, Eike Nagel, Johan HC Reiber, and Rob J van der Geest. Quantification in cardiac mri: advances in image acquisition and processing. *The international journal of cardiovascular imaging*, 26(1):27–40, 2010.

[3] Inc. Blausen Medical Communications. Myocardial infarction or heart attack. "https://commons.wikimedia.org/wiki/File:Blausen_0463_HeartAttack.png#/media/File:Blausen_0463_HeartAttack.png", 2013.

[4] Dan Ciresan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in neural information processing systems*, pages 2843–2851, 2012.

[5] Wlodzislaw Duch. What is computational intelligence and what could it become. *Computational Intelligence, Methods and Applications Lecture Notes NTU, Singapour*, 2003.

[6] Włodzisław Duch. What is computational intelligence and where is it going? In *Challenges for computational intelligence*, pages 1–13. Springer, 2007.

[7] Mariusz Flasiński. *Introduction to artificial intelligence*. Springer, 2016.

[8] Kasuga Huang. Modern high field clinical mri scanner. "https://commons.wikimedia.org/wiki/File:Modern_3T_MRI.JPG", 2006.

[9] Ludovico La Grutta, Giovanni Gentile, Giuseppe Runza, Massimo Galia, Filippo Cademartiri, and Massimo Midiri. Heart anatomy. In *Clinical Applications of Cardiac CT*, pages 93–113. Springer, 2012.

[10] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*, 2016.

[11] Darcy Mason and pydicom contributors. Using pydicom with tkinter. "`https://pydicom.github.io/pydicom/stable/viewing_images.html`", 2008-2019.

[12] Federico E Mordini, Tariq Haddad, Li-Yueh Hsu, Peter Kellman, Tracy B Lowrey, Anthony H Aletras, W Patricia Bandettini, and Andrew E Arai. Diagnostic accuracy of stress perfusion cmr in comparison with quantitative coronary angiography: fully quantitative, semiquantitative, and qualitative assessment. *JACC: Cardiovascular Imaging*, 7(1):14–22, 2014.

[13] National Electrical Manufacturers Association (NEMA). Dicom estandard, 2019.

[14] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[16] Jeanette Schulz-Menger, David A Bluemke, Jens Bremerich, Scott D Flamm, Mark A Fogel, Matthias G Friedrich, Raymond J Kim, Florian von Knobelsdorff-Brenkenhoff, Christopher M Kramer, Dudley J Pennell, et al. Standardized image interpretation and post processing in cardiovascular magnetic resonance: Society for cardiovascular magnetic resonance (scmr) board of trustees task force on standardized post processing. *Journal of Cardiovascular Magnetic Resonance*, 15(1):35, 2013.

Apéndice A

Codigos utilizados

Se procede a mostrar los códigos más importantes usados durante la tesis

Listing A.1: Calculo de Area

```
def calculo_area_curva(data, time):
    """
    Entregado el tiempo y la data, calcula el area de la
    curva utilizando trampz de Numpy
    :param data: Array con los valores de los promedios de las
                 intensidades
    :param time: Array con los valores de tiempo de las imagenes
    :return: area calculada, redondeada al segundo decimal
    """
    input_data = np.array(data)
    input_time = np.array(time)
    input_time = input_data.astype(np.float)
    area = np.round(np.trapz(input_data, input_time), 2)
    return area
```

Listing A.2: Calculo de Maximo

```
def calculo_maximo(data, time):
    """
    Calcula el mayor valor de la data entrante, y devuelve el tiempo
    y la posicion en la que se encuentra este mayor
    :param data: Data en la que se busca el max
    :param time: Array de tiempo, corresponde al tiempo de cada dato
                 de data
    :return: maximo valor, tiempo en el que ocurre, y la posicion
             del array
    """
    input_data = np.array(data)
    pos_max = signal.find_peaks_cwt(input_data, np.arange(1, 150))[0]
```

```
    time_max = np.array(time)[pos_max]
    max_element = np.round(input_data[pos_max], 2)

    return max_element, time_max, pos_max
```

Listing A.3: Calculo de Pendiente

```
def calculo_pendiente(data, time):
    """
    Devuelve la pendiente inicial, considerando el rise time de la
    curva con el momento donde la data sube desde el 10%
    del maximo al 90\% del maximo
    :param data: Data de entrada, eje X
    :param time: Data de tiempo, eje Y
    :return: valor de la pendiente, tiempo inicial, tiempo final,
    valor inicial, valor final
    """
    max_data, time_max, pos_max = calculo_maximo(data, time)
    por10 = max_data*0.1
    por90 = max_data*0.9
    first = 0
    last = 0
    for valor in np.array(data):
        if valor > por10 and first == 0:
            first = valor
            continue
        if valor < por90:
            last = valor
            continue
        else:
            break
    pos_first = np.where(np.array(data) == first)[0][0]
    pos_last = np.where(np.array(data) == last)[0][0]
    time_first = np.array(time).astype(np.float)[pos_first]
    time_last = np.array(time).astype(np.float)[pos_last]
    try:
        p = np.round((last-first)/(time_last-time_first), 2)
    except ZeroDivisionError:
        p = 0
    return p, time_first, time_last, first, last
```

Listing A.4: Calculo de transformación WW y WL

```
import numpy as np


def refactor_dicom_file(dicom_array, ww, wl):
    """
    Ingresada la imagen, procesa los limites segun los valores de
    ww y wl, para escalar el dicom_array a
    valores de escala de grises de 8 BITS
```

```
    :param dicom_array: array de la imagen con los valores
                        invariantes
    :param ww: Window Width
    :param wl: Window Level
    :return: Imagen en escala de grises
    """
    #  Intervalos de escala de grises
    min_val = 0
    max_val = 255

    #  Estandar DICOM, ww >= 1
    ww = max(1, ww)

    #  Formato float64 para manejo de imagenes
    wl, ww = np.float64(wl), np.float64(ww)
    interval = np.float(max_val) - min_val
    input_arr = dicom_array.astype(np.float64)

    minval = wl - 0.5 - (ww - 1.0) / 2.0
    maxval = wl - 0.5 + (ww - 1.0) / 2.0

    min_mask = (minval >= input_arr)
    to_scale = (input_arr > minval) & (input_arr < maxval)
    max_mask = (input_arr >= maxval)

    if min_mask.any():
        input_arr[min_mask] = min_val
    if to_scale.any():
        input_arr[to_scale] = ((input_arr[to_scale] - (wl - 0.5)) /
                               (ww - 1.0) + 0.5)*interval + min_val
    if max_mask.any():
        input_arr[max_mask] = max_val

    return np.rint(input_arr).astype(np.uint8)
```

Listing A.5: Separación Miocardio

```

def detectar_separacion(self):
    """
    Detectar que parte del miocardio es parte del endocardio y
    cual del epicardio
    Buscar valores 1 de self.predict
    - Si mas cercano es 0: ENDOCARDIO
    - Si mas cercano es 2: EPICARDIO
    - Si ambos cercanos: EPICARDIO
    :return: guarda los resultados en self.epicardio y self.endocardio
    """
    for i in range(len(self.predict)):
        x_end = []
```

```
14          x_epi = []
15          x_san = []
16          for j in range(len(self.predict[i])):
17              is0 = False
18              is2 = False
19              if self.predict[i][j] == 1:
20                  dif = 1
21                  while not is0 and not is2:
22                      is0, is2 = self.mas_cercano(i, j, dif)
23                      if is2 and not is0:
24                          x_end.append(1)
25                          x_epi.append(0)
26                          x_san.append(0)
27                      elif is0:
28                          x_end.append(0)
29                          x_epi.append(1)
30                          x_san.append(0)
31                      dif += 1
32              elif self.predict[i][j] == 2:
33                  x_san.append(1)
34                  x_end.append(0)
35                  x_epi.append(0)
36              else:
37                  x_epi.append(0)
38                  x_end.append(0)
39                  x_san.append(0)
40          self.endocardio.append(np.array(x_end))
41          self.epicardio.append(np.array(x_epi))
42          self.sangre.append(np.array(x_san))
43
44  def mas_cercano(self, pos_x, pos_y, dist):
45      """
46      Encuentra el punto mas cercano de interes al punto de analisis.
47      Se debe encontrar fuera del miocardio (0) o
48      sangre (2). Se buscar en cuadrados, con centro en el punto
49      :param pos_x: posicion x del punto
50      :param pos_y: posicion y del punto
51      :param dist: distancia desde el punto hacia donde analizar
52      :return: is0 True si detecto miocardio,
53                   False caso contrario.
54               is2 True si detecto sangre,
55                   False, caso contrario
56      """
57      is0 = False
58      is2 = False
59      # Fijar pos_y-dist
60      if pos_y - dist >= 0:
61          for i in range(dist):
62              if pos_x - i >= 0:
63                  if self.predict[pos_x - i][pos_y - dist] == 0:
```

```
                    is0 = True
                elif self.predict[pos_x - i][pos_y - dist] == 2:
                    is2 = True
            if pos_x + i < len(self.predict):
                if self.predict[pos_x + i][pos_y - dist] == 0:
                    is0 = True
                elif self.predict[pos_x + i][pos_y - dist] == 2:
                    is2 = True
    # Fijar pos_y + dist
    if pos_y + dist < len(self.predict[0]):
        for i in range(dist):
            if pos_x - i >= 0:
                if self.predict[pos_x - i][pos_y + dist] == 0:
                    is0 = True
                elif self.predict[pos_x - i][pos_y + dist] == 2:
                    is2 = True
            if pos_x + i < len(self.predict):
                if self.predict[pos_x + i][pos_y + dist] == 0:
                    is0 = True
                elif self.predict[pos_x + i][pos_y + dist] == 2:
                    is2 = True
    # Fijar pos_x-dist
    if pos_x - dist >= 0:
        for i in range(dist):
            if pos_y - i >= 0:
                if self.predict[pos_x - dist][pos_y - i] == 0:
                    is0 = True
                elif self.predict[pos_x - dist][pos_y - i] == 2:
                    is2 = True
            if pos_y + i < len(self.predict[i]):
                if self.predict[pos_x - dist][pos_y + i] == 0:
                    is0 = True
                elif self.predict[pos_x - dist][pos_y + i] == 2:
                    is2 = True
    # Fijar pos_x + dist
    if pos_x + dist >= len(self.predict):
        for i in range(dist):
            if pos_y - i >= 0:
                if self.predict[pos_x + dist][pos_y - i] == 0:
                    is0 = True
                elif self.predict[pos_x + dist][pos_y - i] == 2:
                    is2 = True
            if pos_y + i < len(self.predict[i]):
                if self.predict[pos_x + dist][pos_y + i] == 0:
                    is0 = True
                elif self.predict[pos_x + dist][pos_y + i] == 2:
                    is2 = True
    return is0, is2
```

Listing A.6: Deteccion cuadrante

```
def detectar_cuadrante(self, p):
    """
    Dado un punto p, calcula el cuadrante al que pertenece, tomando
    en consideracion que el (0,0) del sistema creado es el punto de
    radio del pool de sangre. En caso de que toque un eje, se
    devuelve un par con los cuadrantes que toca.
    :param p: punto ingresado para analizar (x, y)
    :return: String con el valor del cuadrante donde se encuentra
    """
    x = p[0]
    y = p[1]
    if x > self.radio[0]:
        if y > self.radio[1]:
            return 'IV'
        elif y < self.radio[1]:
            return 'I'
        else:
            return 'I-IV'
    elif x < self.radio[0]:
        if y > self.radio[1]:
            return 'III'
        elif y < self.radio[1]:
            return 'II'
        else:
            return 'II-III'
    else:
        if y > self.radio[1]:
            return 'III-IV'
        elif y <= self.radio[1]:
            return 'I-II'
```

Listing A.7: Ingreso de division

```

def ingresar_punto(self, pos, pend, a1, a2, a3, a4):
    """
    Dado el punto, se ingresa el punto al array que le corresponde
    como un 1, mientras que en los otros se ingresa un 0,
    para mantener la forma de imagen y luego poder ingesarla
    :param pos: posicion con respecto al cuadrante
    :param pend: pendiente del punto, sirve para comparar y ver
                 a cual segmento va
    :param a1: primer array de forma
    :param a2: segundo array de forma
    :param a3: tercer array de forma
    :param a4: cuarto array de forma
    :return: void, se finaliza con el punto agregado donde
             corresponde, y el resto con 0
    """
```

```
    if pos == '0':
        a1.append(0); a2.append(0); a3.append(0); a4.append(0)
    else:
        if self.m["init"] == 'I':
            if pos == 'I':
                if pend < self.m["I"]:
                    a1.append(1); a2.append(0);
                    a3.append(0); a4.append(0)
                else:
                    a1.append(0); a2.append(0);
                    a3.append(0); a4.append(1)
            elif pos == 'II':
                if pend < self.m["II"]:
                    a1.append(0); a2.append(0);
                    a3.append(0); a4.append(1)
                else:
                    a1.append(0); a2.append(0);
                    a3.append(1); a4.append(0)
            elif pos == 'III':
                if pend < self.m["III"]:
                    a1.append(0); a2.append(0);
                    a3.append(1); a4.append(0)
                else:
                    a1.append(0); a2.append(1);
                    a3.append(0); a4.append(0)
            elif pos == 'IV':
                if pend < self.m["IV"]:
                    a1.append(0); a2.append(1);
                    a3.append(0); a4.append(0)
                else:
                    a1.append(1); a2.append(0);
                    a3.append(0); a4.append(0)
            elif pos == 'I-II':
                a1.append(0); a2.append(0);
                a3.append(0); a4.append(1)
            elif pos == 'II-III':
                a1.append(0); a2.append(0);
                a3.append(1); a4.append(0)
            elif pos == 'III-IV':
                a1.append(0); a2.append(1);
                a3.append(0); a4.append(0)
            elif pos == 'I-IV':
                a1.append(1); a2.append(0);
                a3.append(0); a4.append(0)
        elif self.m["init"] == 'II':
            if pos == 'I':
                if pend < self.m["I"]:
                    a1.append(0); a2.append(1);
                    a3.append(0); a4.append(0)
                else:
```

```
67                      a1.append(1); a2.append(0);
68                      a3.append(0); a4.append(0)
69              elif pos == 'II':
70                  if pend < self.m["II"]:
71                      a1.append(1); a2.append(0);
72                      a3.append(0); a4.append(0)
73                  else:
74                      a1.append(0); a2.append(0);
75                      a3.append(0); a4.append(1)
76              elif pos == 'III':
77                  if pend < self.m["III"]:
78                      a1.append(0); a2.append(0);
79                      a3.append(0); a4.append(1)
80                  else:
81                      a1.append(0); a2.append(0);
82                      a3.append(1); a4.append(0)
83              elif pos == 'IV':
84                  if pend < self.m["IV"]:
85                      a1.append(0); a2.append(0);
86                      a3.append(1); a4.append(0)
87                  else:
88                      a1.append(0); a2.append(1);
89                      a3.append(0); a4.append(0)
90              elif pos == 'I-II':
91                  a1.append(1); a2.append(0);
92                  a3.append(0); a4.append(0)
93              elif pos == 'II-III':
94                  a1.append(0); a2.append(0);
95                  a3.append(0); a4.append(1)
96              elif pos == 'III-IV':
97                  a1.append(0); a2.append(0);
98                  a3.append(1); a4.append(0)
99              elif pos == 'I-IV':
100                 a1.append(0); a2.append(1);
101                 a3.append(0); a4.append(0)
102         elif self.m["init"] == 'III':
103             if pos == 'I':
104                 if pend < self.m["I"]:
105                     a1.append(0); a2.append(0);
106                     a3.append(1); a4.append(0)
107                 else:
108                     a1.append(0); a2.append(1);
109                     a3.append(0); a4.append(0)
110             elif pos == 'II':
111                 if pend < self.m["II"]:
112                     a1.append(0); a2.append(1);
113                     a3.append(0); a4.append(0)
114                 else:
115                     a1.append(1); a2.append(0);
116                     a3.append(0); a4.append(0)
```

```
117             elif pos == 'III':
118                 if pend < self.m["III"]:
119                     a1.append(1); a2.append(0);
120                     a3.append(0); a4.append(0)
121                 else:
122                     a1.append(0); a2.append(0);
123                     a3.append(0); a4.append(1)
124             elif pos == 'IV':
125                 if pend < self.m["IV"]:
126                     a1.append(0); a2.append(0);
127                     a3.append(0); a4.append(1)
128                 else:
129                     a1.append(0); a2.append(0);
130                     a3.append(1); a4.append(0)
131             elif pos == 'I-II':
132                 a1.append(0); a2.append(1);
133                 a3.append(0); a4.append(0)
134             elif pos == 'II-III':
135                 a1.append(1); a2.append(0);
136                 a3.append(0); a4.append(0)
137             elif pos == 'III-IV':
138                 a1.append(0); a2.append(0);
139                 a3.append(0); a4.append(1)
140             elif pos == 'I-IV':
141                 a1.append(0); a2.append(0);
142                 a3.append(1); a4.append(0)
143         elif self.m["init"] == 'IV':
144             if pos == 'I':
145                 if pend < self.m["I"]:
146                     a1.append(0); a2.append(0);
147                     a3.append(0); a4.append(1)
148                 else:
149                     a1.append(0); a2.append(0);
150                     a3.append(1); a4.append(0)
151             elif pos == 'II':
152                 if pend < self.m["II"]:
153                     a1.append(0); a2.append(0);
154                     a3.append(1); a4.append(0)
155                 else:
156                     a1.append(0); a2.append(1);
157                     a3.append(0); a4.append(0)
158             elif pos == 'III':
159                 if pend < self.m["III"]:
160                     a1.append(0); a2.append(1);
161                     a3.append(0); a4.append(0)
162                 else:
163                     a1.append(1); a2.append(0);
164                     a3.append(0); a4.append(0)
165             elif pos == 'IV':
166                 if pend < self.m["IV"]:
```

```
                    a1.append(1); a2.append(0);
                    a3.append(0); a4.append(0)
                else:
                    a1.append(0); a2.append(0);
                    a3.append(0); a4.append(1)
            elif pos == 'I-II':
                a1.append(0); a2.append(0);
                a3.append(1); a4.append(0)
            elif pos == 'II-III':
                a1.append(0); a2.append(1);
                a3.append(0); a4.append(0)
            elif pos == 'III-IV':
                a1.append(1); a2.append(0);
                a3.append(0); a4.append(0)
            elif pos == 'I-IV':
                a1.append(0); a2.append(0);
                a3.append(0); a4.append(1)
```