INSTITUTO TECNOLÓGICO DE BUENOS AIRES – ITBA

ESCUELA DE INGENIERÍA Y GESTIÓN

Evaluación del entorno de desarrollo Golang como herramienta para la detección de onda dícrota

AUTOR/ES: Boschetti, Marco (Leg. N° 55266)
Mogni, Guido Matías (Leg. N° 55156)

DOCENTE/S TITULAR/ES O TUTOR/ES: Madorno, Matías

TRABAJO FINAL PRESENTADO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN INFORMÁTICA

Lugar: Av. Eduardo Madero 399, CABA
Fecha: 22/07/2019

ITBA
Instituto Tecnológico de Buenos Aires

Índice

1. Resumen / Abstract

El objetivo principal de este trabajo fue probar el leguaje de desarrollo GoLang en una situación concreta, buscando dar una solución a un problema real. GoLang, también conocido como Go, fue desarrollado por Google en 2007 con el objetivo de mejorar la productividad de los desarrolladores, destacándose por su manejo de conexión de redes de información y multi-procesos concurrentes. [1]

Se eligió abordar esa situación actual de la tecnología de la salud, centrada en el análisis de señales fisiológicas en tiempo real. En ese entorno, puntualmente, se probó la detección de la onda dícrota en la señal pletismografía de pulso.

Se evaluó el entorno Golang como desarrollo con resultados prometedores y mostrando que tiene potencial para el procesamiento de las señales mencionadas anteriormente y proporciona varias herramientas de caracter nativo que ayudaron durante desarrollo de este trabajo.

2. Introducción

Hoy dentro del área de la medicina se generan cada vez más volúmenes de datos, haciendo necesario desarrollar nuevos abordajes y herramientas que permitan analizar esa gran cantidad de información para poder obtener resultados cada vez más útiles. Estos instrumentos deben estar preparados para procesar un gran volumen de datos en tiempo real, dando un resultado lo más rápido posible.

Un área de la medicina donde se generan estas situaciones es el de las señales fisiológicas, las cuales en su mayoría no son procesadas, sino solo leídas e interpretadas por los profesionales de la medicina. Este instrumento puede procesar este tipo de señales en tiempo real y dar una respuesta que sin duda sería de gran ayuda para la mejora en la atención de los pacientes. [2]

Dentro del gran espectro que representan las señales fisiológicas si hizo foco en una clase particular, las de pletismografía de pulso. Estas señales se obtienen mediante un aparato conocido como oxímetro de pulso, que determina el porcentaje de saturación de oxígeno en la sangre arterial (SpO2).

En la señal generada por el oxímetro de pulso existe una característica particular llamada onda dícrota. Era por demás ilustrativo poder encontrar la onda dícrota [3] en la morfología de la señal de pletismografía de pulso, ya que permitiría tener una noción de la presión arterial del paciente [4].

En este trabajo se utilizó lenguaje de desarrollo GoLang para construir un sistema de procesamiento y monitoreo de la señal de pletismografía del pulso. Este sistema debe consumir y graficar los valores de entrada de una pletismografía de pulso (tanto a partir de archivos previamente generados como de un oxímetro comercial conectado en tiempo real). En base a estos datos, debe identificarse cada pulso y su onda dícrota correspondiente.

El objetivo final es que mediante el entorno de desarrollo Go, se implemente un sistema donde la señal de pletismografía de pulso le permita a cualquier profesional de la medicina, tener mayor cantidad información y precisión del estado del paciente , tanto de manera presencial como remota.

3. Estado del arte

Lo que se buscó en este desarrollo es utilizar Golang para resolver un problema real, como es el análisis de la señal de pletismografía de pulso.

Existen diversos materiales relacionados con dicho proceso (papers, publicaciones científicas, etc). El hecho de que ya exista un volumen de información respecto del análisis de la señal de pletismografía de pulso, muestra el interés que hay por parte de la comunidad médica en estas señales.

Esta señal tiene información de la luz absorbida en el dedo. Además de poder calcular el valor de oximetría esta señal tiene información de sistema vascular en cuanto a la variación de presión arterial [4] y el flujo de sangre que circula por el capilar [5].

A continuación se muestran algunos ejemplos de las publicaciones existentes.

Publicación en la Revista Cubana de Informática Médica [6]

En esta publicación de la Revista Cubana de Informática Médica se detalla un algoritmo que tiene como objetivo la detección de los puntos de interés en la onda cardíaca. Esta publicación muestra que, de todos los puntos de interés propuestos para buscar, el punto más difícil de detectar con precisión para su algoritmo es el de la incisura dícrota. Este obstáculo se debe la gran cantidad de formas en la que esta incisura puede aparecer y que, en varias ocasiones, puede ser confundida con ruido de la señal.

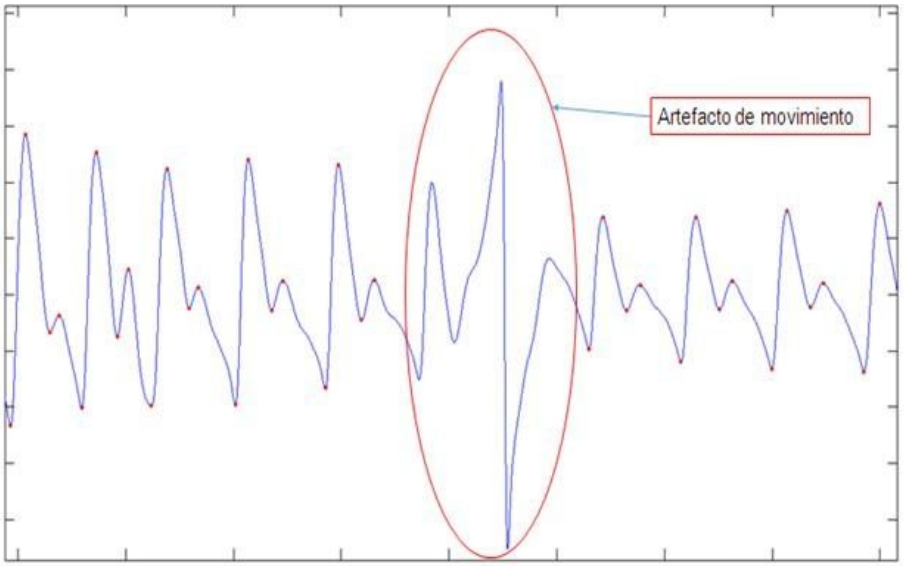


Figura 1- Algoritmo de detección de puntos de la Revista Médica Cubana

Otro detalle a destacar es que las señales cardíacas usadas para probar los resultados del método no presentan gran variabilidad, teniendo bien delimitados los puntos de interés a encontrar.

Obtención de la onda dícrota mediante simulación numérica utilizando el modelo de Ogden [6]

En un informe de la Universidad de Guanajuato, se destaca la necesidad de utilizar una simulación numérica y un modelo matemático , para suponer el comportamiento del flujo sanguíneo, para poder encontrar la onda dícrota. Es una perspectiva completamente distinta a la publicación anterior ya que se aplica sobre modelos teóricos en lugar de señales reales.

Si bien no se preveen usos prácticos para este modelo, Esta publicación deja abierta la posibilidad de llevar a detectar la onda dícrota desde un ambiente de simulación a un entorno real para ver su comportamiento.

4. Objetivos

El objetivo principal del trabajo es utilizar el entorno de desarrollo Golang para resolver un problema real como lo es la de detección de onda dícrota a partir del análisis de señales de pletismografía de pulso.

Siguiendo este objetivo principal, se definieron diferentes módulos a implementar en el lenguaje Golang para abordar el problema mencionado:

- Diseñar e implementar una herramienta que permita la lectura de diversas señales de pletismografía de pulso.
    - Esta herramienta debe ser capaz de obtener dichas señales tanto de un set de datos previamente registrado, como de un oxímetro de pulso comercial en tiempo real.
- Diseñar e implementar un algoritmo de análisis de señales de pletismografía de pulso y detección de onda dícrota basado en la morfología de las mismas.
    - Comparar los resultados obtenidos mediante el enfoque de análisis por morfología de la señal con una estrategia basada redes neuronales
- Proveer una herramienta integrada a la aplicación que permita a expertos evaluar el desempeño del algoritmo de análisis de señales en diferentes escenarios.

Finalmente, se buscará concluir si Golang es una propuesta viable para resolver el problema planteado. Esta conclusión se hará mediante la evaluación de expertos.

5. Materiales y Métodos

Premisas

Para realizar el desarrollo del presente trabajo se establecieron determinadas premisas para dar un contexto más preciso sobre las tareas que debe realizar el sistema planteado.

- Toda onda proveniente de una pletismografía de pulso tiene una y solo una onda dícrota identificable [4].
- La posición de la onda dícrota varía junto con la tensión arterial
- La variación en la amplitud y frecuencia de los pulsos medidos es gradual a lo largo del tiempo [5]

Desarrollo

Características principales de Golang

Golang, como se menciona en la página oficial, es un lenguaje de código abierto orientado al desarrollo de software simple, confiable y eficiente desarrollado por Google y una comunidad *open source*.

La principales características de Golang son:

- **Velocidad de desarrollo**:

Dada la sintaxis simple, Golang permite iterar rápidamente entre diferentes versiones de código de forma simple.

En un proyecto de investigación esta característica resulta fundamental para poder comparar resultados arrojados por diferentes alternativas.

- **Manejo simple de concurrencia**:

Una característica fundamental para el desarrollo de cualquier sistema complejo es la capacidad de procesamiento de información de manera

simultánea. Golang permite mediante herramientas de caracter nativo, un manejo muy simple de distintos procesos que consumen los mismos recursos.

Para esta implementación se explotaron los mecanismos de concurrencia provistos por Golang (en particular *channels* y *goroutines*) con el objetivo de modularizar las responsabilidades de cada componente. Además de ser una buena práctica de programación, esta modularización en el contexto de un trabajo de investigación permitió modificar un único módulo y medir su impacto en el resto del sistema.

- **Performance:**

Una de las características principales de este lenguaje de desarrollo es la performance general del mismo comparado con otros lenguajes de alto nivel. Si bien es posible obtener un desempeño igual o superior en otros lenguajes comerciales (como Java o C#), Golang permite un buen desempeño sin desarrollo adicional.

Arquitectura general de la solución

Con el objetivo de implementar las herramientas definidas previamente, se propuso una arquitectura general que contenga a dichas herramientas y permita la comunicación entre las mismas para alcanzar la solución al problema (*figura 2*).

Esta arquitectura cuenta con un módulo de lectura que se comunica con el de procesamiento de señales mediante un esquema *producer-consumer*.

En este proceso, el módulo de *Data Input* ingresa datos de los datasets preexistentes en el filesystem del servidor, o bien los obtiene en tiempo real a través del puerto serie al que se conecta el oxímetro de pulso. Estos valores se ingresan en una *queue* consumida por el módulo de *Pulse Processor*.

El módulo de procesamiento de señales obtiene la data del módulo de *Data Input,* procesa valores en un buffer interno y los analiza hasta completar un pulso. Una vez identificado, se busca la onda dícrota y se encola el mensaje para ser consumido por el módulo de visualización y evaluación.

El módulo de visualización y evaluación tiene la opción de mostrar los datos procesados del módulo de procesamiento de señales. También permitirá que un usuario experto de *feedback* sobre la calidad del módulo de procesamiento.

La integración entre los diferentes módulos se ejecutan de forma asíncrona.

Cabe destacar que la implementación de las *queues* se realizó por medio de *go channels* mientras que los módulos de *Data Input, Pulse Processor* y cada *query HTTP* se ejecutan en *goroutines* independientes. Esto permite que un delay en la capa de transporte o al procesar un pulso en particular no afecte los tiempos de todo el sistema.

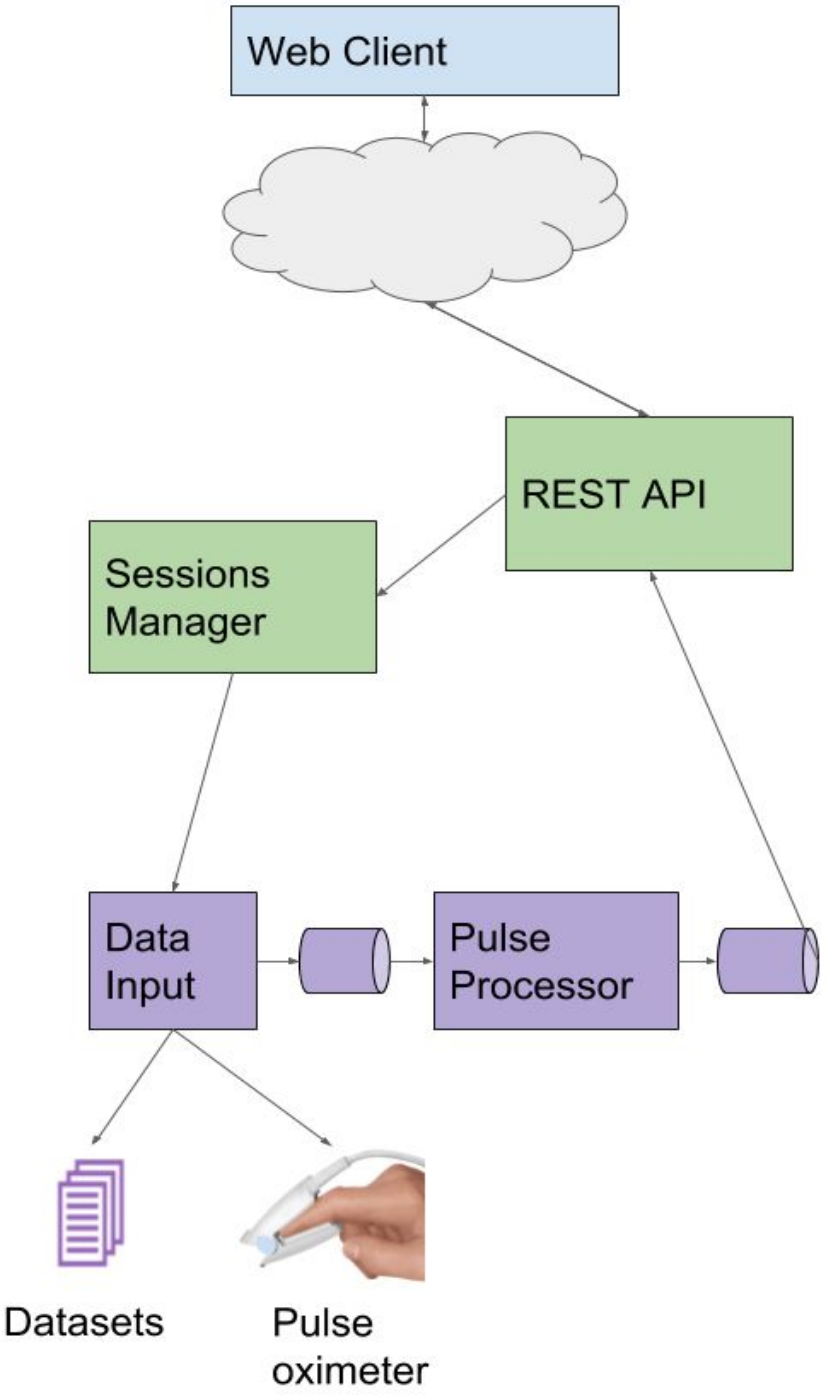


Figura 2 - Diseño de la arquitectura de la aplicación, contemplando todos los módulos individuales y la relación entre los mismos.

Módulo de lectura de señales de pletismografía de pulso

Integración con placa comercial

Uno de los principales objetivos del presente trabajo es obtener y procesar las señales de pletismografía de pulso a partir un oxímetro de pulso en tiempo real. Con este fin, se integró a la aplicación implementada la placa comercial **Integrated Analog Front End (AFE) for Pulse Oximeters** (AFE4490) de Texas Instrument (*figura 3*). Como se describe en la página de producto de Texas Instrument [7] :

El dispositivo consta de un canal receptor de bajo ruido con un convertidor analógico-digital (ADC) de 22 bits, una sección de transmisión de LEDs y diagnósticos para la detección de fallas de sensores y LED. La flexibilidad de configuración permite al usuario tener control completo de las características de temporización del dispositivo. El dispositivo se comunica con un microcontrolador externo o un procesador host mediante una interfaz SPI™.

Figura 3 - Placa comercial AFE4490 de Texas Instrument.

Implementación

Como se detalló al desarrollar la arquitectura de la aplicación en la Propuesta 2, la generación de datos de entrada se encuentra modularizada y abstraída del algoritmo de procesamiento de pulsos y del cliente web que muestra los valores obtenidos. Gracias a estas características de diseño, es posible reemplazar el módulo original que carga muestras de un *dataset* preexistente por uno que las obtenga del puerto serie al cual está conectada la placa.

Si bien la arquitectura propuesta evita propagar las modificaciones realizadas al resto de los módulos en la aplicación, fue necesario desarrollar estrategias para mitigar la latencia en la lectura de las ondas (impuesta por la capa de tráfico y la velocidad del cliente para consumir pulsos procesados y renderizarlos correctamente). Para ajustar la latencia, se optó por reducir la cantidad de muestras encoladas al procesador de pulsos, en función de la cantidad de valores encolados en un momento dado.

Para interactuar con la placa AFE4490 la aplicación instancia una *goroutine* que, en paralelo a los otros flujos de la aplicación definidos previamente, controla los procesos de *input* y *output* a través del puerto serie. Esta rutina escribe en la placa instrucciones para recibir un conjunto de paquetes, y luego lee de la misma desencodeando cada paquete y enviándolo por un *channel* a la instancia del *pulse processor* asignada.

Es importante mencionar que, dado que en este caso el valor que se busca analizar no es la saturación de oxígeno en sangre sino las características del pulso, se decidió utilizar la lectura de un único led (en este caso rojo) y dejar de lado las lecturas del segundo. También cabe mencionar que la placa utilizada cuenta con sensores de luz ambiental para poder eliminar el ruido que esta tiene en la lectura de absorción.

A nivel de la interfaz gráfica, la modularización del módulo de generación de datos permitió que esta integración fuera transparente. A la hora de solicitar los archivos disponibles del *dataset*, el servidor intenta abrir la conexión con el puerto serie. En caso de que esta conexión esté habilitada, se agrega la entrada “Real Time Oximeter” dentro de las opciones disponibles

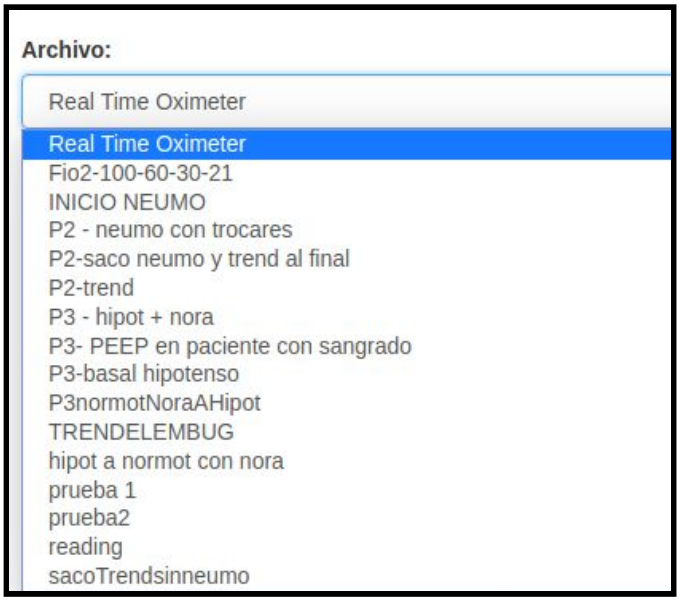


Figura 4 - Lista de posibles orígenes de datos para el algoritmo

Resultados

En la *figura 5* se ve una muestra de un paciente en condiciones normales, con un estado de reposo y la mano en la que se encuentra el oxímetro a la altura del corazón, se puede observar que la señal obtenida presenta características similares a las presentes en los sets de datos iniciales. Como consecuencia, se lograron validar las premisas del proyecto. En este caso se puede observar la correcta división en pulsos por parte del procesador, así como una acertada detección de la onda dícrota.

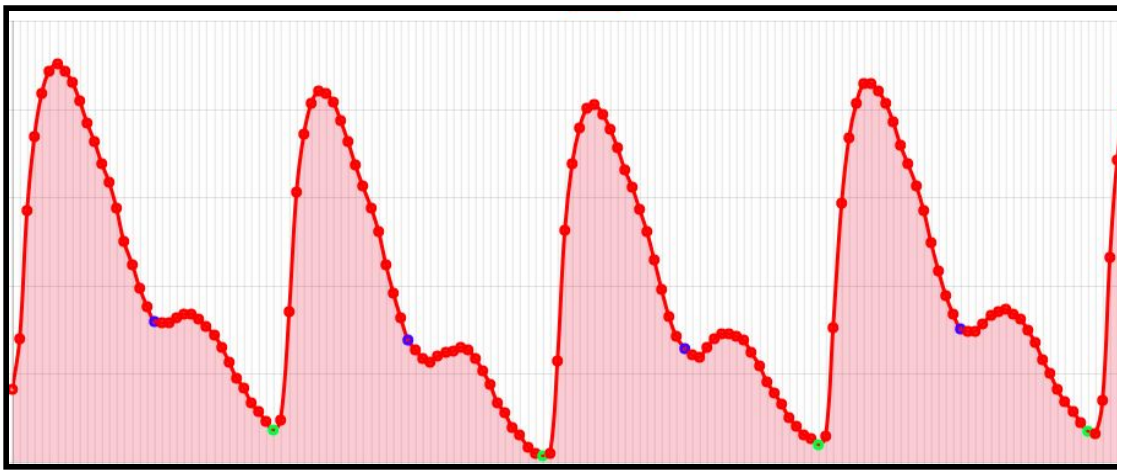


Figura 5 - Muestra de paciente en reposo

Otro escenario evaluado con la placa comercial AFE4490 es el de levantar el brazo en el que se encuentra el oxímetro de pulso. Una muestra de esta caso se puede ver en la *figura 6*. El comportamiento esperado es ver un cambio marcado en los pulsos de entrada causado por el volumen de sangre desplazándose del brazo, lo cual se comporta igual que una disminución en la volemia y por lo tanto en la cantidad de sangre capaz de absorber la luz emitida por el oxímetro.

Si bien esto puede ser corroborado cuando el cambio en la altura de la mano se realiza gradualmente, un cambio brusco genera errores en las lecturas del procesador de pulsos. Es importante recordar que la aplicación implementada utiliza información obtenida de pulsos previos para configurar y ajustar parámetros de división de pulsos y detección de onda dícrota.

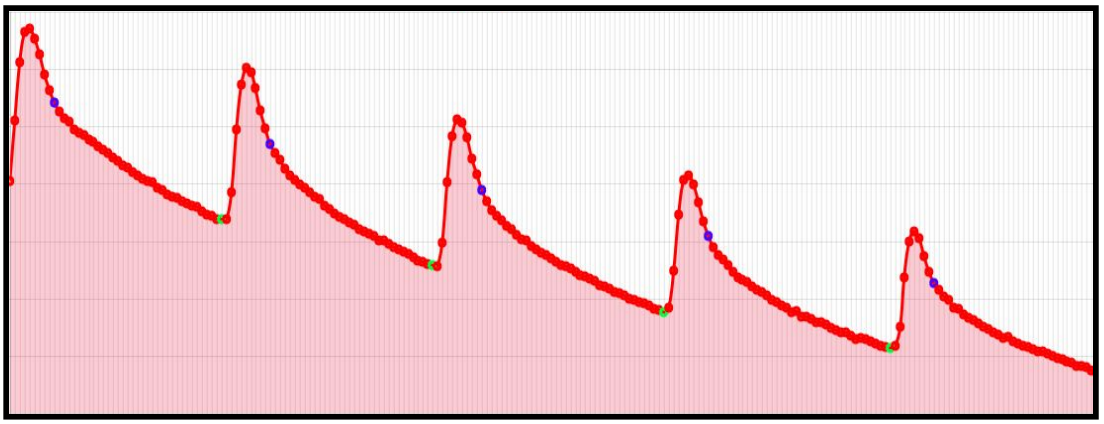


Figura 6 - Muestra de paciente con brazo levantado

Un tercer escenario evaluado es el un paciente con un aumento en el ritmo cardíaco como se muestra en la *figura 7*. En este caso, se observa que la reflexión de la onda es mucho menos marcada. Si bien el detector de pulsos y de onda dícrota funciona correctamente (dado que el indicador principal es la derivada segunda, sensible al cambios en la pendiente), aumentar la frecuencia de muestreo podría arrojado datos más precisos.

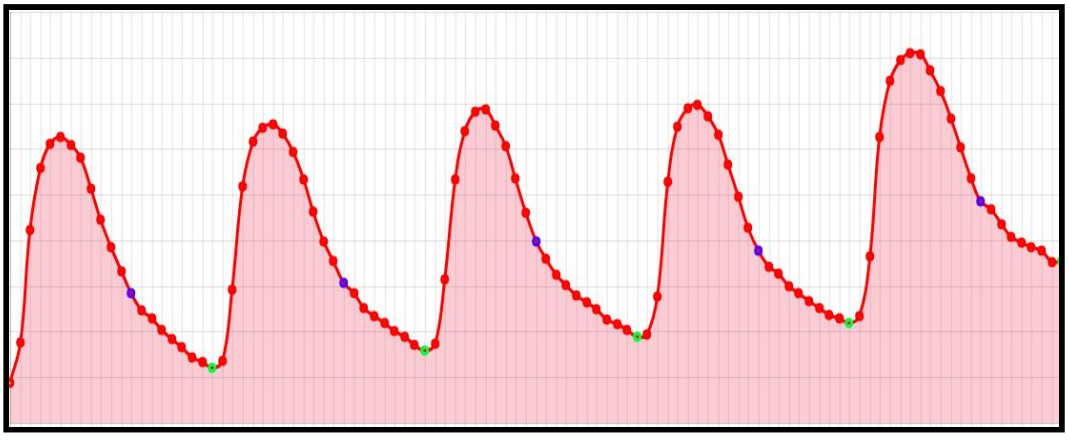


Figura 7 - Muestra de paciente con alto ritmo cardíaco.

Limitaciones

En las pruebas realizadas se identificó la necesidad que el oxímetro de pulso esté conectado cuando se toman las primeras muestras. Esta limitación es inherente al funcionamiento del procesador de pulsos, puesto que depende de la información obtenida de los pulsos previos para ajustar sus parámetros.

Módulo de procesamiento de señales de pletismografía de pulso

Algoritmo de detección basado en la morfología de las señales

Este algoritmo tiene como objetivo consumir y procesar señales de pletismografía de pulso, identificando pulsos individuales y su onda dícrota correspondiente.

Este implementación del módulo de procesamiento de señales se basa en dos características principales de las mismas:

- Cada señal presentaba una morfología similar en cada pulso: una campana con un máximo absoluto en los primeros valores de la señal del pulso y luego una cola larga a derecha hasta finalizar el pulso.

- Luego del máximo absoluto del pulso se puede ver una alteración en la onda distinguible a simple vista, dicha alteración es la onda dícrota que se está buscando. Cabe destacar que la alteración mencionada en alguno casos es mucho más marcada que otros.

Este módulo, en primera instancia, busca dividir la señal original en pulsos individuales que la componen, para luego buscar la onda dicrótica particular de cada uno de esos pulsos.

Para la división en pulsos se decidió analizar la señal original para poder identificar los puntos de la señal que definían el principio y el fin de un pulso. El objetivo de esto era poder caracterizar estos puntos para poder distinguirlos programáticamente y así poder hacer la división de la señal en pulsos.

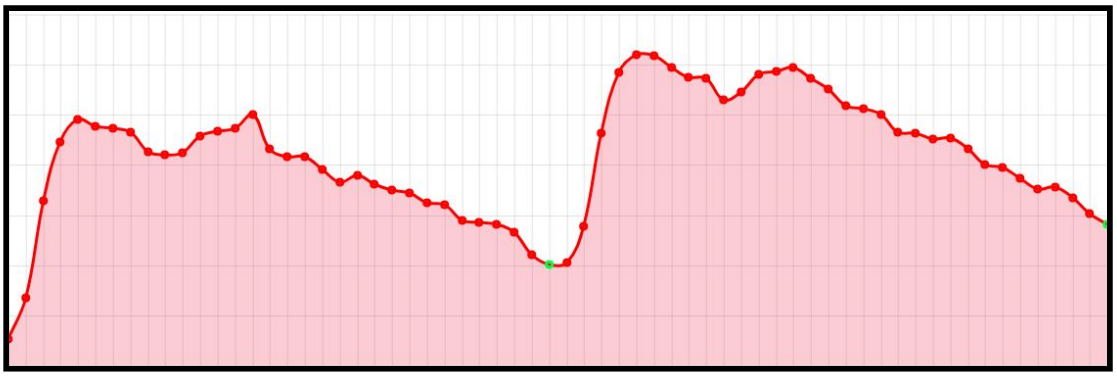


Figura 8 - Captura de pantalla de la morfología de dos pulsos a partir de la herramienta. En verde se marca la muestra considerada por la aplicación como la división entre los dos pulsos.

En la *figura 8* se pueden ver dos pulsos y el punto de color verde que marca el fin del primer pulso y el inicio de segundo. Con base en esta imágen se llevó a cabo un análisis de la morfología de un pulso y de la morfología de la unión de cada pulso. De este análisis podemos ver que el punto se busca es un mínimo local que cumple con dos características:

- los puntos a izquierda se pueden interpolar en una recta con una pendiente decreciente poco pronunciada.
- los puntos a derecha se pueden interpolar en una recta con una pendiente creciente muy pronunciada.

Con base en estas características se dispone a encontrar un forma de distinguir estos mínimos locales en la señal original.

○ Detección de picos para procesar la señal original

Dados los resultados que se obtuvieron en el análisis de la morfología de la señal de pletismografía de pulso, se procedió al desarrollo de una herramienta en el lenguaje GoLang que permitiera la detección de los puntos que representan las división de los pulsos.

Se hizo una búsqueda previa de otros lenguajes que ya implementaran funciones de detección de picos de manera nativa. El lenguaje que se encontró fue Matlab [8]; este lenguaje tiene implementado en su SDK nativo una función

que detecta máximos y mínimos en una señal. Esta función tiene como parámetro un vector que contiene todos los valores de la señal y devuelve un vector con todos los picos de la señal de entrada.

- Derivación de la señal original

Se decidió hacer un preprocesamiento de la señal original de pletismografía de pulso para facilitar la búsqueda de los puntos que marcan el inicio y fin de cada pulso. Este preprocesamiento se basó en derivar la señal original para que resaltar los picos que pudieran aparecer en la señal original.

$$u''(x) \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}$$

Dado que la señal proveniente de la pletismografía de pulso es discreta, para derivar la muestra original se utilizó el método de diferencia finitas centradas.

Figura 9 - (Arriba) Captura de pantalla de pulsos obtenidos de una muestra de pletismografía de pulso. Se marca en azul la detección de onda dícrota, en verde la separación en pulsos. (Abajo) Derivada segunda de la pletismografía.

Para la selección del h se probaron valores entre 1 y 4, teniendo en cuenta que, a mayor h se reduce el ruido pero se pierde precisión en la salida de la función. Finalmente se eligió 3 como valor de h debido a que un valor menor genera mucho ruido en la señal de salida y esto dificulta la detección de picos.

En la *figura 9* se ve una comparativa entre la señal original y su correspondiente segunda derivada con $h = 3$.

- Análisis de la señal original y su derivada

Al realizar un análisis comparativo entre la señal original y su correspondiente derivada segunda punto a punto se observó que la derivada tiene un máximo local que coincide con el mínimo local de la señal de entrada. Esto se debe a que la función en torno al mínimo de la señal de entrada presenta gran variación de pendiente, mientras que en el resto del pulso la pendiente tiende a mantenerse constante.

Figura 10 - (Arriba) Dos pulsos del set de datos “Prueba 2”. (Abajo) la derivada segunda correspondiente.

En la *figura 10* se puede observar la relación entre la onda original y su derivada segunda. Cómo se desarrolló previamente, la derivada segunda se implementó mediante un esquema de diferencias finitas centradas. El código de colores utilizado en esta figura es el siguiente:

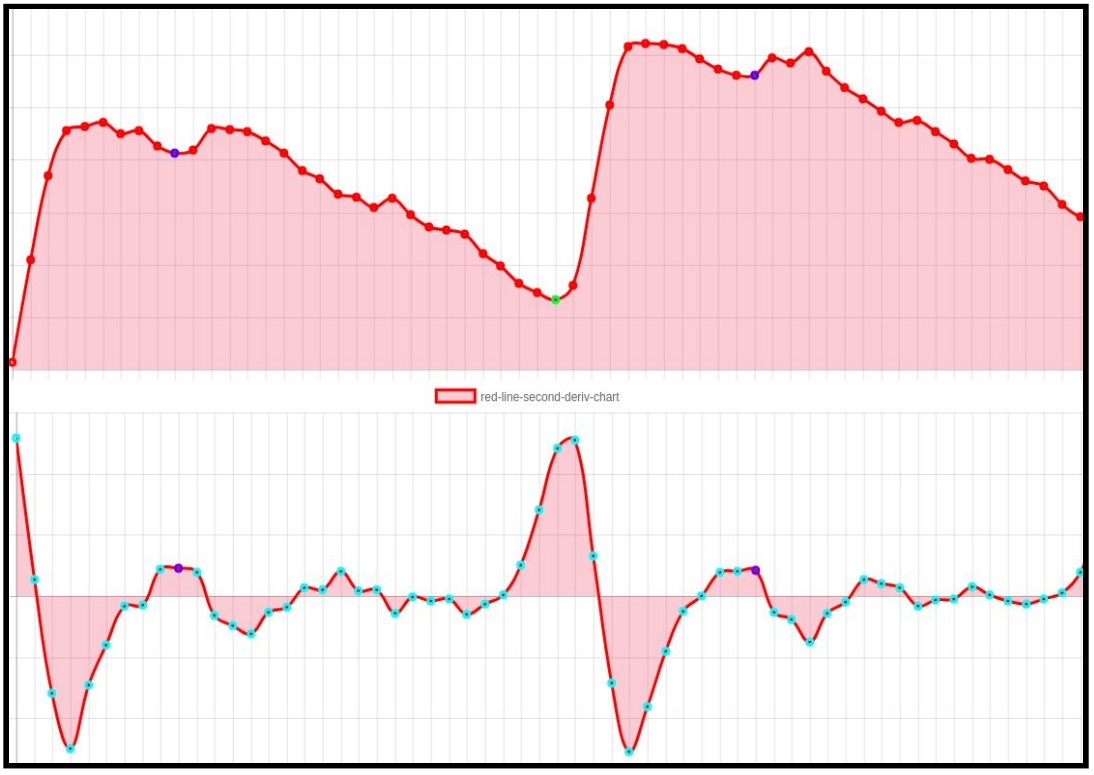
- Los valores finales de cada pulso se colorean en verde con el objetivo de definir a simple vista cada uno.

- Se encuentran coloreados en azul los puntos de la derivada segunda coincidentes con los mínimos de la onda original.

Dado este esquema de colores, es fácilmente identificable la relación entre el máximo de la derivada segunda y el mínimo en el pulso original.

A partir de lo observado en la *figura 10*, y confirmando que este comportamiento es consistente en todas las muestras disponibles, este valor es el que se tomará para definir el inicio de un pulso y el final del otro.

De forma análoga a la propiedad anterior, se identificó que la derivada segunda de un pulso presenta, de forma consistente, un máximo local en el valor correspondiente al inicio de lo onda dícrota del pulso original.



Figur **Figura 11** *- Dos pulsos de la muestra "Prueba 2" (arriba) y la derivada segund correspondiente (abajo) con los inicios de las ondas dícrotas correspondientes.*

A partir de esta propiedad, resulta simple identificar el inicio de cada onda dícrota a partir de un único pulso y su derivada segunda. En la *figura 11* puede verse en azul (en la imagen de arriba) el inicio de la onda dícrota detectado automáticamente por la aplicación. En el gráfico de abajo, se identificaron en

azul los valores máximos (locales) de cada pulso coincidentes con estas ondas.

En la versión final de la aplicación, estos son los puntos que se toman como válidos para la identificación de la onda dícrota.

- Implementación de una función de detección de picos (ad-hoc)

Una parte fundamental para la implementación del módulo de procesamiento de señales es la la función de detección de picos. Esta función es la encargada de marcar los puntos para dividir de la señal original en los pulsos que la componen y luego encontrar la onda dícrota de cada pulso.

En la implementación realizada en este proyecto la función itera sobre un conjunto discreto de valores (en particular, los acumulados en el buffer del procesador de pulsos). Para cada valor del arreglo de entrada, la función compara los valores adyacentes y determina si está en presencia de un máximo local estricto o de un mínimo local estricto.

Esta función presenta orden temporal O(N), siendo N el tamaño del buffer, y además de los valores de máximos y mínimos retorna los índices en los que se encontraron estos valores.

Una vez definida la función para identificar puntos máximos y mínimos locales dentro de la segunda derivada del pulso original, el algoritmo busca un subconjunto con aquellos elementos que se encuentran por sobre un valor determinado. Formalizando este procedimiento, se parte del conjunto E_{max} con los máximos locales identificados mediante la función *peakDetect* previamente definida. Luego se define el conjunto:

$$E'_{max} = \{x \mid x \, \varepsilon \, E_{max} \wedge x \geq \alpha \times v_N\}$$

Donde:

- α es un valor real menor a 1.0 obtenido empíricamente.
- v_N es el promedio de los últimos N máximos de la derivada segunda obtenidos para delimitar pulsos.

En en caso de que, para una determinada señal, el algoritmo no encuentre ningún valor que cumpla con las características previamente mencionadas y $E'_{max} = ø$, entonces al algoritmo disminuye progresivamente el valor de α hasta que exista al menos un valor que cumpla con las características para pertenecer a E'_{max}.

De esta manera se asegura que el algoritmo siempre obtenga al menos un resultado posible. Esta determinación se basó en una de las premisas que tenía las estrategia de detección en la que se asume que una señal va a tener, al menos un pulso con una onda dícrota.

Luego de agregar esta validación se procedió a realizar pruebas para ver el comportamiento del algoritmo con las señales de prueba. Un caso de error que aparecía recurrentemente era el de la *figura 12*.

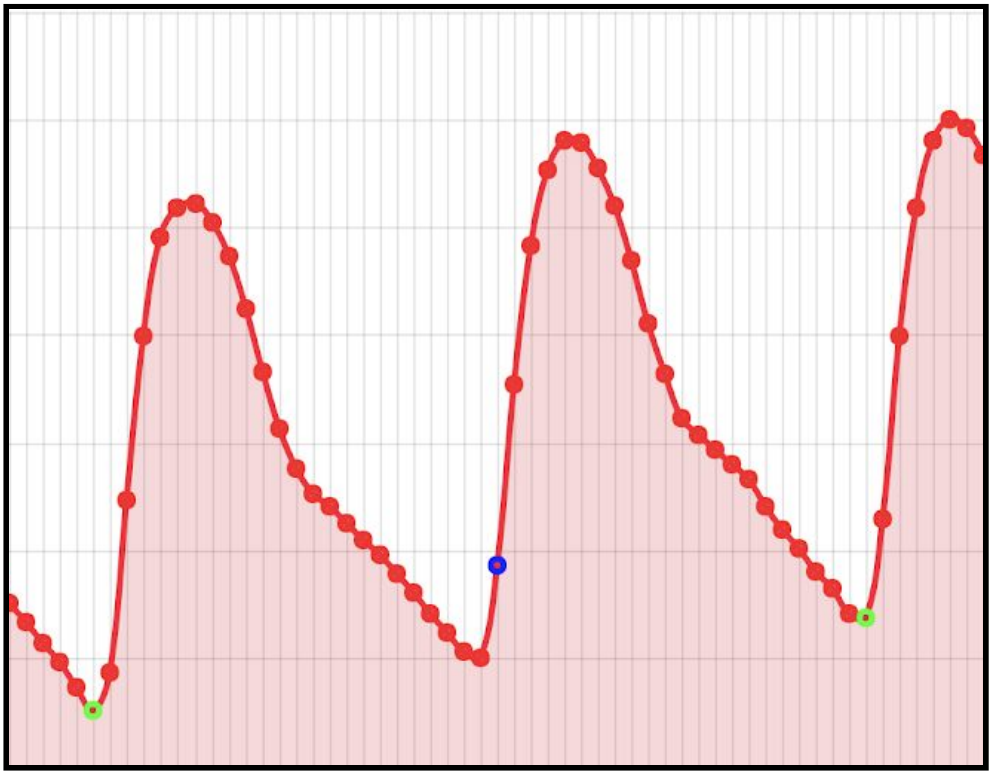


Figura 12 - Dos pulsos mal divididos

Los puntos verdes representan los puntos de división de pulsos hallados por la función de detección de picos. En este caso la función encontró correctamente 2 puntos correspondientes a la división de pulsos, pero omitió el hecho de existía otro punto entre los encontrados que también marcaba el inicio y fin de un pulso.

Por lo tanto para evitar esta selección equivocada de los puntos de la división de pulsos por parte del algoritmo se decidió establecer la siguiente validación:

$l_i \leq ß \times l_N$

Donde:

- l_N es el promedio de la longitud de los últimos N pulsos encontrados.
- l_i es la longitud del último pulso encontrado por el algoritmo
- ß es un valor real mayor a 1.0 y menor que 2.0 obtenido empíricamente.

En el caso de que $l_i > ß \times l_N$, se aplica el mismo procedimiento que en el caso de $E'_{max} = ø$. Se disminuye el valor de α hasta que exista al menos un valor que cumpla con la condición de $l_i \leq ß \times l_N$.

Al agregar esta validación se corrige este error que se encontraba en varios casos.

- Detección de la onda dícrota en cada pulso

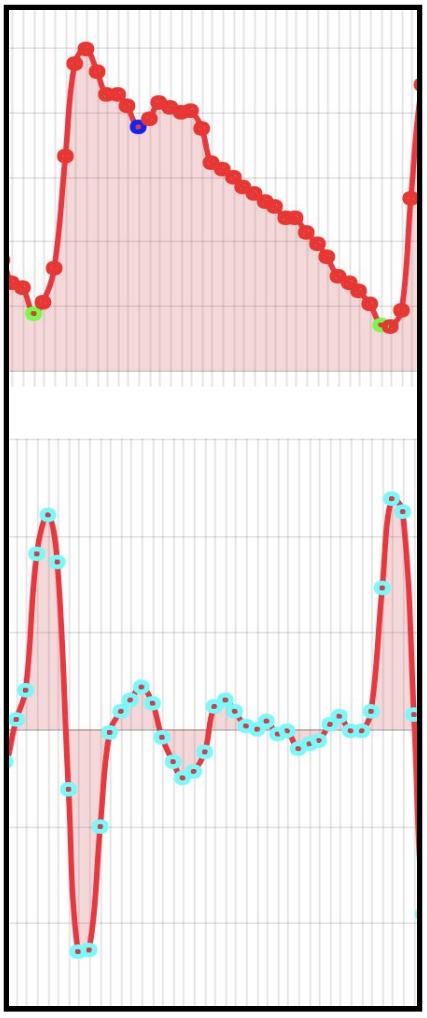


Figura 13 - Muestra de un pulso y su derivada segunda

Una vez discriminado cada pulso, se prosiguió con la búsqueda de la onda dícrota en cada pulso en particular.

Se puede ver en al *figura 13* que la el punto que representa onda dícrota en la señal original, se manifiesta en la segunda derivada como el mayor máximo presente dentro de un pulso (dejando de lado los máximos que delimitan a cada pulso).

Con esta consideración se decidió tomar como la onda dícrota de cada pulso al mayor máximo presente en la segunda derivada de cada pulso. Claro está que se no se tuvieron en cuenta los máximos que señalan el inicio y el fin de cada pulso.

- Procesamiento completo de la señal original

Con todo lo se mencionó anteriormente quedó determinado el procesamiento de la señal de pletismografía de pulso. Este procedimiento detalla el todo el proceso que aplica el módulo de procesamiento sobre la señal recibida:

1. Se recibe la señal original.
2. Se divide la señal recibida en los pulsos que la componen mediante el uso de la segunda derivada de la señal recibida.
3. Se busca la onda dícrota en cada pulso.
4. Se retorna un array de pulsos. El pulso contiene los puntos de la señal original que corresponden a ese pulso, el índice de la posición de la onda dícrota y la segunda derivada de la señal original.

- Uso de funciones ya implementadas de GoLang para detección de picos

Se decidió buscar opciones diferentes para la función detección de picos y así obtener más resultados. Se realizó una búsqueda y una investigación de los

distintos repositorios que poseían este tipo de herramientas, se decidió elegir PeakDetect [4].

Se analizó el código fuente de la herramienta de detección de picos de este repositorio y se vió que poseía un funcionamiento muy similar al de la función nativa de Matlab.

A su vez se empezó a pensar un nueva estructura para el funcionamiento completo del proyecto, más allá de la función de detección de la onda dícrota. Dentro de esta estructura se tuvo en cuenta, como iba a ser la generación de datos, se pensó en una manera de que el sistema

La función PeakDetect posee dos parámetros, uno es el vector con todos los valores de la señal y el otro es un delta. Este delta actúa como threshold para limitar la cantidad de puntos que pueden ser considerados como picos.

Esta función siempre comienza buscando un máximo cuando encuentra uno que supere el delta que recibió como parámetro lo agrega a la array de resultados y luego busca el mínimo siguiente a ese máximo.

El valor de retorno de esta función es una tupla de cuatro arrays. El primer y tercer array son todos los índices de los puntos encontrados por la función (mínimos y máximos respectivamente). El segundo y cuarto array corresponden a los valores de los picos encontrados por la función (mínimos y máximos respectivamente).

Con esta función de detección de picos se procedió primero a implementar la división de pulsos. Se puede ver en la figura X que cada mínimo correspondiente a la finalización de un pulso en la señal original coincide con un máximo fácilmente distinguible en la segunda derivada. Estos máximos en la segundo derivada se filtraban por la función de PeakDetect poniendo un delta equivalente al 70% del mayor máximo presente en la segunda derivada.

Redes Neuronales

La estrategia de uso de redes neuronales consiste en identificar los valores de la onda dícrota a partir de la entrenada mediante aprendizaje supervisado. Este

modelo tiene como objetivo identificar la onda dícrota en función del pulso de entrada y patrones previamente generados.

- Generación de input

El primer enfoque propuesto es implementar un perceptrón multicapa con N neuronas en la capa de entrada (una por cada uno de los N punto obtenidos del pulso inicial) y una neurona en la capa de salida: un valor en el rango [0,1] que indique en qué sección del pulso se encuentra el inicio de la onda dícrota. Frente a este escenario, es necesario contar con una serie de pulsos de longitud N para obtener resultados homogéneos e independientes de la velocidad de muestreo y el tiempo entre latidos.

Luego de identificar que los pulsos presentes en las muestras disponibles no tienen esta propiedad (dada la naturaleza del ritmo cardíaco), se optó por implementar una función para transformar un pulso individual compuesto por una cantidad arbitraria (M) de puntos en uno análogo de longitud fija (en particular N).

Esta función, que llamaremos *normalizarPulso*, puede definirse entonces como:

$$normalizarPulso(i_1, ..., i_M) = (o_1, ..., o_N)$$

La primera aproximación a la implementación de la función *normalizarPulso* se realizó como un promedio ponderado de los puntos de entrada.

En la *figura 14* se muestra un ejemplo con una entrada de longitud M = 29 y una salida normalizada de longitud N = 50. En el ejemplo se puede notar que la forma de la onda es equivalente entre la entrada y la salida, lo cual permite utilizar esta función para el entrenamiento del perceptrón multicapa sin perder las características relevantes del pulso original.

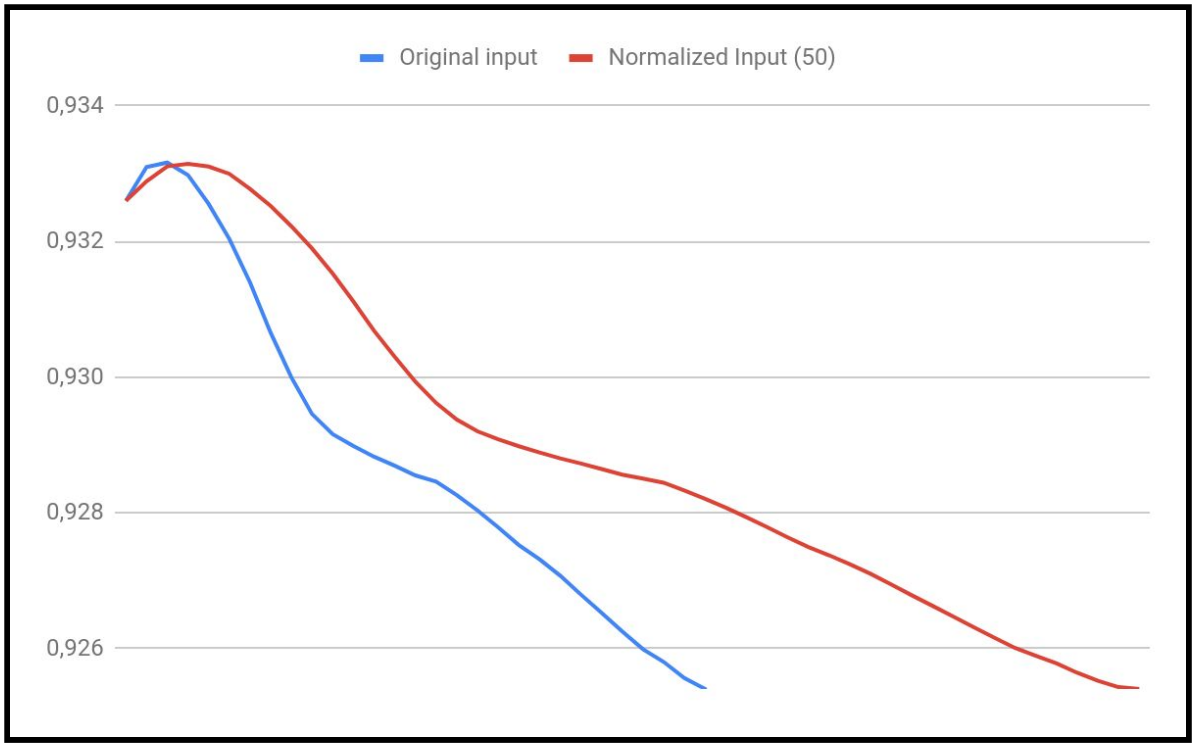


Figura 14 - Ejemplo de normalización de pulso utilizando la función *normalizarPulso* para M = 29 y N = 50

- Framework usado para las redes neuronales

Como prueba de concepto, se propone un perceptrón multicapa. En la práctica, utilizamos una implementación nativa en Golang de código abierto llamada GoBrain [9].

Esta librería incluye una implementación de una red *fast forward*. En la misma se pueden configurar la cantidad de neuronas en cada capa así como los parámetros para las optimizaciones de *momentum factor* y *learning rate*.

Esta decisión marca una limitación de los resultados obtenidos, ya que existe una gran cantidad de modelos de redes neuronales diferentes, y dentro de cada uno existen numerosas implementaciones. Incluso en lo que respecta a la utilización de GoBrain, los parámetros previamente enumerados pueden ajustarse de forma diferente.

Otras implementaciones que se analizaron fueron GoLearn (que cuenta con un clasificador KNN nativo en Golang) y TensorFlow, framework creado por

Google que cuenta con soporte para múltiples lenguajes y una gran variedad de modelos implementados.

Se optó GoBrain por la facilidad de uso, y porque el código resulta simple de leer. Si bien carece de funcionalidades básicas (como la posibilidad de persistir un modelo entrenado), estas no son indispensables para la prueba de concepto que se buscada y son fácilmente implementables de ser requeridos en un futuro.

○ Generación de patrones y entrenamiento

GoBrain define el tipo de un patrón individual como una dupla de vectores (`pattern [][]float64`). El primer elemento de esta dupla es un vector con tantos valores como neuronas de entrada, y el segundo vector representa los valores esperados en las neuronas de salida. En particular, en este proyecto todos los patrones poseen N valores en la primera posición del patrón (capa de entrada) y un único valor en la capa de salida.

Con el objetivo de implementar estos patrones, es necesario conocer dónde se encuentra la onda dicrótica en un conjunto de pulsos dado.

Para esto se implementó un servidor web que separa muestra original en pulsos individuales, los normaliza a una cantidad predefinida de puntos (con el método desarrollado previamente) y lo muestra en una interface web. Cuando usuario selecciona el valor correspondiente a la onda dicrótica, los valores del pulso normalizado y el índice de la onda dicrótica son persistidos en formato JSON en el servidor.

Una vez realizado este procedimiento, el servidor utiliza los patrones persistidos para entrenar el modelo y luego devuelve la predicción de ondas dicróticas para un archivo de entrada deseado. Estas predicciones se devuelven por la API HTTP en formato JSON, y son representadas en forma de gráficos lineales por el cliente web.

○ Implementación

Para la función *normalizarPulso* mencionada previamente, se implementó una función de orden N (tamaño de la salida normalizada que se espera). Esta implementación es independiente del tamaño de la muestra original y admite

que esta sea de longitud mayor o menor que la de la salida normalizada esperada.

El servidor utilizado para levantar pulsos de un filesystem, normalizarlos y servirlos mediante una API REST se implementó con la librería Gin Gonic (https://github.com/gin-gonic/gin), un web framework que permite montar un servidor HTTP concurrente. Este mismo servidor recibe el input del usuario, entrena el modelo fast forward y muestra en una interfaz web las ondas dicróticas encontradas.

Para el cliente web se busca simplicidad y flexibilidad de desarrollo. Dado que la funcionalidad principal del cliente es consumir una API REST y los pulsos obtenidos del servidor, se decidió implementarlo en Javascript sin frameworks adicionales. Se incluyó jQuery para simplificar la interacción con las estructuras de datos y con el el servidor, y GraphJS para renderizar los gráficos de pulsos y obtener valores del usuario.

Evaluación por expertos

Con el objetivo de determinar el correcto funcionamiento del módulo de procesamiento de señales, se realizaron modificaciones a los módulos de lectura de señales y de visualización de pulsos.

La versión modificada del módulo de lectura de señales selecciona, aleatoriamente, uno de treinta sets posibles. Cada set está comprendido por un conjunto de pulsos consecutivos de los muestreos originales. En particular, se componen por 27 pulsos:

- Los primeros 6 pulsos del set son procesados y no se muestra al evaluador. Esto busca generar información contextual que, como se describió previamente, ajusta parámetros del algoritmo dinámicamente
- Los 15 pulsos del medio del set son los que se muestran al evaluador. La interfaz, una vez asignado un set al azar, permite seleccionar en un pulso el valor correspondiente a la onda dícrota.
- Los ùltimos 6 pulsos en el set se agregaron para poder visualizarlos a la hora de seleccionar el valor del último pulso evaluado. Esto busca ayudar al evaluador a tener una referencia visual de la señal completa.

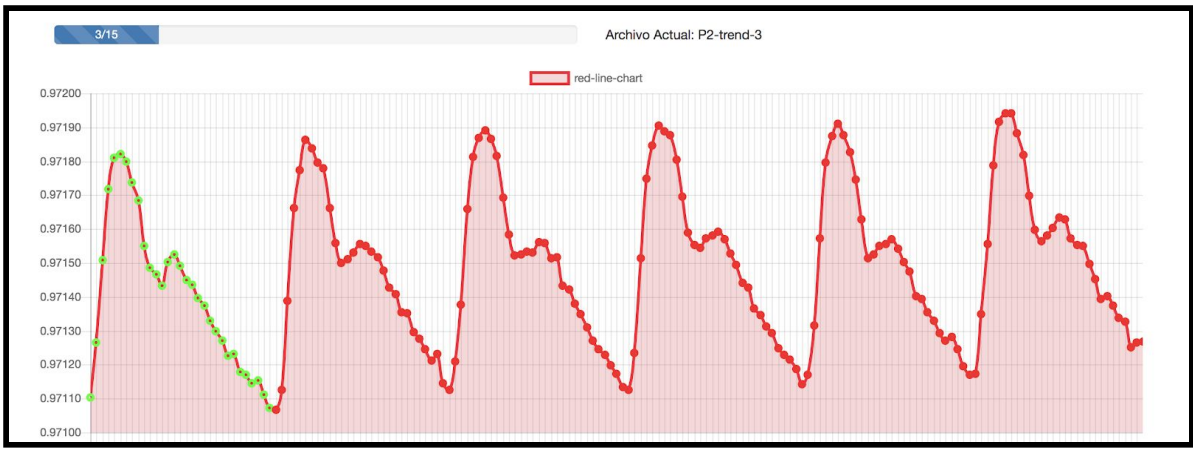


Figura 15 - Captura de pantalla de la interfaz gráfica provista para la evaluación por expertos del dataset "*P2-trend-3*".

Los cambios principales al módulo de visualización consistieron en agregar la posibilidad de seleccionar un pulso, avanzar el *consumer* de pulsos a demanda, seleccionar un set al azar y mostrar el avance de la evaluación. También se coloreó el pulso a ser evaluado en verde para diferenciarlo de los resto de la muestra incluida para dar contexto (como se puede ver en la *figura 15*).

Una vez que el experto completa la evaluación de los 15 pulsos para el set, estos son persistidos mediante una request HTTP en una base de datos relacional para ser evaluados posteriormente. Por otro lado, al evaluador se le presenta un set diferente para continuar el proceso.

6. Resultados

Para probar las 3 estrategias descritas en la sección anterior se tomaron los sets de señales disponibles y se analizaron con todas las estrategias. Estos sets se encuentran distintos tipos de señales que varían tanto en forma como en amplitud.

Antes de presentar los resultados de cada estrategia se definen los siguientes conceptos:

- **Latidos Verdaderos**: aquellos que coinciden el resultado de la estrategia con la onda dícrota seleccionada por el equipo.

- **Latidos falsos**: el resultado de la estrategia no coincide con el valor de la onda dícrota seleccionada por el equipo.

○ Implementación de función de detección de picos ad-hoc

Los resultados obtenidos para el módulo de análisis de morfología de la pletismografía de pulso mediante una función de detección de picos muestran la alta tasa de acierto con la que herramienta logra separar pulsos e identificar el índice de la onda dícrota por cada uno.

Latidos Verdaderos	93.3%
Latidos Falsos	7.7%

En la *figura 16* se pueden observar muestras de diferentes conjuntos de datos analizados con esta estrategia. La escala de colores es la que se describió previamente en este informe (coloreando en verde la división en pulsos y en azul la posición de la onda dícrota).

En las muestras incluidas en la *figura 16*, puede apreciarse una variedad de características. Si bien no es incluye la derivada segunda correspondiente a cada muestra, es importante destacar que la misma es utilizada para la división de los pulsos y la detección de la onda dícrota de cada uno.

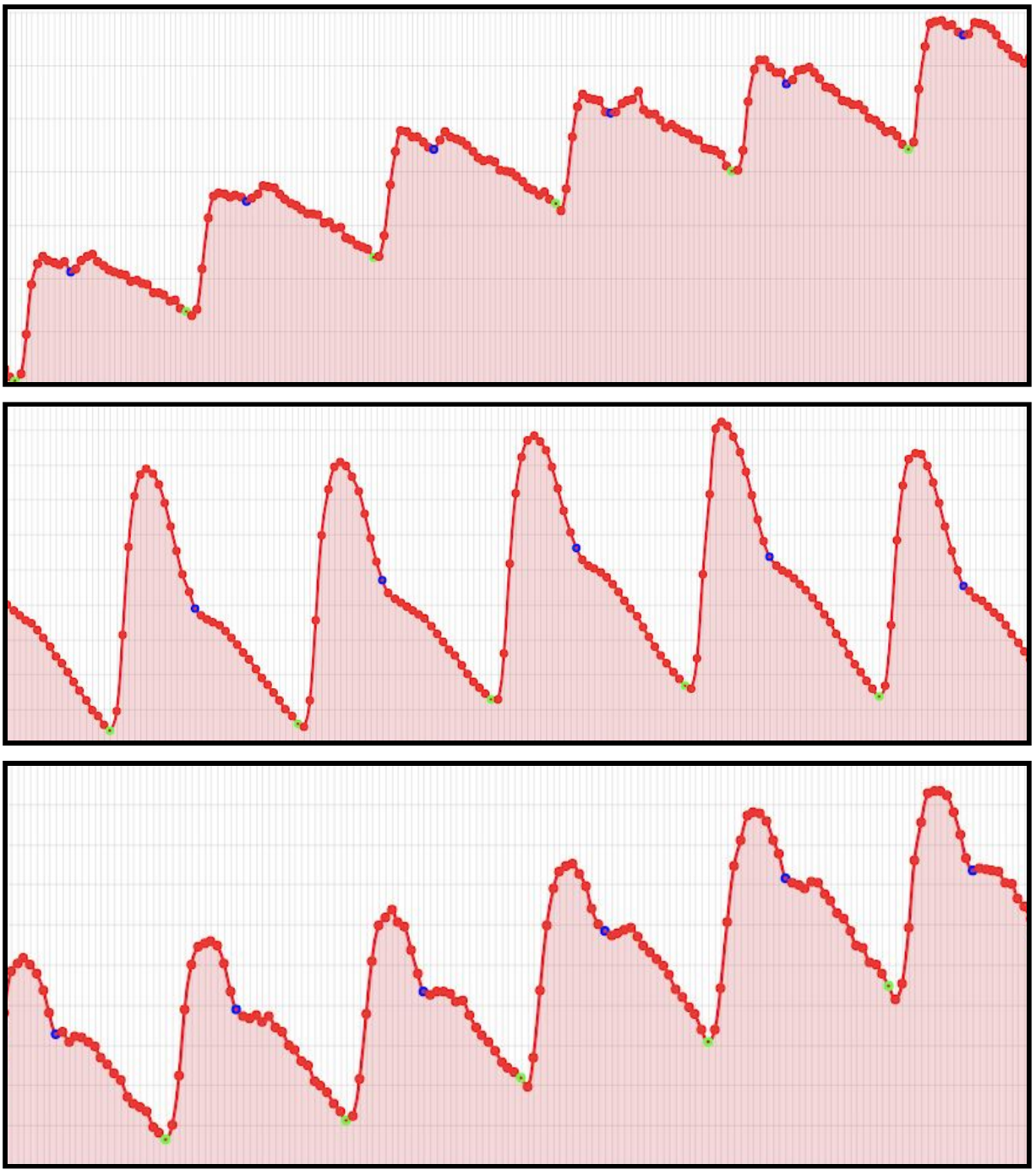


Figura 16 - Resultados de la estrategia *ad hoc* de detección de picos para diferentes muestras, utilizando la escala de colores desarrollada previamente.

○ Implementación de una función de detección de picos Nativa de GoLang

Con la utilización de la herramienta PeakDetect para la detección de picos se obtuvieron los siguientes resultados.

Latidos Verdaderos	65.3%
Latidos Falsos	34.7%

En las pruebas realizadas se vió que en varios casos de prueba los resultados eran muy satisfactorios y el algoritmo realizaba de manera correcta, tanto la división de pulsos como la detección de la onda dícrota. Pero luego se notó que existían otros casos de señales en los que el algoritmo no sólo no encontraba correctamente la onda dícrota, sino que ni siquiera realizaba correctamente la división en pulsos.

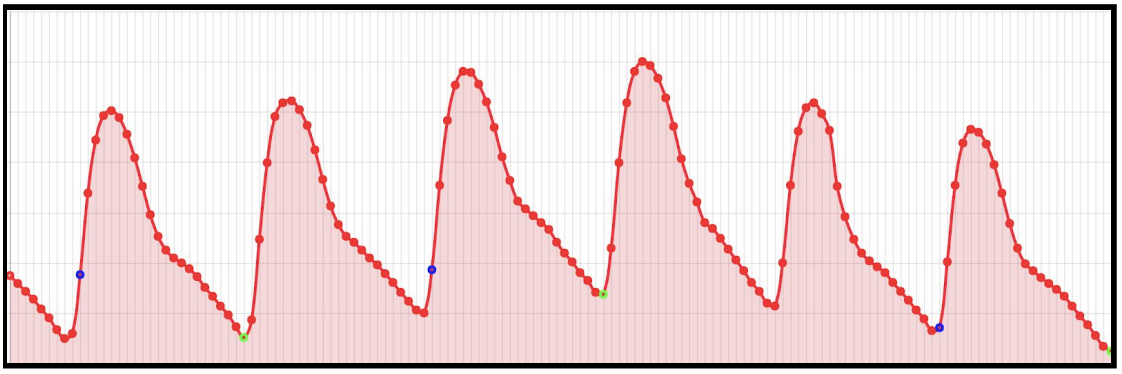


Figura 17 - División incorrecta de pulsos

En la *figura 17* se puede ver que la división en pulsos (señala por los puntos de color verde) es incorrecta. Esta división equívoca genera también que error en la búsqueda de la onda dícrota. Ante esta falla, se optó por cambiar el delta que recibía como parámetro la herramienta PeakDetect buscando obtener mejores resultados.

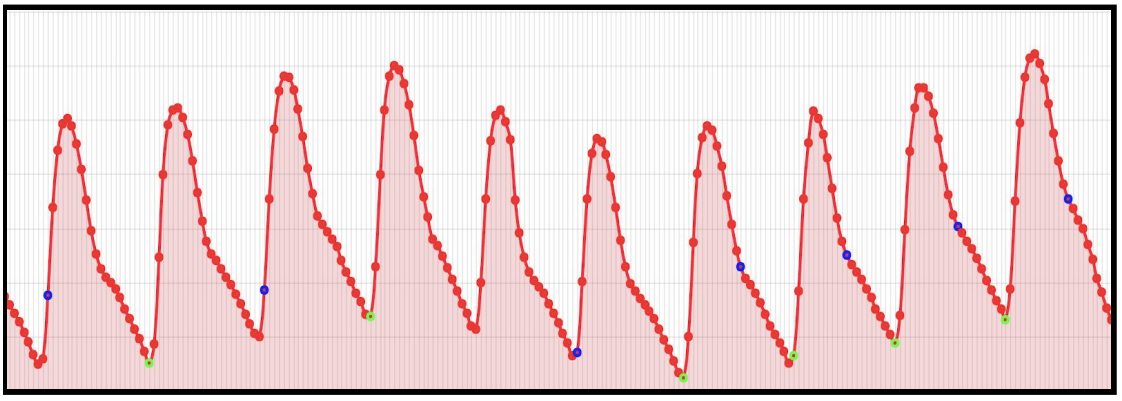


Figura 18 - División con variaciones en el delta de la función PeakDetect

En la *figura 18* se muestra que, si bien hubo mejores resultados con las variaciones del parámetro delta de la herramienta PeakDetect, siguen habiendo varios casos para los cuales la división en pulsos es errónea.

Otra variación que se tuvo en cuenta para tratar de mejorar los resultados que se estaban obteniendo fue ampliar el rango de derivación en el método de diferencias finitas. Se aumentó el valor de h, previamente definido en 3, a 4 y 5. Con este cambió lo que se buscó es reducir lo suficiente el ruido de la señal original para que los picos más pronunciado de la señal original se destaquen en la segunda derivada.

Más allá de los cambios que se implementaron en el delta de la función de derivación, el porcentaje de resultados correctos no aumentó.

Redes Neuronales

En este proyecto se realizaron pruebas con los siguientes parámetros de entrenamiento del perceptrón multicapa previamente definido (a los que se llegó de forma empírica):

Total de iteraciones	1.000.000
Total de patrones de entrenamiento	900
Learning rate	0.005
Momentum factor	0.1

Con estos valores, se logró un error sobre el conjunto de patrones de entrenamiento de 0.18%. Con el modelo entrenado, se procedió a evaluar las respuestas luego del entrenamiento para pulsos fuera del conjunto de patrones de entrenamiento.

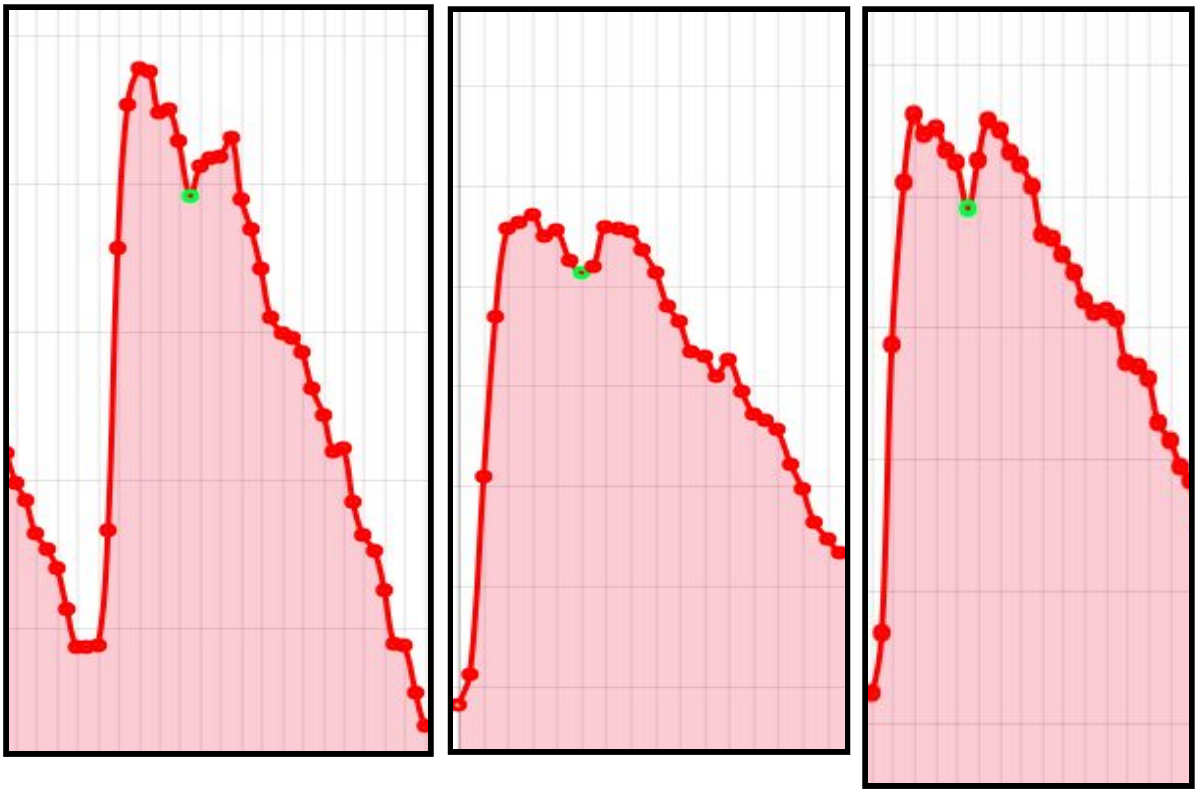


Figura 19 - Resultados de la estrategia de redes neuronales

Resultados de la red neuronal para los cuales la predicción (en verde) es correcta, con los parámetros definidos en la *figura 19*

Para evaluar la precisión del modelo entrenado con los parámetros definidos previamente, el equipo analizó la predicción de la red para 1000 pulsos diferentes. Los mismos se dividieron en tres categorías:

Latidos Verdaderos	19.2%
Latidos Falsos	80.8%

Análisis de evaluación por expertos

Al analizar los 280 valores obtenidos de tomar la diferencia entre el índice de la dícrota obtenido por la aplicación y por expertos, se obtuvo que el 90.3% de los valores cuenta con una diferencia absoluta menor o igual a 3 puntos muestreados (o 300 ms) del pulso original, y 95.5% tienen una diferencia menor a 4 puntos (o 400 ms).

Para realizar un análisis estadístico del conjunto de datos, se utilizó la mediana en aquellos casos en los que se obtuvieron evaluaciones por expertos redundantes (más de una para un mismo set de datos). Esto permitió eliminar *outliers* de este conjunto de datos que es tomado como fuente de verdad.

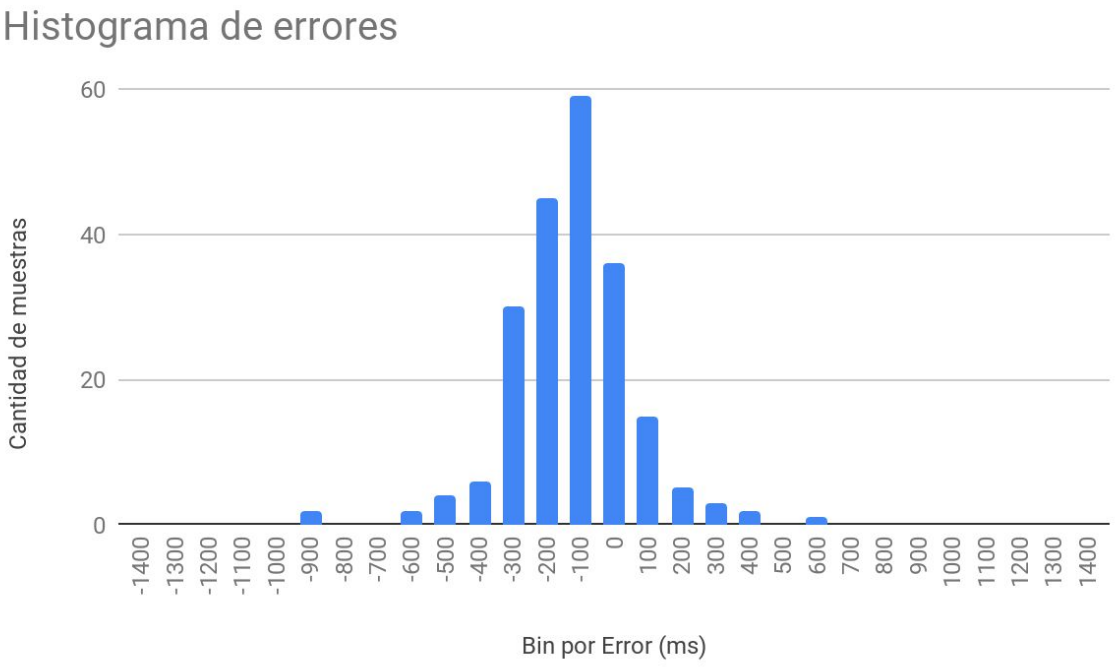


Figura 20 - Histograma de errores sobre los conjuntos de datos muestreados.

Definiendo el error como la diferencia entre el valor de referencia de los expertos con el punto del método se puede cuantificar la medida de error, del

análisis del histograma generado a partir de los errores individuales de cada pulso para el conjunto de muestras (*figura 20*), se puede observar una distribución similar a la normal. Para corroborar esta observación, se pueden extraer los siguientes indicadores sobre el conjunto de datos de errores (en ms):

Media	-130 ms
Mediana	-100 ms
Moda	-100 ms
Desvío estándar	186 ms

Una vez obtenidos los indicadores, se puede generar la distribución normal a partir de la media y el desvío estándar para posteriormente superponerlo en un gráfico con el histograma obtenido. Es importante destacar que la distribución normal generada fue escalada por un factor de 210 (el total de muestras obtenido luego de utilizar la redundancia en la evaluación por expertos para descartar outliers).

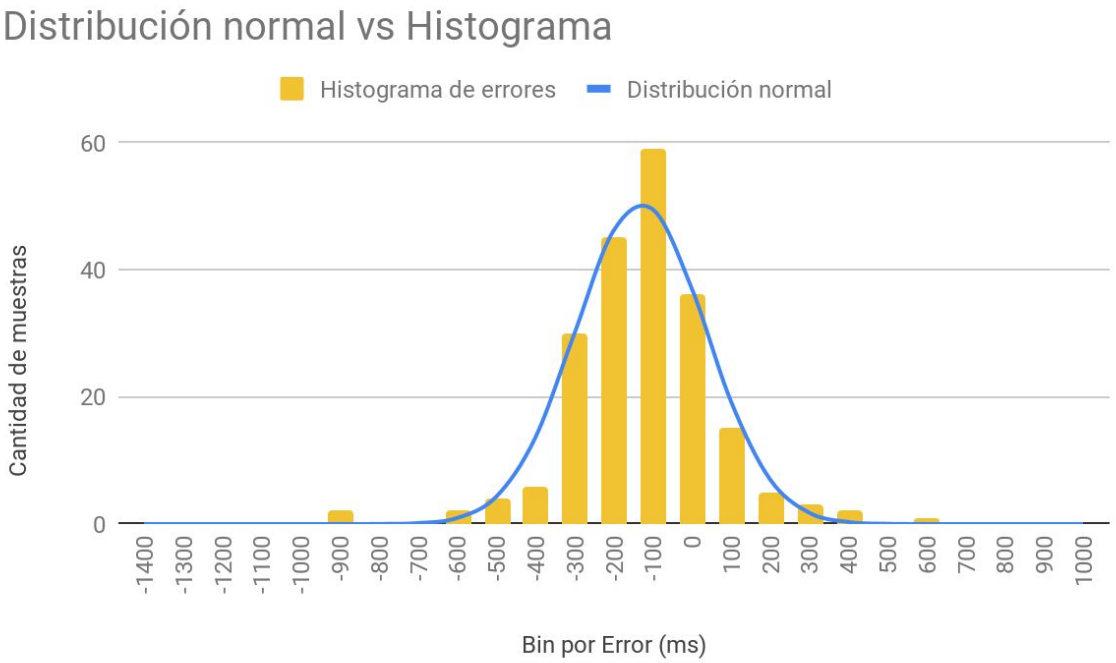


Figura 21 - Histograma de errores sobre los conjuntos de datos muestreados (amarillo) superpuesto con la distribución normal generada a partir de los indicadores estadísticos del conjunto de datos original.

Al comparar la distribución normal generada a partir de la mediana y desvío estándar con el histograma real de errores (*figura 21*), se puede observar la tendencia de la curva de errores a esta distribución.

7. Discusión

Las 3 estrategias de detección fueron sometidas a las mismas pruebas, con las mismas señales de pletismografía de pulso y luego se compararon sus resultados.

Como se mencionó previamente, dado que se disponen de pocas muestras, la evaluación se realizó sobre un conjunto de señales pequeño.

Redes

Latidos Verdaderos	19.2%
Latidos Falsos	80.8%

Implementación de una función de detección de picos ad-hoc

Latidos Verdaderos	65.3%
Latidos Falsos	34.7%

Implementación de una función de detección de picos nativa de Golang

Latidos Verdaderos	93.3%
Latidos Falsos	7.7%

De las pruebas realizadas se puede ver que la estrategia de detección por medio del uso de redes neuronales obtuvo el menor porcentaje de aciertos a la hora de detectar la onda dícrota en cada pulso. No se sabe con certeza a que se deben los malos resultados obtenidos por esta propuesta. Una hipótesis para estos resultados es que, debido a la variabilidad de las señales de la

pletismografía de pulso, se le hacía muy difícil a la red adquirir la especificidad suficiente para realizar mejores predicciones.

Un último aspecto que cuestiona la estrategia de redes neuronales es que, según una de las premisas establecidas, se asume que todo pulso de las señal de entrada posee una onda dícrota asociada, y esta estrategia obtuvo resultados nulos (no se encontró onda dícrota) para algunas señales de prueba.

En contraste con los resultados de la estrategia de redes neuronales se muestran los resultados de la estrategia de detección con funciones de detección de picos nativas de GoLang. Esta estrategia mostró resultados mucho más acertados a la hora de las detección de la onda dícrota en la señales de prueba. Cabe destacar que esta estrategia no presentó resultados nulos y cumplió con todas las premisas que se establecieron de forma previa al desarrollo.

Más allá de la mejora en los resultados obtenidos por la estrategia de detección con funciones de detección de picos nativas de GoLang respecto de la estrategia de redes neuronales, el porcentaje de resultados fallidos en esta propuesta sigue siendo muy elevado (a razón de 1 de cada 3 resultados).

Es importante mencionar que gran parte del desarrollo de las estrategia de división en pulsos con uso de funciones de picos se fue dedicado a la implementación de la arquitectura de productor-consumidor. Esta arquitectura permitió abstraer perfectamente la lógica de obtención de los datos de las señales de pletismografía de pulso y por esta razón se decidió conservar la misma arquitectura en las estrategias de basadas en funciones de detección de picos.

Un aspecto interesante es que la estrategia con uso de funciones nativas Golang para detección de picos y la estrategia con uso de funciones de ah-doc son muy similares en estructura y funcionamiento. Mas allá de esta similitud, los resultados que se obtuvieron con ambas estrategias son muy distintos, esto demostró la importancia de la función de detección de picos para el funcionamiento del sistema.

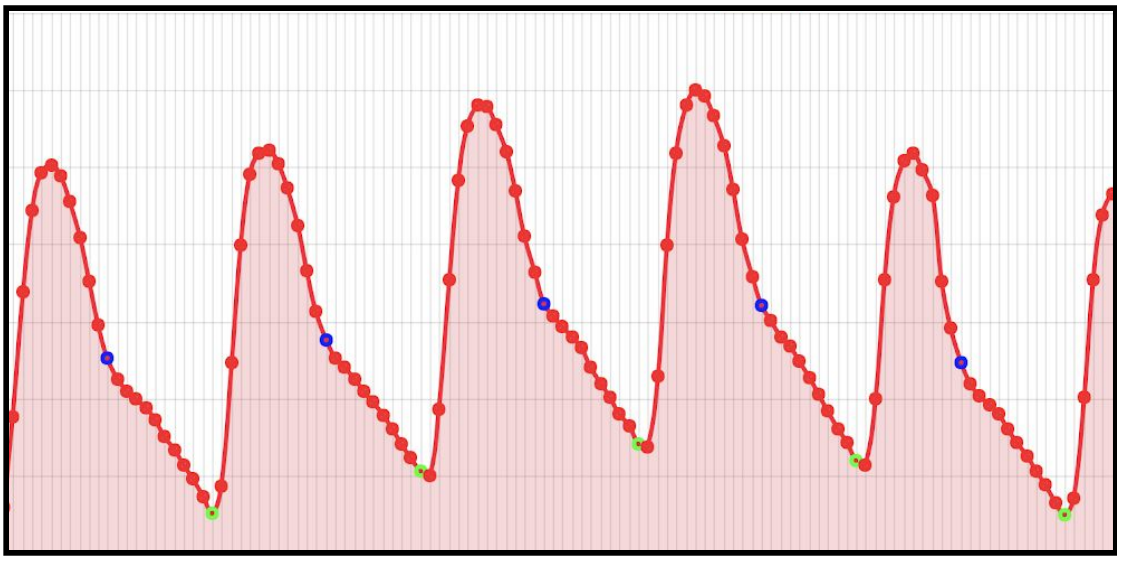


Figura 22 - caso de falla de otra estrategia de detección con funciones ad-hoc que fallaba con funciones nativas de GoLang

En la *figura 22* se puede ver que para el mismo caso en el que la estrategia con funciones de Golang fallaba en la división de pulsos, esta la estrategia con funciones ah-doc respondió correctamente y logró dividir correctamente cada pulso y encontrar u onda dícrota.

Los resultados obtenidos por la estrategia de división de pulsos con uso de funciones de detección de picos ah-doc fueron muy superiores a los resultados mostrados por las demás estrategias y se decidió presentarla como propuesta final para la evaluación con expertos.

Los resultados muestran el potencial de esta herramienta en el ámbito del análisis de señales orientado a la resolución de un problema médico. Para una validación clínica, es necesario llevar a cabo un análisis mucho más exhaustivo que tenga en cuenta casos extremos como vasodilatación y vasoconstricción severa, que excede a los intereses de este trabajo.

Casos de interés

Las pruebas que se realizaron con diferentes tipos de señales arrojaron resultados muy positivos. A pesar de esos se detectaron algunos pocos casos en los que la detección de la onda dícrota no fue correcta.

En este caso la división en pulsos fue correcta, el error está en la detección de la onda dícrota. Esta falla se debe a que el mayor máximo presente dentro de cada pulso no se corresponde con la onda dícrota y por ello el algoritmo señala erróneamente el punto correspondiente a la onda dícrota.

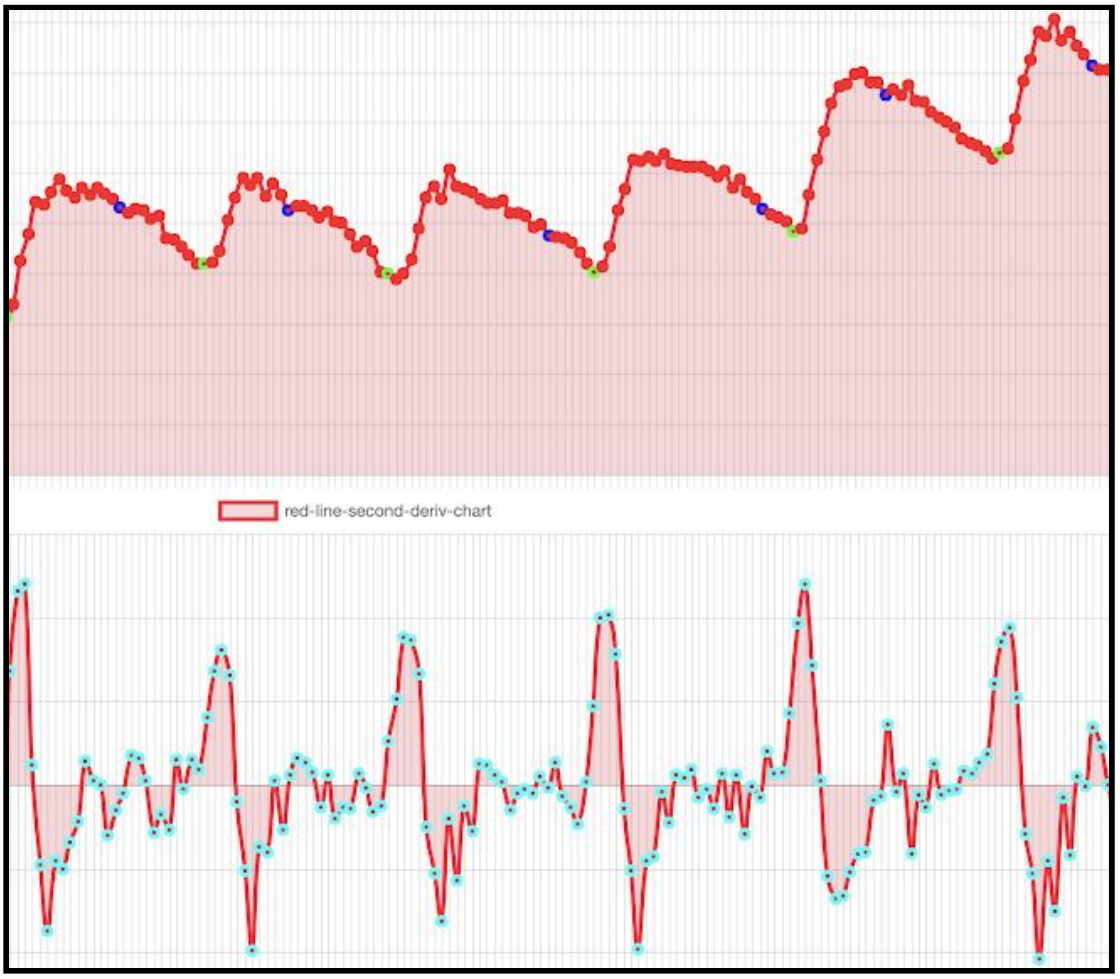


Figura 23 - Error en el cálculo de la posición de la onda dícrota

Debido a las pocas ocurrencias de este fenómeno en todas las muestras usadas para probar el algoritmo de detección, se decidió mantener el criterio utilizado de selección de la onda dícrota.

Como se vió anteriormente la señal de la pletismografía de pulso tiende a ser similar pulso a pulso, tanto en su forma como en su frecuencia. Si bien la forma y la frecuencia varían con el tiempo, si se comparan dos pulsos inmediatamente adyacentes estos tienden a ser de similares características.

Una ocasión de falla del algoritmo se encuentra ante el procesamiento de una señal que posee pulsos de extrema irregularidad respecto de los pulsos inmediatamente anteriores. En estos casos el algoritmo no pudo dividir correctamente el pulso en encontrar la onda dícrota correctamente.

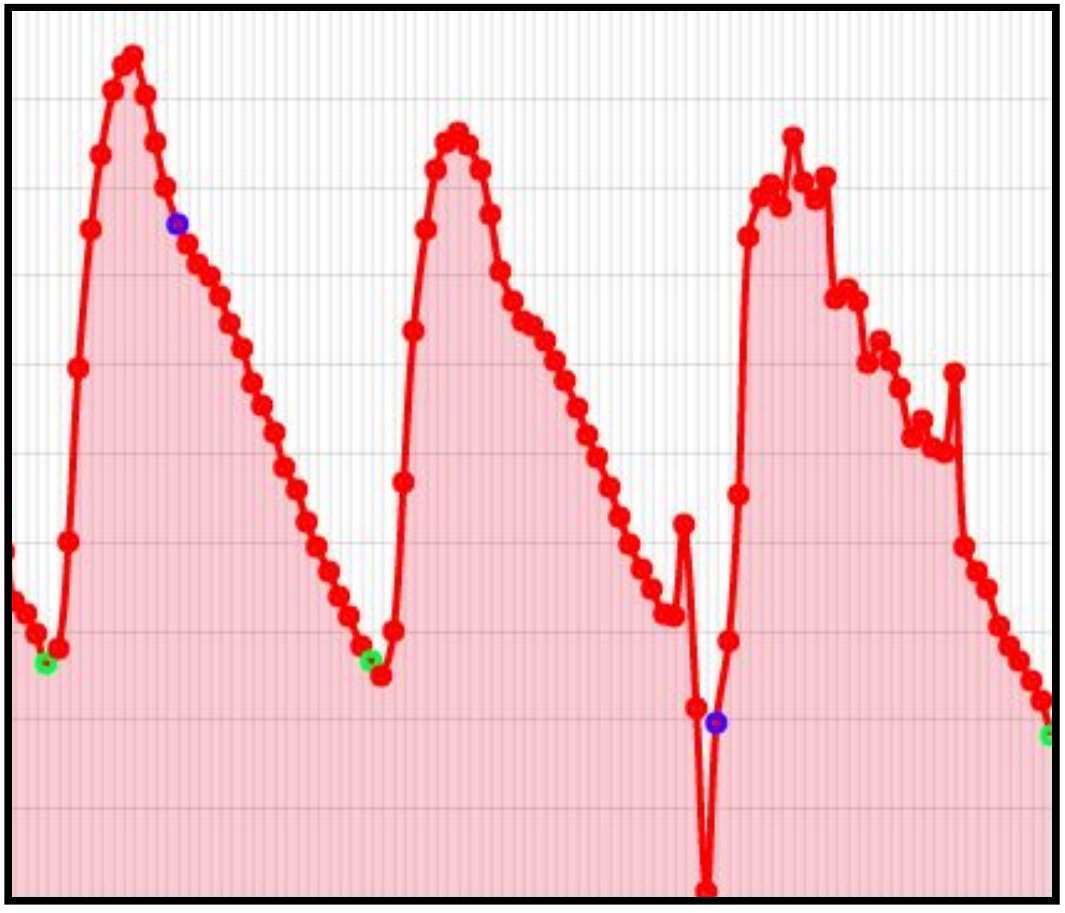


Figura 24 - Muestra con pico abrupto

En el caso de la *figura 24* en el final del segundo pulso se produce un descenso muy abrupto, a comparación de los valores previos de la señal. Por esta razón el algoritmo no puede realizar correctamente ni la división ni la búsqueda de la onda dícrota.

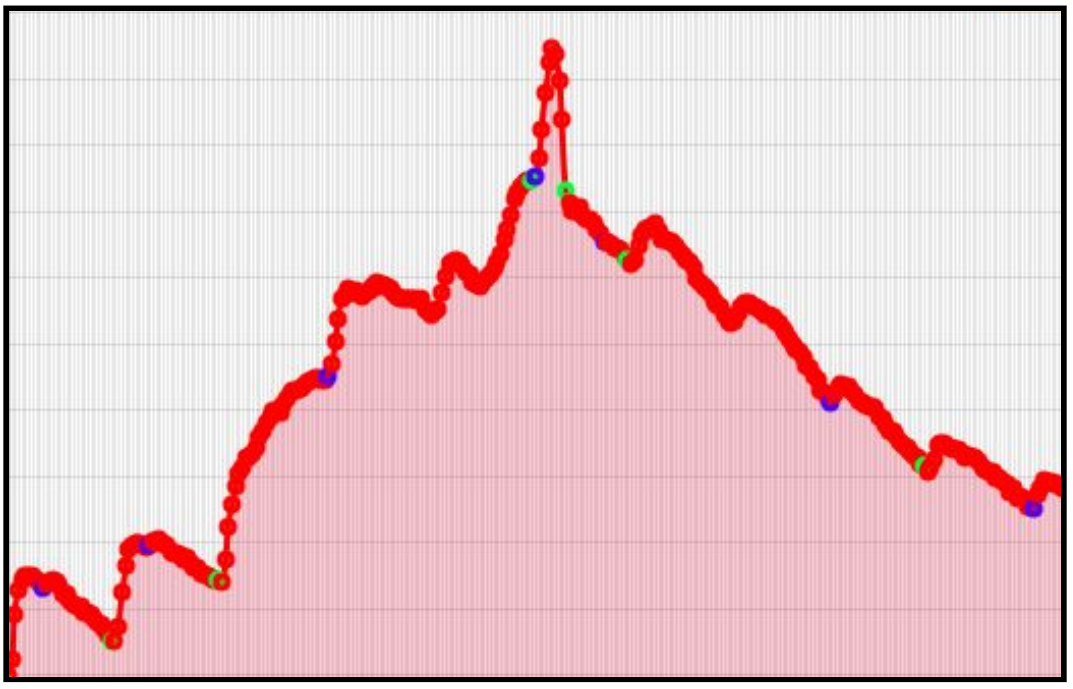


Figura 25 - Señal con pico pronunciado

Se puede ver en *figura 25* una parte de una señal de prueba que posee dos características muy distintivas. La primera y más visible característica es que uno de los pulsos centrales de la señal está claramente desfasado respecto del resto. La segunda característica distintiva de esta señal es que la cantidad de puntos por pulso era menor al promedio, ocasionando pérdida de información al hacer la derivación de la señal.

Por estas características particulares de la señal se deducen los errores de división y de búsqueda de la onda dícrota.

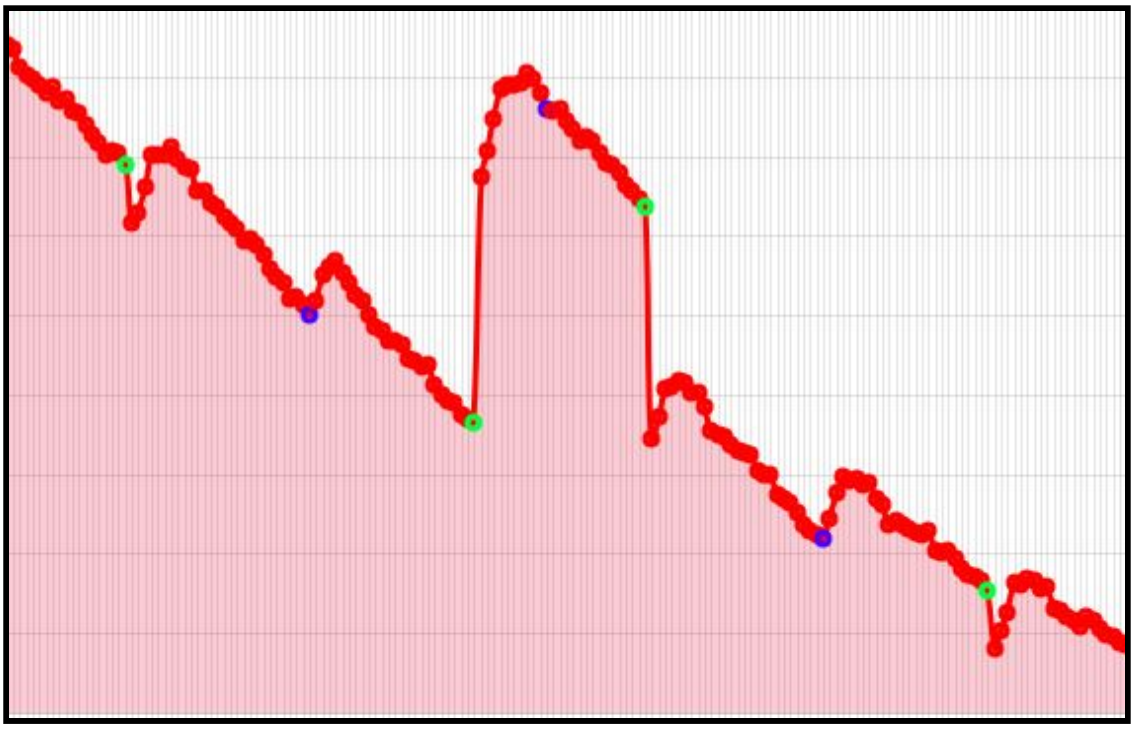


Figura 26 - Señal con error de muestreo

En la *figura 26* se puede ver que los dos primeros pulsos son similares y se procesaron de manera correcta. El resto de los pulsos posee una forma completamente distinta e irregular de manera que el algoritmo no puede determinar correctamente el fin y el comienzo de cada pulso.

No se sabe realmente cual es el motivo de estos cambios tan abruptos de señal, pero se le puede atribuir a un error de medición por parte de aparato. No obstante cabe destacar que en estos casos de señales irregulares, cuando la señal se estabiliza nuevamente, el algoritmo de detección también lo hace.

Cabe destacar que las señales con artefactos mencionadas son de baja frecuencia, por lo que no afectaría al usuario.

Limitaciones de la implementación

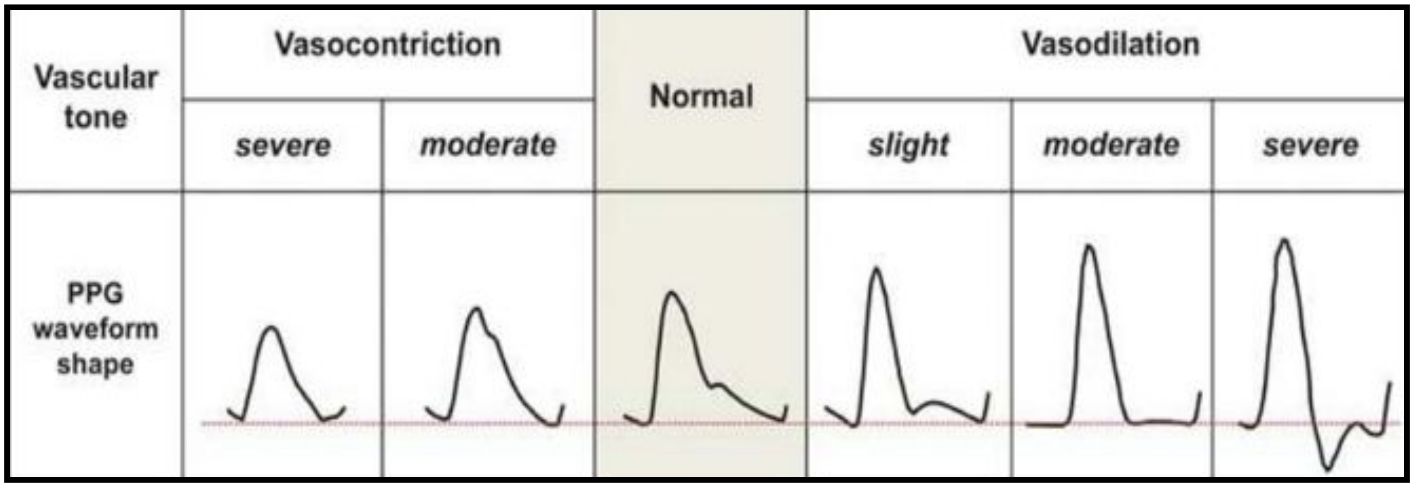


Figura 27 - Tipos de onda de pletismografía de pulso en diferentes condiciones vasculares.[4]

El cuadro comparativo que se muestra en la *figura 27* con la forma de los distintos tipos de señales en base al tamaño de las sección transversal de los vasos sanguíneos. En la estrategia presentada para la evaluación por expertos se contemplan los casos Normal, Vasoconstricción moderada y Vasodilatación leve y moderada.

En el caso de Vasoconstricción severa, la presión sanguínea es alta obstaculizando la propagación de onda mecánica a través de los vasos sanguíneos, y evitando la reflexión de onda sistólica en el capilar. El resultado de la vasoconstricción severa es la supresión completa de la onda dícrota en cada pulso. No obstante en este caso, esta estrategia de detección va a obtener un resultado de una posible onda dícrota cuando no debería.

El caso de Vasodilatación severa es completamente opuesto al anterior, la presión sanguínea en esta caso es bastante baja y ocasiona una gran separación entre los pulsos. Debido a esta separación entre pulsos la reflexión de la onda sistólica en el capilar, se vé en la señal tomada por el aparato como otro pulso independiente. Esto ocasiona que el algoritmo de la estrategia

presentada tome a la reflexión de la onda sistólica como un pulso más cuando no lo es.

Dado que estos errores en la aplicación se producen en latidos aislados, se puede disminuir fácilmente el impacto de las mismas mediante un método de ventana deslizante con una estrategia de mediana

8. Conclusiones

A continuación detallan conclusiones particulares de cada una de las estrategias de detección expuestas en este trabajo.

○ Estrategia con funciones de detección de picos ad-hoc

Esta estrategia obtuvo el mejor porcentaje de latidos verdaderos en las pruebas realizadas con todo tipo de señales, que varían tanto en forma como en frecuencia.

La particularidad de esta estrategia es que posee cierto tipo de “memoria”, ya que las validaciones para un pulso en particular se hacen en base a los pulsos encontrados anteriormente. Esta memoria fue el factor que diferenció a esta estrategia de respecto de la estrategia que usaba funciones nativas de GoLang.

Otro factor importante a destacar de esta estrategia es la capacidad de reponerse ante una situación de falla. Como se pudo ver en la sección de casos de falla, la propuesta logró volver a dividir correctamente los pulsos luego de un tiempo pasada la señal irregular.

○ Estrategia con funciones de detección de picos nativas de GoLang

Como conclusión para esta estrategia se puede decir que, mediante el uso del método de diferencias finitas para derivar la señal original, se pudieron obtener latidos verdaderos en determinados tipos de señales de entrada. Pero para otros tipos de señales, esta estrategia arrojó una gran cantidad de latidos falsos.

Se destaca que los latidos falsos obtenidos con esta estrategia se dieron en casos con señales que no poseían ninguna característica distintiva o particular y que estaban dentro de los casos contemplados para este trabajo.

Se debe hacer un análisis más intensivo para determinar posibles mejoras para esta estrategia debido a que, pesar de haber modificado varios parámetros de

la función de PeakDetect y el valor de *h* en el cálculo de la segunda derivada, los resultados obtenidos mantuvieron el porcentaje de latidos falsos.

○ Estrategia con Redes Neuronales

Respecto de esta estrategia que consiste en utilizar un perceptrón multicapa con entrenamiento *fast forward* con el objetivo de identificar la onda dícrota para un pulso dado, no se pudieron obtener resultados concluyentes.

Si bien el modelo logró obtener un 19.2% de latidos verdaderos y es superior al 3% que se obtendría utilizando una variable aleatoria con distribución uniforme de 32 puntos por pulso, dicho porcentaje no se consideró suficiente para dar por finalizada el estudio de esta estrategia.

La propuesta de entrenar un modelo puede ser sujeta a diversas modificaciones, tanto de parámetros como del modelo seleccionado para entrenar y el mecanismo de generación de patrones, por lo que los resultados obtenidos no son concluyentes.

Una extensión posible para esta estrategia sería la especialización de varias redes neuronales para cada tipo de onda dícrota. De esta manera se podría entrenar la red solo con un tipo de pulso determinado y obtener mejores resultados en la detección de la onda dícrota. Otro aspecto a tener en cuenta en esta extensión es que se deberá tener un mecanismo que reciba la señal de pletismografía de pulso y seleccione qué red es la más adecuada para encontrar la onda dícrota de dicha señal.

○ Conclusiones finales

Con los resultados que Obtenidos se puede concluir que el lenguaje de desarrollo GoLang posee un gran potencial para brindar soluciones al problema del análisis de señales de pletismografía de pulso.

Se desarrolló un sistema completo que logró consumir datos de señales de pletismografía de pulso, procesarlas para obtener su onda dícrota y mostrar los resultados en en tiempo real mediante un servidor web.

El lenguaje Golang aportó varias herramientas de caracter nativo que funcionaron de manera óptima en este trabajo. Se destaca el uso de los

channels para manejar el flujo de información y las *rutinas* para el manejo de concurrencias.

9. Trabajo Futuro Referencias

A lo largo de la de la realización del presente trabajo, se detectaron posibles mejoras a futuro que permitirían mejorar los resultados obtenidos.
A continuación, se enumeran las principales oportunidades de mejora para desarrollos futuros:

+ Soportar cambios bruscos en morfología de la señal, sin que estos afecten el procesamiento de pulsos posteriores.
+ Soportar condiciones iniciales incorrectas, ajustando dinámicamente los parámetros utilizados en el análisis
+ Usar valores de ambos leds para tener redundancia en las lecturas en lugar de únicamente el proveniente del led en espectro rojo.
+ Ajustar dinámicamente la cantidad de valores muestreados si no se encuentra con confianza la onda dícrota en el pulso evaluado.

10. Bibliografía

[1] "Golang," *The Go Programming Language*. [Online]. Available: https://golang.org/. [Accessed: 16-Jul-2019]

[2] "Oximetría de pulso," *American Thoracic Society*, Dec. 2013.

[3] D. V. Nieto, "Onda dícrota," *Portales Medicos*. [Online]. Available: https://www.portalesmedicos.com/diccionario_medico/index.php/Onda_dicrota. [Accessed: 16-Jul-2019]

[4] G. Tusman, C. M. Acosta, S. Pulletz, S. H. Böhm, A. Scandurra, J. M. Arca, M. Madorno, and F. S. Sipmann, "Photoplethysmographic characterization of vascular tone mediated changes in arterial pressure: an observational study.," *J. Clin. Monit. Comput.*, Dec. 2018.

[5] L. B. Cook, "Extracting arterial flow waveforms from pulse oximeter waveforms apparatus.," *Anaesthesia*, vol. 56, no. 6, pp. 551–555, Jun. 2001.

[6] R. Carrazana-Escalona. Facultad de Medicina 1. Universidad de Ciencias Médicas de Santiago de Cuba. Santiago de Cuba. Cuba., B. Taimy Ricardo-Ferro. Centro de Biofísica Médica. Universidad de Oriente. Santiago de Cuba. Cuba., R. R. Fernández-de la Vara Prieto. . Centro de Biofísica Médica. Universidad de Oriente. Santiago de Cuba. Cuba., and M. E. Sánchez-Hechavarríal. Facultad de Medicina 1. Universidad de Ciencias Médicas de Santiago de Cuba. Santiago de Cuba. Cuba., " Algoritmo para la detección de puntos clínicos," *Revista Cubana de Informática Médica*. [Online]. Available: http://Algoritmo para la detección de puntos clínicos. [Accessed: 16-Jul-2019]

[7] "AFE4490 Integrated Analog Front-End for Pulse Oximeters datasheet (Rev. H)," *Texas Instruments*. [Online]. Available: http://www.ti.com/product/AFE4490. [Accessed: 16-Jul-2019]

[8] M. América Latina, "Example List - MATLAB & Simulink." [Online]. Available: https://la.mathworks.com/help/simbio/examples.html?s_cid=doc_ftr. [Accessed: 16-Jul-2019]

[9] J. Trevisan, *GoBrain*. github.com, 2019 [Online]. Available: https://github.com/goml/gobrain. [Accessed: 16-Jul-2019]