

UNIVERSIDAD POLITÉCNICA
DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN



GRADO EN INGENIERÍA DE
TECNOLOGÍAS Y SERVICIOS DE
TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

DISEÑO Y DESARROLLO DE UN SISTEMA DE
AYUDA AL DIAGNÓSTICO DE ENFERMEDADES
CARDIOVASCULARES A PARTIR DE
ELECTROCARDIOGRAMAS:
APRENDIZAJE PROFUNDO SEMISUPERVISADO.

PABLO MARTÍN REDONDO

2020

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Trabajo de fin de grado

Título: Diseño y desarrollo de un sistema de ayuda al diagnóstico de enfermedades cardiovasculares a partir de electrocardiogramas: Aprendizaje profundo semisupervisado

Autor: Pablo Martín Redondo

Tutora: Gema García Sáez

Departamento: Tecnología Fotónica y Bioingeniería

Miembros del tribunal

Presidente:

Vocal:

Secretario:

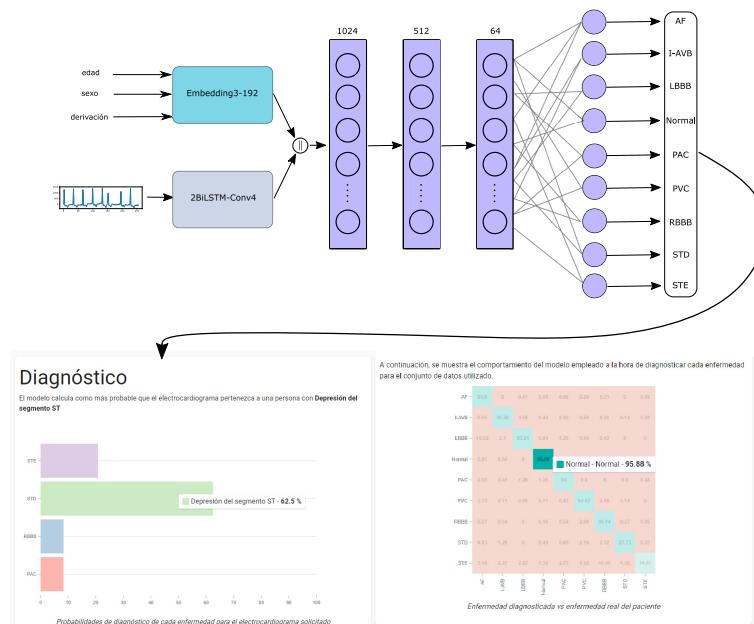
Suplente:

Los miembros del tribunal arriba nombrados acuerdan otorgar una calificación de:

Madrid, a _____ de _____ de 20____

Diseño y desarrollo de un sistema de ayuda al diagnóstico de enfermedades cardiovaseculares a partir de electrocardiogramas:

Aprendizaje profundo semisupervisado.



Alumno: Pablo Martín Redondo
Tutor: Gema García Sáez
*Grupo de Bioingeniería y Telemedicina,
Departamento de Tecnología Fotónica y Bioingeniería
Universidad Politécnica de Madrid*

Resumen

Las enfermedades cardiovasculares son la principal causa de muerte a nivel mundial, provocando un 15% de las defunciones totales (413,153 millones). El grupo de mayor riesgo es el de las personas de la tercera edad. Esto, junto con el hecho de que vivimos en una sociedad cada vez más envejecida, son las motivaciones que llevan a ingenieros y médicos a trabajar en soluciones más eficientes en el diagnóstico para agilizar la detección y reducir el número de defunciones.

La principal prueba de diagnóstico en enfermedades cardiovasculares es el electrocardiograma. Una vez realizado, un cardiólogo experimentado lo revisa a mano y emite un diagnóstico. Los recientes avances en aprendizaje automático, hacen de esta técnica una herramienta muy interesante para la ayuda al diagnóstico de enfermedades cardiovasculares. Así, en este Trabajo de Fin de Grado, se propone el diseño y desarrollo de un sistema de ayuda al diagnóstico a partir de aprendizaje profundo (también conocido como Deep Learning).

De esta forma, el objetivo ha sido, a partir de un electrocardiograma, informar de la probabilidad de que un paciente padezca ninguna, una o varias enfermedades identificando entre nueve clases posibles: fibrilación auricular, bloqueo auriculoventricular de primer grado, bloqueo de rama izquierda, ritmo sinusal normal, complejo auricular prematuro, complejo ventricular prematuro, bloqueo de rama derecha, depresión del segmento ST y elevación del segmento ST. Para abordar este problema, se ha dividido el trabajo en tres fases.

En primer lugar, se han procesado los datos de señales de electrocardiograma. Se ha filtrado la señal del electrocardiograma para eliminar ruido o interferencias. Además, algunas enfermedades son menos comunes que otras y los datos de los que se dispone no están balanceados. Por ello, se ha propuesto el aumento de datos a través de dos técnicas: Redes Generativas Adversarias y la generación de nuevas señales transformando las señales reales.

En segundo lugar, se ha resuelto un problema de clasificación de series temporales. Para ello, se ha creado un modelo de diagnóstico automático basado en aprendizaje profundo con una arquitectura mixta: redes neuronales convolucionales (CNN) y redes neuronales recursivas (LSTM). Esto ha permitido que la red neuronal aprenda tanto de la forma como de la temporalidad de la señal. Además, se han tenido en cuenta parámetros demográficos del paciente: el sexo y la edad.

Por último, se pretende que la herramienta de diagnóstico de enfermedades cardiovasculares pueda utilizarse en un entorno clínico. Para ello, se ha desarrollado un sistema con arquitectura orientada a servicios que permita ejecutar el modelo en un servidor y visualizar los resultados de la herramienta de diagnóstico en el navegador del personal médico.

Palabras clave: Aprendizaje profundo semi-supervisado, Ayuda, CNN, Diagnóstico, Docker, GAN, Ingeniería biomédica, LSTM, Modelos secuenciales, Redes generativas adversarias, Redes neuronales convolucionales, Tensorflow

Abstract

Cardiovascular diseases are the main global death cause, being responsible of a 15% of the mortality (413.153 M). The most in-danger groups are the elderly. This, along to the fact of living in an increasingly aged society, are the motivations for engineers and doctors to work in solutions more efficient in the diagnosis for making the detection more agile and reducing the number of losses.

The main diagnostic test for cardiovascular diseases is the electrocardiogram. Once done, an experienced cardiologist reviews it by hand and provides a diagnosis. The recent progress on deep learning, makes this technique very interesting for the cardiovascular diseases diagnosis help. Hence, this End of Degree Project proposes the design and development of a diagnosis helper tool using Deep Learning.

With this path on mind, the goal has been, given an electrocardiogram, to inform of the probability for a patient to have none, one or multiple diseases between nine different classes: atrial fibrillation, first-degree atrioventricular block, left bundle branch block, normal sinus rhythm, premature atrial complex, premature ventricular complex, right bundle branch block, ST-segment depression and ST-segment elevation. To face the problem, the project has been divided in three phases.

First of all, the data of electrocardiogram signals have been processed. The signal has been filtered to eliminate noise and inferences. Furthermore, some diseases are less common than others and the data set is not balanced. Due to this fact, data augmentation through two different techniques is proposed: Generative Adversarial Networks and new signals generation through transformation of real signals.

In the second phase, a temporal series classification problem has been needed to be resolved. To do this, it has been proposed to create an automatic diagnosis model based on deep learning with a mixed architecture: convolutional neural networks and recurrent neural networks. This allows the neural network to learn not only the shape but also the time evolution of the signal. Additionally, demographic data (sex and age) has been taken into account.

Finally, the intention is to use the tool in a clinical environment. With this goal, a service oriented architecture has been developed. This architecture allows the model to be executed in a server and the results to be viewed at the medical staff's browser.

Key words: Biomedical Engineering, CNN, Convolutional Neural Networks, Diagnosis, Docker, GAN, Generative Adversarial Networks, Help, LSTM, Semi-supervised deep learning, Sequence Models, Tensorflow

Agradecimientos

A Gema García, gracias por creer en mí, por tu apoyo y por tus consejos. Eres en gran medida responsable de mi primer trabajo en investigación y de este Trabajo de fin de grado. A todo el equipo del GBT en general, gracias por cómo me habéis tratado siempre, me sentí pronto dentro de una gran familia. A todos los profesores de la escuela, grandes profesionales que me han enseñado mucho. Gracias, en general, a todos los profesionales que han conseguido inspirarme y levantar en mí la pasión que hoy tengo por la Ciencia y la Tecnología.

A Carlos, Guti, en general gracias a los Ramuras, a mis amigos de la Universidad y de Ayllón. Gracias por ser el lugar en el que refugiararme. Gracias por todos los momentos de desconexión y alegría. También he aprendido mucho de vosotros y quiero seguir aprendiendo. No habría llegado aquí si no hubieseis formado parte de mi vida.

A mi familia, en especial a mi madre, a mi padre, a mi hermana y a mis abuelos. Sois el suelo en el que atenerme, mi punto de apoyo, os debo la vida. Me enseñasteis que la familia es la única constante. No olvidaré nunca esa lección, siempre me tendréis para todo.

En general gracias a todos, porque aunque esto también es fruto de mi esfuerzo, no estaría donde estoy si no hubierais estado a mi lado. Comienza una nueva etapa, prometo estar a la altura.

Índice

Lista de tablas	x
Lista de algoritmos	xi
Lista de figuras	xii
Glosario	xv
Acrónimos	xvii
I Introducción	1
1 Contexto	1
2 Diagnóstico. El Electrocardiograma	2
2.1 Definición	2
2.2 La señal	3
2.3 Procesado	4
3 Las enfermedades cardiovasculares	5
3.1 Ritmo sinusal normal	5
3.2 Fibrilación auricular	5
3.3 Bloqueo auriculoventricular de primer grado	6
3.4 Bloqueo de rama izquierda	7
3.5 Complejo auricular prematuro	7
3.6 Complejo ventricular prematuro	8
3.7 Bloqueo de la rama derecha	9
3.8 Depresión del segmento ST	9
3.9 Elevación del segmento ST	10
II Objetivos	11
4 Potencial del trabajo	11
III Aprendizaje profundo	12
5 Definición	12
6 Entrenamiento	13
6.1 Propagación hacia delante	13
6.2 Descenso del gradiente	13
6.3 Propagación hacia atrás	14
6.4 Algoritmo de aprendizaje	15

7 Funciones, optimización y regularización	15
7.1 Funciones de activación	15
7.2 Normalización y regularización	17
7.3 Optimización	18
8 Arquitecturas	18
8.1 Redes neuronales convolucionales	18
8.2 Modelos secuenciales	20
9 Redes generativas adversarias	22
IV Procesamiento de datos	25
10 Origen de los datos: PhysioNet	25
11 Los datos de entrenamiento	25
12 Preprocesado de los datos	26
12.1 Ruido	26
12.2 Armonización de la entrada	27
12.3 Normalización y resultado final	27
13 Aumento de datos	28
13.1 Generación de datos mediante GANs	28
13.1.1 Arquitectura	29
13.1.2 Entrenamiento y resultados	31
13.2 Generación de datos mediante transformaciones	32
V Modelo de clasificación	34
14 Arquitectura	34
14.1 Primera entrada. Datos cualitativos	34
14.2 Segunda entrada. Electrocardiograma	35
14.3 Modelo final	36
15 Resultados	37
15.1 Entrenamiento	37
15.2 Utilización del modelo y resultados	37
VI Herramienta de ayuda al diagnóstico	41
16 Componentes	41
16.1 Estilo arquitectónico	41
16.1.1 Diagrama de secuencia del sistema (SSD)	41
16.2 Proveedor	42
16.3 Consumidor	42
16.3.1 Vista de solicitud del diagnóstico	43
16.3.2 Vista de ayuda al diagnóstico	43

17 Despliegue	46
17.1 Arquitectura de despliegue	46
17.2 Resultado final	47
VII Conclusiones y líneas futuras	48
18 Conclusiones	48
19 Líneas futuras	49
Bibliografía	50
Anexo A Impacto y responsabilidades	54
A.1 Impacto social	54
A.2 Impacto económico	54
A.3 Impacto medioambiental	54
A.4 Responsabilidad ética, legal y profesional	55
Anexo B Presupuesto	57
Anexo C Búsqueda de un modelo convergente basado en GANs	59
Anexo D Búsqueda de un modelo clasificador	73
Anexo E Manual de despliegue	78
E.1 Obtención de un certificado	78
E.2 Cambio de nombre del dominio	78
E.3 Puesta en marcha	79

Lista de tablas

1	Distribución de enfermedades de los datos de entrenamiento	26
2	Distribución de enfermedades de los datos de entrenamiento tras su procesado	28
3	Precisión de cada modelo y precisión final en el conjunto de datos total.	39
4	Inversión inicial	57
5	Gastos recurrentes mensuales	58
6	Discriminador del modelo entrenado el 22/03/2020	59
7	Generador del modelo entrenado el 22/03/2020	60
8	Generador del modelo entrenado el 23/03/2020	62
9	Discriminador del modelo entrenado el 23/03/2020	63
10	Generador del modelo entrenado el 01/04/2020	65
11	Discriminador del modelo entrenado el 05/04/2020	66
12	Generador del modelo entrenado el 05/04/2020	67
13	Generador del modelo entrenado el 10/04/2020	69
14	Discriminador del modelo entrenado el 10/04/2020	70
15	Clasificador entrenado el 24/04/2020	74
16	Bloque de Fully Connected Layers del modelo clasificador entrenado el 08/05/2020	75
17	Tuneado de hiperparámetros, combinaciones probadas.	76

Lista de algoritmos

1	Propagación hacia delante	13
2	Propagación hacia atrás	15
3	Entrenamiento	15
4	Un paso del entrenamiento por lotes de la GAN	31
5	Diagnóstico a partir de un electrocardiograma	38

Lista de figuras

1	Principales causas de muerte a nivel mundial según la OMS.	1
2	Muertes de Enfermedades cardiovasculares (ECV) por edad. Datos recogidos por el Instituto nacional de estadística (INE).	1
3	Evolución de la pirámide de población en España en los últimos 48 años. Datos recogidos por el INE.	2
4	Electrocardiograma real sacado del subconjunto de datos de entrenamiento. Ondas, intervalos y segmentos.	3
5	Filtrado realizado por el algoritmo Pan-Tompkins.	4
6	Representación de seis segundos de una onda sinusoidal normal extraída del conjunto de datos de entrenamiento. Derivación I	5
7	Representación de 6 segundos de una onda de un paciente con fibrilación auricular extraída de los datos de entrenamiento. Derivación I	6
8	Representación de 6 segundos de una onda de un paciente con bloqueo auriculoventricular de primer grado extraída de los datos de entrenamiento. Derivación I	6
9	Representación de 3 segundos de dos ondas de pacientes con bloqueo de rama izquierda extraída de los datos de entrenamiento. A la izquierda, la derivación aVL. A la derecha, la derivación V1	7
10	Representación de 6 segundos de un electrocardiograma de un paciente con complejo auricular prematuro extraída de los datos de entrenamiento. Derivación I	8
11	Representación de 6 segundos de la onda de un paciente con complejo ventricular prematuro extraída de los datos de entrenamiento. Derivación I	8
12	Representación de 3 segundos de la onda de un paciente con bloqueo de la rama derecha extraída de los datos de entrenamiento. Derivación V1	9
13	Representación de 1 segundo de las ondas de pacientes con depresión del segmento ST extraída de los datos de entrenamiento. Derivación V1	10
14	Representación de 6 segundos de la onda de un paciente con elevación del segmento ST extraída de los datos de entrenamiento. Derivación V2	10
15	Representación de una red neuronal clásica.	12
16	Funciones sigmoide y tangente hiperbólica	16
17	Funciones ReLu y Leaky ReLu	16
18	Representación de la operación de convolución.	19
19	Ejemplo de una arquitectura basada en redes neuronales convolucionales inspirada en la arquitectura VGG-16.	20
20	A la izquierda, la representación de una red neuronal clásica en formato de celdas. A la derecha, su equivalente convertido a una red neuronal recurrente.	21
21	Representación de una celda LSTM. Debajo cómo sería la conexión temporal.	22
22	Representación de una red generativa adversaria tradicional.	23
23	Distribución de los datos de entrenamiento por edad	26

24	A la izquierda, la respuesta en frecuencia del filtro. A la derecha, una derivación de un electrocardiograma de los datos de entrenamiento antes y después de filtrar	27
25	Proceso de separación y submuestreo de señales	28
26	Representación de una AC-GAN.	29
27	Representación del generador.	30
28	Representación del discriminador.	31
29	Muestras generadas normalizadas a trozos tras 200 épocas de entrenamiento por el modelo generativo.	32
30	Transformación de una señal de los datos de entrenamiento que había sido recortada anteriormente.	32
31	Bloque de entrada de datos cualitativos.	34
32	Bloque de entrada de electrocardiogramas.	35
33	Modelo completo.	36
34	Evolución de la precisión del modelo en cada época de entrenamiento. A la izquierda, con ampliación de datos con transformaciones. A la derecha, con redes generativas adversarias.	37
35	Matrices de confusión. A la izquierda el modelo entrenado con transformaciones utilizando el algoritmo de diagnóstico sin filtros. En el medio con filtros. A la derecha, el modelo entrenado con GANs	39
36	Matriz de confusión del modelo final	39
37	Diagrama de secuencia del sistema(SSD).	41
38	Diagrama de estados del servidor	42
39	Vista de la página de inicio de la aplicación web	43
40	Vista de la página de ayuda al diagnóstico de la aplicación web. Modo oscuro	44
41	Vista de la página de ayuda al diagnóstico de la aplicación web. Detalle en la visualización de las derivaciones.	45
42	Vista de la página de ayuda al diagnóstico de la aplicación web. Detalle en la visualización del diagnóstico.	45
43	Arquitectura de despliegue.	46
44	Gráficas de monitorización del modelo entrenado el 22/03/2020 .	61
45	Gráficas de monitorización del modelo entrenado el 23/03/2020 .	62
46	Gráficas de monitorización del modelo entrenado el 05/04/2020 .	68
47	Gráficas de monitorización del modelo entrenado el 10/04/2020 .	71
48	Gráficas de monitorización del modelo entrenado el 11/04/2020 .	71
49	Matriz de confusión del modelo clasificador del 08/05/2020 con el conjunto de datos ampliado con transformaciones y con filtro de sesgos.	76

Glosario

Alpine Linux Alpine Linux es una distribución Linux basada en musl y BusyBox, que tiene como objetivo ser ligera y segura por defecto sin dejar de ser útil para tareas de propósito general. Alpine Linux, desde la versión 3.8 deja de usar los parches PaX y grsecurity en el núcleo por defecto. Está diseñado principalmente para routers x86, cortafuegos, VPNs, VoIP y servidores. xiv, 46, 47

Cardiomiocitos Los cardiomiocitos son células del músculo cardíaco capaces de contraerse de forma espontánea e individual. Las ramificaciones características y las uniones estrechas entre estas células, conforman una sólida red de fibras miocárdicas, que determina la función de bomba cardíaca y el sistema celular eléctrico de conducción, que permite que esta bomba funcione. xiv, 7

D3.js D3.js (o simplemente D3 por las siglas de Data-Driven Documents) es una biblioteca (informática) de JavaScript para producir, a partir de datos, infogramas dinámicos e interactivos en navegadores web. Hace uso de tecnologías bien sustentadas como SVG, HTML5, y CSS. Esta biblioteca es sucesora de la biblioteca Protovis. En contraste con muchas otras bibliotecas, D3.js permite tener control completo sobre el resultado visual final. xiv, 44

Diástole Movimiento de relajación y expansión del corazón y las arterias que se produce cuando la sangre purificada entra en ellas. xiv, 4

Docker Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos . xiv, 46, 47

Flask Flask es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código. Está basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2 y tiene una licencia BSD. xiv, 42, 47, 49

Material-UI Framework popular para crear interfaces de usuario con React con componentes que siguen las directrices del lenguaje de diseño Material Design. xiv, 42

NGINX Nginx es un servidor web/proxy inverso ligero de alto rendimiento y un proxy para protocolos de correo electrónico (IMAP/POP3) . xiv, 46, 78

Node.js Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación JavaScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web. xiv, 47

Nodo auriculoventricular El nodo auriculoventricular (Nodo A-V), nodo atrioventricular, o nódulo de Aschoff-Tawara está formado por células cardíacas especializadas en la formación y la conducción de impulsos eléctricos cardíacos y se encuentra situado en la porción inferior del surco interauricular próximo al septo membranoso interventricular, en el vértice superior del triángulo de Koch (espacio entre el seno coronario, la válvula septal tricuspídea y el tendón de Todaro) . xiv, 7

Nodo sinoauricular El nodo sinoauricular (NSA o nodo SA) o nodo de Keith y Flack, nodo seno-auricular o nodo sinoatrial recibe el nombre común de marcapasos cardíaco, y es una de las estructuras del miocardio que compone el sistema de conducción eléctrica del corazón. En los cardiomiocitos de este nodo sinoauricular es donde se origina el impulso eléctrico que genera un latido cardíaco. xiv, 7, 8

Python Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma. xiv, 42, 47

React React (también llamada React.js o ReactJS) es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Es mantenido por Facebook y la comunidad de software libre. xiv, 42, 44, 47

Sístole Movimiento de contracción del corazón y de las arterias para empujar la sangre que contienen. xiv, 4

Tabique interventricular El tabique interventricular es un tabique, septo o septum membrano-muscular que, en condiciones normales, divide al corazón en dos cámaras independientes una de la otra: una cámara ventricular izquierda (o corazón izquierdo) y una cámara ventricular derecha (o corazón derecho). xiv, 7

Tensorflow TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. xiv, 42

Acrónimos

- AC-GAN** Auxiliary Classifier GAN. xiv, 29, 72
- AF** Fibrilación auricular. xiv, 5, 25
- API** Interfaz de programación de aplicaciones. xiv, 41, 42, 46, 49
- aVF** augmented vector foot. xiv, 3
- aVL** augmented vector left. xiv, 3, 7, 9
- aVR** augmented vector right. xiv, 2, 5
- BiLSTM** Bidirectional Long short-term memory. xiv, 35
- cGAN** Conditional Generative Adversarial Network. xiv, 29, 61
- CNN** redes neuronales convolucionales. xiv, 20, 22, 30, 34–36, 64
- DCGAN** Deep Convolutional Generative Adversarial Network. xiv, 30, 69
- ECG** Electrocardiograma. xiv, 2, 3, 6, 7, 11, 25–27, 29, 30, 34, 35, 37, 41, 55, 61, 70
- ECV** Enfermedades cardiovasculares. xii, xiv, 1–3, 11, 25, 54
- GAN** Red generativa adversaria. xiv, 22, 29, 31, 48, 70, 72, 73
- GANs** Redes generativas adversarias. xiv, 22, 24, 31, 37, 38, 48, 59, 77
- GBT** Grupo de Bioingeniería y Telemedicina. xiv, 11
- GCP** Google Cloud Platform. xiv, 58
- GDPR** Reglamento General de Protección de Datos. xiv, 55
- I-AVB** Bloqueo auriculoventricular de primer grado. xiv, 6, 25
- INE** Instituto nacional de estadística. xii, xiv, 1, 2, 50
- JSON** Javascript Object Notation. xiv, 42, 43
- LBBB** Bloqueo de rama izquierda. xiv, 7, 25
- LOPDGDD** Ley Orgánica de Protección de Datos Personales y garantía de los derechos digitales. xiv, 55
- LSTM** Long short-term memory. xiv, 21, 22, 35, 64
- NIH** National Institutes of Health. xiv, 25
- NLP** Procesamiento del lenguaje natural. xiv, 34

- NSVT** Taquicardia ventricular no sostenida. xiv, 9
- OMS** Organización mundial de la salud. xiv, 1, 50
- PAC** Complejo auricular prematuro. xiv, 7, 25
- PVC** Complejo ventricular prematuro. xiv, 8, 25, 26
- PWA** Aplicación web progresiva. xiv, 42
- RBBB** Bloqueo de rama derecha. xiv, 9, 25
- REST** Representational state transfer, en castellano transferencia del estado representacional. xiv, 41, 42, 46
- RNN** redes neuronales recurrentes. xiv, 21, 34
- STD** Depresión del segmento ST. xiv, 9, 25
- STE** Elevación del segmento ST. xiv, 10, 25, 26
- TFG** Trabajo de Fin de Grado. xiv, 11, 48
- WGAN** GAN de Wasserstein. xiv, 68
- WGAns** GANs de Wasserstein. xiv, 24
- WSGI** Web Server Gateway Interface. xiv, 42, 47

Capítulo I

Introducción

1 Contexto

Según la Organización mundial de la salud (OMS), se entiende por Enfermedad Cardiovascular (ECV) [1] a un grupo de desórdenes del corazón y de los vasos sanguíneos. Muchas de las ECV son enfermedades muy comunes. Hoy en día, constituyen la principal causa de defunción a nivel mundial. De los 2.668,475 millones de fallecimientos que tuvieron lugar en 2016, 413,153 millones fueron debido a ECV, lo que constituye un 15% de todas las muertes a nivel mundial [2].



Figura 1: Principales causas de muerte a nivel mundial según la OMS.

En España, el panorama es similar. Cada año mueren mas de 100.000 personas por ECV, siendo la principal causa de muerte. Los grupos de mayor riesgo son las personas de la tercera edad, que sufren la mayor parte de las muertes por ECV [3].

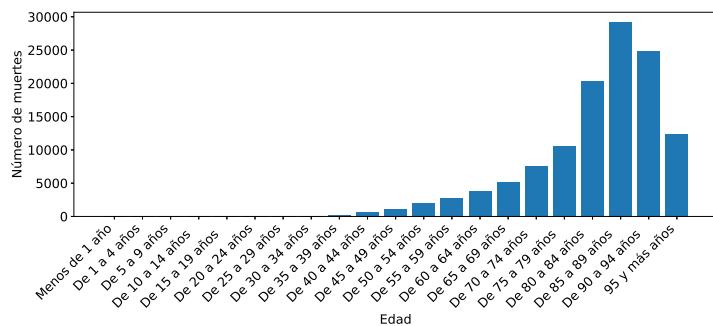


Figura 2: Muertes de ECV por edad. Datos recogidos por el INE.

Por otro lado, la pirámide de población se está invirtiendo [4]. Cada vez la población está mas envejecida debido a la incesante reducción de la natalidad [5]

y a la mayor esperanza de vida.

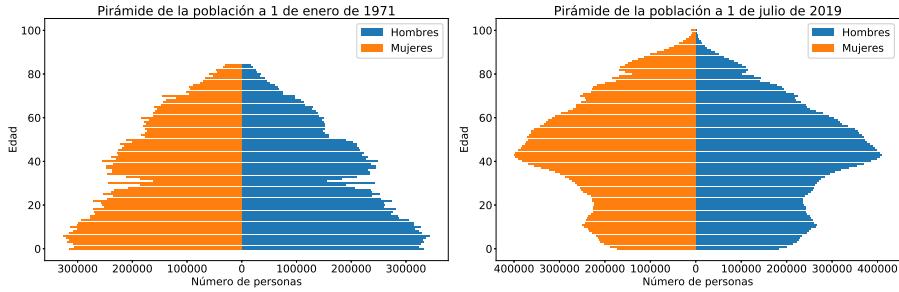


Figura 3: Evolución de la pirámide de población en España en los últimos 48 años. Datos recogidos por el INE.

Con todo esto, se presenta el reto de: por un lado, mejorar el diagnóstico para detectar a tiempo las ECV y así lograr una reducción en las defunciones por esta causa; y por el otro, agilizarlo para poder atender a la creciente demanda.

2 Diagnóstico. El Electrocardiograma

2.1 Definición

La prueba de diagnóstico de ECV más común es el Electrocardiograma (ECG). Un ECG, es una representación temporal de la actividad eléctrica del corazón mediante el uso de un conjunto de electrodos que se sitúan sobre la superficie corporal [6]. Estos electrodos detectan los pequeños cambios eléctricos producto de la depolarización y repolarización del músculo cardíaco durante cada ciclo o latido cardíaco.

A cada una de las medidas de voltaje, entendido como diferencia de potencial entre dos electrodos, se le llama derivación. Estas derivaciones permiten distintas representaciones de la actividad cardíaca, pues son tomadas entre distintos puntos del cuerpo humano. En la actualidad, está extendido el uso de 12 derivaciones, con 10 electrodos simultáneos:

- Derivación I: del brazo derecho y el izquierdo.
- Derivación II: del brazo derecho y la pierna izquierda.
- Derivación III: del brazo izquierdo y pierna izquierda.

En las siguientes tres derivaciones, se utiliza la combinación de señales en el electrodo negativo para potenciar el electrodo positivo:

- La derivación *augmented vector right (aVR)* tiene el electrodo positivo en el brazo derecho. El electrodo negativo es una combinación del electrodo del brazo izquierdo y el electrodo de la pierna izquierda.

- La derivación *augmented vector left (aVL)* tiene el electrodo positivo en el brazo izquierdo. El electrodo negativo es una combinación del electrodo del brazo derecho y la pierna izquierda.
- La derivación *augmented vector foot (aVF)* tiene el electrodo positivo en la pierna izquierda. El electrodo negativo es una combinación del electrodo del brazo derecho y el brazo izquierdo.

Por último, tenemos las 6 derivaciones precordiales (V1, V2, V3, V4, V5, y V6). Estas derivaciones, están colocadas en distintos puntos del pecho y se consideran unipolares. Esto es porque su potencial se mide respecto al electrocardiógrafo.

Los ECG son de uso muy extendido y común debido a que se requieren pocos medios para poder hacerlos. Es por ello que es la prueba principal de diagnóstico para ECV. Una vez enfermería ha realizado la prueba, un cardiólogo experimentado analiza, generalmente sobre el papel, las características de la señal y diagnostica al paciente.

El tiempo dedicado a la interpretación del ECG podría ser mejor aprovechado por el cardiólogo. Si se contara con una herramienta de detección automática, se aligeraría la carga de tiempo dedicado al diagnóstico, pudiendo reutilizar ese tiempo en otras actividades. Además, podría ser una herramienta muy útil en medicina general, permitiendo detectar enfermedades de difícil diagnóstico. Esto, aligeraría aún mas la carga, pues evitaría falsos negativos que llevan a complicaciones y más visitas a los centros médicos.

2.2 La señal

Un latido se compone fundamentalmente por: onda P, puntos Q, R y S y onda T.

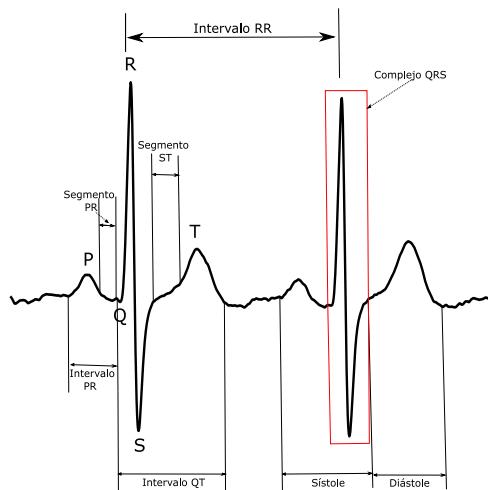


Figura 4: Electrocardiograma real sacado del subconjunto de datos de entrenamiento. Ondas, intervalos y segmentos.

Los análisis que el cardiólogo realiza sobre la señal se centran en dos características:

- La periodicidad de la señal. Las irregularidades son signo de algunas enfermedades, como las arritmias.
- La forma de señal. Las desviaciones en lo que se considera un pulso estándar pueden ser síntoma de algunas enfermedades.

Una vez se tienen estos puntos identificados, se pueden medir intervalos de señal, cuyas irregularidades también aportan información. El intervalo PR, el segmento PR, el complejo QRS, el intervalo QT y el segmento ST aportan información sobre la forma de la señal. El intervalo RR, que es la distancia entre picos de pulsos consecutivos, y nos da información sobre su periodicidad [7].

La onda P es consecuencia de la despolarización auricular (Sístole). A continuación viene el complejo QRS, consecuencia de la despolarización ventricular. Por último viene la onda T, resultado de la repolarización ventricular (Diástole).

2.3 Procesado

Los modelos tradicionales de detección automática de enfermedades se basan, en su gran mayoría, en la detección de irregularidades en la periodicidad de la señal. Para ello, se procesa la señal eliminando ruido y potenciando los picos R. Así, luego se detectan los picos y se miden los latidos por minuto. El algoritmo más extendido es el desarrollado por Jiapu Pan y Willis J. Tompkins [8].



Figura 5: Filtrado realizado por el algoritmo Pan-Tompkins.

1. Se eliminan los ruidos procedentes del cuerpo humano y otras fuentes externas con un filtro paso banda entre los 5 y los 15 Hz.
2. Se deriva la señal para obtener información sobre las pendientes.
3. Se eleva e integra la señal para potenciar los picos dominantes (los del complejo QRS) y así reducir la posibilidad de confundir las ondas T como ondas R.
4. Tras este procesado, un algoritmo basado en límites inferiores, detecta los picos R.

Este algoritmo sirve de precedente y ayuda a tomar algunas decisiones a la hora de elaborar el preprocesado de este trabajo. Sin embargo, es un algoritmo centrado en detectar complejos QRS. Así, elimina partes de la señal que pueden ser útiles a la hora de detectar ciertas enfermedades, como la onda T.

3 Las enfermedades cardiovasculares

Como ya se ha expuesto en el apartado anterior, las enfermedades cardiovasculares se pueden diagnosticar atendiendo a anomalías en las diferentes partes de la onda: anomalías de duración, forma o polarización. A continuación, se profundiza un poco más en las anomalías a las que atienden los cardiólogos a la hora de detectar las enfermedades que se pretenden diagnosticar con el modelo que se va a diseñar y desarrollar [9].

3.1 Ritmo sinusal normal

Antes de explicar por qué se caracteriza la onda de cada patología, hay que definir cuál es el estándar con el que se compara. El estándar es el paciente sin patologías cardíacas, cuya onda se denomina *ritmo sinusal normal*.

En este caso, la frecuencia cardíaca es de 60 a 100 pulsaciones por minuto. El complejo QRS viene precedido por una onda P, que es ascendente en las derivaciones I y II e invertida en la derivación aVR. El intervalo PR se mantiene constante y los complejos QRS duran menos de 100 milisegundos.

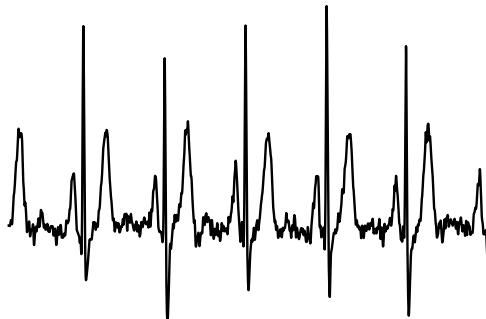


Figura 6: Representación de seis segundos de una onda sinusoidal normal extraída del conjunto de datos de entrenamiento. Derivación I

3.2 Fibrilación auricular

La Fibrilación auricular (AF) se caracteriza por un ritmo irregularmente irregular. Esto es que, dentro de que un ritmo sea irregular, esta irregularidad no sigue un patrón. Además una onda cardíaca de un paciente con AF no tiene ondas P, y no tiene línea base isoeléctrica, es decir, no tiene una base recta sobre la que surgen las ondas T o los complejos QRS.

Además, las ondas de AF tienen un ritmo ventricular variable, su complejo QRS dura menos de 120 milisegundos. Por último, pueden presentar ondas de

fibrilación, las cuales se pueden confundir con ondas P y llevar a diagnósticos erróneos.

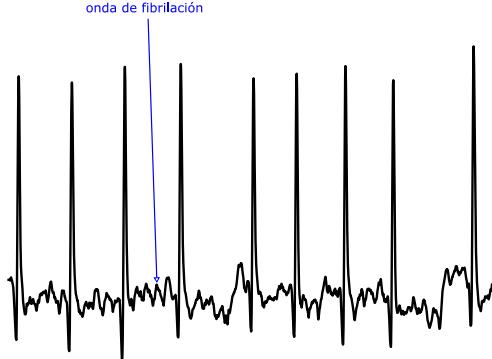


Figura 7: Representación de 6 segundos de una onda de un paciente con fibrilación auricular extraída de los datos de entrenamiento. Derivación I

3.3 Bloqueo auriculoventricular de primer grado

El ECG del Bloqueo auriculoventricular de primer grado (I-AVB) se caracteriza por un intervalo PR mayor a 200 milisegundos. Esto ocasiona que la onda P pueda superponerse con la onda T del ciclo anterior, provocando que ambas sean irreconocibles.

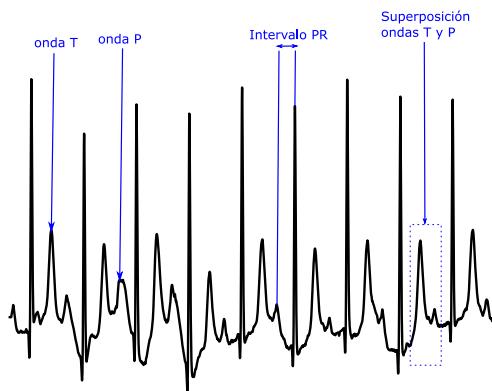


Figura 8: Representación de 6 segundos de una onda de un paciente con bloqueo auriculoventricular de primer grado extraída de los datos de entrenamiento. Derivación I

Este cambio de la onda es consecuencia de un retraso en la conducción

del impulso eléctrico por un bloqueo entre el Nodo sinoauricular y el Nodo auriculoventricular [10] del corazón. Por sí solo es asintomático, pero combinado con otras anomalías en el ECG puede llegar a suponer problemas de salud para el paciente.

3.4 Bloqueo de rama izquierda

En el ECG del Bloqueo de rama izquierda (LBBB), los segmentos ST y las ondas T muestran una "*discordancia apropiada*", es decir, tienen una dirección opuesta al vector principal del complejo QRS. Esto produce elevación del segmento ST y onda T positiva en ondas con complejo QRS negativo y depresión del segmento ST e inversión de la onda T en ondas con complejo QRS positivo.

Normalmente, el Tabique interventricular se depolariza de izquierda a derecha. Sin embargo, en pacientes con LBBB se invierte, extendiendo la duración del complejo QRS por encima de los 120 milisegundos. Por último, este cambio en la dirección de depolarización produce cambios en las derivaciones:

- Altos picos R en derivaciones I, V5 y V6.
- Profundas ondas S en derivaciones V1, V2 y V3.
- Complejos QRS anchos con forma de M/W en las derivaciones laterales (I, aVL, V5, V6).

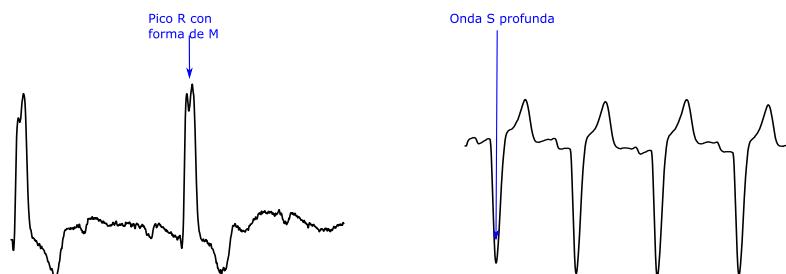


Figura 9: Representación de 3 segundos de dos ondas de pacientes con bloqueo de rama izquierda extraída de los datos de entrenamiento. A la izquierda, la derivación aVL. A la derecha, la derivación V1

3.5 Complejo auricular prematuro

El Complejo auricular prematuro (PAC) consiste en pulsos fuera de lugar que surgen en los tejidos de Cardiomiocitos [11] que están dentro de la aurícula. Su ECG se caracteriza por una onda P anormal, que tiene una morfología distinta a una de ritmo sinusal normal.

Esta onda P distinta, se puede manifestar de formas distintas dependiendo de la localización en la aurícula de esos Cardiomiocitos que se comportan de forma errónea.

- La onda P podría quedar sumada a la onda T, siendo indistinguibles.
- La onda P podría invertirse y quedar un intervalo PR relativamente corto (menor a 120 milisegundos).
- Si se alcanza el Nodo sinoauricular [12], podría ocasionar un intervalo mayor hasta que el próximo pulso cardíaco ocurra. A esto se le llama "*reseteo*" del Nodo sinoauricular.

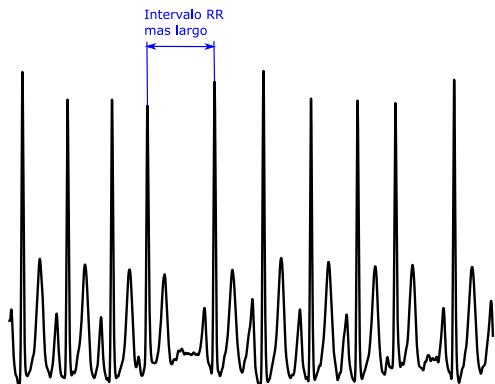


Figura 10: Representación de 6 segundos de un electrocardiograma de un paciente con complejo auricular prematuro extraída de los datos de entrenamiento. Derivación I

3.6 Complejo ventricular prematuro

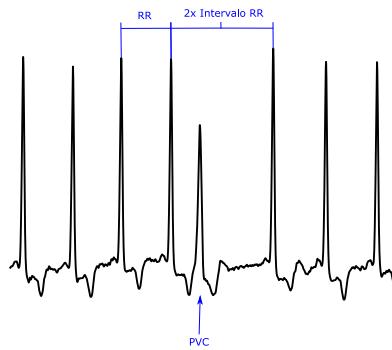


Figura 11: Representación de 6 segundos de la onda de un paciente con complejo ventricular prematuro extraída de los datos de entrenamiento. Derivación I

El Complejo ventricular prematuro (PVC) consiste en pulsos fuera de lugar localizados en los ventrículos. Una de sus características es la presencia de complejos QRS anchos (mayores de 120 milisegundos), con una morfología anormal

y que ocurren antes de lo que se esperaría en un ritmo sinusal normal.

Se manifiesta por intervalos ST discordantes y cambios en la onda T. Normalmente, les sigue una pausa compensatoria. Suelen ocurrir en patrones: uno de cada dos pulsos (bigeminismo), cada 3 pulsos (trigeminismo), uno de cada cuatro (cuadrigeminismo), en pares o 3 de cada 30, conocido como Taquicardia ventricular no sostenida (NSVT).

3.7 Bloqueo de la rama derecha

El patrón del electrocardiograma para pacientes con Bloqueo de rama derecha (RBBB) se caracteriza por una depresión del intervalo ST y la inversión de la onda T en las derivaciones V1, V2 y V3.

Se debe al retraso en la activación del ventrículo derecho. Esto produce un pico R secundario (llamado R') en las derivaciones V1, V2 y V3 y una onda S ancha en las derivaciones laterales. De esta forma, puede diagnosticarse por:

- Un complejo QRS ancho (mayor de 120 milisegundos).
- Un pico R secundario en forma de M en las derivaciones V1, V2 y V3.
- Una onda S ancha en las derivaciones I, aVL, V5 y V6.

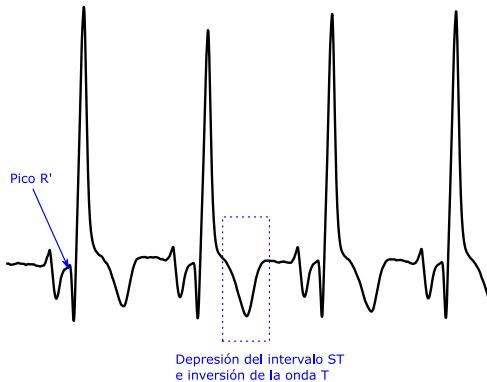


Figura 12: Representación de 3 segundos de la onda de un paciente con bloqueo de la rama derecha extraída de los datos de entrenamiento. Derivación V1

3.8 Depresión del segmento ST

La característica del electrocardiograma en pacientes con Depresión del segmento ST (STD) se indica en el propio nombre. La pendiente de esta depresión puede ser: descendente, ascendente y horizontal.

Una depresión descendente u horizontal indica isquemia miocárdica. Una depresión ascendente puede indicar la oclusión de la arteria descendente anterior izquierda.

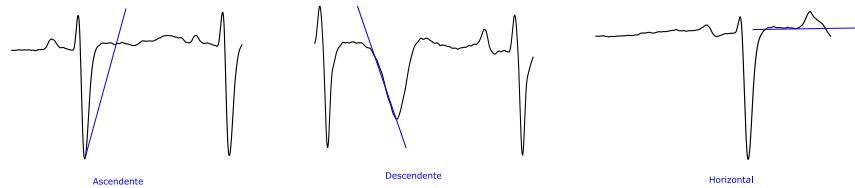


Figura 13: Representación de 1 segundo de las ondas de pacientes con depresión del segmento ST extraída de los datos de entrenamiento. Derivación V1

3.9 Elevación del segmento ST

La característica de la onda en pacientes con Elevación del segmento ST (STE) se describe en el propio nombre. La forma de esta elevación puede ser cóncava, convexa u oblicua. La STE indica infarto agudo de miocardio.

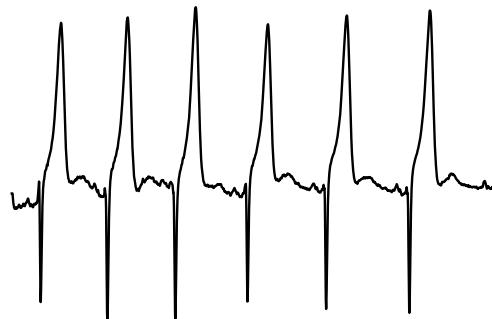


Figura 14: Representación de 6 segundos de la onda de un paciente con elevación del segmento ST extraída de los datos de entrenamiento. Derivación V2

Capítulo II

Objetivos

El objetivo de este Trabajo Fin de Grado es diseñar y desarrollar un sistema de ayuda al diagnóstico de enfermedades cardiovasculares a partir de registros de ECG mediante aprendizaje profundo (también conocido como Deep Learning). Dentro de este desarrollo, se observan tres retos.

En primer lugar se deberá preprocesar los datos. Esto se traduce en que la señal de entrada deberá ser tratada para eliminar todo aquello que no aporta información al modelo. Además, habrá que conseguir que la distribución del conjunto de datos sea uniforme. Para ello, se tratará de generar datos nuevos que permitan entrenar el modelo con una colección de datos con clases balanceadas. Se propone utilizar dos métodos diferentes para aumentar el conjunto de datos disponible: el uso de Redes Generativas Adversarias y de transformaciones.

A continuación, se presenta el problema de la identificación automática de enfermedades a partir de los datos. Este será el problema más complejo a tratar, pues se debe diseñar y desarrollar un sistema capaz de predecir la clase de diagnóstico a partir de las señales de ECG. Se explorarán y validarán diferentes arquitecturas basadas en la combinación de redes neuronales convolucionales, redes neuronales recurrentes y redes neuronales clásicas. El objetivo es obtener un modelo de gran precisión en el diagnóstico.

Por último, se pretende desarrollar una aplicación que permita al personal sanitario enviar los datos correspondientes al registro de un electrocardiograma. El resultado de esta petición será la visualización de la información contenida en el registro y una propuesta de diagnóstico, que incluirá las probabilidad estimada de que se padezca una de las enfermedades detectadas de forma automática por el modelo basado en aprendizaje profundo.

Como se puede observar, se presenta un problema muy completo, que servirá para aplicar todos los conocimientos aprendidos durante el grado.

4 Potencial del trabajo

El Grupo de Bioingeniería y Telemedicina (GBT) está en continua colaboración con el Servicio de Cardiología del Hospital 12 de Octubre de Madrid. Es de gran interés validar el modelo y la herramienta de diagnóstico que resulten de este Trabajo de Fin de Grado (TFG) con registros ECG obtenidos en este hospital.

Esto podría potencialmente resultar, a partir de este trabajo o de futuras investigaciones a raíz del mismo, en una herramienta de gran valor para el diagnóstico de pacientes con ECV y, por lo tanto, en una aportación útil a nuestra comunidad. Por ello, está entre los objetivos alcanzar un sistema extrapolable a otros escenarios mas allá del conjunto de datos de entrenamiento.

Capítulo III

Aprendizaje profundo

5 Definición

El aprendizaje profundo [13], conocido como Deep Learning, es un subconjunto de métodos del aprendizaje automático basado en redes neuronales artificiales. Puede ser supervisado, semi supervisado o no supervisado.

Una red neuronal artificial es una colección de neuronas artificiales conectadas, inspirada en las redes neuronales biológicas. Una neurona artificial, representa una función: tiene una puerta de entrada y una puerta de salida. Esta normalmente pasa a continuación por una función de activación. En secciones posteriores se mostrarán algunas. La conexión de estas neuronas, formando redes, permite la representación abstracta de funciones no lineales complejas.

Sea $x \in \Re^{n_x}$ el vector de entrada a una neurona, $w \in \Re^{n_x}$ el vector de pendientes de la neurona, $b \in \Re$ la ordenada en el origen, g la función de activación e \hat{y} la salida [14]:

$$\hat{y} = g(w^T x + b) \quad (1)$$

Generalizando, una capa intermedia, llamada capa oculta, de una red neuronal se representa por el vector $a^{[l]} = g(W^{[l]}a^{[l-1]} + b^{[l]})$ y una neurona se representa por $a_i^{[l]}$. Aquí, l se referirá a la capa oculta correspondiente e i a la neurona dentro de esa capa. $W \in \Re^{j,k}$ y $b \in \Re^j$, donde k es el número de entradas (o el número de salidas de la capa anterior) y j es el número de neuronas de esta capa. También podemos establecer entonces que $a \in \Re^j$.

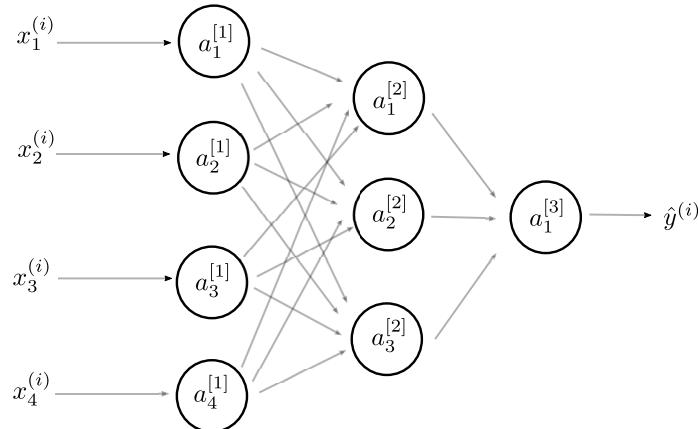


Figura 15: Representación de una red neuronal clásica.

La vectorización del problema es interesante, pues ahorra carga computacional al ser posibles las operaciones vectoriales con computación paralela. Esto

ha llevado a realizar todas las operaciones en formato matricial.

De esta forma, una red neuronal se puede representar en formato matricial [15]. Sea una red neuronal de L capas, $X \in \Re^{n_x, m}$ una entrada de n_x valores por ejemplo y m ejemplos, $A \in \Re^{j, m}$ e $Y \in \Re^{n_y, m}$ una salida real de n_y valores de salida y m ejemplos: A

$$A^{[l]} = g^{[l]}(W^{[l]}A^{[l-1]} + b^{[l]}), \quad A^{[0]} = X, \quad \hat{Y} = A^{[L]} \quad (2)$$

6 Entrenamiento

6.1 Propagación hacia delante

Definidas las redes neuronales, falta por definir su proceso de aprendizaje. En primer lugar, se inicializan los pesos de las matrices $W^{[l]}$ y $b^{[l]}$. Esta inicialización suele ser aleatoria y acotada en valores pequeños, de esta forma se asegura la actualización de la red que se explicará en los próximos pasos.

A continuación, se sigue un proceso iterativo que comienza con la propagación hacia delante [16], conocida comúnmente por su nombre en inglés: *forward propagation*. Este paso consiste en el cálculo de las salidas capa por capa a partir de la entrada para los datos de entrenamiento.

Algoritmo 1: Propagación hacia delante

```

Result:  $\hat{Y}$ 
 $A^{[0]} \leftarrow X;$ 
for  $l \leftarrow 1$  to  $L$  do
     $A^{[l]} \leftarrow g^{[l]}(W^{[l]}A^{[l-1]} + b^{[l]});$ 
 $\hat{Y} \leftarrow A^{[L]};$ 
```

Con esto, se ha calculado una iteración de la propagación hacia delante de todas las muestras de entrenamiento. Esto calcula una predicción \hat{Y} , que en aprendizaje supervisado queremos que se parezca lo máximo posible a la salida real esperada Y .

6.2 Descenso del gradiente

Con el objetivo de minimizar la diferencia entre Y e \hat{Y} se calcula la función de pérdidas $\mathcal{L}(\hat{y}, y)$, que mide la distancia entre la salida real y la salida calculada. A continuación podremos calcular a partir de cada función de pérdidas, la función de coste de la iteración:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \quad (3)$$

Donde w y b son el conjunto de parámetros de la red neuronal. La función de coste nos permite saber cómo de bien se está comportando nuestro modelo con los datos de entrenamiento. El objetivo será minimizar esta función de coste para que nuestro modelo sea capaz de predecir una salida en función de su entrada con la mayor exactitud posible, es decir, que la función de coste llegue a

un punto cercano al mínimo global.

Para ello, utilizamos el algoritmo del descenso de gradiente [17], conocido comúnmente por su nombre en inglés: *gradient descend*. Como su propio nombre indica, este calculará:

$$\nabla J(w, b) = \left(\frac{\partial J}{\partial w^{[1]}}, \dots, \frac{\partial J}{\partial w^{[L]}}, \frac{\partial J}{\partial b^{[1]}}, \dots, \frac{\partial J}{\partial b^{[L]}} \right) \quad (4)$$

Donde $\frac{\partial J}{\partial x} = \frac{1}{m} \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial x_i}$. A continuación, se modifican los pesos de las neuronas a partir de las derivadas parciales de la función de coste respecto a $J(w, b)$. Sea α el ratio de aprendizaje, es decir, cómo de pronunciada queremos que sea la actualización de los pesos, la actualización de una capa l de una red neuronal sería:

$$w^{[l]} = w^{[l]} - \alpha \frac{\partial J}{\partial w^{[l]}} \quad (5)$$

$$b^{[l]} = b^{[l]} - \alpha \frac{\partial J}{\partial b^{[l]}} \quad (6)$$

Sin embargo, $w^{[l]}$ no es una variable independiente sobre la que se calcula $J(w, b)$. Aplicando la regla de la cadena, tenemos que $\frac{\partial J}{\partial w^{[l]}} = \frac{\partial J}{\partial z^{[l]}} \frac{\partial z^{[l]}}{\partial w^{[l]}}$. Así, de forma iterativa:

$$\frac{\partial J}{\partial w^{[l]}} = \frac{\partial J}{\partial z^{[l]}} a^{[l-1]} = \frac{\partial J}{\partial a^{[l]}} g'^{[l]}(z^{[l]}) a^{[l-1]} = w^{[l+1]T} \frac{\partial J}{\partial z^{[l+1]}} g'^{[l]}(z^{[l]}) a^{[l-1]} \quad (7)$$

6.3 Propagación hacia atrás

Como se ha podido observar, la derivada de $J(w, b)$ respecto de los parámetros de una capa depende de la capa siguiente. Por tanto, se debe comenzar desde la última capa y calcular las derivadas hacia atrás. Esto da nombre al algoritmo usado en el siguiente paso, la propagación hacia atrás [18], conocida comúnmente por su nombre en inglés: *backward propagation*.

La derivada de la función de pérdidas depende de qué función de pérdidas se utilice. Para mostrar este algoritmo, supóngase que la derivada respecto a la última capa para una muestra i es:

$$\frac{\partial \mathcal{L}^{(i)}}{\partial a^{[L](i)}} = -\frac{y^{(i)}}{a^{(i)}} + \frac{1 - y^{(i)}}{1 - a^{(i)}} \quad (8)$$

Supóngase además que se realiza esta operación para todos las muestras del conjunto de datos de entrenamiento y que se agrupan en formato matricial $\frac{\partial J}{\partial A^{[L]}}$. Para facilitar la lectura, $\frac{\partial J}{\partial V^{[l]}}$ se representa por $dV^{[l]}$ siendo $V^{[l]}$ una matriz cualquiera de la capa l . Por último supóngase que dW y db son el conjunto formado por las matrices $dW^{[l]}$ y $db^{[l]}$. Con todo esto, se procede de la siguiente manera:

Algoritmo 2: Propagación hacia atrás

Data: $dA^{[L]}$
Result: dW, db
for $l \leftarrow L$ **to** 1 **do**
 $dZ^{[l]} \leftarrow dA^{[l]} * g'^{[l]}(Z^{[l]})$ // producto elemento a elemento
 $dW^{[l]} \leftarrow \frac{1}{m} dZ^{[l]} A^{[l-1]T}$
 /* media de las filas de $dZ^{[l]}$ */
 $db^{[l]} \leftarrow \frac{1}{m} \sum_{i=1}^m dZ^{[l]}[i]$
 $dA^{[l-1]} \leftarrow W^{[l]T} dZ^{[l]}$

6.4 Algoritmo de aprendizaje

Por último, se actualizan los parámetros de las neuronas artificiales tal y como se ha comentado anteriormente. Este proceso se repite de manera iterativa, hasta que se considera que la red neuronal ha *aprendido* a realizar la tarea para la que se le ha entrenado.

Hay diferentes formas de realizar este proceso, la mas común es iterar en subconjuntos aleatorios pequeños de tu conjunto de datos de entrenamiento, actualizando la red neuronal. Además, se realiza este proceso N veces. A cada subconjunto se le denomina comúnmente *mini-batch* (mini lote) y a cada iteración i de las N iteraciones totales *epoch* (época). A este algoritmo de entrenamiento [19] se le denomina como *mini-batch training*.

Algoritmo 3: Entrenamiento

Data: X, Y , batch_size, neural_network, α
for $epoch \leftarrow 1$ **to** N **do**
 for $b \leftarrow 0$ **to** $length(X)/batch_size$ **do**
 $batch_x, batch_y \leftarrow get_mini_batch(X, Y)$
 $\hat{Y} \leftarrow forward_prop(neural_network, batch_x)$
 $J \leftarrow calcular_coste(\hat{Y}, batch_y)$
 $dW, db \leftarrow backward_prop(neural_network, J, \hat{Y})$
 actualizar_pesos(neural_network, dW, db, α)

7 Funciones, optimización y regularización

Para desarrollar un buen modelo, se deben de tener en cuenta multitud de variables. Los resultados pueden variar considerablemente cambiando en la arquitectura la función de activación, la función de pérdidas o añadiendo capas de regularización que mejoran la optimización o evitan el sobre-ajuste (conocido por el término en inglés *overfitting*)

7.1 Funciones de activación

Como se comentó en la sección 5, las funciones de activación aumentan la no linealidad de las redes neuronales. Resultan un elemento esencial en todas las capas de redes neuronales clásicas, ya que además permiten acotar la salida,

evitando una explosión de gradiente. Por otro lado, se cuentan con ciertas intuiciones y reglas obtenidas empíricamente para elegir qué función resultará mejor para cada parte de la arquitectura.

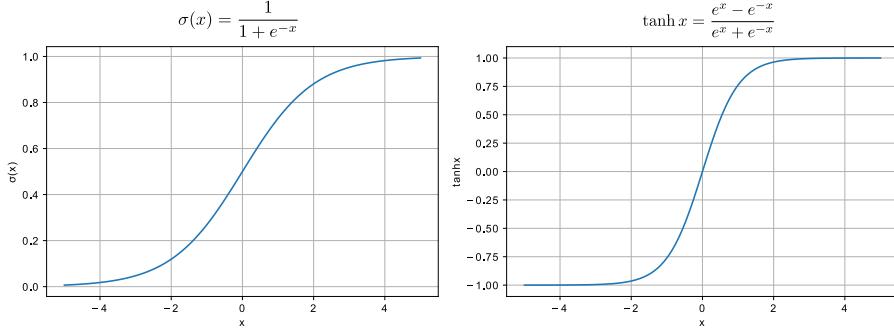


Figura 16: Funciones sigmoide y tangente hiperbólica

Entre las mas utilizadas, se encuentra la función sigmoide. Se caracteriza por su no linealidad y por permitir acotar salidas entre 0 y 1. Esto la hace la función estándar para salidas en modelos de clasificación binaria. Parecida a esta es la tangente hiperbólica, acotada entre -1 y 1. Precisamente su no linealidad hace que su uso esté muy extendido en las puertas de las células de los modelos secuenciales (modelos dependientes del tiempo).

Otras muy relevantes son los rectificadores. Estas funciones son lineales y a trozos. Son muy usadas en arquitecturas de redes neuronales profundas y en redes convolucionales. La mas extendida es la función ReLu (unidad lineal rectificada), conocida en el campo del procesamiento de señal como función rampa. En los últimos años, está proliferando el uso de una modificación de esta que permite un pequeño gradiente positivo para valores negativos, conocida como *Leaky ReLu*

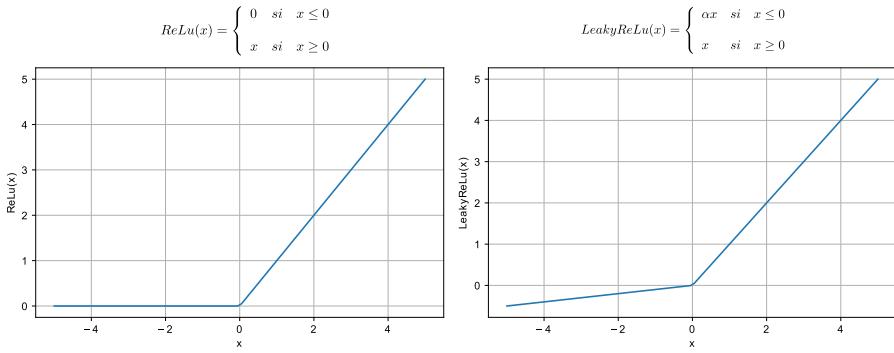


Figura 17: Funciones ReLu y Leaky ReLu

Por último, cabe mencionar la función softmax [20]. Esta es muy utilizada

como función de salida de aquellas redes neuronales que deben clasificar mas de dos clases, es decir, clasificación no binaria. Para ello, convierte valores en probabilidades de actuación. Así, sean $K \in \mathbb{N}$ clases a predecir, la última capa de la red neuronal de tamaño K $z^{[L]} = w^{[L]}a^{[L-1]} + b^{[L]}$ y $z_i^{[L]}$ la salida de una de las K neuronas, la salida de la función softmax sería:

$$a^{[L]} = \frac{e^{z^{[L]}}}{\sum_{i=1}^K e^{z_i^{[L]}}} \quad (9)$$

7.2 Normalización y regularización

Existe un conjunto de buenas prácticas que permiten conseguir modelos mas robustos, evitar el *overfitting* y agilizar el entrenamiento. Una de ellas, es la normalización [21]. La normalización a la entrada permite que la escala de los datos sea uniforme en todas las direcciones, logrando una J mas simétrica en todas las dimensiones (w, b). Esto facilita el trabajo al algoritmo del descenso del gradiente y evita problemas de explosión/desvanecimiento del mismo.

Se puede realizar una operación similar en las capas de una red neuronal, lo que se conoce comúnmente como *batch normalization* [22] (normalización de lotes). El objetivo es el mismo que se persigue con la normalización de las entradas. Dados unos valores intermedios de una capa de la red neuronal $z^{[l](i)}$, se normalizan con la media y la desviación estándar:

$$\mu = \frac{1}{m} \sum_{i=0}^m z^{[l](i)} \longrightarrow \sigma^2 = \frac{1}{m} \sum_{i=0}^m (z^{[l](i)} - \mu)^2 \longrightarrow z_{norm}^{[l](i)} = \frac{z^{[l](i)} - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad (10)$$

Nótese el término ϵ , que evita la división por cero en el caso de que $\sigma^2 = 0$. Con esto, se calcula el conjunto de valores que se pasarán a la función de activación $\hat{z}^{[l](i)} = \gamma z_{norm}^{[l](i)} + \beta$, donde γ y β son parámetros entrenables como lo son w o b .

Además, también se puede tunear la importancia que se le da a esta variación de la media y la varianza a lo largo de las distintas iteraciones. Para ello, se define un parámetro denominado *momentum*, y se pondera la importancia del momento actual con respecto a todos los anteriores. Así, μ y σ^2 quedarían:

$$\begin{aligned} \mu &= \alpha * \mu_{anterior} + (1 - \alpha) * \mu_{actual} \\ \sigma^2 &= \alpha * \sigma_{anterior}^2 + (1 - \alpha) * \sigma_{actual}^2 \end{aligned} \quad (11)$$

Todo esto agiliza y facilita el entrenamiento. Sin embargo, hay otros problemas que se deben tratar, como el caso de sobre-entrenar un modelo, lo que le haría sobre-especializarse en el conjunto de datos de entrenamiento y no generalizar bien el problema que se desea resolver (*overfitting*). Para estos problemas se utiliza la regularización. Hay muchas maneras de regularizar, entre ellas destacan:

- Anular en cada iteración algunas neuronas de forma aleatoria durante el entrenamiento. Esto evita la dependencia de unas sobre las otras, lo que contribuye a que el modelo no se sobre-especialice. Es lo que se conoce comúnmente como *dropout* [23].

- Aumentar el conjunto de datos de entrenamiento generando nuevos o realizando transformaciones a los existentes. Así, el modelo se entrena con más datos y generaliza mejor.
- En clasificación binaria, entrenar con valores de salida próximos a 1 y 0 pero no exactamente 1 y 0. Es lo que se conoce como *label smoothing* (suavizado de etiquetas) [24].

7.3 Optimización

Con el objetivo de agilizar el algoritmo del descenso del gradiente surgen los algoritmos optimizadores. Estos se basan en guardar información del resultado de los gradientes en iteraciones anteriores y usar esta información para mejorar la actualización de los valores en la iteración actual.

De esta manera, se potencia el descenso del gradiente en aquellas direcciones que permiten obtener una menor función de coste y se atenúa en aquellas que no. Entre estos algoritmos destacan *RMSprop* y *Adam*. El primero, actualiza los pesos como sigue:

$$\begin{aligned} s_{dw} &= \beta s_{dw} + (1 - \beta)(\frac{\partial J}{\partial w})^2 \longrightarrow w = w - \alpha \frac{\partial J}{\partial w} \frac{1}{\sqrt{s_{dw} + \epsilon}} \\ s_{db} &= \beta s_{db} + (1 - \beta)(\frac{\partial J}{\partial b})^2 \longrightarrow b = b - \alpha \frac{\partial J}{\partial b} \frac{1}{\sqrt{s_{db} + \epsilon}} \end{aligned} \quad (12)$$

Por otro lado, *Adam* [25] usa las actualizaciones de *RMSprop* y les añade momento. Así, el β anterior pasa a ser β_2 y se introducen los siguientes términos:

$$\begin{aligned} v_{dw} &= \beta_1 v_{dw} + (1 - \beta_1) \frac{\partial J}{\partial w} \longrightarrow v_{dw}^{correlada} = v_{dw} / (1 - \beta_1^T) \\ v_{db} &= \beta_1 v_{db} + (1 - \beta_1) \frac{\partial J}{\partial b} \longrightarrow v_{db}^{correlada} = v_{db} / (1 - \beta_1^T) \end{aligned} \quad (13)$$

Así, la actualización de los pesos de las neuronas queda:

$$w = w - \alpha \frac{v_{dw}^{correlada}}{\sqrt{s_{dw} + \epsilon}}, b = b - \alpha \frac{v_{db}^{correlada}}{\sqrt{s_{db} + \epsilon}} \quad (14)$$

8 Arquitecturas

Hasta ahora se han planteado algunas de las operaciones a realizar en redes neuronales clásicas. Sin embargo, a partir de estas se han ido construyendo bloques más complejos que han resultado en un avance vertiginoso de los modelos de aprendizaje profundo en campos como el procesamiento del lenguaje natural y la visión por ordenador.

8.1 Redes neuronales convolucionales

En el campo de la visión por ordenador, el uso de una arquitectura de redes neuronales clásicas sería intratable. Una imagen de baja resolución, como por

ejemplo 100 píxeles de alto, 100 píxeles de ancho y 3 canales RGB supondrían un vector de entrada de 30000 parámetros. Si escalamos esto a imágenes de resolución normal, que suele ser mucho mas alta, nos encontramos con una cantidad de parámetros a entrenar computacionalmente demasiado costoso [26].

En este contexto se introduce la operación convolución a las redes neuronales. Esta consiste en operar una entrada con un filtro (*conocido como kernel*) para extraer características de la misma. Así, en vez de operar valores uno a uno con sus parámetros entrenables en cada neurona, se opera sobre conjuntos de valores.

Para mostrar la operación, imagínese una imagen, o cualquier otra entrada, de 4 de ancho por 4 de alto y un solo canal y un filtro de 3 de ancho y 3 de alto. Supóngase además un salto entre operaciones de 1 y que no se le añade relleno. La operación se podría visualizar como:

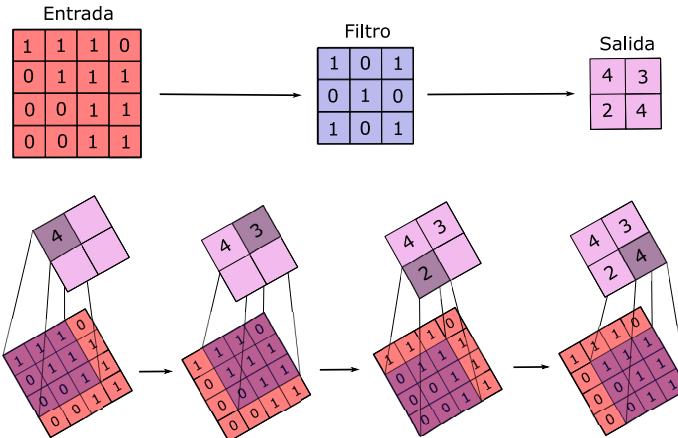


Figura 18: Representación de la operación de convolución.

Nótese que los valores tanto de la entrada como del filtro pertenecerán a \mathbb{R} . Los valores del filtro serán los parámetros entrenables, a los que luego se les añadirá una ordenada b y el resultado pasará por una función de activación, como en las redes neuronales tradicionales. Además, se debe de tener en cuenta que tanto la entrada como el filtro pueden tener mas de un canal. A esto, se le añade un conjunto de operaciones y decisiones derivadas de esta forma de red neuronal:

- Se pueden rodear los bordes de la entrada con ceros, de esta manera se solucionará el problema de no pasar todos los valores de la entrada el mismo número de veces por el filtro. Por ejemplo: en la figura anterior, el valor de la entrada en la posición (1,1) se computa solo una vez, mientras que el de la posición (2,2) se computa 4. A esto se le denomina relleno (*padding*)
- La operación de la convolución mostrada en la figura anterior, realiza un salto de una posición. Sin embargo, este paso (*stride*) podría configurarse con un valor mayor.

- Se pueden añadir filtros de agrupación (*pooling*) entre filtros de convolución para reducir las dimensiones. Las agrupaciones más comunes son seleccionando el valor máximo (*MaxPooling*) y calculando la media (*Average Pooling*).

La combinación de todos estos elementos hará variar tanto las dimensiones de salida como los resultados de las redes neuronales convolucionales (CNN) [27]. En una CNN, a continuación de las capas convolucionales, se aplana el vector de salida de la última capa para que tenga una sola dimensión y se hace pasar por varias capas de redes neuronales clásicas. La salida final es normalmente una función de clasificación *softmax*. A continuación se muestra un ejemplo de una arquitectura [28].

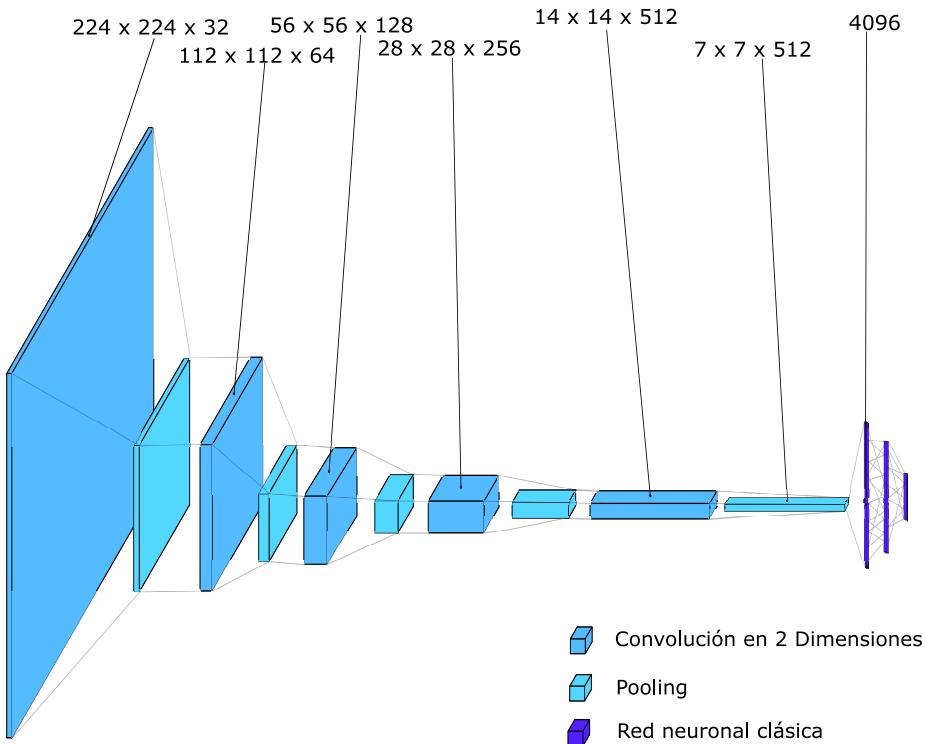


Figura 19: Ejemplo de una arquitectura basada en redes neuronales convolucionales inspirada en la arquitectura VGG-16.

Habitualmente, estas operaciones se realizan en 2 dimensiones. Sin embargo, las operaciones y conceptos son extrapolables tanto a 1 dimensión (señales) como a 3 dimensiones(vídeos, fotografías médicas en 3 dimensiones,...).

8.2 Modelos secuenciales

Como se ha podido observar con las CNN, se puede utilizar como base las redes neuronales artificiales clásicas y crear nuevos elementos y arquitecturas que se adaptan mejor a determinadas clases de datos. Otro ejemplo de esto son los modelos secuenciales. Los modelos secuenciales se utilizan para aprendizaje

automático aplicado a datos dependientes del tiempo, es decir, a series temporales. El principal caso de uso en la actualidad es el procesamiento del lenguaje natural [29].

En esta clase de modelos, se utilizan redes neuronales clásicas introduciendo como elemento la temporalidad. Para ello, se calcula la salida de una determinada capa no solo a partir de la entrada, sino a partir de un dato calculado en el instante de tiempo anterior. Es lo que se denomina redes neuronales recurrentes (RNN) [30].

De esta manera, sea un instante de tiempo t y una capa de la red neuronal recurrente l , el cómputo de la salida para una celda (una capa en un instante de tiempo) sería:

$$\hat{y}^{[l]<t>} = g(w_a^{[l]}[a^{[l]}<t-1>, a^{[l-1]}<t>] + b_a^{[l]}) \quad (15)$$

Donde $[a^{[l]}<t-1>, a^{[l-1]}<t>]$ se refiere a la concatenación de ambas matrices. Para calcular el algoritmo del descenso del gradiente, se procedería a calcular la función de pérdidas en cada instante de tiempo para luego computar la suma y realizar la propagación hacia atrás de las capas.

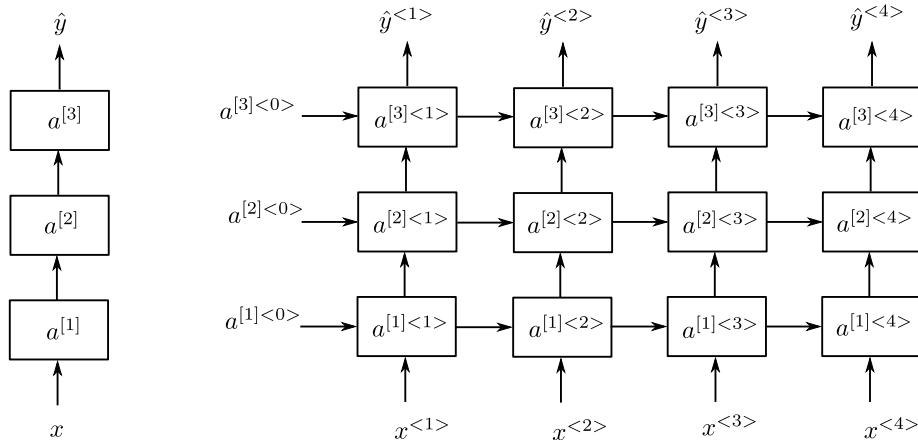


Figura 20: A la izquierda, la representación de una red neuronal clásica en formato de celdas. A la derecha, su equivalente convertido a una red neuronal recurrente.

La celda de uso más extendido es la Long short-term memory (LSTM) [31], cuya traducción sería algo así como *larga memoria a corto plazo*. Su extendido uso se debe a su capacidad para *recordar* características de la entrada en un instante t muchos instantes de tiempo después de haber ocurrido sin ocasionar desvanecimiento de gradientes.

Como se puede observar en la figura siguiente, esta arquitectura añade complejidad a lo que sería una capa de una red neuronal clásica. El concepto de las RNN prevalece: hacer a las entradas temporalmente dependientes. Para ello se introducen: la puerta de olvido con salida τ_f , la puerta de actualización

con salida τ_u , la puerta de cómputo parcial de $c^{<t>}$ con salida $\hat{c}^{<t>}$ y la puerta de salida τ_o .

La combinación de estas salidas dará la salida utilizada para hacer depender las celdas en el tiempo $c^{<t>}$ y la salida de la capa $a^{<t>}$, que será además utilizada con una función similar a $c^{<t>}$. Supóngase todo lo anterior y que nos encontramos en la capa l . No se añadirá ese superíndice para facilitar la notación.

$$\begin{aligned}\tau_u &= \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) & \hat{c}^{<t>} &= \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \\ \tau_f &= \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) & c^{<t>} &= \tau_u * \hat{c}^{<t>} + \tau_f * c^{<t-1>} \\ \tau_o &= \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) & a^{<t>} &= \tau_o * \tanh c^{<t>}\end{aligned}\tag{16}$$

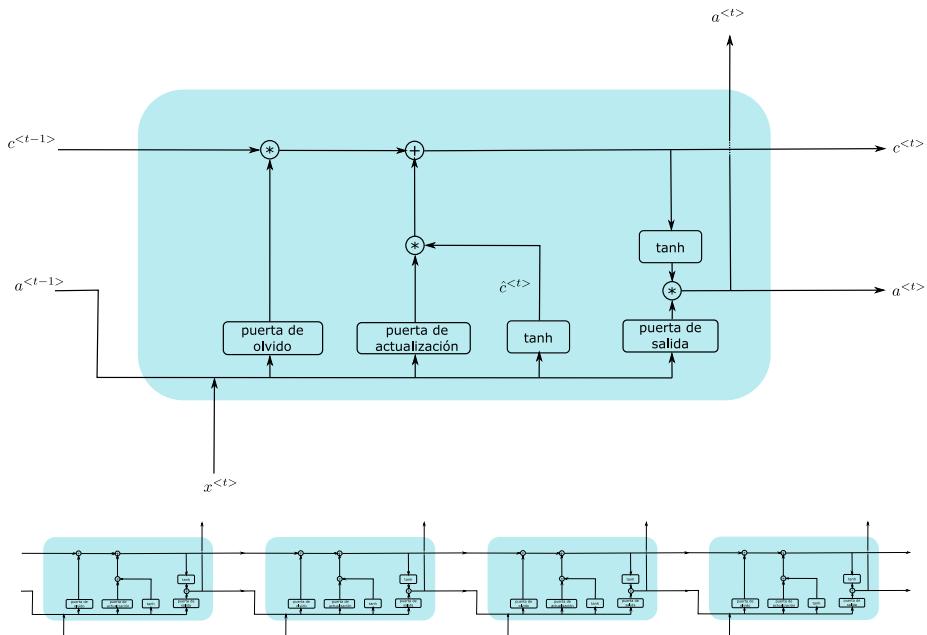


Figura 21: Representación de una celda LSTM. Debajo cómo sería la conexión temporal.

9 Redes generativas adversarias

A un nivel superior en las arquitecturas de redes neuronales se encuentra la combinación de modelos para obtener resultados concretos. Por ejemplo, se puede combinar una red de LSTM y una de CNN para detección de acciones en vídeo. Una arquitectura en auge son las Redes generativas adversarias (GANs).

Una Red generativa adversaria (GAN) [32] es una red generativa de aprendizaje profundo no supervisado basada en la teoría de juegos. La idea es la

combinación de una red neuronal generadora G , que calcula una salida que intenta engañar a una red discriminadora D , que intenta discernir si la entrada introducida corresponde a un dato real o a un dato generado.

La entrada a G es un vector latente aleatorio. Esto, junto a técnicas de regularización, evita que G llegue a un estado en el que solo genera 1 salida que engaña al discriminador, independientemente de la entrada y las veces que se realice el cálculo. La consecuencia de esto es un entrenamiento estocástico, en el que cada intento genera un resultado distinto.

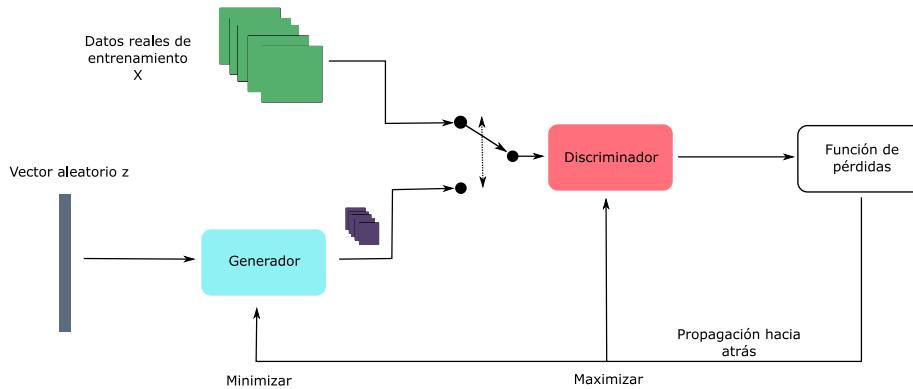


Figura 22: Representación de una red generativa adversaria tradicional.

El objetivo es llegar al conocido en teoría de juegos como equilibrio de Nash. Este equilibrio supone que ambos jugadores (discriminador y generador) han adoptado sus mejores estrategias, es decir, el generador no puede generar datos más realistas y el discriminador no puede identificar mejor los datos reales.

Entrenar una red generativa adversaria es una tarea difícil, hay que cuidar muchas variables. Por un lado, las arquitecturas tanto del generador como del discriminador deben de ser adecuadas para la tarea que se quiere resolver. El entrenamiento para un objetivo que requiere la abstracción de ambas arquitecturas a una conjunta supone un aumento notable de la complejidad.

A esta complejidad se le añade escoger unos buenos hiper-parámetros para los optimizadores y regularizadores de ambos modelos y una función de pérdidas que refleje bien la competición entre generador y discriminador. Para ello se usan funciones de pérdidas como la introducida en el paper original [32]. Sea $D(x)$ la estimación hecha por el discriminador para datos reales (en probabilidad), $G(z)$ los datos generados por el generador y $D(G(z))$ la estimación hecha por el discriminador para estos datos, la función de pérdidas será:

$$E_z[\log(D(x))] + E_z[\log(1 - D(G(z)))] \quad (17)$$

El generador tratará de minimizar esta función de pérdidas mientras que el discriminador tratará de maximizarla. Cabe añadir que los autores recomendaron más tarde modificar la función de pérdidas del generador para que trate de maximizar $\log(D(G(z)))$. Mas adelante, en nuevos trabajos [33] sugerían algunas

técnicas de mejora de entrenamiento de GANs como un término de promedio histórico $\|\theta - \frac{1}{t} \sum_{i=1}^t \theta^{<i>}\|^2$, siendo θ los parámetros (w, b) .

Muy prometedoras son también las GANs de Wasserstein (WGANs) [34], con funciones de pérdidas aún más simples. El discriminador intenta maximizar la diferencia entre su predicción de datos reales y falsos $D(x) - D(G(z))$, mientras que el generador intenta maximizar la salida del discriminador $D(G(z))$.

Problemas de explosión del gradiente obligan a utilizar técnicas de recorte del gradiente a modo de normalización de las salidas. Mas adelante, se publicó un paper [35] que trataría de mejorar esto eliminando el recorte de gradiente y añadiendo una penalización de gradiente a la función de pérdidas del discriminador:

$$D(G(z)) - D(x) + \lambda(\|\nabla D(\hat{x})\| - 1)^2 \quad (18)$$

Siendo $\hat{x} = \epsilon x + (1 - \epsilon)x$ una media aleatoria entre datos reales y datos generados.

Capítulo IV

Procesamiento de datos

10 Origen de los datos: PhysioNet

PhysioNet es una red de recursos de investigación financiada por el Instituto de Salud en Estados Unidos (*National Institutes of Health (NIH)*). Su objetivo es conducir y catalizar la investigación y educación biomédica ofreciendo acceso gratuito a grandes colecciones de datos fisiológicos y clínicos, y a software libre relacionado [36].

Cada año esta red, junto con la organización *Computing in Cardiology*, publica unos desafíos denominados *Computing in Cardiology Challenge* [37]. Estos desafíos pretenden potenciar el avance en tecnologías que mejoren el diagnóstico de enfermedades cardíacas. La publicación del desafío de este año era una buena oportunidad para realizar un proyecto como el que se pretende. Aunque no se participará en el desafío debido a los tiempos de entrega, si se participará activamente en la comunidad.

El desafío de este año consiste en la identificación de diagnósticos clínicos de ECV a partir de 12 derivaciones de ECG. Las clases a detectar son: AF, I-AVB, LBBB, Ritmo sinusal normal, PAC, PVC, RBBB, STD, STE.

No todas las enfermedades son igual de comunes, por lo tanto, los datos no están balanceados. Es decir, no hay el mismo número de datos de personas con AF, que es bastante común, que con STE. Este será un gran reto a tratar para poder desarrollar un sistema fiable.

Se aporta información sobre cómo fue recogida la señal (frecuencia, resolución de amplitud), además de datos demográficos y de diagnóstico sobre el paciente al que se le realizó el ECG. Estos datos se analizarán en profundidad en próximas secciones.

11 Los datos de entrenamiento

Se comentó superficialmente en la sección 7 cuando se habló de normalización: para llegar a un buen modelo necesitas tener un conjunto de datos adecuado. Para ello conviene, no solo que los datos estén normalizados, si no que sean suficientes en cantidad y diversidad y que hayan sido obtenidos de una misma fuente o con un criterio unificado.

Para abordar el problema de clasificación, antes se deben conocer bien los datos. Se dispone de 6.877 ECG. De ellos, 3.699 son de hombres y 3.178 de mujeres. Todos han sido capturados a 500 Hz y con sus 12 derivaciones. La longitud de las muestras varía entre las 3.000 (6 segundos) y las 72.000 (144 segundos).

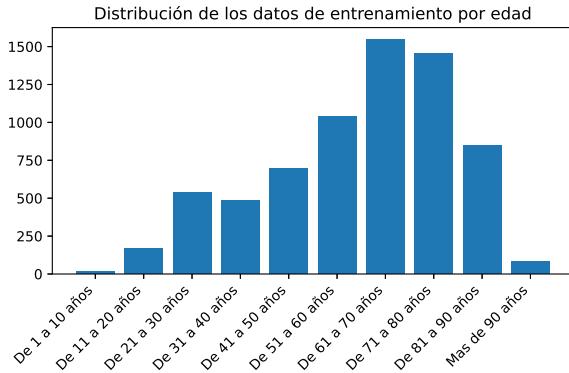


Figura 23: Distribución de los datos de entrenamiento por edad

La distribución de los diagnósticos tampoco es uniforme. Mientras se dispone de 1857 muestras de PVC, solo se tiene acceso a 220 muestras de STE. Cabe por último también mencionar que los datos son multiclas, es decir, hay ECG cuyo diagnóstico es más de una enfermedad. En concreto, encontramos 470 ECG que dan como resultado 2 patologías y 6 que dan 3.

Tabla 1: Distribución de enfermedades de los datos de entrenamiento

	AF	I-AVB	LBBB	Normal	PAC	PVC	RBBB	STD	STE
Cantidad	1.221	722	236	918	616	700	1.857	869	220

Será deseable que los datos tengan la misma longitud a la entrada del modelo, así como que se disponga de una distribución uniforme de enfermedades. El cómo se han abordado estas cuestiones es el tema a tratar en las siguientes secciones.

12 Preprocesado de los datos

Debido a la escasa cantidad de datos, resulta inviable entrenar un modelo que tome por separado distintas derivaciones de un ECG y las procese de forma paralela. Por ello, en la fase de entrenamiento cada derivación será un ECG en sí mismo.

12.1 Ruido

Los ECG de los que se dispone no han sido tratados. Así, la señal incluye ruido procedente de la respiración y los movimientos del cuerpo a bajas frecuencias y de agentes externos a altas frecuencias. Por ello, lo primero que se hace en la

etapa de preprocesado es filtrar el ruido.

Se ha utilizado un filtro de Butterworth paso banda de orden 1. Para ello se transforman las frecuencias de corte 0,001Hz y 15Hz a frecuencias normalizadas dividiendo entre la frecuencia de Nyquist $f_N = \frac{f_s}{2}$ y se utiliza la función de filtrado de *butter* definida en la librería *scipy*. Por último, se corrige el retraso introducido por el filtro.

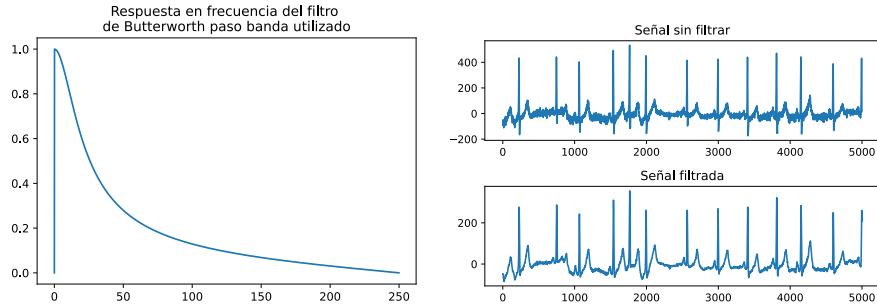


Figura 24: A la izquierda, la respuesta en frecuencia del filtro. A la derecha, una derivación de un electrocardiograma de los datos de entrenamiento antes y después de filtrar

12.2 Armonización de la entrada

Todos los datos tendrán la misma longitud. Así, los ECG de más de 3.000 muestras serán separados en ECG independientes de 3.000 muestras cada uno (6 segundos). El resultado es un aumento del número de datos debido a esta separación de señales en subseñales de 6 segundos y al tratamiento de las derivaciones de una señal como señales distintas.

Como se puede apreciar, las señales están muestreadas a gran calidad. Se necesitaría un modelo con mucha complejidad para clasificar datos de longitud 3.000. Sin embargo, se puede submuestrear la señal para disminuir su tamaño sin perder apenas calidad.

Se ha decidido submuestrear en un factor de 11,72 a 1. Así, la señal de 6 segundos pasa de tener una longitud de 3.000 a una longitud de 256 muestras. Se ha decidido este factor porque resulta conveniente en términos de optimización de los recursos computacionales utilizar números en potencias de 2.

12.3 Normalización y resultado final

A continuación se ha normalizado la señal entre 0 y 1. Esto permite un aprendizaje mas rápido, como ya se comentó en la sección 7.

$$\vec{x}_{norm} = \frac{\vec{x} - min(\vec{x})}{max(\vec{x}) - min(\vec{x})} \quad (19)$$

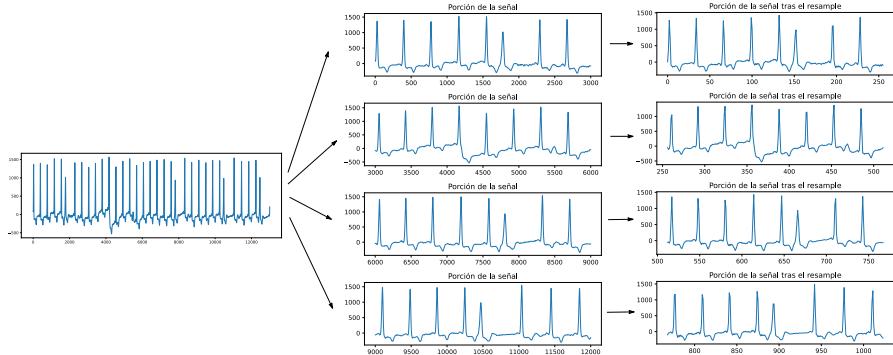


Figura 25: Proceso de separación y submuestreo de señales

Tras todo esto, los datos quedan listos para ser utilizados en el proceso de entrenamiento. Se dispone de 169.502 vectores de una longitud de 256 valores por vector acotados entre 0 y 1 y con un diagnóstico multiclase. La distribución de datos sigue estando desbalanceada. Solucionar esto es la materia de las siguientes secciones.

Tabla 2: Distribución de enfermedades de los datos de entrenamiento tras su procesado

	AF	I-AVB	LBBB	Normal	PAC	PVC	RBBB	STD	STE
Cantidad	27.789	15.537	5.289	21.705	19.401	24.202	41.013	21.154	5.950

13 Aumento de datos

Se podría tratar de entrenar al modelo con la distribución de datos con la que se cuenta. Sin embargo, contar con una distribución uniforme es más que deseable para obtener un buen modelo: que cumpla su función para todas las clases de enfermedades y que generalice adecuadamente. Para ello, en aprendizaje profundo se utiliza una técnica denominada aumento de datos. Esta consiste en generar más datos a partir de los datos de los que se dispone.

Para aumentar los datos, se pueden utilizar técnicas muy variadas. Para este trabajo se han utilizado dos técnicas: transformaciones y redes generativas adversarias.

13.1 Generación de datos mediante GANs

Como se comentó en la sección 9, entrenar una red generativa adversaria no es tarea fácil. Es una arquitectura no supervisada relativamente nueva y encontrar un modelo que no diverja es realmente complicado. En el caso de este proyecto, se ha invertido dos meses de tiempo ininterrumpido en un proceso iterativo de

investigación, prueba y error hasta que se ha conseguido un modelo que convergiese y diera unos resultados relativamente razonables. Se comenta esto, pues es difícil valorarlo cuando sólo se presentan los resultados finales. En el anexo C se explica detalladamente el proceso seguido para obtener el resultado presentado en esta sección.

Entrenar un modelo capaz de generar 12 derivaciones de 9 enfermedades distintas podría ser el objeto de investigación de un grupo entero, el cual debería contar además con una cantidad mucho mayor de datos y de recursos computacionales de los que se dispone. El generador obtenido, es capaz de generar señales que parecen una mezcla de distintas derivaciones. Así, los resultados obtenidos están lejos de ser ECG reales. Sin embargo, podrían ser suficientes a la hora de entrenar el modelo clasificador.

Por todo esto, no se evaluará el modelo según su capacidad para obtener resultados reales sino por su capacidad para mejorar el modelo clasificador. De cualquier modo, en este apartado se expondrá la arquitectura utilizada y los resultados obtenidos.

13.1.1 Arquitectura

Se requiere un modelo generativo multiclasa, pues se necesita una arquitectura de GAN capaz de generar datos condicionados a la clase que se desea obtener. Para ello, se ha desarrollado una GAN inspirada en una Conditional Generative Adversarial Network (cGAN), es decir, una red generativa adversaria condicional [38]. Concretamente, está basada en una Auxiliary Classifier GAN (AC-GAN), es decir, una red generativa adversaria con un clasificador auxiliar [39].

Esta red parte de una GAN como la original explicada en la sección 9, a la que se le añade en la entrada del generador información sobre la clase que se desea generar y una salida adicional en el discriminador que discierne entre las clases correspondientes. Así, la función de pérdidas consistirá en una parte correspondiente a la entropía cruzada binaria y otra parte correspondiente a la entropía cruzada categórica.

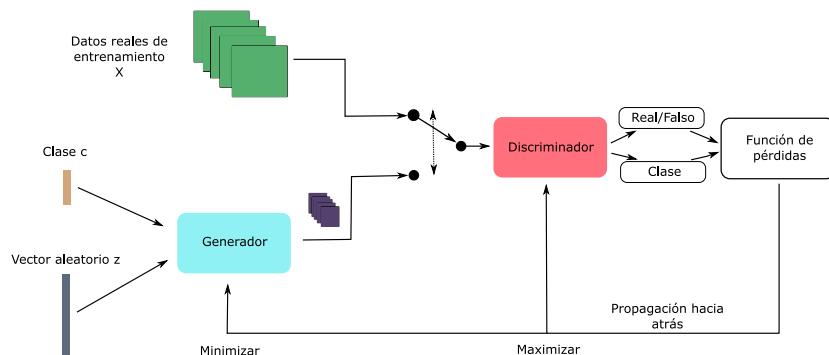


Figura 26: Representación de una AC-GAN.

Tanto el generador como el discriminador son CNN de una dimensión siguiendo las directrices de optimización de la arquitectura mas exitosa a día de hoy, Deep Convolutional Generative Adversarial Network (DCGAN) [40]. Esta incluye unos parámetros de optimización que han sido utilizados tanto para el discriminador como para el generador: un optimizador Adam con $\alpha = 10^{-4}$, $\beta_1 = 0,5$ y $\beta_2 = 0,999$

El generador consiste en una CNN invertida, que generará el ECG falso. Tiene dos entradas: la entrada del vector aleatorio z de 128 valores, que se procesa por una capa de red neuronal clásica para extender su tamaño; y la entrada c , que pasa por una capa de incrustación(*embedding*) que convierte valores en vectores. Ambas se remodelan y concatenan y entran a una sucesión de 3 capas que consisten en ampliar las dimensiones del vector al doble para luego pasar por una capa de convolución. Esta unión de operaciones sería el equivalente a la convolución traspuesta.

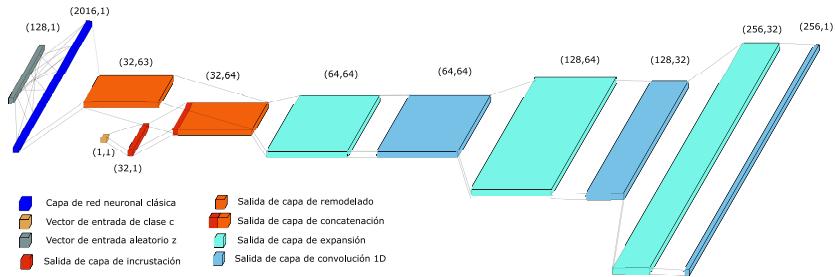


Figura 27: Representación del generador.

Además, siguiendo las recomendaciones del paper de DCGAN [40] y las de [33] se han ajustado la regularización y optimización de la siguiente manera:

- No se regulariza ni normaliza ni en la primera ni en la última capa.
- Se utiliza *batch normalization* con un momentum $\alpha = 0.8$ y *dropout* al 25% en el resto de capas.
- La función de activación utilizada es *ReLU* para todas las capas menos para la última, que es una *tanh*.
- Los filtros de convolución tienen una longitud de 3 unidades.
- Se utiliza el *padding* necesario para que la longitud de la señal a la salida de la convolución sea la misma que a la entrada.

El discriminador por su parte consiste en una CNN con 5 capas de convoluciones seguidas de dos capas totalmente conectadas (capas de redes neuronales clásicas). Se usa una *stride* de 2 para extraer las características de la señal disminuyendo su longitud a la mitad y aumentando sus canales. La segunda capa de las capas totalmente conectadas será las dos salidas: una para clasificación binaria (real o falso) y otra de clasificación categórica con 9 salidas (los posibles diagnósticos) mediante una función softmax.

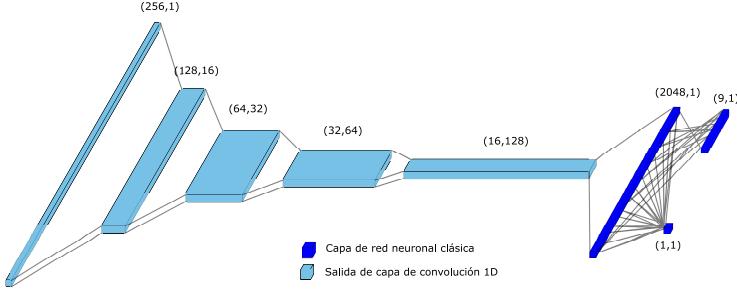


Figura 28: Representación del discriminador.

Siguiendo las recomendaciones del paper de DCGAN [40] y las de [33] se han ajustado la regularización y optimización de la siguiente manera:

- No se regulariza ni normaliza ni en la primera ni en la última capa.
- Se utiliza un *dropout* al 25% en el resto de capas.
- La función de activación utilizada es *LeakyReLu* para todas las capas.
- Los filtros de convolución tienen una longitud de 3 unidades.

13.1.2 Entrenamiento y resultados

Se ha entrenado a la red neuronal en una máquina virtual de la plataforma online *Google Colab* mediante un entrenamiento por lotes durante 200 épocas con un tamaño de 64 vectores por lote. Para ello, se debían guardar los pesos de manera periódica, pues el tiempo de ejecución de *Google Colab* está acotado (un máximo de uso de 12 horas). Se han utilizado 4 sesiones de *Google Colab*.

Algoritmo 4: Un paso del entrenamiento por lotes de la GAN

Data: x, y, z

```

c = calcular_vector_clases (y)
ecgs_falsos = generar_ecgs ([z,c])
fake_labels, fake = dicriminar (ecgs_falsos)
valid_labels, valid = dicriminar (x)
loss_d = calcular_loss (y, valid, fake, valid_labels, fake_labels)
loss_g = calcular_loss (y, fake, fake_labels)
backpropagation (calcular_gradienes (loss_d))
backpropagation (calcular_gradienes (loss_g))

```

La evolución de la función de pérdidas en las GANs no es relevante, pues el entrenamiento de una GAN trata de llegar al *Nash equilibrium* y no de minimizar la función de pérdidas. Así, para evaluar el resultado del entrenamiento de una GAN se utilizan otras métricas que se basan en el parecido entre las muestras reales y las generadas. Como ya se comentó en la sección 13.1, en el caso de este proyecto no se evaluará así, sino por su utilidad.

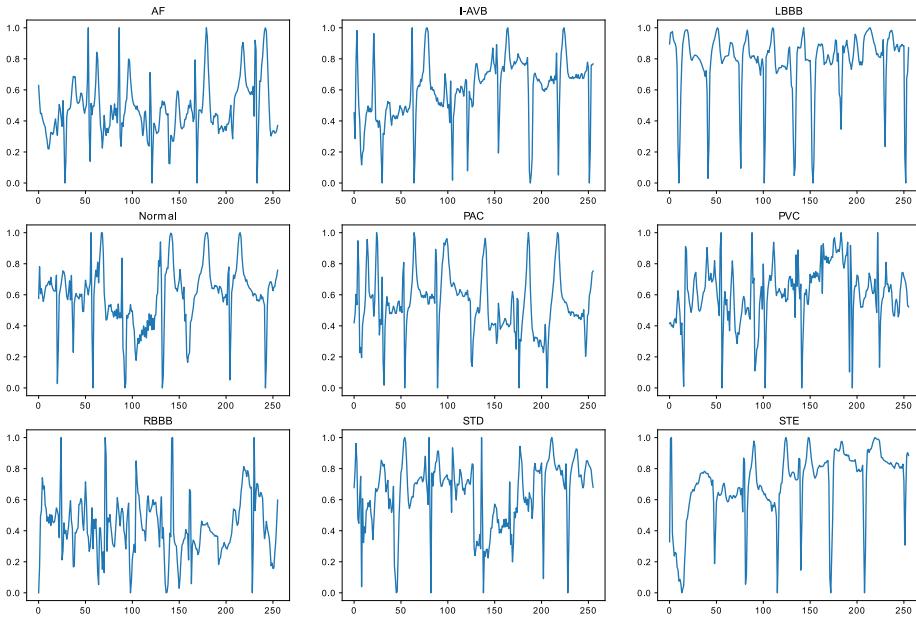


Figura 29: Muestras generadas normalizadas a trozos tras 200 épocas de entrenamiento por el modelo generativo.

13.2 Generación de datos mediante transformaciones

Esta forma de aumento resulta mucho más sencilla que la anterior. Podría llegar a complicarse si se decidiese aplicar transformaciones más complejas. Sin embargo, no es el caso. Se cree que las transformaciones aplicadas son suficientes para aumentar los datos de manera que parezcan de la misma distribución que los datos reales y con suficiente varianza respecto a los mismos.

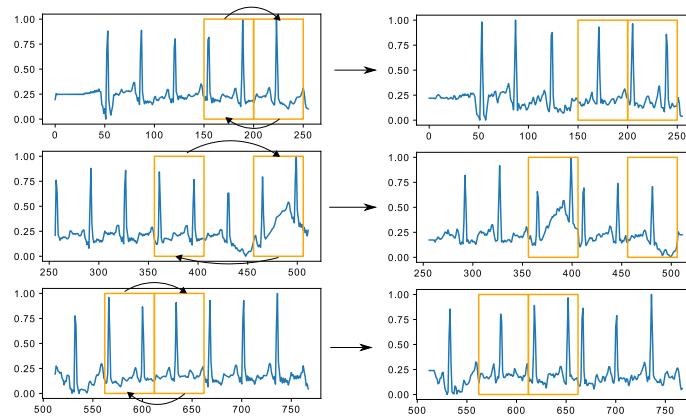


Figura 30: Transformación de una señal de los datos de entrenamiento que había sido recortada anteriormente.

El proceso es el siguiente:

1. Se toman dos ventanas aleatorias de 50 muestras.
2. Se intercambian estas dos ventanas
3. Se añade a la señal una señal de ruido gaussiano de amplitud entre -0,02 y 0,02
4. Se filtra y normaliza la señal resultante

Capítulo V

Modelo de clasificación

14 Arquitectura

Una vez se cuenta con una distribución de datos uniforme, se puede comenzar la búsqueda de un modelo clasificador que permita ofrecer ayuda al diagnóstico de enfermedades a partir del ECG. El diseño de la arquitectura del clasificador, al igual que pasaba en la sección 13.1, es un proceso iterativo largo. Aquí sólo se mostrarán los resultados, pero en el anexo D se detallan las tareas realizadas para conseguirlos.

Lo primero, ha sido investigar sobre el estado del arte de arquitecturas para propósitos similares. Se han encontrado trabajos muy interesantes que sin duda han sido punto de partida a la hora de diseñar la arquitectura. Estos trabajos se basan en el uso de CNN [41], RNN [42] o en la combinación de ambas [43] y [44].

Con esto en mente, se ha realizado un proceso iterativo trabajando con arquitecturas de los tres tipos. El resultado obtenido es una arquitectura mixta que tiene en cuenta: la temporalidad de la señal, la forma de la misma y datos cualitativos (sexo, edad y derivación introducida). Esto implica un modelo de entrada múltiple con capas de redes neuronales convolucionales, redes neuronales recurrentes y redes neuronales clásicas. A continuación, se expone la arquitectura final.

14.1 Primera entrada. Datos cualitativos

Este es un bloque sencillo que recoge los tres parámetros anteriores y los interpreta como un vector de longitud 192. La idea es expresar estos tres datos cualitativos en un vocabulario numérico con 192 números, de forma similar a como se hace con las palabras en Procesamiento del lenguaje natural (NLP).

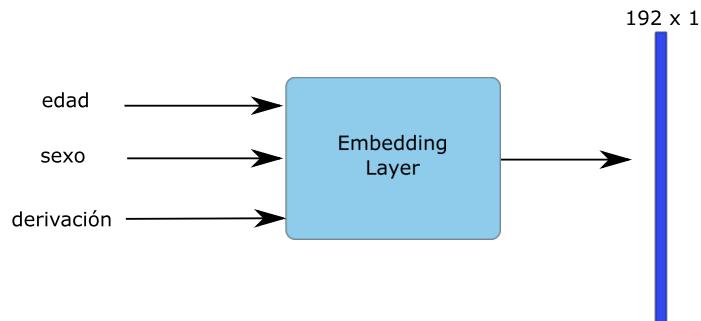


Figura 31: Bloque de entrada de datos cualitativos.

Por dentro, este bloque son capas de redes neuronales clásicas, que se entrenan junto con el resto del modelo para interpretar los datos de la mejor forma posible. A efectos del trabajo, a este bloque se le denominará *Embedding3-192*.

14.2 Segunda entrada. Electrocardiograma

A lo largo de la sección 13 se pudo ver cuál es el tratamiento de los datos y el resultado final del mismo. La entrada del clasificador será un vector de longitud 256, correspondiente a un trozo de una derivación de un electrocardiograma. Este bloque se compone de un sub-bloque de LSTM bidireccionales para luego entrar en un sub-bloque de CNN.

Una capa de *Bidirectional Long short-term memory (BiLSTM)*, son dos capas con la misma entrada de LSTM estándar: una que va en sentido de tiempo ascendente y otra descendente. A continuación, las salidas de estas dos capas pueden o no combinarse. El objetivo es que no solo un instante de tiempo posterior dependa de uno anterior, sino que además un instante de tiempo anterior dependa de uno posterior.

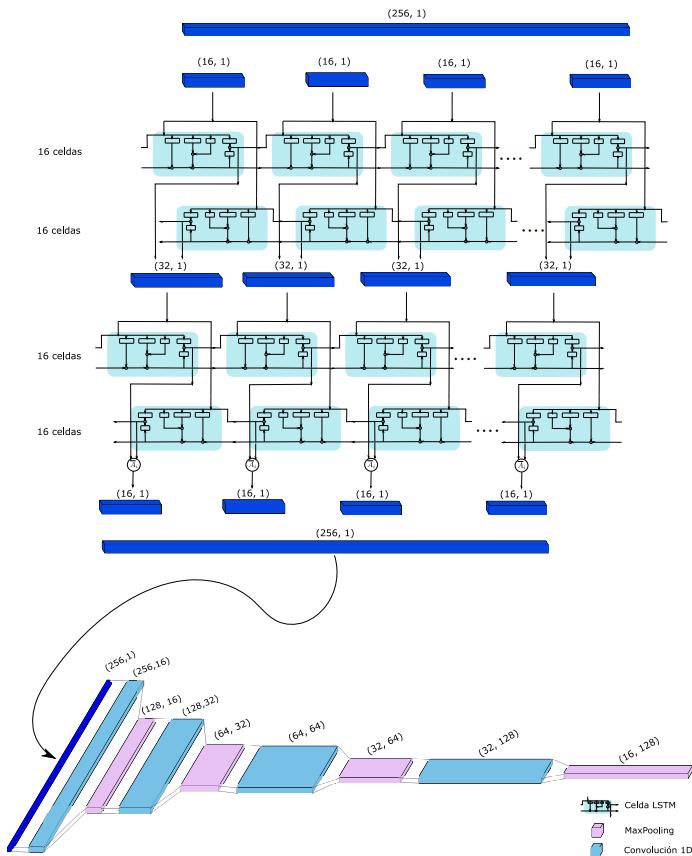


Figura 32: Bloque de entrada de electrocardiogramas.

Así la señal puede tener un contexto conjunto, que es la realidad en el caso de los ECG. En este caso se utilizan dos capas de BiLSTM, cuya salida temporal y antitemporal se usan por separado en la primera capa y se agrupan mediante media aritmética en la segunda.

A continuación, esta salida se introduce en un sub-bloque de CNN de una dimensión. Cada capa convolucional se compone de:

- Un filtro de convolución con tamaño de *kernel* 3, *padding* de tipo *same* y *stride* 1.
- Normalización por lotes (*batch normalization*).
- Función de activación.
- Simplificación por máximos (*Max Pooling*), reduciendo la longitud de los datos a la mitad.
- Regularización con *dropout*.

La salida final es un vector de longitud 2048. A efectos del trabajo, a este bloque se le denominará *2BiLSTM-Conv4*.

14.3 Modelo final

Por último, ambos bloques se concatenan y se introducen a un conjunto de capas de *fully connected layers* (redes neuronales clásicas), cuya capa final es la salida del clasificador.

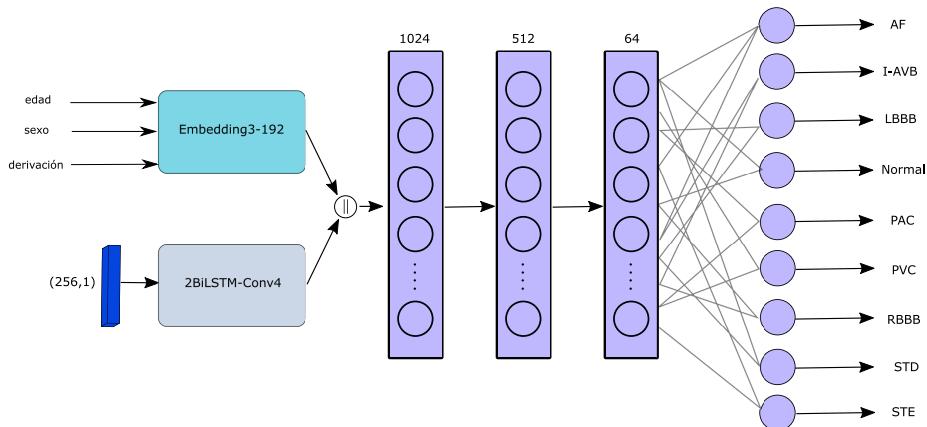


Figura 33: Modelo completo.

Como función de activación se ha usado *ReLU* para todas las capas, excepto para la capa de salida que se ha usado la función *softmax*. La normalización por lotes se ha realizado con un momentum de $\alpha = 0,8$ para todo el modelo. La regularización mediante *dropout* es del 25% para todo el modelo.

15 Resultados

15.1 Entrenamiento

Con la arquitectura definida, se ha procedido al entrenamiento del modelo. Se han entrenado dos modelos, cada uno de ellos con una distribución de datos: uno con la ampliación de datos mediante GANs y el otro mediante transformaciones, como ya se explicó en la sección 13. Se ha destinado un 95% de los datos a entrenamiento y un 5% a validación. Para cada conjunto, las señales se han escogido de forma aleatoria. En una distribución de datos uniforme se estima que esto hace que se mantenga la distribución de datos, haciendo que las clases sigan estando balanceadas.

Se ha entrenado a la red neuronal en una máquina virtual de la plataforma online *Google Colab* mediante un entrenamiento por lotes de tamaño 64 durante 400 épocas para cada modelo. Esto ha supuesto un tiempo de entrenamiento de aproximadamente 7 horas. Como optimizador se ha utilizado Adam con $\alpha = 10^{-4}$, $\beta_1 = 0,9$ y $\beta_2 = 0,999$. Como función de pérdidas se usa la entropía categórica cruzada, que es la función de pérdidas estándar para problemas de clasificación en aprendizaje profundo.

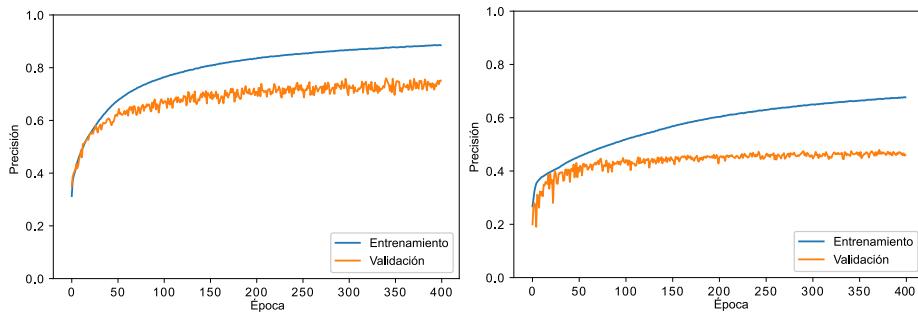


Figura 34: Evolución de la precisión del modelo en cada época de entrenamiento. A la izquierda, con ampliación de datos con transformaciones. A la derecha, con redes generativas adversarias.

Como se puede observar, la precisión del modelo entrenado con datos generados con GANs es bastante baja. Sin embargo, esta precisión tiene en cuenta datos que son falsos y podrían ser *ruido* a la hora de predecir adecuadamente. En el siguiente apartado se tratará cómo se usa este modelo y los resultados finales que se obtienen, que son muy positivos.

15.2 Utilización del modelo y resultados

Con lo anterior, se ha obtenido un modelo que a la entrada recibe una señal de longitud 256 y unos parámetros cualitativos y a la salida devuelve un vector de probabilidades. Sin embargo, como se vio en la sección 12, un ECG se compone de doce derivaciones y, cada una de estas derivaciones, puede tener mas de un segmento. Así, el proceso de diagnóstico consistirá, a grandes rasgos, en hacer la media del resultado de este modelo para todos estos segmentos. El algoritmo

de diagnóstico es muy sencillo:

Algoritmo 5: Diagnóstico a partir de un electrocardiograma

Data: *modelo, ecg*
señales, n_derivaciones, diagnósticos = new lists()
señales, edad, sexo, n_derivaciones = **procesar_ecg** (ecg)
diagnósticos = *modelo.evaluar* (señales, edad, sexo, n_derivaciones)
for *d* \leftarrow *diagnosticos* **do**
 \lfloor *d* = **one_hot_con_maximo** (*d*)
 diagnóstico = **media_aritmetica** (*diagnósticos*)
 diagnóstico = **filtro_de_sesgo** (*diagnósticos*)
 diagnóstico = **get_diagnóstico_con_umbral_o_maximo** (*diagnóstico*)

1. Procesa las señales para obtener una lista de estructuras de datos en la que cada ítem contiene: edad, sexo, nombre de la derivación y señal de longitud 256.
2. Se evalúa cada elemento de esta lista.
3. Para cada diagnóstico, se obtiene un vector representando las 9 enfermedades donde todos los valores son 0 menos el máximo valor obtenido.
4. Se realiza una media aritmética de todos los diagnósticos.
5. Se filtra el sesgo del modelo, es decir, si por ejemplo el modelo tiene una clara desviación por la que identifica las I-AVB como RBBB y el vector tiene como valor máximo RBBB y como segundo valor máximo I-AVB, se intercambian estos valores.
6. Por último, se devuelve un vector donde todos los valores son cero menos aquellos que sean superiores a 0,6. Si ninguno es superior al 0,6, se pone a 1 el máximo valor.

Con este algoritmo y el modelo utilizado, se obtienen unos resultados muy buenos. En el modelo con transformaciones, se utiliza un filtro de sesgo entre la clase *Normal* y la clase *PAC*. En el modelo con GANs, no se utilizan filtros de sesgos.

Los excelentes resultados obtenidos en el modelo de GANs comparado con la evolución de la precisión durante el entrenamiento, lleva a pensar que el hecho de utilizar datos no del todo consistentes (los generados) junto con datos reales en el entrenamiento, evita el overfitting y no perjudica en el entrenamiento de la predicción de aquellos datos reales.

Como se puede ver, cada modelo tiene fortalezas en clases distintas. Para la herramienta final, se ha decidido utilizar ambos modelos. Se ha ponderado su resultado, dando la mitad de importancia a cada uno. El resultado final es uno con una precisión ligeramente superior.

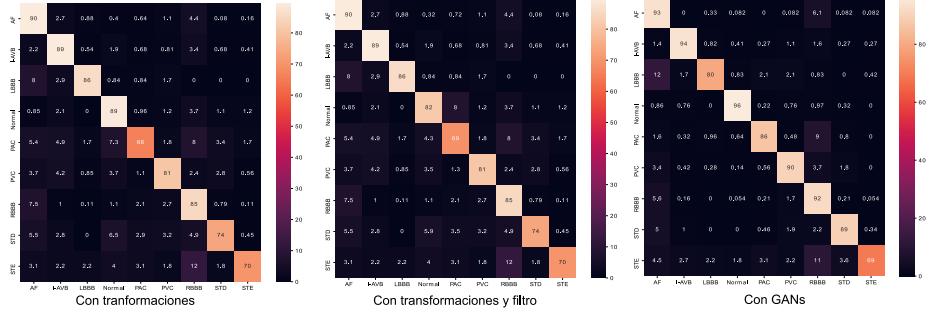


Figura 35: Matrices de confusión. A la izquierda el modelo entrenado con transformaciones utilizando el algoritmo de diagnóstico sin filtros. En el medio con filtros. A la derecha, el modelo entrenado con GANs

Tabla 3: Precisión de cada modelo y precisión final en el conjunto de datos total.

	Transformaciones	GANs	Final
Precisión	88,19%	94,78%	95,11%

La tasa de falsos positivos, tasa de diagnóstico de una enfermedad cuando se debería diagnosticar ritmo sinusal normal, es del 0,45%. La tasa de falsos negativos, tasa de diagnóstico de ritmo sinusal normal cuando se debería diagnosticar una enfermedad, es del 0,47%.

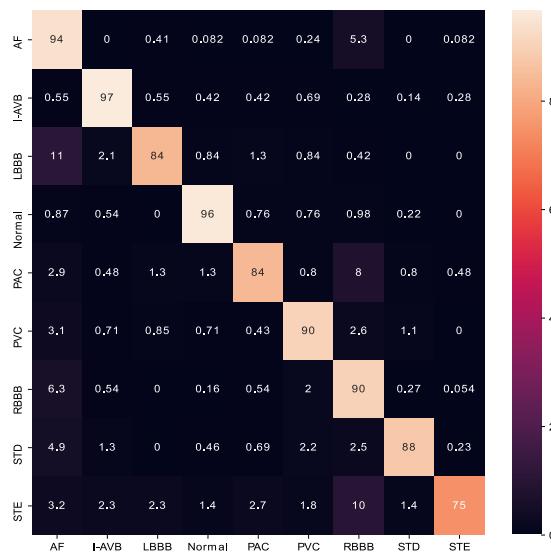


Figura 36: Matriz de confusión del modelo final

El hecho de utilizar modelos con distintas distribuciones de validación y entrenamiento, podría comprometer la capacidad de generalización de este modelo y producir *overfitting*. Esto es un asunto a tener en cuenta en futuras evaluaciones del mismo, además de a la hora de validar los datos. Lo ideal hubiera sido contar con más de una distribución y con mayor cantidad de datos. De esta forma, hubiera sido posible reservar datos para una evaluación final, tras la construcción del modelo conjunto.

Capítulo VI

Herramienta de ayuda al diagnóstico

16 Componentes

16.1 Estilo arquitectónico

Para el desarrollo de la herramienta, se ha considerado que la arquitectura más conveniente es una página web de cliente que hace llamadas a una Interfaz de programación de aplicaciones (API) tipo Representational state transfer, en castellano transferencia del estado representacional (REST), es decir, una arquitectura orientada a servicios. Además, será 100% sin estado: la página web envía los ficheros de un ECG, a lo que el servidor responde con un diagnóstico.

Con el desarrollo realizado, la API REST contará con un recurso, el clasificador. Sin embargo, este modelo de desarrollo permitiría añadir nuevos recursos al servidor si se desea ampliar las funcionalidades de la herramienta en trabajos futuros.

16.1.1 Diagrama de secuencia del sistema (SSD)

Se tiene una visión global de este sistema, que solo tendrá un caso de uso: la ayuda al diagnóstico de enfermedades cardiovasculares a partir de la subida de ficheros con el electrocardiograma e información demográfica. Esto se podría reformular como una *User story*:

Como personal médico quiero poder enviar los ficheros necesarios para obtener una ayuda en el diagnóstico de enfermedades cardiovasculares.

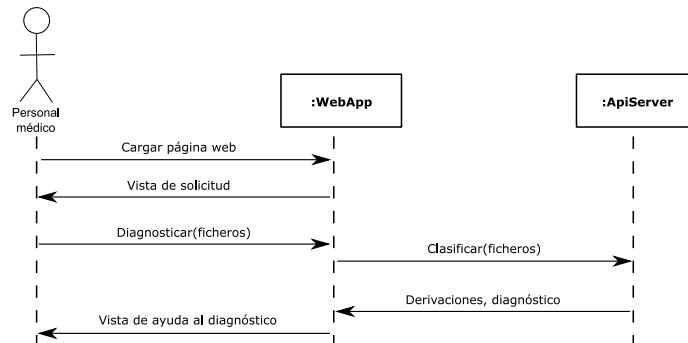


Figura 37: Diagrama de secuencia del sistema(SSD).

16.2 Proveedor

Los modelos están desarrollados en *Python* [45] en su versión 3 con la librería *Tensorflow* [46] en su versión 2. Por ello, se ha considerado que lo mas conveniente resulta utilizar *Python* 3 para la parte servidora. Para ello, se ha escogido el framework *Flask* [47].

Flask está basado en la especificación Web Server Gateway Interface (WSGI), una convención de llamadas a servidores web para *Python*; y en *Jinja*, un motor de plantillas web para este mismo lenguaje. *Flask* permite gestionar llamadas a recursos, lo que lo convertía en una buena opción.

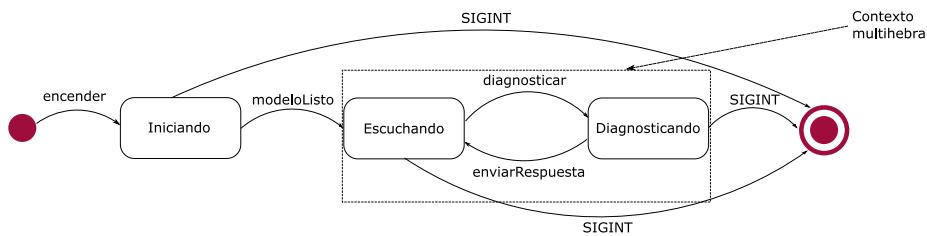


Figura 38: Diagrama de estados del servidor

Cuando el servidor se enciende, crea una clase tipo *Singleton* que almacena el modelo y tiene un método con el que se puede llamar al algoritmo de diagnóstico explicado en la sección 15.2. Esto se hace para que solo se almacene una instancia del mismo. En un contexto de ejecución en paralelo, crear una instancia para cada cliente que hiciera una petición supondría tiempos de espera mas largos, pues la creación del modelo y su almacenamiento en memoria RAM es una operación costosa.

Una vez almacenado este *Singleton*, se expone el recurso */classifier* con un método *POST* en el puerto 3000. Cuando se llama al recurso, se comprueba que los ficheros sean correctos. A continuación se realiza el diagnóstico, que es enviado junto con las doce derivaciones procesadas al cliente para su visualización en formato Javascript Object Notation (JSON).

16.3 Consumidor

El consumidor de las llamadas a la API REST será una Aplicación web progresiva (PWA) diseñada con la biblioteca de Javascript *React* [48]. El objetivo de esta aplicación es tener una interfaz web de usuario sencilla, con la que el personal médico pueda interactuar para solicitar y, posteriormente, visualizar el diagnóstico ofrecido por el modelo.

Para contar con una interfaz de usuario fácil y bonita, se ha utilizado el framework *Material-UI* [49], que nos proporciona componentes listos para ser utilizados basados en el lenguaje de diseño *Material Design*.

16.3.1 Vista de solicitud del diagnóstico

La pantalla principal es una vista sencilla con instrucciones para comenzar el diagnóstico y un formulario para subir los archivos. Se pueden arrastrar los archivos hasta la zona de captura, así como explorar el sistema de ficheros del usuario. Una vez seleccionados los ficheros correspondientes al electrocardiograma, que contienen las doce derivaciones e información sobre edad, sexo, etc; pulsamos en el botón *Diagnosticar*.

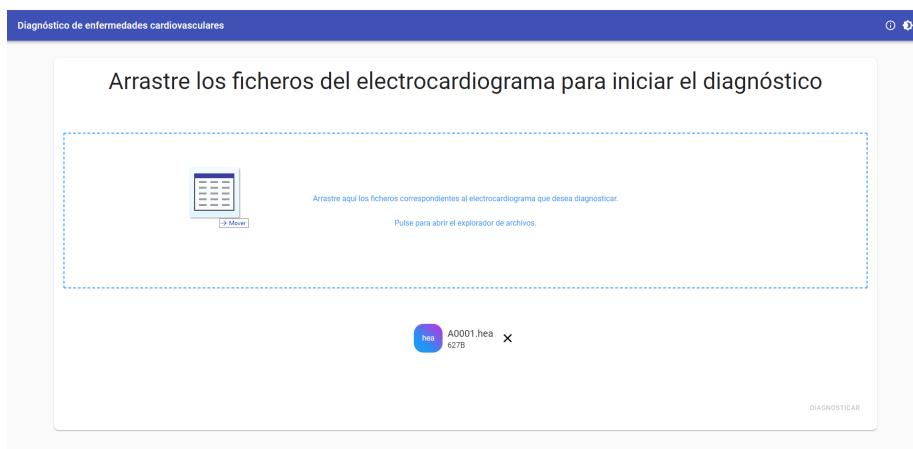


Figura 39: Vista de la página de inicio de la aplicación web

En la barra de navegación, el título nos permite recargar la página con un click, para empezar de cero. El primer botón despliega un diálogo con una leyenda con el significado de cada sigla, por ejemplo: AF - Fibrilación auricular. El último botón permite cambiar la aplicación entre modo oscuro y modo claro.

16.3.2 Vista de ayuda al diagnóstico

Una vez se ha enviado al servidor proveedor de recursos los archivos y este ha devuelto el JSON con toda la información del diagnóstico, la aplicación cliente muestra esta vista. A la izquierda, se ofrece la posibilidad de visualizar las doce derivaciones del electrocardiograma. A la derecha, se ofrece la predicción realizada, así como información sobre cómo se comporta el modelo utilizado.



Figura 40: Vista de la página de ayuda al diagnóstico de la aplicación web. Modo oscuro

Para todas las gráficas de esta parte se ha utilizado *nivo* [50], una librería de componentes de *React* basada en *D3.js* [51]. Esta ofrece una gran cantidad de bonitos componentes interactivos para visualización de datos.

Cada menú de la izquierda puede desplegarse, mostrando la gráfica de la derivación correspondiente para el diagrama utilizado. Además, se puede hacer zoom in y zoom out para una mejor inspección de la señal.

Derivaciones del electrocardiograma a diagnosticar

A continuación se ofrece una vista de las derivaciones del electrocardiograma tras ser filtradas. Sobre una derivación

- Ctrl + Click para hacer zoom in
- Ctrl + Mayus + Click para hacer zoom out



Figura 41: Vista de la página de ayuda al diagnóstico de la aplicación web. Detalle en la visualización de las derivaciones.

En la parte del diagnóstico, primero se muestra la enfermedad diagnosticada y un diagrama interactivo con las probabilidades predecidas por el modelo. A continuación se muestra la matriz de confusión del modelo, con el objetivo de que el personal médico pueda hacerse una idea de la fiabilidad del mismo para el diagnóstico de cada clase.



Figura 42: Vista de la página de ayuda al diagnóstico de la aplicación web. Detalle en la visualización del diagnóstico.

17 Despliegue

Una vez se cuenta con el sistema completo, se debe desplegar. Para ello, se deben satisfacer ciertos requisitos de estabilidad y seguridad que van implícitos a la utilización de la herramienta:

- Se debe poder desplegar la aplicación de forma rápida en cualquier máquina, independientemente de su sistema operativo y configuración.
- La información utilizada es sensible, se debe garantizar la confidencialidad e integridad de los datos.

El hecho de ser una aplicación sin estado, facilita mucho las tareas de mantenimiento. Basta con apagar el sistema en funcionamiento y desplegar la herramienta con las nuevas actualizaciones.

17.1 Arquitectura de despliegue

Para cumplir los requisitos comentados, se ha optado por una arquitectura basada en contenedores (*Docker*) [52]. Esto otorga independencia en la configuración y permite controlar la seguridad del sistema de forma más efectiva.

Continuando con esta vía, se considera adecuado tener en la máquina donde se despliega la herramienta un solo puerto abierto, y con un solo certificado para la conexión sobre *https*. Por ello se ha utilizado la herramienta de definición y despliegue multi-contenedores de Docker: *docker compose*. Así, se podrá tener una red virtual que conecte un contenedor con un proxy inverso *NGINX* [53], que es el que se expondrá al exterior, con el contenedor que sirve la web cliente y el contenedor que sirve la API REST.

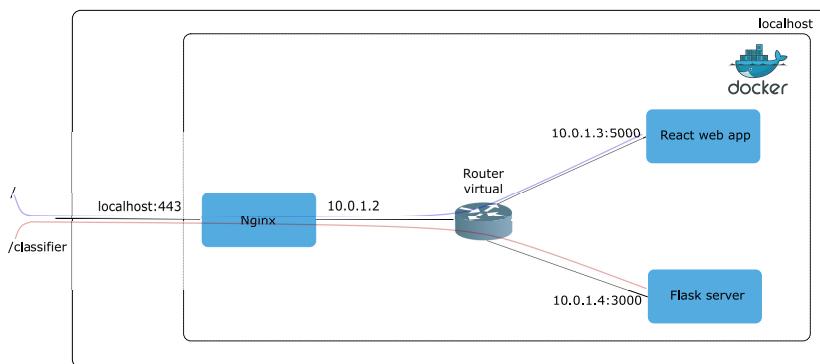


Figura 43: Arquitectura de despliegue.

Para el contenedor con *NGINX*, se ha usado una imagen de *Alpine Linux* [54] configurada para este tipo de servidor proxy. *Alpine Linux* nos permite contar con contenedores linux muy ligeros que cuentan con lo estrictamente necesario para servir una aplicación específica. A este contenedor se le copia el certificado que se ha obtenido previamente en la máquina que alberga los contenedores, así

como la configuración del proxy inverso. Esta configuración incluye la redirección del puerto 80 al 443 así como direcciones de proxy inverso, que permiten dirigir las llamadas a la máquina con ruta / al contenedor de *React* y las llamadas a /*classifier* al contenedor de *Flask*.

Para el contenedor de *React*, se ha utilizado una imagen de *Alpine Linux* configurada para *Node.js* [55]. A esta se le copian los ficheros de la aplicación. A continuación se instalan las dependencias necesarias, se monta una versión de despliegue y se sirve con una herramienta de npm llamada *serve*.

Por último, para el contenedor de *Flask* se utiliza una imagen optimizada por los desarrolladores de *Python* basada en *Debian*. A esta se le copian los ficheros de la aplicación. A continuación, se instalan las dependencias necesarias. Por último, se sirve con un servidor WSGI llamado *gunicorn*.

17.2 Resultado final

Con la herramienta ya lista para el despliegue, solo falta una máquina para desplegarla y un dominio para identificarla. Se ha utilizado una máquina de *Google Compute Engine* con una *CPU* de 1 núcleo y 1,75GB de RAM. La máquina en la que se decida desplegar depende de la cantidad de peticiones que se espere tener. Los requisitos son:

- Una máquina con un sistema operativo que permita virtualización.
- Tener *Docker* y *docker compose* instalados.
- Haber añadido el nombre del dominio a la lista de hosts en /etc/hosts.
- Tener un certificado para ese nombre de dominio.
- Abrir los puertos 80 y 443.

En este caso, se ha utilizado *duckdns* [56]: una página que ofrece nombres de dominio con su extensión de forma completamente gratuita. El dominio se puede visitar: tfg-ecg.duckdns.org.

Capítulo VII

Conclusiones y líneas futuras

18 Conclusiones

El objetivo de este TFG era diseñar y desarrollar una herramienta de ayuda al diagnóstico de enfermedades cardiovasculares a partir de electrocardiogramas, usando para ello aprendizaje profundo. Cuando se comenzó este proyecto, se contaba con poco más de 4 meses para su realización. Se ha trabajado muy duro para llegar al objetivo marcado con un resultado de la mayor calidad posible. En ese sentido, se está muy conforme con los resultados obtenidos.

La primera fase del trabajo fue la documentación. Se parte de una carrera técnica y se pretende dar una solución en el campo de la Ingeniería Biomédica. Fue necesario un proceso de adquisición de conocimiento en enfermedades cardiovasculares, características de los electrocardiogramas y su uso para el diagnóstico, etc. Además, hubo un gran trabajo de documentación en el estado del arte de modelos de aprendizaje automático: modelos secuenciales, convolucionales, generación de datos con GANs, trabajos anteriores en el mismo campo, etc. Se considera que ha sido un proceso bastante enriquecedor.

La fase del trabajo más complicada fue el aumento de datos. Llevó aproximadamente 2 meses de trabajo. Se habían visto trabajos anteriores que buscaban objetivos similares, pero el problema era totalmente nuevo. No se trataba sólo generar un electrocardiograma, sino que tenía que ser de una enfermedad específica. Además, las GANs nacieron en 2014, por lo que se trata de un campo en pañales. La información, aunque de utilidad, era muy limitada en el dominio de la generación de señales por lo que se trató de todo un reto.

La realidad es que hubiera sido de interés continuar esta fase: ya se había conseguido un modelo que generaba electrocardiogramas, pero no se tenía en cuenta la derivación a generar. Hubiera sido interesante continuar en este sentido, pero el tiempo disponible es limitado. Aunque se ha visto que la ampliación de datos con GANs utilizada está lejos de ser ideal, ha ayudado en el proceso de entrenamiento del clasificador.

La fase de clasificación tampoco fue fácil. Sin embargo, se contaba con la experiencia previa del clasificador utilizado en la GAN. Esto permitió evitar muchos errores que sí se cometieron anteriormente. Así, el proceso incremental que permitió encontrar la arquitectura adecuada duró un mes. Los resultados son muy buenos, hubiera sido deseable contar con más datos para validar y es posible que el modelo no generalice bien para otras distribuciones, pero para el dominio en el que se encuentra este trabajo es un gran punto de inicio que se puede retomar en el futuro con más datos y recursos.

Por último, el proceso de despliegue de la aplicación duró poco más de 3 semanas. Salvo algunos problemas puntuales, resultó sencillo. Ya se había trabajado anteriormente con contenedores, desarrollo de aplicaciones web, de-

sarrollo de una API en Python, etc; ya sea en proyectos personales o académicos. Se considera que el resultado final es elegante, bonito, sencillo y útil, lo cual es todo un éxito.

En cuanto a su uso en hospitales, la situación actual no ha permitido una colaboración estrecha con el Hospital 12 de Octubre. Sin embargo, se ha mostrado la herramienta a un cardiólogo del hospital. Este, se ha mostrado entusiasmado con la idea. Esta reacción es una buena noticia, pues abre las puertas a colaboraciones futuras para la mejora y el despliegue de esta herramienta en un entorno real.

19 Líneas futuras

Como se ha dejado ver en conclusiones, se tienen ideas para mejoras incrementales del trabajo realizado. Se desea que investigadores interesados continúen el trabajo comenzado, pues de verdad se cree que esta podría llegar a ser una herramienta de gran utilidad para el personal médico. A continuación, se exponen algunas mejoras.

En la actualidad, esta herramienta está adecuada para el formato de ficheros del conjunto de entrenamiento utilizado. Sin embargo, este formato no es el usado en hospitales. Un trabajo importante para que esta herramienta sea utilizada en hospitales, sería modificarla para que fuese compatible con el formato de las señales generadas por los equipos utilizados.

Otra opción sería ampliar la herramienta para que esta contase con la posibilidad de gestión de pacientes. De esta manera, se podría almacenar la información de un paciente: sexo, edad, señales, anteriores diagnósticos. Esta información podría ser utilizada para nuevos diagnósticos en un hipotético modelo mejorado que tuviese en cuenta diagnósticos anteriores. Para esta línea de mejora, sería necesario cambiar la arquitectura a una cliente-servidor, así como añadir un contenedor como Base de Datos conectado al servidor *Flask*.

En este mismo sentido, se podría contar con una pantalla de visualización de pacientes que incorporara información de diagnósticos anteriores. Esto cambiaría el enfoque de la herramienta, que de ser una herramienta de ayuda al diagnóstico podría llegar a convertirse en una herramienta de seguimiento de pacientes en cardiología, con muchas más funcionalidades.

Otras mejoras a considerar tienen que ver con el modelo de aprendizaje profundo utilizado. Se considera que se ha hecho un buen trabajo y que la arquitectura y técnicas utilizadas tienen potencial. Sin embargo, sería de gran interés conseguir conjuntos de datos de entrenamiento más grandes y diversos entre ellos. Estos podrían incluso incorporar más enfermedades. El objetivo final sería alcanzar un modelo que generalice mucho mejor y que tenga buenos resultados, para hacer de esta herramienta una que se pueda usar de forma global en muchos hospitales.

Bibliografía

- [1] OMS. Enfermedades cardiovasculares. (Click para ir a la web). [Fecha de último acceso: 13/03/2020].
- [2] OMS. Disease burden and mortality estimates. (Click para ir a la web). [Fecha de último acceso: 13/03/2020].
- [3] INE. Defunciones según la causa de muerte. (Click para ir a la web). [Fecha de último acceso: 13/03/2020].
- [4] INE. Población residente por fecha, sexo y edad. (Click para ir a la web). [Fecha de último acceso: 13/03/2020].
- [5] INE. Tasa bruta de natalidad por provincia. (Click para ir a la web). [Fecha de último acceso: 13/03/2020].
- [6] Electrocardiograma. (Click para ir a la web). [Fecha de último acceso: 13/03/2020].
- [7] Eder Iván Zamarrón López. Electrocardiografía dirigida para áreas críticas i. (Click para ir a la web). [Fecha de último acceso: 13/03/2020].
- [8] J. Pan and W. J. Tompkins. A real-time qrs detection algorithm. *IEEE Transactions on Biomedical Engineering*, BME-32(3):230–236, 1985.
- [9] LITFL. Ecg basics. (Click para ir a la web). [Fecha de último acceso: 27/03/2020].
- [10] Nodo atrioventricular. (Click para ir a la web). [Fecha de último acceso: 27/03/2020].
- [11] Cardiomocito. (Click para ir a la web). [Fecha de último acceso: 27/03/2020].
- [12] Nódulo sinoauricular. (Click para ir a la web). [Fecha de último acceso: 27/03/2020].
- [13] Aprendizaje profundo. (Click para ir a la web). [Fecha de último acceso: 04/04/2020].
- [14] Andrew Ng. Logistic regression. (Click para ir a la web). Vídeo online como parte del curso *Neural Networks and Deep Learning*[Fecha de último acceso: 04/04/2020].
- [15] Andrew Ng. Vectorizing logistic regression. (Click para ir a la web). Vídeo online como parte del curso *Neural Networks and Deep Learning*[Fecha de último acceso: 04/04/2020].
- [16] Andrew Ng. Forward propagation in a deep network. (Click para ir a la web). Vídeo online como parte del curso *Neural Networks and Deep Learning*[Fecha de último acceso: 04/04/2020].
- [17] Andrew Ng. Gradient descent for neural networks. (Click para ir a la web). Vídeo online como parte del curso *Neural Networks and Deep Learning*[Fecha de último acceso: 04/04/2020].

- [18] Andrew Ng. Forward and backward propagation. (Click para ir a la web). Vídeo online como parte del curso *Neural Networks and Deep Learning*[Fecha de último acceso: 04/04/2020].
- [19] Andrew Ng. Mini-batch gradient descent. (Click para ir a la web). Vídeo online como parte del curso *Neural Networks and Deep Learning*[Fecha de último acceso: 04/04/2020].
- [20] Andrew Ng. Softmax regression. (Click para ir a la web). Vídeo online como parte del curso *Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization*[Fecha de último acceso: 07/04/2020].
- [21] Andrew Ng. Normalizing inputs. (Click para ir a la web). Vídeo online como parte del curso *Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization*[Fecha de último acceso: 07/04/2020].
- [22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
- [23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.
- [24] Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. When does label smoothing help? *CoRR*, abs/1906.02629, 2019.
- [25] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.
- [26] Andrew Ng. Computer vision. (Click para ir a la web). Vídeo online como parte del curso *Convolutional Neural Networks*[Fecha de último acceso: 14/04/2020].
- [27] Andrew Ng. Convolutional neural networks. (Click para ir a la web). Curso *Convolutional Neural Networks*[Fecha de último acceso: 14/04/2020].
- [28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.
- [29] Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, volume 2, pages 1045–1048, 01 2010.
- [30] Andrew Ng. Recurrent neural network model. (Click para ir a la web). Vídeo online como parte del curso *Sequence Models*[Fecha de último acceso: 14/04/2020].
- [31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [32] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

- [33] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.
- [34] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.
- [35] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.
- [36] PhysioNet. About. (Click para ir a la web). [Fecha de último acceso: 13/03/2020].
- [37] PhysioNet/Computing in Cardiology. Classification of 12-lead ecgs: the physionet/computing in cardiology challenge 2020. (Click para ir a la web). [Fecha de último acceso: 13/03/2020].
- [38] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.
- [39] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651. JMLR. org, 2017.
- [40] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.
- [41] U Rajendra Acharya, Shu Lih Oh, Yuki Hagiwara, Jen Hong Tan, Muhammad Adam, Arkadiusz Gertych, and Ru San Tan. A deep convolutional neural network model to classify heartbeats. *Computers in biology and medicine*, 89:389–396, 2017.
- [42] Ronald Salloum and C-C Jay Kuo. Ecg-based biometrics using recurrent neural networks. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2062–2066. IEEE, 2017.
- [43] Jen Hong Tan, Yuki Hagiwara, Winnie Pang, Ivy Lim, Shu Lih Oh, Muhammad Adam, Ru San Tan, Ming Chen, and U Rajendra Acharya. Application of stacked convolutional and long short-term memory network for accurate identification of cad ecg signals. *Computers in biology and medicine*, 94:19–26, 2018.
- [44] Shu Lih Oh, Eddie YK Ng, Ru San Tan, and U Rajendra Acharya. Automated diagnosis of arrhythmia using combination of cnn and lstm techniques with variable length heart beats. *Computers in biology and medicine*, 102:278–287, 2018.
- [45] Python. (Click para ir a la web). [Fecha de último acceso: 10/06/2020].
- [46] Tensorflow. (Click para ir a la web). [Fecha de último acceso: 10/06/2020].
- [47] Flask. (Click para ir a la web). [Fecha de último acceso: 10/06/2020].

- [48] React. (Click para ir a la web). [Fecha de último acceso: 10/06/2020].
- [49] Material-ui. (Click para ir a la web). [Fecha de último acceso: 13/06/2020].
- [50] Nivo. (Click para ir a la web). [Fecha de último acceso: 13/06/2020].
- [51] D3.js. (Click para ir a la web). [Fecha de último acceso: 13/06/2020].
- [52] Docker (software). (Click para ir a la web). [Fecha de último acceso: 10/06/2020].
- [53] Nginx. (Click para ir a la web). [Fecha de último acceso: 10/06/2020].
- [54] Alpine linux. (Click para ir a la web). [Fecha de último acceso: 10/06/2020].
- [55] Node.js. (Click para ir a la web). [Fecha de último acceso: 10/06/2020].
- [56] Duckdns. (Click para ir a la web). [Fecha de último acceso: 13/06/2020].
- [57] Google. Environmental report. (Click para ir a la web). [Fecha de último acceso: 09/06/2020].
- [58] Outer Vision. Power supply calculator. (Click para ir a la web). [Fecha de último acceso: 09/06/2020].
- [59] Google. Google cloud pricing calculator. (Click para ir a la web). [Fecha de último acceso: 10/06/2020].

A Impacto y responsabilidades

Todo proyecto que se desee poner en funcionamiento conlleva una serie de impactos directos o indirectos de los que se debe tener constancia. Los profesionales técnicos deben analizar las implicaciones del uso de sus herramientas, pues las consecuencias son, para bien o para mal, su responsabilidad.

A.1 Impacto social

Como se expuso en I, un diagnóstico semi-automatizado permitiría reducir los tiempos dedicados por los cardiólogos a la inspección de electrocardiogramas para su diagnóstico. Este tiempo podría ser utilizado en otras actividades clínicas más complicadas de automatizar. Además, las ECV son la principal causa de muerte a nivel mundial y el grupo de mayor riesgo son las personas mayores en una sociedad cada vez más envejecida.

En un mundo donde las ECV están al alza, esta herramienta podría permitir reducir considerablemente los tiempos de diagnóstico y aumentar la eficiencia. El tiempo en la salud de las personas es fundamental: cuanto más rápido sea el diagnóstico, cuanto más tiempo tenga el cardiólogo para otras actividades clínicas, más vidas se salvarán.

Por ello se considera que un conjunto de pequeños pasos, pequeñas herramientas de telemedicina que permitan mejorar el sistema de salud, son de gran impacto social. Un incremento en la eficiencia de los diagnósticos y del tiempo invertido por los profesionales sanitarios se traduce en un incremento de las vidas salvadas.

A.2 Impacto económico

La herramienta es económicamente viable. La parte más costosa, la del desarrollo, ya está hecha. Es indudable que se necesitarán mejoras y mantenimiento, y que para que sea de utilidad debería ser gestionada por alguien en nómina, estos asuntos se tratarán en el anexo siguiente. Sin embargo, teniendo en cuenta lo fácil que es de escalar y el aumento en productividad que podría suponer para los hospitales, el impacto económico es sin duda positivo.

Así, la inversión en herramientas que permitan aumentar la eficiencia del sistema nacional de salud, es un ahorro económico a medio plazo debido al aumento de la productividad. Este ahorro económico se podrá reconducir a reducir el gasto o a hacer inversiones en mejores equipos o nuevos proyectos, lo que resulta en un balance económico final muy favorable.

A.3 Impacto medioambiental

El impacto medioambiental en este proyecto es casi inexistente. Los equipos en el lado del cliente ya estaban consumiendo electricidad, con el impacto al medioambiente en la producción de energía por métodos contaminantes que esto conlleva. Que los empleados del hospital utilicen una aplicación web no

supone un consumo extra.

Donde sí podría haber un consumo extra de energía, y por lo tanto una producción de la misma por métodos contaminantes es en el lado del servidor. Si el despliegue se realiza como en el trabajo, en una máquina virtual en Google Cloud, entonces el impacto es mínimo: Google ha sido neutral en carbono desde hace una década y ahora compensan el 100% de su consumo energético en la compra de energía limpia [57].

En el caso de contar con un servidor propio, se ha estimado un hardware con: una CPU Intel Core i7-10700K, memoria ram de 4GB DDR4 y con un almacenamiento SSD. Con una herramienta [58] se ha calculado qué fuente de alimentación se debería utilizar. La recomendación han sido 234W, lo que sería un consumo mensual de 67,4 kWh. La producción por vías contaminantes de esta electricidad, será el máximo impacto medioambiental que el servidor de la herramienta podrá ocasionar.

Un último factor medioambiental serán los registros de ECG. Normalmente, el cardiólogo imprime y revisa a mano los ECG. Si el uso de la herramienta reduce la impresión y revisión manual de los mismos, al ser suficiente con la herramienta de ayuda al diagnóstico y la visualización en la web, esto supondrá un menor uso de papel, con el consiguiente impacto en nuestros bosques.

A.4 Responsabilidad ética, legal y profesional

Cuando se desarrollan herramientas en el campo de la salud, se debe de tener mucho cuidado y ser responsable, un error puede poner en peligro la vida de un paciente. Por ello, se debe ser consciente de la fiabilidad de las herramientas desarrolladas y no caer en el error de afirmar que la tecnología está lista para tomar decisiones autónomas en el campo de la salud.

Es por esta razón que desde el inicio se contempló la herramienta como una ayuda al diagnóstico. Es decir, se concibe una herramienta para hacer la vida más fácil al personal sanitario pero no para sustituirlo.

Se cree que muchos trabajos, incluido este, contribuyen a un futuro mucho más ambicioso en el que la tecnología y la medicina se hacen uno, donde los profesionales sanitarios y los ingenieros colaboran para hacer herramientas de predicción y prevención de enfermedades de forma automatizada. Sin embargo, nos encontramos en un presente en el que los profesionales se pueden apoyar en herramientas como esta, pero en ningún caso pueden contar con ellas como única fuente de diagnóstico.

Otro campo al que se ha debido de prestar atención es al de la confidencialidad de los datos. El artículo 5.1 de la Ley Orgánica de Protección de Datos Personales y garantía de los derechos digitales (LOPDGDD) emitida a raíz del Reglamento General de Protección de Datos (GDPR) europeo, establece que:

Los responsables y encargados del tratamiento de datos así como todas las personas que intervengan en cualquier fase de este estarán sujetas al deber

de confidencialidad al que se refiere el artículo 5.1.f) del Reglamento (UE) 2016/679.

Además, los datos clínicos son especialmente sensibles. Estos están protegidos por la Ley 41/2002. En su artículo 7 se indica:

Artículo 7. El derecho a la intimidad.

- 1. Toda persona tiene derecho a que se respete el carácter confidencial de los datos referentes a su salud, y a que nadie pueda acceder a ellos sin previa autorización amparada por la Ley.*
- 2. Los centros sanitarios adoptarán las medidas oportunas para garantizar los derechos a que se refiere el apartado anterior, y elaborarán, cuando proceda, las normas y los procedimientos protocolizados que garanticen el acceso legal a los datos de los pacientes.*

Se han realizado todas las acciones necesarias para cumplir estas reglas de dominio. La aplicación no tiene estado, no almacena datos, por lo tanto no es necesario hacer ningún tratamiento con los mismos. Además, las comunicaciones con el servidor son encriptadas bajo el protocolo https con claves RSA de 2048 bits, evitando ataques de intercepción de la información en tránsito.

B Presupuesto

El presupuesto necesario para cubrir los costes de esta herramienta consta de una inversión inicial y de costes recurrentes necesarios para su mantenimiento en el tiempo. La inversión inicial consiste en la mano de obra que ha sido necesaria para el desarrollo inicial de la aplicación, así como la adquisición de un portátil para su desarrollo. Se ha escogido como equipo de desarrollo un Huawei Matebook X Pro de 13,9 pulgadas.

Tabla 4: Inversión inicial

Coste de la mano de obra (Coste directo)	Horas	Precio/hora	Total	
Ingeniero Técnico de Telecomunicación	400h	15 €	6.000 €	
Coste de recursos materiales (Coste directo)	Precio de compra	Uso en meses	Amortización (en años)	Total
Huawei Matebook X Pro	1085,31 €	4	5	72,35 €
Subtotal mano de obra				6.000 €
Subtotal recursos materiales				72,35 €
Total				6.072,35 €
Gastos generales	15%			910,85 €
Beneficio industrial	6%			364,34 €
Subtotal presupuesto				7.347,55 €
IVA Aplicable				21% 1.542,99 €
Total presupuesto				8.890,53 €

Si se quiere un proyecto extendido en el tiempo: mantenimiento, mejoras incrementales, soporte, etc; se necesitará tener a alguien en nómina cuyo trabajo sea este. Así, los costes recurrentes estarán formados por lo que cuesta mantener la máquina virtual de *Google Cloud Engine* en la que se ha desplegado la herramienta, sumado al salario del personal.

Tabla 5: Gastos recurrentes mensuales

	Coste por hora	Coste mensual
Salario bruto mensual	15 €	1.500 €
Máquina GCP	0,021 €/h	15,21 €
Subtotal		1515,21 €
Subtotal		1515,21 €
IVA Aplicable	21%	318,19 €
Total		1.833,40 €

La máquina que se está utilizando es una tipo *g1-small* con IP pública efímera. El coste efectivo por hora ronda los 0,021€. El coste estimado mensual se ha estimado con una calculadora de costes ofertada por Google Cloud Platform (GCP) [59]. Para el salario del desarrollador, supóngase en un escenario realista que se le contrata a jornada parcial, con 25 horas semanales.

C Búsqueda de un modelo convergente basado en GANs

A continuación, se tratará de hacer un resumen del proceso seguido para la obtención del modelo utilizado para generación de datos mediante GANs. Muchos de los modelos probados no se conservan. Por ello, lo que se muestra a continuación es una reconstrucción temporal aproximada que no representa el total del trabajo realizado. Se comenzó a guardar resultados después de varias semanas de desarrollo.

Durante un largo periodo de tiempo, se trató de encontrar un modelo generativo que tomase como entrada una señal de 3000 muestras. Mas tarde, se cayó en la cuenta de que para ello se hubiera necesitado un modelo mucho mas grande. Además, no se entrenaba con clases condicionadas. Por todo ello, las configuraciones que se muestran a continuación fueron un completo fracaso.

Tabla 6: Discriminador del modelo entrenado el 22/03/2020

Capa	Dimension de salida	Parámetros
conv1d_176 (Conv1D)	(None, 3000, 32)	128
leaky_re_lu_57 (LeakyReLU)	(None, 3000, 32)	0
dropout_57 (Dropout)	(None, 3000, 32)	0
batch_normalization_196 (Batch Normalization)	(None, 3000, 32)	128
conv1d_177 (Conv1D)	(None, 1500, 64)	6208
leaky_re_lu_58 (LeakyReLU)	(None, 1500, 64)	0
dropout_58 (Dropout)	(None, 1500, 64)	0
batch_normalization_197 (Batch Normalization)	(None, 1500, 64)	256
conv1d_178 (Conv1D)	(None, 750, 128)	24704
leaky_re_lu_59 (LeakyReLU)	(None, 750, 128)	0
dropout_59 (Dropout)	(None, 750, 128)	0
batch_normalization_198 (Batch Normalization)	(None, 750, 128)	512
conv1d_179 (Conv1D)	(None, 750, 256)	98560
leaky_re_lu_60 (LeakyReLU)	(None, 750, 256)	0
dropout_60 (Dropout)	(None, 750, 256)	0
flatten_14 (Flatten)	(None, 192000)	0
Parámetros totales		130.496
Parámetros entrenables		130.048
Parámetros no entrenables		448

Tabla 7: Generador del modelo entrenado el 22/03/2020

Capa	Dimension de salida	Parámetros
dense_28 (Dense)	(None, 24064)	4836864
reshape_25 (Reshape)	(None, 188, 128)	0
batch_normalization_121 (Batch Normalization)	(None, 188, 128)	512
up_sampling1d_74 (UpSampling)	(None, 376, 128)	0
cropping1d_5 (Cropping1D)	(None, 375, 128)	0
conv1d_100 (Conv1D)	(None, 375, 128)	49280
activation_108 (Activation)	(None, 375, 128)	0
batch_normalization_122 (Batch Normalization)	(None, 375, 128)	512
up_sampling1d_75 (UpSampling)	(None, 750, 128)	0
conv1d_101 (Conv1D)	(None, 750, 128)	49280
activation_109 (Activation)	(None, 750, 128)	0
batch_normalization_123 (Batch Normalization)	(None, 750, 128)	512
up_sampling1d_76 (UpSampling)	(None, 1500, 128)	0
conv1d_102 (Conv1D)	(None, 1500, 64)	24640
activation_110 (Activation)	(None, 1500, 64)	0
batch_normalization_124 (Batch Normalization)	(None, 1500, 64)	256
up_sampling1d_77 (UpSampling)	(None, 3000, 64)	0
conv1d_103 (Conv1D)	(None, 3000, 1)	193
activation_111 (Activation)	(None, 3000, 1)	0
batch_normalization_125 (Batch Normalization)	(None, 3000, 1)	4
activation_112 (Activation)	(None, 3000, 1)	0
Parámetros totales		4.962.053
Parámetros entrenables		4.961.155
Parámetros no entrenables		898

Como se puede observar, se trataba de modelos puramente convolucionales, como el que al final se acabó utilizando. A continuación, se muestra la imagen con la que se iba comprobando la evolución del modelo. Esta muestra cómo la red neuronal simplemente es incapaz de aprender a generar lo que se está buscando.

El discriminador detecta sin problemas que las señales generadas son falsas prácticamente desde el principio. No corre la misma suerte con las reales, que es incapaz de discernir que lo son. Esto parece indicar que el discriminador

simplemente divergía, su salida era siempre 0. La muestra generada está muy alejada de un ECG.

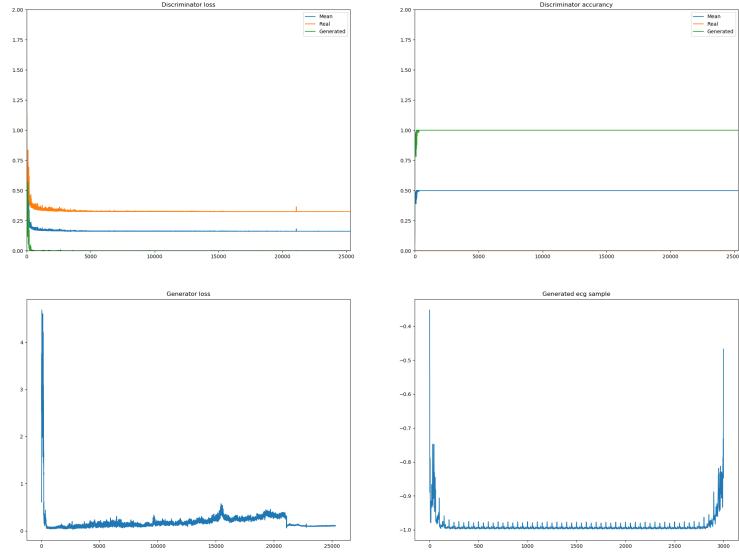


Figura 44: Gráficas de monitorización del modelo entrenado el 22/03/2020

La arquitectura sin embargo no anda del todo desencaminada, es púramente convolucional tanto en el discriminador como en el generador, como la que al final se acabó utilizando. Estaba basada en una arquitectura de un modelo de cGAN. Pero a estas alturas del proceso aún no se había investigado tanto como se haría mas adelante.

Al día siguiente se probó algo distinto: un modelo secuencial en el generador y uno convolucional en el discriminador. El discriminador también diverge, aunque al menos aquí las funciones de coste oscilan, lo que era una buena señal. El resultado acabó siendo igual de decepcionante.

Tabla 8: Generador del modelo entrenado el 23/03/2020

Capa	Dimension de salida	Parámetros
bidirectional (Bidirectional)	(None, 60, 100)	40400
bidirectional_1 (Bidirection	(None, 60, 50)	60400
activation (Activation)	(None, 60, 50)	0
dropout (Dropout)	(None, 60, 50)	0
time_distributed (TimeDistri	(None, 60, 50)	2550
activation_1 (Activation)	(None, 60, 50)	0
reshape (Reshape)	(None, 3000, 1)	0
Parámetros totales		103.350
Parámetros entrenables		103.350
Parámetros no entrenables		0

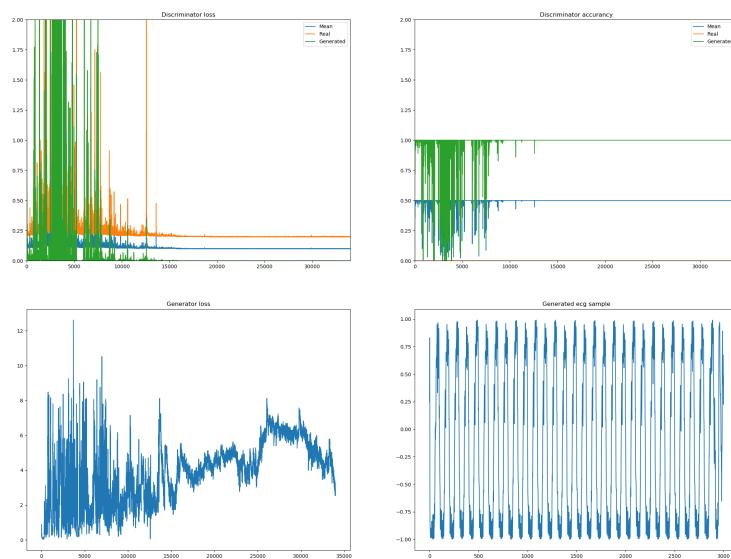


Figura 45: Gráficas de monitorización del modelo entrenado el 23/03/2020

Tabla 9: Discriminador del modelo entrenado el 23/03/2020

Capa	Dimension de salida	Parámetros
conv1d_8 (Conv1D)	(None, 3000, 32)	128
max_pooling1d_8 (MaxPooling1	(None, 1500, 32)	0
batch_normalization_8 (Batch	(None, 1500, 32)	128
leaky_re_lu_8 (LeakyReLU)	(None, 1500, 32)	0
dropout_10 (Dropout)	(None, 1500, 32)	0
conv1d_9 (Conv1D)	(None, 1500, 64)	6208
max_pooling1d_9 (MaxPooling1	(None, 750, 64)	0
batch_normalization_9 (Batch	(None, 750, 64)	256
leaky_re_lu_9 (LeakyReLU)	(None, 750, 64)	0
dropout_11 (Dropout)	(None, 750, 64)	0
conv1d_10 (Conv1D)	(None, 750, 128)	24704
max_pooling1d_10 (MaxPooling	(None, 375, 128)	0
batch_normalization_10 (Batc	(None, 375, 128)	512
leaky_re_lu_10 (LeakyReLU)	(None, 375, 128)	0
dropout_12 (Dropout)	(None, 375, 128)	0
conv1d_11 (Conv1D)	(None, 375, 256)	98560
max_pooling1d_11 (MaxPooling	(None, 187, 256)	0
batch_normalization_11 (Batc	(None, 187, 256)	1024
leaky_re_lu_11 (LeakyReLU)	(None, 187, 256)	0
dropout_13 (Dropout)	(None, 187, 256)	0
flatten_2 (Flatten)	(None, 47872)	0
Parámetros totales		131.520
Parámetros entrenables		130.560
Parámetros no entrenables		960

Se comenzó entrenando los modelos en una máquina virtual en Google Cloud, más tarde se acabaron los créditos y se migró todo a Google Colab. Tras una semana de refactorización del código y cambio de plataforma, se trató un modelo muy complejo que combinaba LSTM con CNN distribuidas en el tiempo. Este intento fue un absoluto fracaso, no vale la pena profundizar mucho más en él, pero se muestra el generador a modo de ilustración.

Tabla 10: Generador del modelo entrenado el 01/04/2020

Capa	Dimension de salida	Parámetros
bidirectional (Bidirectional)	(None, 60, 120)	125280
bidirectional_1 (Bidirection	(None, 60)	86880
activation (Activation)	(None, 60)	0
dropout (Dropout)	(None, 60)	0
reshape (Reshape)	(None, 60, 1, 1)	0
time_distributed (TimeDistri	(None, 60, 3, 1)	0
time_distributed_1 (TimeDist	(None, 60, 3, 256)	1024
time_distributed_2 (TimeDist	(None, 60, 3, 256)	1024
activation_1 (Activation)	(None, 60, 3, 256)	0
dropout_1 (Dropout)	(None, 60, 3, 256)	0
time_distributed_3 (TimeDist	(None, 60, 6, 256)	0
time_distributed_4 (TimeDist	(None, 60, 6, 128)	98432
time_distributed_5 (TimeDist	(None, 60, 6, 128)	512
activation_2 (Activation)	(None, 60, 6, 128)	0
dropout_2 (Dropout)	(None, 60, 6, 128)	0
time_distributed_6 (TimeDist	(None, 60, 12, 128)	0
time_distributed_7 (TimeDist	(None, 60, 12, 64)	24640
time_distributed_8 (TimeDist	(None, 60, 12, 64)	256
activation_3 (Activation)	(None, 60, 12, 64)	0
dropout_3 (Dropout)	(None, 60, 12, 64)	0
time_distributed_9 (TimeDist	(None, 60, 24, 64)	0
time_distributed_10 (TimeDis	(None, 60, 24, 32)	6176
time_distributed_11 (TimeDis	(None, 60, 24, 32)	128
activation_4 (Activation)	(None, 60, 24, 32)	0
dropout_4 (Dropout)	(None, 60, 24, 32)	0
time_distributed_12 (TimeDis	(None, 60, 25, 32)	0
time_distributed_13 (TimeDis	(None, 60, 50, 32)	0
time_distributed_14 (TimeDis	(None, 60, 50, 1)	97
time_distributed_15 (TimeDis	(None, 60, 50, 1)	4
activation_5 (Activation)	(None, 60, 50, 1)	0
dropout_5 (Dropout)	(None, 60, 50, 1)	0
reshape_1 (Reshape)	(None, 3000)	0
scale_layer (ScaleLayer)	(None, 3000)	0
reshape_2 (Reshape)	(None, 3000, 1)	0
Parámetros totales		344.453
Parámetros entrenables		343.491
Parámetros no entrenables		962

El siguiente modelo que merece la pena mencionar combinaba una modificación del generador anterior, con más capas secuenciales, y un modelo convolucional en el discriminador.

Tabla 11: Discriminador del modelo entrenado el 05/04/2020

Capa	Dimension de salida	Parámetros
conv1d_5 (Conv1D)	(None, 1500, 4)	16
leaky_re_lu (LeakyReLU)	(None, 1500, 4)	0
dropout (Dropout)	(None, 1500, 4)	0
conv1d_6 (Conv1D)	(None, 375, 16)	336
batch_normalization_8 (Batch	(None, 375, 16)	64
leaky_re_lu_1 (LeakyReLU)	(None, 375, 16)	0
dropout_1 (Dropout)	(None, 375, 16)	0
conv1d_7 (Conv1D)	(None, 94, 32)	2592
batch_normalization_9 (Batch	(None, 94, 32)	128
leaky_re_lu_2 (LeakyReLU)	(None, 94, 32)	0
dropout_2 (Dropout)	(None, 94, 32)	0
conv1d_8 (Conv1D)	(None, 16, 64)	14400
batch_normalization_10 (Batch	(None, 16, 64)	256
leaky_re_lu_3 (LeakyReLU)	(None, 16, 64)	0
dropout_3 (Dropout)	(None, 16, 64)	0
conv1d_9 (Conv1D)	(None, 4, 128)	41088
batch_normalization_11 (Batch	(None, 4, 128)	512
leaky_re_lu_4 (LeakyReLU)	(None, 4, 128)	0
dropout_4 (Dropout)	(None, 4, 128)	0
conv1d_10 (Conv1D)	(None, 1, 256)	164096
batch_normalization_12 (Batch	(None, 1, 256)	1024
leaky_re_lu_5 (LeakyReLU)	(None, 1, 256)	0
dropout_5 (Dropout)	(None, 1, 256)	0
flatten_1 (Flatten)	(None, 256)	0
dense (Dense)	(None, 60)	15420
Parámetros totales		239.932
Parámetros entrenables		238.940
Parámetros no entrenables		992

Tabla 12: Generador del modelo entrenado el 05/04/2020

Capa	Dimension de salida	Parámetros
bidirectional (Bidirectional)	(None, 60, 120)	90720
bidirectional_1 (Bidirection	(None, 60, 120)	86880
batch_normalization (BatchNo	(None, 60, 120)	480
bidirectional_2 (Bidirection	(None, 60, 120)	86880
batch_normalization_1 (Batch	(None, 60, 120)	480
bidirectional_3 (Bidirection	(None, 60)	86880
batch_normalization_2 (Batch	(None, 60)	240
activation (Activation)	(None, 60)	0
reshape (Reshape)	(None, 60, 1, 1)	0
time_distributed (TimeDistri	(None, 60, 2, 1)	0
time_distributed_1 (TimeDist	(None, 60, 2, 64)	256
time_distributed_2 (TimeDist	(None, 60, 2, 64)	256
activation_1 (Activation)	(None, 60, 2, 64)	0
time_distributed_3 (TimeDist	(None, 60, 6, 64)	0
time_distributed_4 (TimeDist	(None, 60, 6, 32)	6176
time_distributed_5 (TimeDist	(None, 60, 6, 32)	128
activation_2 (Activation)	(None, 60, 6, 32)	0
time_distributed_6 (TimeDist	(None, 60, 12, 32)	0
time_distributed_7 (TimeDist	(None, 60, 12, 16)	1552
time_distributed_8 (TimeDist	(None, 60, 12, 16)	64
activation_3 (Activation)	(None, 60, 12, 16)	0
time_distributed_9 (TimeDist	(None, 60, 24, 16)	0
time_distributed_10 (TimeDis	(None, 60, 24, 4)	196
time_distributed_11 (TimeDis	(None, 60, 24, 4)	16
activation_4 (Activation)	(None, 60, 24, 4)	0
time_distributed_12 (TimeDis	(None, 60, 25, 4)	0
time_distributed_13 (TimeDis	(None, 60, 50, 4)	0
time_distributed_14 (TimeDis	(None, 60, 50, 1)	13
time_distributed_15 (TimeDis	(None, 60, 50, 1)	4
activation_5 (Activation)	(None, 60, 50, 1)	0
time_distributed_16 (TimeDis	(None, 60, 50)	0
scale_layer (ScaleLayer)	(None, 60, 50)	0
reshape_1 (Reshape)	(None, 3000, 1)	0
Parámetros totales		361.221
Parámetros entrenables		360.387
Parámetros no entrenables		834

Este modelo merece una mención especial, porque fue uno de los primeros en el que el discriminador no divergía (se habían hecho pruebas muy similares que no vale la pena mencionar previas al modelo del día 01/04/2020). También se puede observar a continuación esa *lucha* entre generador y discriminador en la función de costes, al menos al principio. Sin embargo, el generador resulta incapaz de crear señales de electrocardiogramas de longitud 3000.

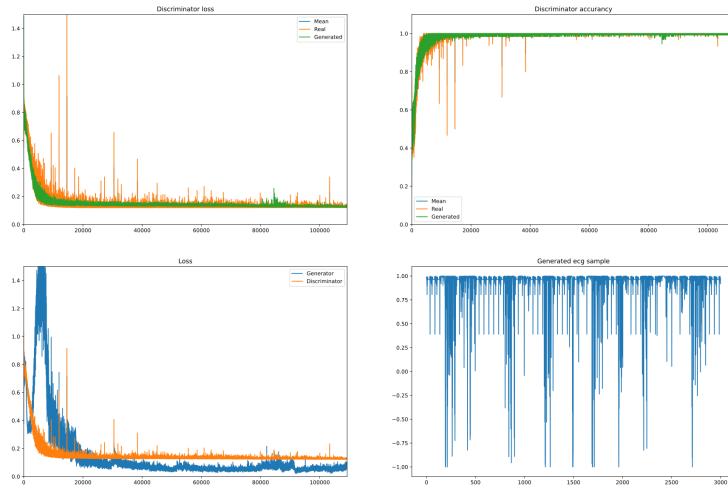


Figura 46: Gráficas de monitorización del modelo entrenado el 05/04/2020

Hasta este momento solo se habían usado funciones de pérdidas clásicas, del paper original. Después del fracaso de este modelo se produjo un gran salto teórico. Se investigó mucho mas de lo que se había investigado sobre las diferentes arquitecturas posibles.

Se implementó una GAN de Wasserstein (WGAN) y se realizaron pruebas, que resultaron no satisfactorias. La función de costes funciona bien en determinados contextos, pero en este divergía completamente. Fue en estos días cuando también se cayó en la cuenta de que se podía hacer un downsampling de la señal sin apenas perder calidad. Lamentablemente, no se consevaban ficheros de estos días. El primer modelo que mostró resultados esperanzadores se entrenó el 10 de abril.

Tabla 13: Generador del modelo entrenado el 10/04/2020

Capa	Dimension de salida	Parámetros
dense (Dense)	(None, 2048)	264192
activation (Activation)	(None, 2048)	0
reshape (Reshape)	(None, 32, 64)	0
up_sampling1d (UpSampling1D)	(None, 64, 64)	0
conv1d (Conv1D)	(None, 64, 64)	12352
batch_normalization (BatchNo	(None, 64, 64)	256
activation_1 (Activation)	(None, 64, 64)	0
dropout (Dropout)	(None, 64, 64)	0
up_sampling1d_1 (UpSampling1	(None, 128, 64)	0
conv1d_1 (Conv1D)	(None, 128, 32)	6176
batch_normalization_1 (Batch	(None, 128, 32)	128
activation_2 (Activation)	(None, 128, 32)	0
up_sampling1d_2 (UpSampling1	(None, 256, 32)	0
conv1d_2 (Conv1D)	(None, 256, 1)	97
activation_3 (Activation)	(None, 256, 1)	0
Parámetros totales		283.201
Parámetros entrenables		283.009
Parámetros no entrenables		192

Para este modelo, se utilizaron los datos de entrenamiento donde la longitud de las señales era de tamaño 256. Además, se implementó una DCGAN, siguiendo al pie de la letra las directrices de [40]. La arquitectura del discriminador y el generador se redujeron a su mínima expresión, volviendo a ser modelos convolucionales.

Tabla 14: Discriminador del modelo entrenado el 10/04/2020

Capa	Dimension de salida	Parámetros
conv1d_3 (Conv1D)	(None, 128, 16)	64
leaky_re_lu (LeakyReLU)	(None, 128, 16)	0
dropout_1 (Dropout)	(None, 128, 16)	0
conv1d_4 (Conv1D)	(None, 64, 32)	1568
leaky_re_lu_1 (LeakyReLU)	(None, 64, 32)	0
dropout_2 (Dropout)	(None, 64, 32)	0
conv1d_5 (Conv1D)	(None, 32, 64)	6208
leaky_re_lu_2 (LeakyReLU)	(None, 32, 64)	0
dropout_3 (Dropout)	(None, 32, 64)	0
conv1d_6 (Conv1D)	(None, 16, 128)	24704
leaky_re_lu_3 (LeakyReLU)	(None, 16, 128)	0
dropout_4 (Dropout)	(None, 16, 128)	0
flatten (Flatten)	(None, 2048)	0
dense_1 (Dense)	(None, 1)	2049
Parámetros totales		34.593
Parámetros entrenables		34.593
Parámetros no entrenables		0

A continuación se muestra la evolución de este modelo. Las funciones de pérdidas son completamente oscilantes, que es lo que se espera de una GAN. Además, se puede observar el ECG generado, que ya se comienza a parecer a la señal de un electrocardiograma.

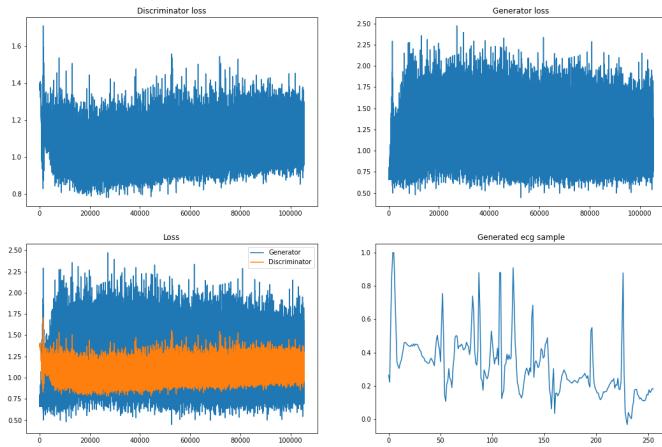


Figura 47: Gráficas de monitorización del modelo entrenado el 10/04/2020

El día siguiente se encontró un bug en el código que evitaba que este último modelo se entrenara con todo el dataset. Al arreglarlo el resultado era también positivo.

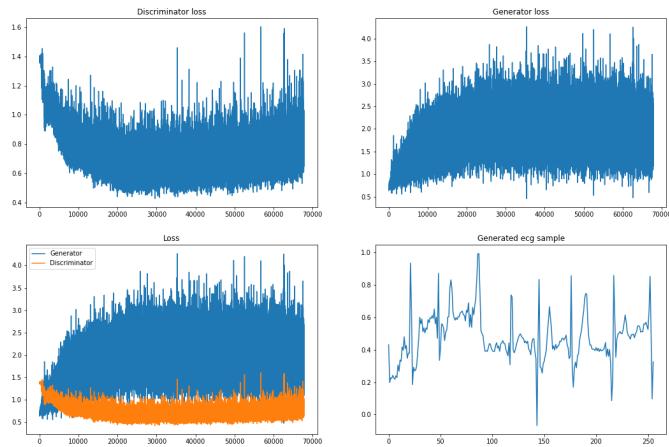


Figura 48: Gráficas de monitorización del modelo entrenado el 11/04/2020

Se puede apreciar en la señal generada cómo el generador, sin tener ninguna información del conjunto de datos, ha aprendido que debe generar un vector que represente una señal periódica, con picos y valles. Aunque ya son públicos resultados mucho mejores en generación de imágenes, resulta fascinante ver cómo

esto es posible.

A partir de este resultado se decidió que era necesario generar por clase de enfermedad para que el modelo generativo tuviese sentido, al fin y al cabo el objetivo de todo esto era la ampliación de datos por tipo de enfermedad. Es entonces cuando se comenzaron las pruebas con AC-GAN. La arquitectura convolucional, que había dado tan buen resultado, se mantuvo, añadiendo los elementos necesarios para convertirla en una condicional con clasificador auxiliar. El resultado final es el que se mostraba en la sección 13.1.2.

Al final, un tratamiento adecuado de los datos, una función de costes coherente con lo que se pretendía obtener, unos hiperparámetros ajustados a lo que ya había funcionado en otros contextos y una cantidad de parámetros comedida, habían sido las claves para la resolución del problema. Con toda esta adquisición de conocimiento, ahora se partiría de una posición mucho más favorable a la hora de desarrollar un proyecto basado en GAN.

D Búsqueda de un modelo clasificador

Al contrario que para el apéndice anterior, para ilustrar el proceso iterativo para conseguir el modelo clasificador final, no se conservan los ficheros de los clasificadores probados. Sin embargo, este desarrollo es mucho más reciente. Se comentará de la forma más precisa posible el proceso que llevó a obtener el modelo clasificador utilizado en la herramienta.

Se comenzó con un modelo clasificador púramente convolucional, parecido al discriminador utilizado en la GAN. El resultado no fue muy bueno. Tampoco lo fue el uso de un modelo exclusivamente secuencial. Los primeros resultados positivos se obtuvieron con un clasificador que combinaba un modelo secuencial con un modelo convolucional. La arquitectura se muestra en la siguiente tabla.

Tabla 15: Clasificador entrenado el 24/04/2020

Capa	Dimension de salida	Parámetros
reshape_2 (Reshape)	(None, 16, 16)	0
lstm (LSTM)	(None, 16, 16)	2112
lstm_1 (LSTM)	(None, 16, 16)	2112
reshape_3 (Reshape)	(None, 256, 1)	0
conv1d_7 (Conv1D)	(None, 256, 16)	64
batch_normalization_2 (Batch)	(None, 256, 16)	64
activation_5 (Activation)	(None, 256, 16)	0
max_pooling1d (MaxPooling1D)	(None, 128, 16)	0
dropout_5 (Dropout)	(None, 128, 16)	0
conv1d_8 (Conv1D)	(None, 128, 32)	1568
batch_normalization_3 (Batch)	(None, 128, 32)	128
activation_6 (Activation)	(None, 128, 32)	0
max_pooling1d_1 (MaxPooling1	(None, 64, 32)	0
dropout_6 (Dropout)	(None, 64, 32)	0
conv1d_9 (Conv1D)	(None, 64, 64)	6208
batch_normalization_4 (Batch)	(None, 64, 64)	256
activation_7 (Activation)	(None, 64, 64)	0
max_pooling1d_2 (MaxPooling1	(None, 32, 64)	0
dropout_7 (Dropout)	(None, 32, 64)	0
conv1d_10 (Conv1D)	(None, 32, 128)	24704
batch_normalization_5 (Batch)	(None, 32, 128)	512
activation_8 (Activation)	(None, 32, 128)	0
max_pooling1d_3 (MaxPooling1	(None, 16, 128)	0
dropout_8 (Dropout)	(None, 16, 128)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_4 (Dense)	(None, 512)	1049088
batch_normalization_6 (Batch)	(None, 512)	2048
activation_9 (Activation)	(None, 512)	0
dense_5 (Dense)	(None, 9)	4617
Parámetros totales		1.093.481
Parámetros entrenables		1.091.977
Parámetros no entrenables		1.504

Sin embargo, esta arquitectura quedaba lejos de los objetivos planteados para la herramienta. Se entrenó con el conjunto de datos generado por transformaciones durante 200 épocas y con un tamaño de lote de 64 muestras. Los hiperparámetros y el optimizador utilizados fueron los mismos que para el modelo final. La precisión obtenida en el conjunto de validación no llegaba al 0,6.

Es entonces cuando se decidió incorporar los datos cualitativos: la derivación, el sexo y la edad. Tenía sentido, sobre todo dar información de la derivación: una señal cambia mucho según la derivación que se esté visualizando. Así, se llegó a una arquitectura muy parecida a la final, con una ligera diferencia en el bloque final de redes neuronales clásicas. La arquitectura de esta parte se muestra a continuación.

Tabla 16: Bloque de Fully Connected Layers del modelo clasificador entrenado el 08/05/2020

Capa	Dimension de salida	Parámetros
concatenate (Concatenate)	(None, 2240)	0
dense (Dense)	(None, 512)	1147392
batch_normalization_4 (Batch	(None, 512)	2048
activation_4 (Activation)	(None, 512)	0
dense_1 (Dense)	(None, 64)	32832
batch_normalization_5 (Batch	(None, 64)	256
activation_5 (Activation)	(None, 64)	0
dense_2 (Dense)	(None, 9)	585
Parámetros totales		1.183.113
Parámetros entrenables		1.181.961
Parámetros no entrenables		1.152

Con esta incorporación, la mejora fue substancial. Este clasificador tenía una precisión de más del 70% para casi todas las clases. La matriz de confusión se muestra en la siguiente página.

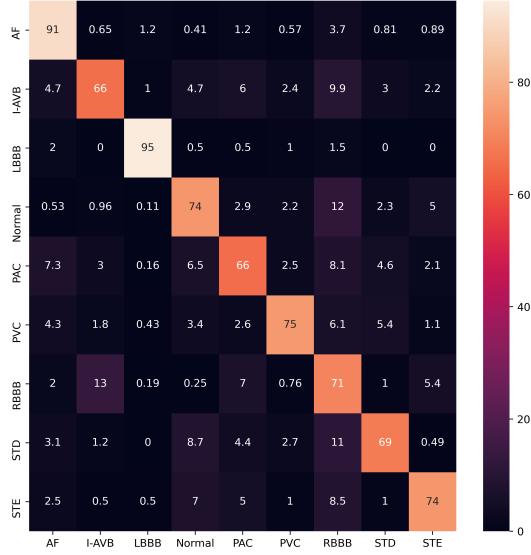


Figura 49: Matriz de confusión del modelo clasificador del 08/05/2020 con el conjunto de datos ampliado con transformaciones y con filtro de sesgos.

Se consideraba que esta arquitectura era la adecuada, así que se procedió al tuneado de hiperparámetros que permitiese mejorar el modelo final. Este proceso se centró en cuatro variables: el ratio de entrenamiento, el momentum de la normalización por lotes (*batch normalization*), el porcentaje de dropout y el número de capas ocultas del bloque de *fully connected layers*. Las combinaciones realizadas se muestran a continuación, en negrita la combinación final escogida.

Tabla 17: Tuneado de hiperparámetros, combinaciones probadas.

α	Momentum	Dropout	Número de capas
0,0001	0,8	25%	3
0,00005	0,8	25%	3
0,005	0,8	25%	3
0,001	0,8	25%	3
0,001	0,6	30%	3
0,001	0,9	50%	3
0,001	0,8	15%	3
0,001	0,8	25%	4

Tanto el momentum como el porcentaje de *dropout*, ya se encontraban en valores razonables. Aumentar por 10 el ratio de entrenamiento supuso una pequeña mejora. La mejora producto de añadir una capa de 1024 al bloque final, fue la mas notable de todas. Este modelo, es el modelo final. Es el que sería entrenado con la distribución de datos aumentada mediante GANs y mediante transformaciones para luego ser combinados y usados en la herramienta final de ayuda al diagnóstico. Los resultados pueden revisarse en la sección 15.1.

E Manual de despliegue

El objetivo de este anexo es facilitar el despliegue de la herramienta en otras máquinas a la persona que reciba este proyecto como herencia. A continuación se darán una serie de claves necesarias para hacerlo sin incidentes.

E.1 Obtención de un certificado

Para obtener el certificado https, se recomienda utilizar la herramienta certbot. A continuación se detallan los pasos para Debian 10 (buster):

1. Asegurarse de que no hay ningún contenedor activo
2. En la máquina servidora, añadir el nombre del dominio en /etc/hosts con una línea nueva de la siguiente manera:
127.0.0.1 <nombre del dominio sin "http://" >
3. Asegurarse también de tener abiertos los puertos 80 y 443.
4. Instalar NGINX en la máquina servidora
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install nginx
5. Debería estar NGINX ya disponible en el puerto 80. Si no lo está, comprueba su status con:
sudo systemctl status nginx
Enciende el servicio con:
sudo systemctl start nginx
6. Seguir los pasos de instalación del certificado que se detallan en [certbot](#) seleccionando como software NGINX y como sistema, el correspondiente a la máquina servidora. Seleccionar el nombre de dominio correcto durante el proceso de instalación en la cli. Permitir a certbot sobreescribir la configuración de NGINX
7. Deshabilitar NGINX
sudo systemctl stop nginx
sudo systemctl disable nginx
8. En el *docker-compose.yml* se añade como volumen la carpeta donde está alojado el certificado en la máquina servidora bajo la etiqueta *volume*, para que así el contenedor proxy pueda utilizarlo. Asegurarse de que el archivo *default.conf* que se copia en el contenedor del proxy apunta al archivo creado con el certificado. Se puede comparar la parte de ficheros utilizados para el certificado con la configuración local que hay en */etc/nginx/sites-enabled/default*

E.2 Cambio de nombre del dominio

Si se decide cambiar el nombre del dominio, se deberán hacer una serie de modificaciones en los ficheros para que todo siga funcionando con normalidad:

1. Será necesario la obtención de un nuevo certificado tal y como se indica en el apartado anterior.
2. En el *docker-compose.yml* se enlaza el localhost del proxy con el nombre del dominio con la etiqueta *extra_hosts*. Esta habrá que cambiarla si se cambia el nombre de dominio.
3. El archivo de configuración del proxy *default.conf* también hace referencia a este nombre de dominio, habrá que cambiar el nombre de dominio actual por el que se desea usar.
4. En *physionet-challenge-2020/webtool/frontend/src/App.js* de la aplicación React, se hace la llamada al clasificador. Habrá que cambiar el nombre del dominio a quien hace la llamada para que siga funcionando con normalidad, concretamente en la línea 31.
5. En la aplicación de *Flask*, se habilita la comunicación entre máquinas (protocolo *CORS*) desde el nombre de dominio antiguo. Así, en el fichero *physionet-challenge-2020/webtool/apiserver/app.py* habrá que modificar en la línea 14 el nombre de dominio para que permita las llamadas desde el nuevo.

E.3 Puesta en marcha

Para la puesta en marcha, se debe instalar Docker y docker-compose. Todos los ficheros funcionan para la versión 19.03.11 de Docker y la versión de docker-compose 1.23.1. Las instrucciones de instalación se pueden encontrar en internet para el sistema concreto de forma fácil. Asegúrese también de tener abiertos los puertos 80 y 443.

Una vez instaladas ambas herramientas, solo será necesario correr el script *run_servers.sh* que se encuentra en *physionet-challenge-2020/webtool*:

```
cd physionet-challenge-2020/webtool
./run_servers.sh
```

Puede haber problemas si Docker se ha instalado como usuario normal y no super-usuario. Si da un error de esta naturaleza, se tendrá que eliminar los *sudo* de las llamadas que hace el script. Si se realizan cambios en los ficheros, siempre se puede volver a hacer una llamada a *run_servers.sh* para aplicar los cambios en los contenedores desplegados.