UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS
"MAT. LUIS MANUEL RIVERA GUTIÉRREZ"

CLASIFICACIÓN DE ELECTROCARDIOGRAMAS CON MÉTODOS DE INTELIGENCIA ARTIFICIAL

T E S I S

QUE PARA OBTENER EL GRADO DE:

LICENCIADO EN CIENCIAS FÍSICO MATEMÁTICAS

PRESENTA:

FABRIZIO AARÓN RIVERA SÁNCHEZ

DIRECTOR DE TESIS:

DR. JOSÉ ANTONIO GONZÁLEZ CERVERA

MORELIA, MICHOACÁN. OCTUBRE DEL 2018

A mi familia.

AGRADECIMIENTOS

Quiero agradecer a cada una de las personas que me enseñaron algo a lo largo de mi carrera. A mi asesor, José Antonio González Cervera, por su gran paciencia y apoyo en todo momento. A los profesores de la Facultad de Ciencias Físico Matemáticas, por cambiar mi visión del mundo. A toda mi familia, en especial a mis papás, Fabrizio Rivera Alcántar y Mariana Sánchez Hernández. Agradezco a todos mis amigos, que también me brindaron su apoyo en todo momento.

Morelia, Michoacán., Octubre del 2018

Índice general

Resumen

Se presentan los resultados del desempeño de los diferentes métodos de clasificación sobre los datos de Electrocardiogramas (ECG) obtenidos del CinC challenge 2017 de la fuente physionet. Los métodos implementados comprenden las redes neuronales artificiales (RNA), redes neuronales convolucionales (RNC) y máquinas de soporte vectorial (MSV). Se obtienen resultados satisfactorios, sobre la base de datos del CinC Challenge 2017, con precisión del 94.33 %, 99.0 % y 100 % al implementar cada uno de los métodos mencionados anteriormente. Para llevar a cabo la clasificación en dos de los métodos, se realizó la transformación de las series de tiempo en imágenes por medio de la matriz GASF (Gramian Angular Summation Field), abriendo la posibilidad de un nuevo enfoque para la solución al difícil problema de la detección de enfermedades del corazón bajo el análisis de ECGs, siendo la red convolucional la solución más prometedora.

Palabras Clave— Aprendizaje de Máquina, Redes Neuronales Artificiales, Máquinas de Soporte Vectorial, Redes Neuronales Convolucionales, Matriz Gramiana.

Abstract

We present the performance of different classifiers using ECG time series, extracted from the physionet CinC challenge 2017 database. The algorithms to classify the ECG recordings comprise of Feed Forward Neural Networks, Convolutional Neural Network and Support Vector Machines. The results on the classification of the ECG database are satisfying, obtaining an overall accuracy of 94.33 %, 99.0 % and 100 % for each one of the algorithms mentioned above. In order to classify the ECG recordings in two of the algorithms a transformation to images of the signals was made, namely, the GASF (Gramian Angular Summation Field), providing with a new approach to solve the difficult problem of the detection of heart diseases by analizing ECGs, with the CNN being one of the best algorithms.

Capítulo 1

Introducción

1.1. Electrocardiogramas

El electrocardiograma es una grabación de la actividad eléctrica del corazón, dicho de otra manera, es una señal eléctrica que varía con el tiempo. Fue introducido en el área clínica hace más de 100 años por Einthoven [1] y desde entonces ha sido de gran utilidad, al crear una nueva ventana de observación para el estudio del corazón. Esta técnica tiene una gran ventaja, ya que es un método no invasivo y rico de información cardiaca. La morfología de la señal varía de acuerdo a la posición de los electrodos lectores, sin embargo, siempre hay una similitud en dichas señales. Un pulso completo está conformado por un latido y tiene una forma bien conocida.

La técnica de la electrocardiografía se utiliza para el estudio y diagnóstico de pacientes con dolor en el pecho, arritmias, entre otros [1]. A su vez, es importante su uso para dar seguimiento a los tratamientos implementados para el control o cura de padecimientos implicados.

Desafortunadamente, no siempre resulta en un diagnóstico acertado, también se han registrado casos en los que las enfermedades varían de acuerdo a los síntomas y se observa una señal ECG completamente normal, incluso con algún padecimiento peligroso presente. Más aún, es posible tener un ECG normal durante meses o años después de un infarto de miocardio [1], por ejemplo, conocido coloquialmente como paro cardiaco. Esto puede deberse a una mala interpretación de las señales realizada por los médicos o es posible que se tengan patrones en las señales que resultan invisibles a los ojos de los médicos.

Para ilustrar la importancia del diagnóstico de enfermedades cardiacas, analicemos la enfermedad de la Fibrilación Auricular (FA o AF por sus siglas en inglés), la cual es la arritmia más frecuente en la práctica clínica, caracterizada por latidos más rápidos e irregulares y la cual, es detectada mayoritariamente en personas de edad avanzada. Aproximadamente el 1 % de las personas con FA son menores a 60 años, mientras que el 12 % se encuentran entre los 75 y 84 [2] años y un 33 % corresponde a mayores de 80 años [3, 4].

En la FA intervienen varios factores y su estudio en clínicas resulta complejo, con un rango desde cero síntomas, hasta síntomas severos. La FA incrementa en cinco veces el riesgo de sufrir un infarto [5], el cual incrementa aún más con la edad [6]. También el riesgo de sufrir falla del corazón (enfermedad crónica donde el corazón no bombea sangre de forma eficiente) se triplica [7–9]. Por si fuera poco, duplica el riesgo en demencia [10] y mortalidad [5].

Los tratamientos para la enfermedad de FA han evolucionado al igual que los de otras enfermedades, desde el uso de fármacos antiarrítmicos [11], hasta la implementación de marcapasos. Los primeros generan efectos secundarios o indeseables, los cuales pueden llegar a afectar de manera grave al paciente, incrementando el riesgo vital si el tratamiento no es el adecuado. Además incrementa el costo de hospitalización [11]. En Estados Unidos se tiene un número mayor a 467,000 hospitalizaciones anuales por FA [12], además, contribuye con 99,000 muertes al año [12]. Otro aspecto negativo es el del costo por el tratamiento de FA, el cual, en México, duplica los costos [11] y en Estados Unidos incrementa el costo por paciente 8,700 dólares al padecer FA. Esta enfermedad afecta entre 2.7 millones y 6.1 millones de personas y se espera que esta cifra se duplique dentro de los próximos 25 años [12].

Considerando todos los tipos de padecimientos cardiacos juntos, las cifras aumentan de una forma exagerada, sumando la miríada de padecimientos actuales y los costos anuales por pacientes, nos enfrentamos ante un problema serio. Antes que tener un buen tratamiento para la cura de estas enfermedades, el primer paso es tener un buen diagnóstico y es aquí donde surge un área de investigación orientada a la detección y clasificación efectiva de las enfermedades del corazón.

Actualmente se cuenta con una tasa de mortalidad alta por enfermedades del corazón. Una detección temprana y precisa de las enfermedades es necesaria. El diagnóstico realizado por los médicos requiere de un apoyo para llevar a cabo mejores tratamientos, dada la información necesaria y suficiente [13].

La clasificación de ECG por medio de la inteligencia artificial (IA) es una solución muy prometedora que se ha desarrollado en los últimos años [14]. Este apoyo posee un rol importante, ya que el poblema es realmente difícil, a consecuencia de que las características presentes en los ECG no son tan evidentes como uno pensaría. No es como clasificar manzanas y peras, personas altas y bajas, perros y gatos. Todas estas categorías llamadas clases en el ámbito de la IA, son estipuladas por los seres humanos. Pero si se tienen ejemplos que puedan pertenecer a varias categorías o los ejemplos de una sola categoría difieren en cierto grado, el problema se vuelve complejo [13].

La base de datos de Physionet [15] ofrece acceso a una gran variedad de colecciones de grabaciones de señales fisiológicas y software de open-source (libre uso). Esta pretende promover la investigación actual y nueva sobre el estudio de señales biomédicas y fisiológicas complejas. En cooperación con la conferencia anual *Computing in Cardiology*, Physionet es el anfitrión de la serie de retos anuales, en los cuales los estudiantes e investigadores abordan problemas abiertos de índole clínica o de interés científico básico usando datos y

software otorgado por Physionet, dando lugar al reto edición 2017 para la clasificación de ECG.

1.2. Inteligencia artificial

La inteligencia artificial nace del concepto de aprendizaje y razonamiento en los seres humanos, que, en el área de las ciencias de la computación, incluye a una máquina, donde, al tomar datos del exterior, ésta puede llevar a cabo un procesamiento de la información y conseguir algún cometido u objetivo. Es difícil definir inteligencia artificial debido a que la misma definición de inteligencia es compleja, pero podemos intentarlo. Para Elaine Rich la inteligencia artificial es el estudio de cómo hacer que las computadoras realicen cosas en las que, por el momento, los humanos son mejores [16]. La idea básica es la del aprendizaje de máquina (ML por sus siglas en inglés), que de acuerdo a Elaine Rich, es un subcampo central de la IA. De la misma forma se intenta definir, pero esta vez se expone la definición de Tom Mitchell, la cual afirma que es el estudio de algoritmos computacionales que mejoran automáticamente a través de la experiencia [17]. La inteligencia artificial ha evolucionado desde su origen y va encaminándose más allá del reconocimiento de patrones, hasta los robots autónomos, los cuales son sometidos a pruebas, en cierto ambiente controlado, para la resolución de problemas basados en la toma de decisiones. Los agentes de IA más conocidos hasta ahora son los automóviles autónomos [18] como el Google self-driving car y los Tesla, o los robots autónomos como Atlas, un robot humanoide de Boston Dynamics [19]. Así como los que participan en la competencia RoboCupRescue donde son sometidos a pruebas de rescate o los de servicio como el robot Marvin presentado en el proyecto de investigación AsRoBe [20] para ayudar a personas con discapacidad.

Una de las bases fundamentales del aprendizaje, es una excelente forma de resolver problemas de regresión y clasificación. Éste es el terreno del aprendizaje de máquina [17]. Se han creado diversas estructuras matemáticas que, de las características que se encuentran presentes en los conjuntos de datos recopilados, los agentes consiguen aprender y reconocen dichas características en nuevos datos, ajenos al conjunto del cual aprendieron [17]. De manera que es posible “enseñarle” a la computadora a reconocer la imagen de un gato en una fotografía. Esta capacidad de las estructuras, da lugar a toda un área de investigación con diversas áreas de aplicación como la física, química, astronomía, ingeniería entre otros. Llamaremos “sistema de aprendizaje” a cualquier algoritmo o estructura del aprendizaje de máquina [17].

Algunas de las herramientas más utilizadas en el aprendizaje de máquina son las redes neuronales artificiales, las redes neuronales convolucionales, las máquinas de soporte vectorial, los vecinos más cercanos, entre otros. A continuación se explica de manera breve el área de aplicación y las ventajas y desventajas que presentan cada una de las herramientas antes mencionadas ya que son métodos implementados en esta tesis.

Las redes neuronales artificiales (RNA) son estructuras que sirven como herramientas

del aprendizaje de máquina. Las RNA son utilizadas ampliamente en los ámbitos del reconocimiento de patrones, modelado, problemas de clasificación, optimización y para hacer pronósticos, entre otros. Esta tesis se enfoca en la aplicación de dichas redes en el problema de clasificación con aprendizaje supervisado (el cual se explica más adelante). Su amplio uso se debe a las siguientes ventajas [21]:

- Paralelismo en masa: se refiere a la realización de cálculos de forma simultánea [22].
- Representación y computación distribuida: La computación distribuida hace referencia a la realización de tareas que se lleva a cabo por varias computadoras que se encuentran en comunicación y se coordinan a través de mensajes para lograr un objetivo en común. En cuanto a la representación distribuida, se presenta cuando representamos información utilizando varias unidades (neuronas) y no sólo una [23].
- Habilidad de aprendizaje: Es la capacidad de resolver un problema por medio de la experiencia.
- Habilidad para generalizar lo que aprende: Es la capacidad de resolver un problema nuevo a partir de la experiencia de otros parecidos o de la misma naturaleza.
- Adaptabilidad: Capacidad de corregir o ajustarse a nuevos datos.
- Procesamiento de información contextual inherente: Capacidad de procesar la información que es relevante.
- Tolerancia a los fallos: Es la propiedad de un sistema que permite continuar operando incluso si una parte del mismo falla [24].
- Bajo consumo de energía.

Las redes neuronales convolucionales (RNC), se aplican mayoritariamente al reconocimiento de patrones en imágenes y se están desarrollando nuevos algoritmos para videos (sucesión de imágenes). Así mismo, los datos se procesan con transformaciones, por ejemplo, de series de tiempo a imágenes [25], para ser tratados por una red convolucional posteriormente. Las redes convolucionales pretenden identificar características en regiones de las imágenes por medio de un filtraje, donde filtros correspondientes a una cierta característica barren la imagen, realizando la extracción de las características. Posteriormente se implementa un downsampling, donde se reducen el número de datos a tratar. Finalmente pasan a una etapa donde se aplica una red neuronal artificial de propagación hacia adelante para la clasificación de las características extraídas por los filtros.

Este tipo de redes son aplicadas en tareas como el reconocimiento de dígitos (aumentando la complejidad, las muestras consisten en fotos del número exterior de las casas), donde se alcanza una precisión superior, comparada con los otros métodos de aprendizaje de máquina [26, 27].

Las máquinas de soporte vectorial (MSV) se usan en la clasificación y en el análisis de regresión. La capacidad de generalizar es más notoria en ellas, esto se explicará con mayor detalle en el capítulo 4. Las aplicaciones más recientes abarcan el reconocimiento de patrones como los identificadores de voz [28], la detección de infarto de miocardio en señales tipo ECG [30], entre otros.

Una de las ventajas más notorias de las máquinas de soporte vectorial es que su entrenamiento siempre encuentra una solución global, dicha característica contrasta con la situación en las redes neuronales artificiales donde la solución puede encontrarse en algún mínimo local [31]. Otra ventaja que tienen estas máquinas de aprendizaje es una propiedad muy particular la cual tiene que ver con las funciones Kernel implicadas. En el capítulo 4 se explica esta propiedad, la cual tiene como consecuencia que las MSV evitan las dos formas de la maldición de la dimensionalidad, las cuales consisten en el exceso de parámetros causando problemas complejos que no son posibles abordar y segundo, el exceso de parámetros que pueden producir un sobreajuste del método. Al decir sobreajuste, nos referimos a la incapacidad de generalizar lo aprendido por la máquina. Básicamente lo que sucede es que la máquina se comporta como un ente que al encontrar un mínimo detalle distinto, clasifica erróneamente los datos.

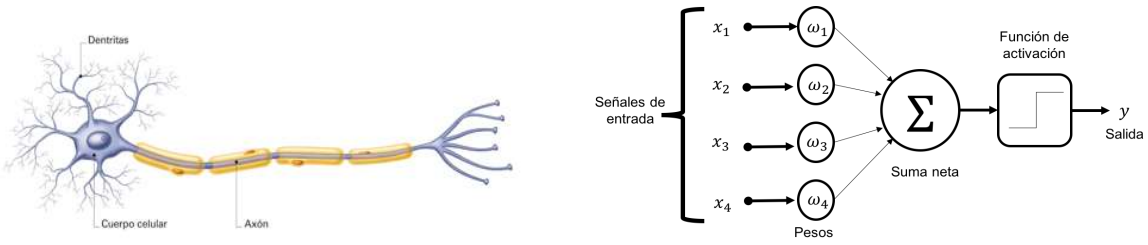
Esta tesis presenta una comparación del desempeño de los métodos mencionados anteriormente sobre los ECG para identificar enfermedades manifestadas en las señales cardiacas. Se utiliza el lenguaje python, por su gran uso en el ámbito del aprendizaje de máquina, también se utilizan las paqueterías enfocadas en esta área como *Tensorflow*. Los códigos elaborados comprenden de algunos hechos desde cero y otros utilizando la paquetería de *Tflearn*, *Tensorflow* y *LIBSVM* [32–34]. Esto se hizo así, pues el tiempo invertido para realizar todos los códigos desde cero es demasiado y va más allá del objetivo de este trabajo. Por lo que el uso de dichas librerías agiliza la implementación, por un lado en el tiempo requerido para escribir los algoritmos y por otro lado el tiempo de cómputo ya que ciertas operaciones son más rápidas usando las funciones de *numpy* [35], que es la paquetería principal de cómputo científico con python. De manera que la utilización de las paqueterías es de gran ventaja al momento de considerar la extensión de los códigos y el tiempo de cómputo.

Esta tesis está organizada de la siguiente forma: En el capítulo 2 se presentan las redes neuronales artificiales, sus bases y su implementación sobre un ejemplo de aplicación. En el capítulo 3 se presentan las máquinas de soporte vectorial con su respectiva implementación y ejemplo de aplicación. En el capítulo 4 se presentan los ECG y los resultados de la clasificación producidos por los métodos introducidos en los capítulos previos, así como un análisis de comparación entre los distintos métodos. Finalmente, en el capítulo 5 se presentan las conclusiones.

Capítulo 2

Redes Neuronales Artificiales

Las RNA fueron inspiradas en las células principales del sistema nervioso. Una neurona o célula nerviosa es una célula que procesa información. Está compuesta principalmente por un cuerpo celular o soma, el axón y las dendritas (figura 2.1a).



(a) (b)

Figura 2.1: (a) Se muestran los modelos de la neurona biológica [36] y (b) la neurona artificial o perceptrón.

La función general de una neurona es recibir señales o impulsos eléctricos provenientes de otras neuronas a través de las dendritas y transmitir las señales creadas por el cuerpo celular a las dendritas de alguna otra neurona, pasando por el axón, las ramas y finalmente las subramas. A esta conexión o estructura elemental entre dos neuronas se le conoce como sinapsis. Cuando la señal llega a la subrama en el momento de la sinapsis, algunos químicos, llamados neurotransmisores, son segregados por la vesícula sináptica, los cuales tienen un efecto directo sobre el comportamiento de las neuronas para mejorar o inhibir la transmisión de los impulsos eléctricos [21]. Este es el hecho en el cual se basa la neurocomputación dando origen a la unidad o neurona de umbral de McCulloch y Pitts [37].

Las RNA fueron desarrolladas formalmente en 1943 por McCulloch y Pitts [37]. La idea general de las RNA es ingresar un conjunto de elementos de entrada, el cual es sometido a una operación de suma asociando cantidades conectoras con cada elemento de entrada llamadas pesos, luego se le aplica una función a la cantidad resultante. A dicha

función se le conoce como función de activación, la cual es no-lineal, obteniendo un elemento de salida.

La neurona umbral de McCulloch y Pitts recibe n valores reales de entrada representando las dendritas que interactúan con distintas neuronas recibiendo los impulsos eléctricos de intensidades variadas. Luego se procesan todos los valores bajo la operación de suma y multiplicando cada entrada por el peso correspondiente obteniendo un número real el cual es ingresado en la función umbral para obtener finalmente el valor de salida ya sea 1 o 0. En la figura 2.1 se muestra la neurona y su análogo artificial.

El próximo paso, de acuerdo a la estructura biológica, es considerar la interacción de varias neuronas dando lugar a un sistema que llamamos red neuronal. De manera que el valor de salida obtenido a través de una se convierte en un valor de entrada de otra neurona. Considerando el análogo artificial, la primera distinción de este tipo de redes se realiza de acuerdo al número de capas que comprende. Llamaremos una capa al conjunto de neuronas o nodos ya sea del elemento de salida o de alguna otra capa que no sea del tipo elemento de entrada, a las capas que no sean las de entrada o la de salida son llamadas capas ocultas.

En esta tesis se trabaja con aprendizaje supervisado por lo que se podría decir, que existe un ente que le dice a la red cuando está equivocada para poder corregirse y así aprender [38]. Usualmente se considera un conjunto de entrenamiento con pares de entrada-salida como ejemplos para la red. Para que cuando se aplique uno de los elementos de entrenamiento de entrada sea posible comparar el resultado con el elemento de salida esperado o correcto y cambiar el valor de los pesos conectores ω_{ij} para minimizar la diferencia entre el elemento de salida calculado por la red y el elemento de salida esperado. Resulta muy útil ingresar elementos de entrada que no estén en el conjunto de entrenamiento para poner a prueba la red y ver su capacidad de generalizar lo aprendido. En la práctica suelen presentarse problemas de clasificación con datos que nunca habían sido registrados de manera idéntica, aú así la red sirve como buscador profundo de patrones por lo que puede llegar a encontrar alguna relación con la información conocida.

Hebb [39] estableció la esencia del aprendizaje de las redes simples, con un conjunto de normas para modificar los pesos de acuerdo a si la red respondía de una manera deseable al elemento de entrada para poder incrementar la probabilidad de la red de responder de la misma manera al ingresar elementos similares o de la misma naturaleza.

Con el objetivo de introducir la notación para el trato de las redes, se presenta el caso de una red neuronal de una capa, la capa de salida. Ésta comprende del conjunto de N elementos en la entrada y una capa de salida, es decir, ninguna capa oculta.

Formalmente, podemos representar una red neuronal artificial como una tupla [40]

$$M = (N, C, \alpha, O, \omega, t, \Phi), \tag{2.1}$$

donde

- N es un conjunto finito no-vacío de neuronas o nodos.

- $C \subseteq N \times N$ es un conjunto no-vacío de aristas orientados entre las neuronas.
- $\alpha_i \subset N$ es un conjunto no vacío de neuronas en la capa de entrada.
- $O_j \subset N$ es un conjunto no vacío de neuronas en la capa de salida.
- $\omega_{ij} : C \mapsto \mathbb{R}$ es una función de peso.
- $t : N \mapsto \mathbb{R}$ es una función para el bias* de la red.
- $\Phi : \mathbb{R} \mapsto \mathbb{R}$ es una función de activación.

El cálculo de los valores en la capa de salida está dado por

$$O_j = \Phi(h_j) = \Phi\left(\sum_{i=0}^{n} \omega_{ij}\alpha_i\right), \tag{2.2}$$

donde n es el número de neuronas o nodos en la capa de entrada y $j = 1, ..., M$ donde M es el número de neuronas en la capa de salida.

La red neuronal más sencilla es el perceptron, el cual consiste de una sólas neurona en la capa de salida, donde estrictamente la función de activación corresponde a la función escalón, es decir, el perceptron es un clasificador binario [40]. Las RNA con multicapas son consideradas aproximadores universales que pueden aproximar una función arbitraria a cualquier grado de precisión [41], por lo que son más utilizadas que el perceptrón simple. Las RNA con varias capas son una arquitectura donde los elementos de salida de la capa de entrada consisten en los elementos de entrada de la capa oculta y así sucesivamente hasta pasar por todas las capas ocultas. En la figura 2.2 se muestra la representación esquemática de una red neuronal con una capa oculta.

La razón de una función de activación proviene del mismo perceptrón el cual en principio recibe la información de entrada para obtener un solo valor de salida y así formar el clasificador binario

$$f(x) = \begin{cases} 1 \quad \text{si} \quad x \cdot \omega + b > 0 \\ \\ 0 \qquad \text{otro caso,} \end{cases}$$

donde x es el vector de valores de entrada, ω es el vector de pesos, y b es un *bias*. En realidad la función implicada es la función escalón de Heaviside [18]. Ahora si pensamos en funciones continuas que tengan aproximadamente el mismo comportamiento que la función de Heaviside, aparecen funciones tales que su rango es $[0, 1]$ como la sigmoide. En

*Parámetro que brinda un grado de libertad extra y evita la pérdida de información.

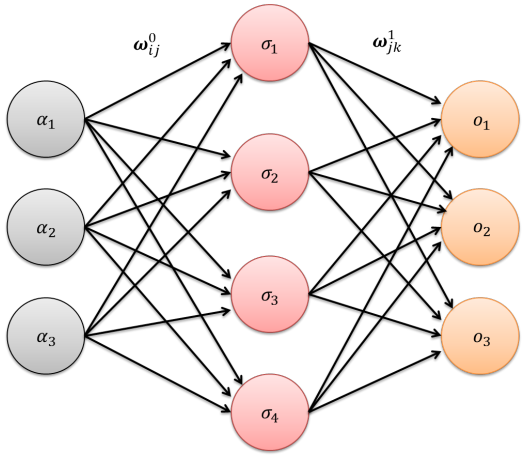


Figura 2.2: Se muestra el diagrama de una red neuronal con una capa oculta con 3 neuronas en la capa de entrada, 4 neuronas en la capa oculta y 3 neuronas en la capa de salida.

los casos en los que se desea resolver un problema de clasificación los elementos de salida son indicadores de la pertenencia del elemento de entrada a cierto conjunto (Se pueden resolver otros problemas, p.e. regresión [42]). A los distintos conjuntos a los cuales puede pertenecer un elemento los llamamos clases. Por ello, establecida la normalización adecuada es posible interpretar al valor de la función de activación como la "probabilidad" de pertenecer a cierta clase. De manera que se puede aproximar una función no-lineal de varias variables, mediante funciones no-lineales de una variable. La función de activación es una función no-lineal, por ejemplo, puede ser una sigmoide o una tanh, entre otras.

Podemos categorizar a las redes en dos grupos, las de propagación hacia adelante (*feed-forward*) y las recurrentes. Si vemos la estructura de las redes como grafos dirigidos con peso en los que las neuronas son nodos y las aristas dirigidas con peso son las conexiones entre las neuronas de entrada y las neuronas de salida [21], las redes de propagación hacia adelante son las que no tienen ciclos y las redes recurrentes tienen ciclos ya que usan retroalimentación de las capas anteriores inmediatas (ver [43]). Para las redes que se propagan hacia adelante, el valor de salida es una función explícita del valor de entrada.

El índice i en la ecuación (2.2) empieza desde el valor $i = 0$, es decir, consideramos una neurona extra. A esa neurona la llamamos bias, que sirve para suprimir los valores de salida con valor nulo, lo cual incrementa el desempeño de una red al evitar la pérdida de información. Por ejemplo, para la suma con pesos de la ecuación (2.2), es posible tener un vector de entrada que de como resultado cero en el valor de salida, dada la naturaleza de la función de activación. Teniendo esta situación, posiblemente, para otros vectores de entrada. Pero al introducir una neurona extra distinta de cero a la suma con pesos, el resultado cambia a ser distinto de cero, lo cual podemos interpretar posteriormente.

Los elementos de entrada, de salida correctos y de salida calculados por la red pueden ser de tipo booleano, variable continua o discreta. La dependencia del tipo de dato en los elementos de salida recae sobre la naturaleza de la función de activación Φ implementada. Sin embargo, resulta una parte fundamental la del preprocesamiento de los datos (adecuarlos para que puedan ser utilizados por la red). Esto incluye normalizar y particionar el conjunto de datos completo. Existen varias técnicas de normalización sin embargo no existe un método estándar. Más adelante se aborda el tema de la partición de los datos.

La clave en el aprendizaje de las redes es la modificación de los pesos de la red. Para ello se desarrollaron conjuntos de normas o reglas para cambiar estos valores de manera conveniente. Existen 4 tipos distintos de reglas de aprendizaje: Corrección de error, Boltzmann, Hebbiano y Aprendizaje Competitivo [21]. En esta tesis nos enfocamos exclusivamente al aprendizaje por corrección del error; no obstante, si se desea una explicación detallada de los otros métodos consultar [21, 44].

En el aprendizaje por corrección de error se considera una función de costo o de error, la cual nos da información sobre qué tan alejados se encuentran los valores calculados por la red de los valores deseados. Se suele utilizar como función de costo al error cuadrático total por su diferenciabilidad y sencillez.

La función de error cuadrático total, para un grupo de patrones de entrada α_i^μ, con función de activación $\Phi(x)$ y sus respectivas etiquetas de salida correctas ζ_i^μ para la clasificación, es

$$E\left[\omega\right] = \frac{1}{2}\sum_{i,\mu}\left(O_i^\mu - \zeta_i^\mu\right)^2. \tag{2.3}$$

En esta tesis se presenta la regla de la corrección sobre la función de costo ya que es de las más utilizadas, para resolver el problema de clasificación. Esta forma de entrenar a la red consiste en la minimización de la función de costo. El método se basa en encontrar la dirección en la que se presenta un mínimo (convenientemente global) y modificar los pesos para que el costo se mueva en esa dirección. Es posible que al iniciar los pesos nos encontremos en una región donde al corregir nos dirijamos hacia un mínimo local y por lo tanto no se tendrá la solución óptima, es decir, con la mejor precisión. Para evitar estos atascos en los mínimos locales se pueden implementar algoritmos de búsqueda sofisticados que salten de región en región para encontrar el mejor mínimo. La forma más sencilla de abordar este inconveniente es hacer una serie de experimentos en los que el valor de los pesos cambie para explorar el espacio de parámetros y posiblemente se encuentre un resultado satisfactorio.

Existe también la posibilidad de modificar los valores a los cuales llamamos bias, de la misma manera que los pesos. Por simplicidad fijamos estos valores como constantes e incorregibles.

Regresando a la partición del conjunto de datos, se suele hacer en tres partes: conjunto de entrenamiento, validación y predicción. El conjunto de entrenamiento es el conjunto de

elementos que participarán en el cálculo de las modificaciones en los pesos. El conjunto de validación es nuestro vigilante, el que nos dice que tan bien está aprendiendo la red y sirve como un criterio de frenado en el aprendizaje en cuanto a ciclos de entrenamiento se refiere y éste no participa en la modificación de los pesos. El conjunto de predicción, siendo información nueva o desconocida para la red, es nuestra prueba final para establecer el desempeño de todo el sistema y la capacidad de la red para generalizar lo que aprendió después de su entrenamiento. A continuación se presenta el algoritmo para entrenar a una RNA, para una explicación más profunda de lo que es una RNA ver [45].

2.1. Entrenamiento de una RNA *Feed-forward*

El procedimiento de corrección a utilizar será la regla delta también llamada “Back propagation”; es decir, propagación hacia atrás o descenso gradiente (para el desarrollo de la generalización a M capas ocultas [44]). La idea general es encontrar el valor mínimo de la función error en cierta región del espacio de parámetros (los pesos). Para ello debemos cambiar el valor de los pesos para que el error tome valores en la dirección contraria del gradiente, es decir la dirección de máximo cambio. Y poco a poco hacer un ajuste en la selección del valor de los pesos y llegar a tener un error satisfactorio para el problema en cuestión. En la figura 2.3 se muestra la geometría detrás del descenso gradiente para un espacio con dos variables independientes.

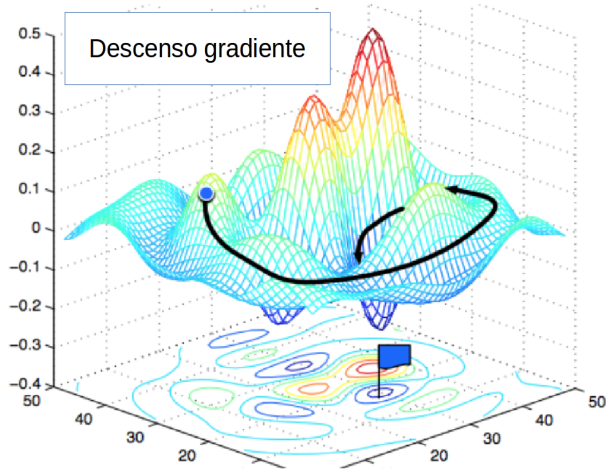


Figura 2.3: Se muestra la idea del descenso gradiente, la cual es encontrar el mínimo de la función error a través del ajuste de los pesos en la dirección contraria al máximo cambio [46].

A continuación se realiza el cálculo de los gradientes respecto a los pesos en la red

neuronal compuesta por una capa de entrada, una capa oculta y una de salida como se muestra en la figura 2.2.

La función de costo será el error cuadrático (ecuación 2.3), estableciendo con la siguiente notación

$$E = \sum_k \frac{1}{2} (T_k - R_k)^2 , \tag{2.4}$$

donde consideramos que se corrige respecto a una sola muestra. De manera que R_k corresponde al resultado esperado (la clase) y T_k es el valor de salida calculado por la red a través de la propagación hacia adelante, donde k es el índice de referencia a los componentes de la capa de salida y corre de 1 a N, el número de clases distintas.

Se utiliza la función de activación $\Phi(x)$, comenzando por los nodos de la capa de entrada propagamos hacia adelante

$$D_j = \sum_{i=1}^{n} \omega_{ij} I_i, \tag{2.5}$$

donde los pesos ω pertenecen a las conexiones entre la capa de entrada y la oculta, por lo tanto el índice i corre sobre el número de nodos de entrada n y el índice j corre sobre el número de nodos en la capa oculta M.

Luego aplicamos la función de activación a la cantidad D_j

$$H_j = \Phi(D_j). \tag{2.6}$$

Se calcula la suma correspondiente a las conexiones con pesos P entre la capa oculta y la capa de salida,

$$S_l = \sum_{j=1}^{M} H_j P_{jl}. \tag{2.7}$$

Finalmente, le aplicamos la función de activación a la cantidad S_l

$$T_l = \Phi(S_l), \tag{2.8}$$

donde el índice $l = 1, ..., N$, donde N es el número de nodos en la capa de salida y el número de clases distintas.

Estamos listos entonces para calcular el gradiente respecto a los pesos de la capa oculta ω y los pesos de la capa de salida P. La derivada respecto al argumento de la función de activación se representará con el signo prima. Consideremos primero los pesos de la capa de salida

$$\begin{aligned}\frac{\partial E}{\partial P_{ml}} &= \sum_k (T_k - R_k)\,\frac{\partial T_k}{\partial P_{ml}} \\ &= \sum_k (T_k - R_k)\,\frac{\partial}{\partial P_{ml}}\left(\Phi\left(S_k\right)\right) \\ &= \sum_k (T_k - R_k)\,\Phi'\left(S_k\right)\sum_{j=1}^{M} H_j \frac{\partial P_{jk}}{\partial P_{ml}} \\ &= \sum_k (T_k - R_k)\,\Phi'\left(S_k\right) H_m \delta_{kl},\end{aligned} \tag{2.9}$$

donde en la última igualdad se utilizó la delta de Kronecker

$$\delta_{\nu\mu} = \begin{cases} 1 \quad \text{si} \qquad \nu = \mu \\ \\ 0 \qquad \text{otro caso.} \end{cases}$$

Por lo tanto, la delta de Kronecker también elimina la suma sobre k, dando como resultado

$$\frac{\partial E}{\partial P_{ml}} = (T_l - R_l)\Phi'(S_l)H_m. \tag{2.10}$$

Calculemos ahora el gradiente respecto a las conexiones de la capa de entrada y la capa oculta

$$\begin{aligned}\frac{\partial E}{\partial \omega_{ml}} &= \sum_k (T_k - R_k)\,\frac{\partial}{\partial \omega_{ml}}(T_k - R_k) \\ &= \sum_k (T_k - R_k)\,\frac{\partial T_k}{\partial \omega_{ml}}. \qquad (2.11)\end{aligned}$$

(2.12)

Ahora calculemos separadamente $\frac{\partial T_k}{\partial \omega_{ml}}$ por simplicidad

$$\begin{aligned}
\frac{\partial T_k}{\partial \omega_{ml}} &= \frac{\partial}{\partial \omega_{ml}} \left(\Phi(S_k)\right) \\
&= \Phi'(S_k) \left(\sum_{j=1}^{M} \frac{\partial (H_j P_{jk})}{\partial \omega_{ml}} \right) \\
&= \Phi'(S_k) \left(\sum_{j=1}^{M} P_{jk} \frac{\partial H_j}{\omega_{ml}} \right) \\
&= \Phi'(S_k) \left(\sum_{j=1}^{M} P_{jk} \Phi'(D_j) \frac{\partial D_j}{\omega_{ml}} \right) \\
&= \Phi'(S_k) \left(\sum_{j=1}^{M} P_{jk} \Phi'(D_j) \sum_{i=1}^{n} I_i \frac{\partial \omega_{ij}}{\partial \omega_{ml}} \right) \\
&= \Phi'(S_k) \sum_{j=1}^{M} P_{jk} \Phi'(D_j) I_m \delta_{lj} \\
&= \Phi'(S_k) P_{lk} \Phi'(D_l) I_m. \qquad (2.13)
\end{aligned}$$

Sustituyendo el valor de $\frac{\partial T_k}{\partial \omega_{ml}}$ en la ecuación (2.11) resulta

$$\begin{aligned}
\frac{\partial E}{\partial \omega_{ml}} &= \sum_k \left(T_k - R_k\right) \frac{\partial T_k}{\partial \omega_{ml}} \\
&= \sum_k \left(T_k - R_k\right) \Phi'(S_k) P_{lk} \Phi'(D_l) I_m \\
&= I_m \Phi'(D_l) \sum_k P_{lk} \Phi'(S_k) \left(T_k - R_k\right). \qquad (2.14)
\end{aligned}$$

A continuación se enlistan los pasos a seguir para lograr que una red con varias capas aprenda y clasifique un grupo de datos de entrada. Se asume que los datos fueron preprocesados para poder ingresarlos en la red, así mismo, se conoce la clase de cada elemento de entrada en cuestión.

Procedimiento de aprendizaje BP [44]:

1. Se inicializan las matrices de pesos con números aleatorios con valores menores a 1 en valor absoluto para cada par de capas de la red.

2. Se elige un patrón de entrada α_i a procesar.

3. Se realiza la propagación hacia adelante hasta la útlima capa, obteniendo un patrón output O_i

$$\sigma_i^m = \Phi(h_i^m) = \Phi\left(\sum_j \omega_{ij}^m \sigma_j^{m-1} \right),$$

donde el índice m identifica a la capa implicada y $m = 0$ corresponde a la capa de entrada, además $O_i = \sigma_i^M$.

4. Al realizar el cálculo del gradiente nos encontramos con cantidades Γ_i^j que pueden verse de manera recursiva. Se calculan las Gammas de cada una de las capas empezando por la de salida

$$\Gamma_i^M = \Phi'\left(h_i^M\right)\left[\zeta_i - \sigma_i^M\right]$$
$$\Gamma_i^{m-1} = \Phi'\left(h_i^{m-1}\right)\sum_j \omega_{ji}^m \Gamma_j^m,$$

donde los valores ζ_i son el arreglo de salida deseado o esperado. Para valores de $m = M, M-1, ..., 1$, hasta calcular todas las Gammas.

5. Usamos la definición de las correcciones $\Delta\omega_{ij}^m = \eta\Delta_i^m\sigma_j^{m-1}$ donde η es un factor llamado razón de aprendizaje y este indica el tamaño del salto en dirección del gradiente para la corrección. De manera que se actualizan todas las conexiones de acuerdo a la regla

$$\omega_{ij}^{\text{nuevo}} = \omega_{ij}^{\text{viejo}} + \Delta\omega_{ij}.$$

6. Regresar al paso 2 y repetir para el siguiente patrón de entrada.

En la práctica tenemos los siguientes modos de entrenamiento debido a la frecuencia con la que ajustamos los pesos:

- Aprendizaje uno a uno: Se estima el gradiente de acuerdo al error presentado por cada ejemplo de entrenamiento de forma individual.
- Aprendizaje por lote: Se corrigen los pesos después de presentar el grupo completo de ejemplos de entrenamiento.
- Aprendizaje por mini-lotes: Se corrigen los pesos después de presentar subconjuntos del grupo de entrenamiento.

Los modos de entrenamiento mencionados anteriormente tienen ventajas y desventajas al momento de su implementación [47]. El aprendizaje uno a uno es conveniente cuando es imposible entrenar a la red ingresando el conjunto de entrenamiento en su totalidad. También suele utilizarse cuando el algoritmo necesita adaptarse de forma dinámica a nuevos patrones en los datos, o cuando los datos son generados como función del tiempo (por ejemplo predicciones de precios de mercancía). Desventaja del aprendizaje uno a uno es que la red tiende a olvidar más rápido lo que ha aprendido. Para el aprendizaje por lotes, se ha afirmado su superioridad debido al uso del verdadero gradiente o su aproximación. En el caso de que no sea posible ingresar el conjunto de datos en su totalidad se recurre a

realizar la corrección de los pesos en grupos o mini-lotes lo cual nos da una aproximación del gradiente global. Esto puede ser de gran ayuda, ya que al ser una aproximación puede evitar caer en un mínimo local.

2.1.1. Clasificando dígitos

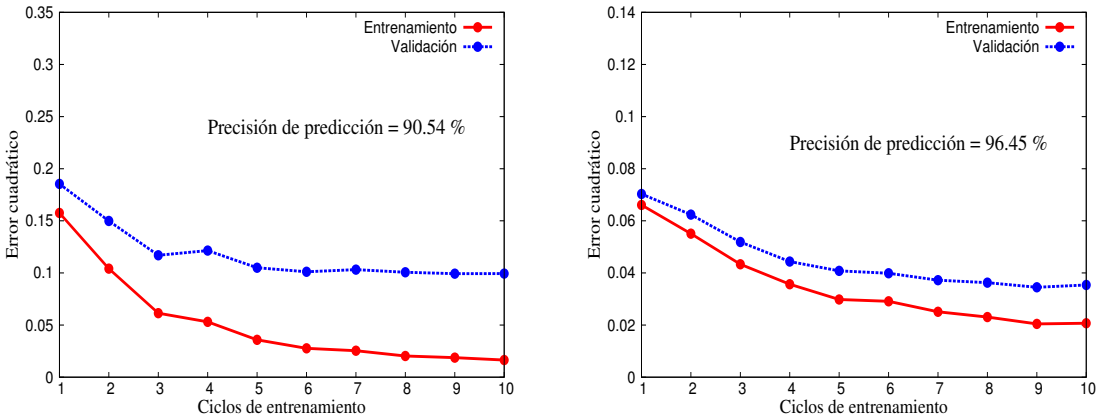
A continuación se presenta un ejercicio que permite ilustar el comportamiento de la función de costo y el aprendizaje de una red

Figura 2.4: Se muestran algunos ejemplos de los digitos escritos a mano de la base de datos de Mnist [48].

Se propone resolver un caso sencillo usando una red neuronal *feed-forward* con una capa oculta y aplicando el método de BP para la corrección de los pesos. El problema es el de la clasificación de dígitos escritos a mano de la base de datos de Mnist [49] obtenidos desde la librería TensorFlow [33] de python, algunos ejemplos se muestran en la figura (2.4). La base de datos consiste en 60,000 ejemplos para el entrenamiento y 10,000 para el conjunto de predicción. Dichos datos se repartieron aleatoriamente en distintos escenarios en los que se modifica la cantidad de datos ingresados al conjunto de entrenamiento en la red con el fin de ilustrar los efectos sobre el desempeño de la red en relación al tamaño del conjunto de entrenamiento. Cada ejemplo es una imagen con los valores numéricos de los pixeles se encuentran en un rango de cero a uno, es decir, una imagen normalizada y centrada de 28×28. Se definen los datos de entrada para la red la imagen "aplanada" convirtiéndola en un arreglo con 784 valores donde cada uno representa la intensidad de cada pixel, una representación unidimensional de una imagen.

En la figura 2.5 se pueden observar las gráficas del error en función del número de ciclos de entrenamiento para los conjuntos de menor (2,222 ejemplos) y mayor (66,666 ejemplos) tamaño. En todos los conjuntos se tiene la siguiente configuración en la partición de los datos de cada escenario: 63 % entrenamiento, 27 % validación, 10 % predicción. Al

comparar la figura 2.5a con la figura 2.5b podemos notar que el error usando el grupo de 66,666 ejemplos es un orden de magnitud menor desde el primer ciclo de entrenamiento. Otra diferencia es que el aprendizaje sobre el grupo de validación en el grupo de 2,222 ejemplos se quedó estancado a partir del ciclo 5, simplemente dejó de aprender. Por otro lado, en el grupo de 66,666 ejemplos se obtiene un aprendizaje continuo siendo mayor en los primeros 4 ciclos.



(a) (b)

Figura 2.5: Se muestra el error en función del número de ciclos de entrenamiento para (a) un grupo aleatorio de 2,222 ejemplos el grupo con el número mínimo de datos y para (b) un grupo aleatorio de 66,666 ejemplos con el número máximo de datos, ambos extraídos de la base de datos de Mnist de dígitos escritos a mano. Los grupos de datos incluyen los grupos de entrenamiento, validación y predicción.

La arquitectura de la red consistió de los siguientes parámetros:

$$\begin{aligned} N &= 100 \\ \eta &= 0.5 \\ \text{Mini lote} &= 10 \text{ ejemplos} \\ W_{ij} &\in [-1, 1] \subset \mathcal{R}, \end{aligned} \tag{2.15}$$

Donde N es el número de neuronas utilizado en la capa oculta, η es la razón de aprendizaje a la cual, en un principio, conviene darle un valor pequeño (menor a 1) para evitar dar saltos bruscos y perder el movimiento en dirección del mínimo. El subgrupo de entrenamiento fue de 10 elementos, de acuerdo al modo de entrenamiento de *mini-lotes* [50]. En esta estructura se incluyen las cantidades conocidas como *bias* generados aleatoriamente usando el mismo rango que los pesos (excluyendo el cero), las cuales sirven para evitar que la red genere en sus cálculos internos valores nulos, como se mencionó anteriormente esto ayuda al aprendizaje y evita la pérdida de información.

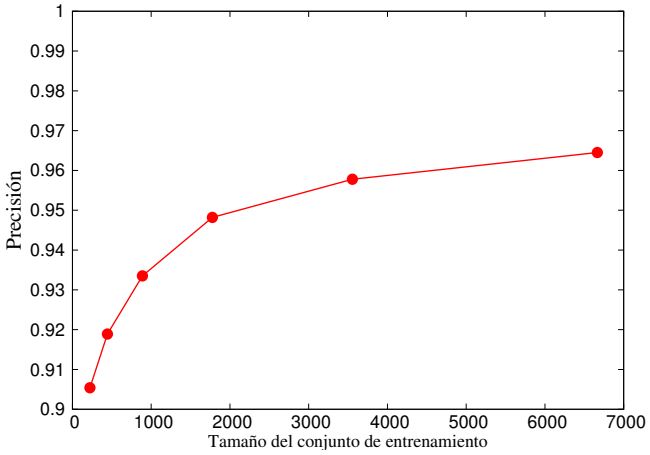


Figura 2.6: Se muestra la precisión de la clasificación de la red sobre el grupo de predicción en función del número de elementos del conjunto de entrenamiento, donde es claro el comportamiento creciente.

En la figura (2.6) se muestra la precisión de la red en función de los diferentes tamaños de los grupos de entrenamiento. Es muy claro el comportamiento de la red, que al tener más elementos de entrenamiento, mejora su desempeño de manera considerable y así hacer efectiva la generalización de su aprendizaje. Con una arquitectura tan sencilla fue posible conseguir una precisión del 96.45 % sobre el conjunto de predicción el cual no fue presentado en la red en la etapa de entrenamiento.

2.2. Los K-vecinos más cercanos

Otro método conocido en el ámbito del aprendizaje de máquina es el llamado los K-vecinos más cercanos. Siendo un método no paramétrico, es decir, no tiene parámetros de los cuales pueda aprender, puede ser usado en problemas que impliquen regresiones y clasificación. Puesto que el caso a tratar es el de aprendizaje supervisado, el método consiste en una comparación uno a uno de todos los elementos conocidos con algún elemento que se desea clasificar. La naturaleza de la comparación se extrae del concepto de distancia mínima euclidiana. Ilustraremos el método de comparación con el siguiente ejemplo básico.

Supongamos que tenemos los pares ordenados que podemos visualizar en el plano cartesiano como en la figura 2.7. Los datos pertenecen a una de dos clases, las cuales diferenciamos por figura. La etiqueta que le asignamos al primer grupo son los círculos rellenos de color rojo y al segundo grupo los cuadrados rellenos de color verde. Ahora, si en algún momento queremos clasificar un nuevo punto, podríamos hacer un análisis cualitativo y decidir en qué grupo será asignado dicho elemento. Naturalmente podríamos elegir el gru-

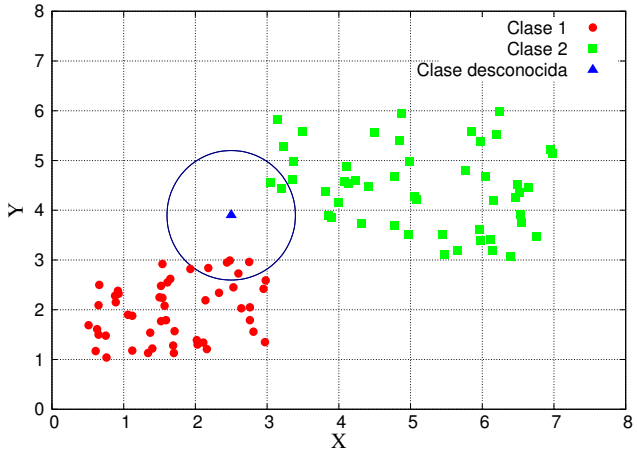


Figura 2.7: Se muestra la gráfica de los pares ordenados pertenecientes a dos clases y el par ordenado que se desea clasificar.

po al cual se encuentra más cerca el elemento nuevo. Este razonamiento puede expresarse matemáticamente y ayudarnos en ciertos escenarios en los cuales no sea tan fácil distinguir a cuál de los grupos pertenece el nuevo elemento. Para ello utilizamos la norma euclidiana:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{2.16}$$

De manera que podemos calcular las distancias entre el elemento a clasificar y los elementos conocidos, y posteriormente el elemento nuevo adoptará la clase del elemento cuya distancia al elemento nuevo sea la mínima (figura 2.7). Ahora mejorando esta estrategia, al intentar evitar ciertos fallos, por ejemplo que existan varios elementos muy cercanos (o varios mínimos), se consideran ahora las tres distancias mínimas, mejorando la confiabilidad del método. Podemos seguir aumentando el número de vecinos más cercanos, llegando así al concepto de los K-vecinos más cercanos, donde se le asocia una probabilidad de que el elemento pertenezca a cierta clase. Se utilizan valores impares para el número de vecinos con la intención de minimizar los empates entre clases. Por lo que al contar la votación, podemos clasificar el elemento de acuerdo al mayor número de elementos de la misma clase dentro del conjunto de los K-vecinos.

Esta forma de tratar los datos se puede generalizar a mayor número de dimensiones, donde no es posible obtener gráficas como en la figura anterior y es en esta situación donde el aprendizaje de máquina es de gran utilidad.

Otro plus para mejorar el método es el siguiente criterio que a su vez sirve de desempate dado el caso. El criterio consta de tomar en cuenta la distancia al momento de las votaciones. Por lo que en la situación que se requiera, podemos darle cierto peso a los votos de acuerdo a la distancia de los vecinos. Esto puede realizarse añadiendo un término de peso ω inversamente proporcional a la distancia del vecino al elemento [51].

Se tienen desventajas en el método de los K-vecinos más cercanos en relación a la dimensión y al número de muestras o ejemplos de entrenamiento. Primero, en cuanto a la dimensión, la distancia euclidiana se vuelve muy similar para dimensiones grandes, por lo que la distancia entre el vecino más cercano y el más lejano pueden llegar a ser muy similares y así provocar alguna clasificación errónea. Segundo, la cantidad de ejemplos o datos por procesar puede llegar a ser exorbitante; no obstante, esto no quiere decir que el algoritmo va a funcionar mejor, ya que depende del conjunto de datos en cuestión. Una forma de mejorar el algoritmo es aplicando una reducción en el número de ejemplos implicados [52], un intento por condensar los datos, absorbiendo aquellos que no contribuyen a obtener una clasificación con mayor precisión.

La pregunta que nos planteamos naturalmente es ¿qué valor de K debemos elegir? o ¿cómo elegir el valor de K que resulte con la mejor precisión posible en la clasificación? Y una respuesta a este problema es un método muy eficiente llamado validación cruzada (VC), el cual es usado en varios métodos de aprendizaje de máquina.

La validación cruzada es un método que nos ayuda a elegir los hiperparámetros de un método, es decir los parámetros que no cambian durante el entrenamiento, sino los que son definidos por el usuario. El método consiste en hacer una partición normalmente balanceada respecto al número de elementos de cada clase. Sea k (minúscula) el número de subconjuntos obtenidos por la partición, de donde obtiene el nombre de validación cruzada de k iteraciones (*k-fold cross-validation*). Se procede con el algoritmo de los K-vecinos más cercanos usando uno de los subconjuntos como grupo de validación y los $k-1$ restantes como grupo de comparación (en el caso de las redes neuronales o las máquinas de soporte vectorial corresponde al grupo de entrenamiento, más adelante se aborda el caso), luego se continúa con el siguiente subconjunto de la partición para tomar el papel del grupo de validación y los $k-1$ restantes para tomar el papel de conjunto de comparación hasta que todos los subconjuntos hayan sido grupo de validación. Se realiza esto un número m de veces para variar los hiperparámetros y así encontrar la mejor configuración de estos. De manera que la VC es un método de selección de modelo [53]. Después de haber entrenado con cada configuración de hiperparámetros se promedia la precisión (o el error) del grupo de validación, y se registran para comparar y encontrar la mejor configuración de hiperparámetros para fijarlos en el entrenamiento pero esta vez con todos los subconjuntos como grupo de entrenamiento para obtener el valor de predicción sobre un conjunto no presentado anteriormente.

Por lo tanto la manera para elegir el mejor valor de vecinos K será utilizando la validación cruzada k, donde se realiza el entrenamiento con un valor fijo de K vecinos fijo y la rotación del grupo de validación. Se obtiene entonces el valor de K para el cual se obtiene la mejor precisión promedio y posteriormente se utiliza ese número de vecinos sobre el grupo de predicción.

Se propone utilizar el método para clasificar imágenes para ilustrar las características que pueden presentar los resultados del método de los vecinos. A continuación se presenta el conjunto de datos utilizados por este método y los resultados.

2.2.1. Conjunto de datos de Mnist

En la sección 2.1.1 se clasificó este conjunto de datos por medio de la red neuronal *feed-forward*, la cual obtuvo una precisión del 96.45 %. En esta sección se resolverá el mismo problema, pero se implementará el método de los K-vecinos más cercanos junto a la validación cruzada para la elección de K.

Para realizar el algoritmo de los K-vecinos se utilizó la librería numpy [35] y en particular, la resta y la función 2-norma para calcular las distancias a los vecinos.

Los resultados de la clasificación se muestran en la tabla 2.1

Tabla 2.1: Valores de la precisión en función del número de vecinos sobre el conjunto de entrenamiento. Se utilizó validación cruzada de 5 iteraciones.

k	precisión %
1	97.51
3	97.31
5	97.12
7	97.03
9	96.81
11	96.69
13	96.57
15	96.38
17	96.23
19	96.17
21	96.01

Por lo que el valor de K óptimo resulta ser igual a 1. Se realizó la clasificación sobre el conjunto de predicción con este valor de K obteniendo una precisión de 96.77 %. Se obtiene un excelente resultado, y el método supera el anterior. Aunque es natural pensar en implementar este método para la solución de la clasificación de un conjunto de datos como el Mnist, resulta que no siempre se obtienen buenos resultados en otros datos y de hecho el método que soluciona el problema de la mejor manera depende de los mismos. En la siguiente sección veremos cómo el método de los K-vecinos falla rotundamente.

2.2.2. Conjunto de datos de CIFAR-10

El conjunto de datos de Cifar [54] es un conjunto de imágenes de 32×32 a color, divididos en 5 lotes de entrenamiento y 1 grupo de predicción con 10,000 imágenes cada uno. Cada uno de los grupos contiene imágenes de las 10 clases posibles: aviones, barcos, trenes,

automóviles, perros, pájaros, venados, gatos, caballos y ranas. Las imágenes pertenecen a una y sólo una clase. Algunos ejemplos se muestran en la figura 2.8.

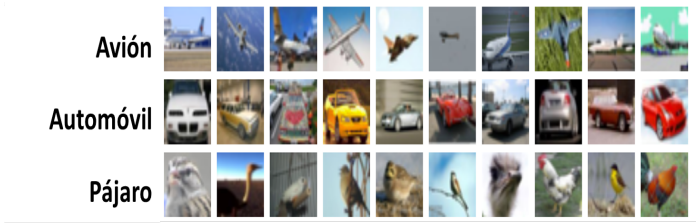


Figura 2.8: Se muestran algunos ejemplos de las imágenes de la base de datos de cifar-10, imagen extraída de [55].

Una vez obtenidos los datos [54], se procedió a convertir los datos en arreglos tipo numpy. Aunque se tienen objetos de tipo tensor de 3er rango (es decir, podemos definir un componente de una imagen al fijar 3 índices) por los tres filtros RGB se calcula de la misma manera la distancia a los vecinos.

Los resultados obtenidos en la clasificación se muestran en la tabla 2.2.

Tabla 2.2: Valores de la precisión en función del número de vecinos sobre el conjunto de entrenamiento. Se utilizó validación cruzada de 5 iteraciones.

k	precisión %
1	33.83
3	32.60
5	33.18
7	33.42
9	33.25
11	32.94
13	32.83
15	32.73
17	32.59
19	32.37
21	32.09

Se concluye que el valor de K óptimo es igual a 1 nuevamente. Se realizó la clasificación sobre el conjunto de predicción con este valor de K obteniendo una precisión de 35.39 %. Estos resultados son deficientes debido al algoritmo de clasificación implementado, ya que al comparar las imágenes de esta manera, el algoritmo no logra diferenciar entre

las características de los objetos en las imágenes, en lugar de ello puede aprender concentraciones de color en ciertas regiones por lo que podría clasificar la imagen de un perro como la de un avión si tienen fondos similares por ejemplo. Este es un caso donde los K-vecinos no demuestran mucho poder de clasificación, por el contrario existen métodos que fueron pensados para analizar imágenes y extraer los patrones relevantes para lograr una buena clasificación y uno de esos métodos es el de las redes neuronales convolucionales. A continuación se presenta su funcionamiento y su aplicación para contrastar la eficiencia con este tipo de datos.

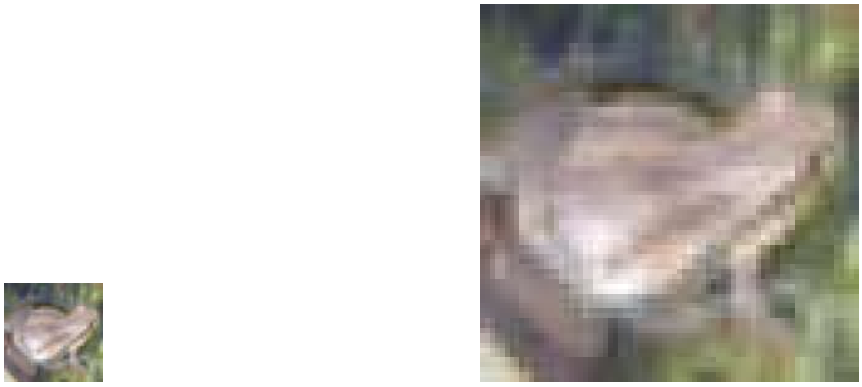
2.3. Redes Neuronales Convolucionales

Las redes neuronales convolucionales fueron pensadas para clasificar imágenes, una de las características importantes que tienen este tipo de redes es su capacidad de cómputo cuando hablamos de velocidad, ya que las imágenes pueden estar compuestas de cantidades grandes de datos de tipo entrada. Pensemos en una imagen RGB de 28×28, esto quiere decir que se tienen 3 matrices o arreglos de 28×28, indicando los niveles de intensidad de color del rojo, azul y verde (ver figura 2.9a). Supongamos que queremos ingresar esta muestra a una red neuronal de tipo propagación hacia delante simple de dos capas, con una estructura de 1024 neuronas en la capa oculta, y en la capa de entrada $3 \times 28 \times 28$ neuronas correspondientes al aplanamiento de las matrices RGB de la imagen, donde el aplanamiento no es más que tomar fila por fila convirtiendo las matrices en un solo arreglo con $3 \times 28 \times 28$ entradas. Consideremos ahora la matriz de pesos resultante, ¡la cual tiene $1024 \times 3 \times 28 \times 28 = 2,408,448$ conexiones! Ni pensar en tener varias capas ocultas. Esto nos da una noción del tiempo que le tomaría a una estructura como la red neuronal artificial de propagación hacia adelante en hacer un análisis con millones de imágenes para clasificar. Esta fue la motivación para pensar en un algoritmo que reduzca el número de parámetros de la máquina de aprendizaje y eso se logra con algo llamado convolución y pooling, donde el primero es la operación base de este tipo de redes, la cual explicaremos más adelante y el segundo es un "downsampling", el término inglés de tomar un muestreo para reducir la cantidad de datos a procesar. Esta reducción del número de parámetros permite aumentar el número de capas, en otras palabras, permite incrementar la profundidad, conviertiéndose en un método de "deep learning" o aprendizaje profundo.

A continuación se explica la derivación de la operación convolución a partir de una capa completamente conectada [51].

Supongamos que se tienen k neuronas en la primer capa oculta, cada neurona H_i^1, $i = 0, ..., k$ está conectada a todos los pixeles, por lo que resulta en $W \times H$ conexiones donde W denota el número de pixeles.

Supongamos también que se tienen imágenes de 16×16 en escala de grises, es decir, una sola matriz de 16×16, conectada a una capa con 7200 neuronas, donde el elemento de entrada es pensado, como se mencionaba antes, aplanado, con $16 \times 16 = 1024$ entradas.



(a) Fotografía de una rana (b) Ampliación de la fotografía

Figura 2.9: (a) Se muestra la fotografía de una rana de tamaño 32×32 del conjunto de datos de cifar [56] y (b) una ampliación de la fotografía la cual permite distinguir los pixeles.

Por lo tanto, existen $1024 \times 7200 = 7,372,800$ parámetros (pesos) distintos.

Por lo que para reducir significativamente esta cantidad reacomodamos las neuronas en 50 bloques de 12×12 neuronas cada uno. Ahora, consideremos la geometría de los pixeles. Ya que los pixeles tienen una correlación principalmente con sus vecinos más cercanos, podemos aplicar cada bloque a toda la imagen operando por regiones. De manera que el bloque va a operar sobre las regiones de la imagen y pasando de una región a otra por medio de saltos. El salto usualmente se asigna de tamaño 1, como el ejemplo en la figura (2.10).

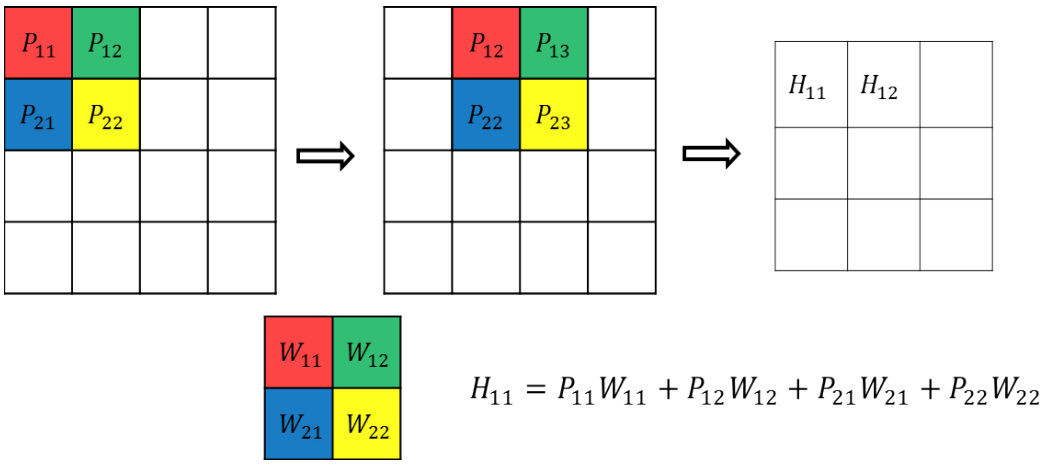


Figura 2.10: Se muestra el desplazamiento por un salto de tamaño 1 de un bloque de neuronas de 2×2 al operar sobre una imagen, nótese que una neurona estará operando sobre una región de 3×3.

De esta forma cada neurona en un bloque aprende solamente de la región por la cual

está operando. Si seguimos el movimiento de una neurona a través de la imagen de 16×16 el tamaño de las regiones donde opera una sola neurona es de 5×5 pixeles, lo que equivale a tener 25 pesos por neurona.

Si hacemos el recuento de los parámetros, tenemos en total $(5 \times 5) \times 50 \times 12 \times 12 = 180,000$. De esta forma se realizó una reducción considerable y más manejable por la computadora.

Es posible reducir aún más el número de parámetros si suponemos que cada neurona en el mismo bloque tiene los mismos valores en los pesos, por lo que esto reduce a $5 \times 5 \times 50 = 1250$ el número de parámetros. A esta técnica se le llama "pesos compartidos", es decir todas las neuronas en un sólo bloque comparten los mismos pesos, por lo que en un bloque de neuronas se tienen 25 valores de pesos distintos.

Consideremos la siguiente notación. Sea $f^l_{p,q}$ la salida de la neurona (p, q) en el bloque l, por lo que el elemento de salida de la neurona está dada por

$$f^l_{p,q} = g\left(\sum_{i=0}^{4}\sum_{j=0}^{4} im(p+i, q+j)\omega^l_{i,j}\right), \tag{2.17}$$

donde g es la función de activación, $\omega^l_{i,j}$ es el peso asociado a la neurona en el bloque l y $p, q = 0, ..., 11$. Nótese que las sumas van de 0 a 4 en representación de las regiones de 5×5 definidas anteriormente. El *campo receptivo* es la región a la cual una neurona está conectada. El elemento de salida por bloque será una matriz de 12×12, el cual conserva las mismas dimensiones de los bloques. Ahora denotando la matriz de salida del bloque l por f^l, esta matriz podemos calcularla haciendo la siguiente operación

$$\mathbf{f}^l(p, q) = g\left(\sum_{i=0}^{4}\sum_{j=0}^{4} im(p+i, q+j)\omega^l_{i,j}\right) \text{ con } p, q = 0, ..., 11. \tag{2.18}$$

Para las personas que estudian óptica es fácil reconocer que la operación anterior se llama convolución. A la matriz de pesos implicada en la operación anterior se le llama filtro, el cual es de 5×5. Por lo que ahora estamos considerando 50 filtros de 5×5, cada uno convolucionándose con la imagen de entrada.

Al proceso que se lleva acabo cuando se convolucionan todos los filtros y se obtienen todos los elementos de salida de cada filtro se le llama capa de convolución. Por lo que la salida de una capa de convolución, llamado mapa característico, será una serie de imágenes donde la cantidad de éstas corresponde a la cantidad de filtros implicados en la convolución. Posteriormente se aplica la función de activación a cada mapa característico de manera independiente. Por tanto, si el tamaño de las imágenes es de $B \times H$ y la capa convolucional comprende de L filtros de tamaño $M \times N$, la salida de la capa convolucional serán L imágenes de tamaño $B - M + 1 \times H - N + 1$ cada una. Cada imagen se obtiene al convolucionar el filtro correspondiente con la imagen de entrada.

Aunque se tienen imágenes con tres canales (RGB) el proceso de convolución nos resulta en una sola imagen y no 3 por lo que se integran los componentes. Por ejemplo, supongamos que se tiene una imagen de entrada $I \in \mathbb{R}^{B\times H\times c}$ y el filtro se diseña de tal forma que la convolución $I * f \in \mathbb{R}^{B\times H\times 1}$. Esto quiere decir que f debe ser un filtro 3-dimensional donde la tercera dimensión es siempre igual al número de canales c. Formalmente, si $f \in \mathbb{R}^{b\times h\times c}$ entonces $I * f \in \mathbb{R}^{B-b+1\times H-h+1\times 1}$. Este hecho, permite diseñar varias capas convolucionales de manera que se va transformando la imagen inicial a través de las diferentes capas ya que, la salida de un capa produce más imagenes correspondientes al número de canales del elemento de entrada en esa capa. Por lo cual, si L es el número de filtros, este corresponde al número de canales de la imagen entrada de la siguiente capa.

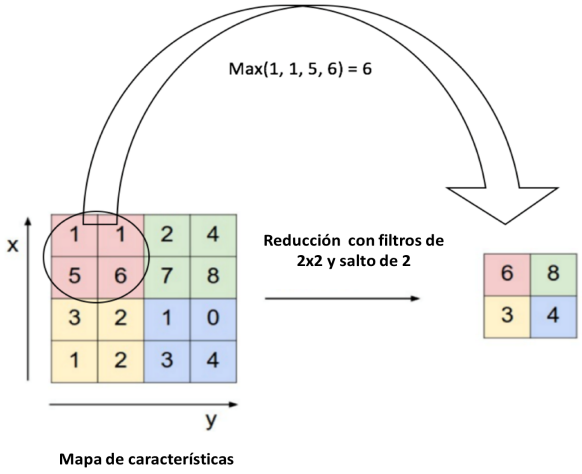


Figura 2.11: Se muestra el diagrama de un ejemplo de "Max pooling" con salto de 2 [59].

Después de obtener la salida de una capa de convolución podemos aplicar un downsampling a cada filtro de forma separada. Para ello suele utilizarse el "Max Pooling" (ver figura 2.11) el cual toma el máximo valor de un grupo de datos reduciendo el tamaño de los filtros. También puede utilizarse el "Average pooling" el cual toma el promedio del grupo de datos, sin embargo Scherer [57] demostró que el max pooling arroja mejores resultados. Por otro lado, investigadores como Springenberg [58] han cuestionado el uso de estas capas de pooling, demostrando que pueden ser reemplazadas por una capa convolucional con valor de salto igual a 2, por lo que tendríamos una arquitectura más simple compuesta por sólo capas convolucionales.

Ahora si, podemos completar la estructura de la red convolucional. Una vez completado el proceso de convolución y downsampling a través de las capas de convolución y pooling respectivamente, se procede introduciendo una o varias capas de neuronas ocultas totalmente conectadas como en la red de propagación hacia adelante usual, de manera que estas capas aprendan las características extraídas en la etapa de convolución. Finalmente, se crea la capa de salida donde se registrarán los resultados de la clasificación en cuestión.

En la figura 2.12 se muestra el diagrama de la arquitectura de una red convolucional por etapas. La primer etapa es la de la convolución donde se extraen las características de las imágenes y en la segunda etapa se clasifican dichas características.

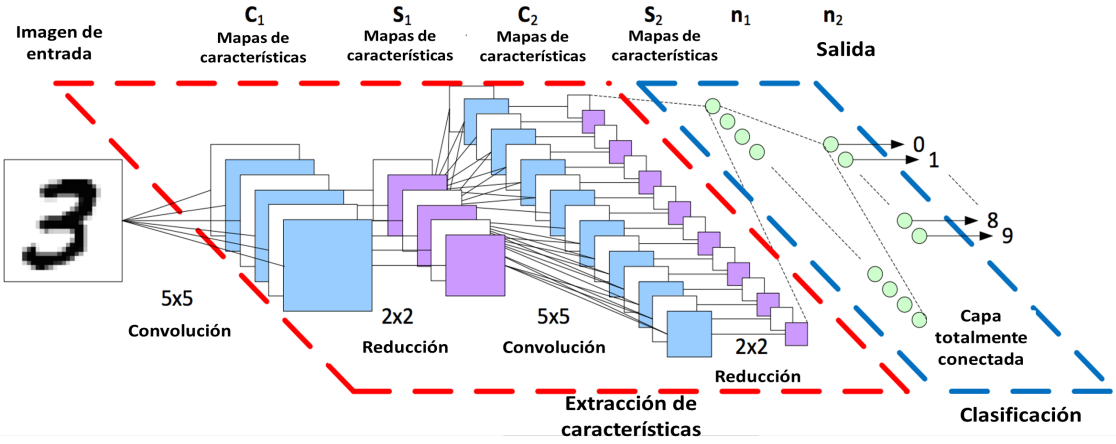


Figura 2.12: Se muestra el diagrama de la arquitectura de la red convolucional [60].

El entrenamiento de las redes convolucionales es exactamente igual al de las redes de propagación hacia adelante, simplemente tomando en consideración la función convolución y el downsampling al momento de calcular el gradiente usando el método de back-propagation, de la misma manera que en los cálculos realizados en la sección (2.1). Para ver los cálculos paso a paso ver [61].

A través de la experiencia, en la implementación de las redes resultó común el sobreajuste, es decir, deficiencia a la hora de generalizar lo aprendido por la red. Esto llevó a la creación de un artificio llamado dropout [62], literalmente significa deshechar. Y dicho artificio consiste en deshechar neuronas con sus respectivos pesos aleatoriamente durante el entrenamiento, resolviendo el problema de recurrir a la variación de versiones de la red para observar y combinar su desempeño, ya que estas pruebas suelen ser costosas en tiempo. Este método para evitar el sobreajuste mejora de manera significativa el desempeño de las redes profundas [62] y en particular, de las redes convolucionales profundas.

2.3.1. Conjunto de datos de CIFAR-10

Una vez explicado el funcionamiento y la gran ventaja que pueden presentar las redes neuronales convolucionales para clasificar imágenes y al mismo tiempo contrastar su desempeño respecto al método de los K-vecinos más cercanos, se presenta su aplicación sobre el conjunto de datos de CIFAR-10. Se utilizó la librería TFlearn [32] para su implementación en *python3*.

La arquitectura de la red propuesta será hecha con base en la regla *rule of thumb* (principio básico o guía basada en la experiencia), la cual corresponde a elegir alguna configura-

ción previa con la cual se hayan resuelto problemas similares con otros conjuntos de datos. En este caso utilizaremos como base la arquitectura creada por LeCun, LeNet-5 [63] para reconocimiento de dígitos la cual comprende de 2 capas de convolución, dos capas de max pooling y una capa totalmente conectada.

Se comenzó fijando valores de hiperparámetros usuales esta vez sin usar VC, ya que en este método de aprendizaje de máquina las posibilidades dentro de la configuración de los parámetros son significativamente mayores que el caso de los K-vecinos más cercanos en el cual únicamente se variaba un sólo parámetro. En cuanto a las redes neuronales convolucionales podemos elegir el número de capas, el tipo de capa, el valor de la razón de aprendizaje, el número de filtros en las capas de convolución, el tamaño del salto en la convolución, el tamaño del filtro del pooling, la función de costo, el número de neuronas en las capas totalmente conectadas, el valor de probabilidad en el dropout, etc. Nótese la complejidad de este tipo de redes en cuanto a la elección de los parámetros. Por esa razón en este ejercicio nos enfocamos únicamente en obtener un resultado razonable, no el mejor. A continuación se muestra el valor de los parámetros y se describe la arquitectura de la red implementada.

La red convolucional consta de dos fases de extracción de características y una de clasificación. Cada fase de extracción tiene una capa de convolución y una capa de *max pooling*, la primera capa de convolución tiene 32 filtros de 3×3 y la segunda tiene 64 filtros de 3×3. Ambas capas de *max pooling* tienen filtros de tamaño 2×2. Después de cada capa de convolución se aplica la función de activación *ReLu*. Después de las dos fases de extracción se tiene una capa totalmente conectada con 512 neuronas y función de activación ReLu. Posteriormente, se utiliza el método de *dropout* donde las neuronas tienen una probabilidad de 0.8 de quedarse activas. Finalmente se tiene otra capa totalmente conectada correspondiente a los valores de salida con 10 neuronas, una para cada clase con función de activación *Softmax*. Para el entrenamiento se utilizó el optimizador *Adam* de la librería *TFlearn*, una razón de aprendizaje de 0.001 y a la función Categorical Cross-entropy como función de costo.

La función de costo Categorical Cross-entropy se define como

$$E[w] = -\sum_{i,\mu} y_i^\mu \log\left(O_i^\mu\right), \tag{2.19}$$

donde la simbología correspode a la utilizada en la ecuación (2.3). La función de costo también afecta el desempeño de una RNA e investigaciones actuales tratan de buscar mejores funciones de costo que ayuden a incrementar la precisión y perfeccionar el desempeño de una RNA [64].

En cuanto a la partición de los datos se dividió el grupo de entrenamiento para obtener el grupo de validación y un nuevo grupo de entrenamiento elegido de manera aleatoria. Obteniendo la siguiente configuración: el grupo de entrenamiento con $50,000$ ejemplos, donde el $5\,\%$ corresponde al conjunto de validación y el grupo de predicción con $10,000$ ejemplos.

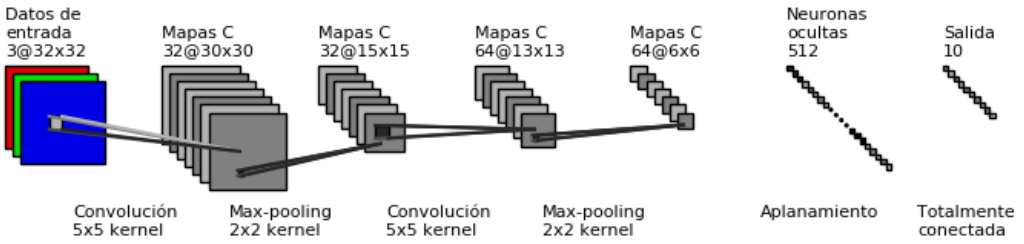


Figura 2.13: Se muestra el diagrama [65] de la arquitectura de la red convolucional implementada para clasificar los datos del conjunto CIFAR-10.

Se inició con 20 ciclos de entrenamiento pero al graficar el error de los conjuntos de entrenamiento y validación, se presenta nuestro semáforo rojo; es decir, es suficiente detener el entrenamiento después del quinto ciclo, ya que después de este la red deja de aprender sobre nuevos datos. La gráfica de ambos errores se muestran en la figura 2.14. Después de entrenar la red se evalúa sobre el grupo de predicción obteniendo nuestro resultado final. Se corrió el código 10 veces para obtener una precisión promedio sobre los conjuntos de entrenamiento, validación y predicción.

Los resultados en cada corrida se muestran en la tabla 2.3, donde la mejor corrida obtuvo una precisión de 73.18 % sobre el conjunto de predicción y una precisión promedio de 72.218 %.

Es claro como el método supera el de los K-vecinos más cercanos. Para mejorar la precisión de la red sobre este conjunto se han implementado el "deep learning" donde el número de capas aumenta en gran número. A su vez, se han implementado técnicas de búsqueda de modelos así como nuevos algoritmos y diferentes tipos de *pooling* [66]. La precisión obtenida es satisfactoria, dada la complejidad del conjunto de imágenes y la sencillez de la estructura elegida para la RNC.

2.3.2. Conjunto de datos de Mnist

Por completez, se realizó la clasificación del conjunto de datos de Mnist usando las RNC con la las funciones internas de *TFlearn*. El tamaño de las imágenes es de 28×28. La red convolucional utilizada para este conjunto consta de dos fases de extracción de características y una de clasificación. Cada fase de extracción tiene una capa de convolución y una capa de *max pooling*, la primera capa de convolución tiene 32 filtros de 2×2 y la segunda tiene 64 filtros de 2×2. Ambas capas de *max pooling* tienen filtros de tamaño 2×2. Después de cada capa de convolución se aplica la función de activación *ReLu*. Después de las dos fases de extracción se tiene una capa totalmente conectada con $1,024$ neuronas y

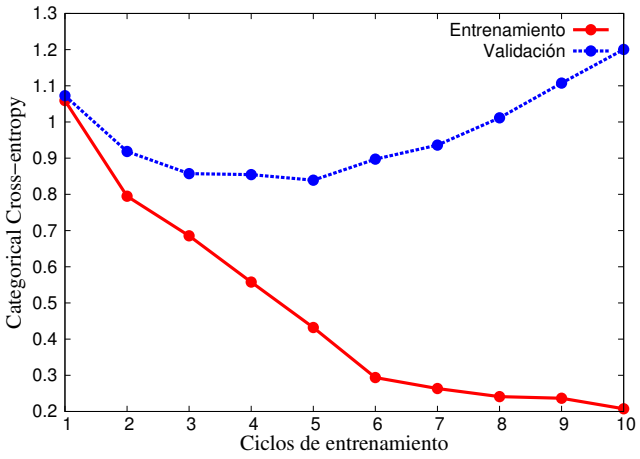


Figura 2.14: Se muestra la gráfica del error en función del número de ciclos de entrenamiento de los conjuntos de validación y de entrenamiento del conjunto de imágenes de CIFAR-10 donde el conjunto de entrenamiento comprende 47,500 imágenes y el de validación está compuesto por 2,500.

Tabla 2.3: Precisión de la red después del entrenamiento sobre los conjuntos de entrenamiento y predicción en cada corrida junto con el promedio.

Corrida	Entrenamiento (%)	Predicción (%)
1	85.43	71.13
2	87.15	71.8
3	87.53	72.73
4	86.7	72.35
5	86.97	72.62
6	85.51	72.35
7	84.87	72.52
8	88.36	73.18
9	84.65	72.38
10	86.62	71.84
Promedio	86.379	72.218

función de activación *ReLu*. Nuevamente se utilizó el método de *dropout* con el mismo valor de 0.8 de probabilidad de las neuronas para quedarse activas. Finalmente se tiene otra capa totalmente conectada correspondiente a los valores de salida con 10 neuronas una correspondiente a cada dígito y función de activación *ReLu*. Esta vez, para el entrenamiento se decidió utilizar otro optimizador, esto con la intención de tratar de tener mayor similitud con la clasificación realizada por la *FFNN*, el optimizador utilizado fue el *sgd* (stochastic gradient descent) extraído de las funciones internas de *TFlearn*, una razón de aprendizaje de 0.05 y a la función Categorical Cross-entropy como función de costo (no se utilizó el error cuadrático ya que la clasificación se veía afectada gravemente, al igual que el tiempo de entrenamiento). El entrenamiento se realizó por mini-lotes de tamaño 128 (default).

El entrenamiento tuvo una duración de 10 ciclos. Se realizó una corrida obteniendo una precisión sobre el conjunto de predicción de 98.16 %, el cual resulta ser un excelente resultado, dado que se aplicó el método más sencillo para elegir la arquitectura de la RNC.

Así queda demostrada la utilidad de este tipo de redes neuronales. A continuación se presenta el capítulo que aborda otro método de inteligencia artificial que también ha sido utilizado con mayor frecuencia en los últimos años.

Capítulo 3

Máquinas de Soporte Vectorial

Otro de los métodos más utilizados recientemente, en el ámbito de la clasificación con el aprendizaje de máquina son las máquinas de soporte vectorial MSV (SVM en inglés). En varias aplicaciones del reconocimiento de patrones, como reconocimiento de dígitos [67], reconocimiento de objetos [68], identificadores de voz [69], la detección de quarks [31] , detección de rostro en imágenes [**?**, 70] y categorización de texto [72]. En la mayoría de estos casos, su rendimiento iguala o supera significativamente a otros [31]. Su desarrollo se llevó a cabo al final de los 70 por Vapnik [67], inspirado en el problema de clasificación.

El método utiliza el concepto de superficie de decisión para llevar a cabo la clasificación, por ejemplo dado un conjunto de datos en dos dimensiones

$$D = \{(5, 1), (6, -1), (7, 3), (1, 7), (2, 8), (3, 8)\}, \tag{3.1}$$

cada uno perteneciendo a una clase $C = \{-1, 1\}$, es decir una clasificación binaria. Supongamos que se tienen los datos clasificados de la siguiente manera, los primeros tres puntos pertenecen a la clase $\{1\}$ y el resto a la clase $\{-1\}$, como se muestra en la figura 3.1.

Ahora con los datos proporcionados queremos clasificar algún otro punto dado posteriormente, por lo que la idea intuitiva es separar los datos de la siguiente forma, usando una línea recta, de manera que todos los puntos que aparezcan del mismo lado serán de la misma clase, dividiendo entonces el espacio en dos partes cada una correspondiente a una clase. A este tipo de datos para los cuales existe dicha recta se les llama separables linealmente. Establecida una recta con esta característica de separar los tipos de datos, se convierte en una línea de decisión, que su análogo tres-dimensional corresponde a un plano de decisión y finalmente la generalización dimensional es una superficie de decisión.

En la tarea de clasificar estos nuevos puntos nos enfocamos en encontrar "la línea recta" que mejor separe los datos para no tener errores al momento de clasificar. Esto puede entenderse mejor si suponemos que alguno de los datos está muy cerca de la línea recta y posteriormente aparece otro dato similar, de la misma clase que el primero pero el cual cae del lado contrario de la línea recta, por ende, estos dos datos realmente pertenecen a la misma clase; sin embargo, la recta indica que son de clases distintas, por lo que quedaría

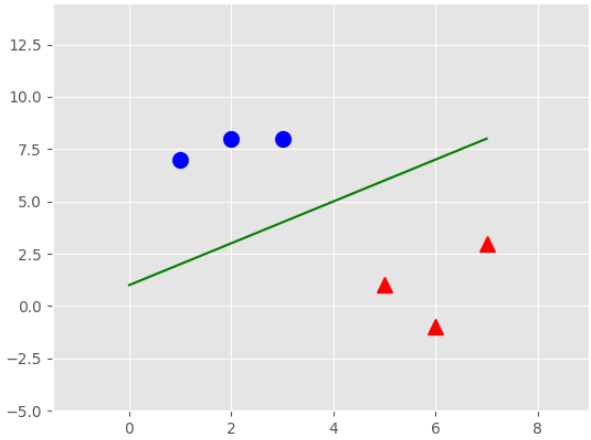


Figura 3.1: Se muestran dos tipos de datos en el plano cartesiano, las coordenadas (x, y) son los atributos o características de los datos, cada uno perteneciendo a una clase, $\{1\}$(círculos azules) o $\{-1\}$(triángulos rojos).

clasificado erróneamente. Tratando de evitar este tipo de situaciones lo que se busca es que la línea recta quede lo más alejada posible de los datos, pues esto mejora la capacidad de generalizar. Cuando hablamos de encontrar el mejor, el mayor o el menor valor, entramos al ámbito de la optimización, y ese es nuestro problema principal. Para encontrar la mejor línea recta se tiene que resolver un problema de maximización de distancia, dadas ciertas condiciones. A esta distancia la llamamos margen y es la distancia más corta de la superficie de decisión al punto más cercano de la clase positiva y de la clase negativa.

3.1. Caso Separable Linealmente

Por simplicidad se analiza en principio, el caso separable linealmente. Veamos ahora la matemática del problema [31]. Recordemos que el conjunto de datos es separable linealmente, a este caso se le conoce por "hard margin". Sea $D = \{\vec{x_1}, \vec{x_2}, ..., \vec{x_N}\}$ un conjunto de datos y sus respectivas clases $C = \{y_1, y_2, ..., y_N\}$, donde $y_i \in \{-1, 1\}$ y $\vec{x_i} \in \mathbb{R}^d$. Supongamos que ya elegimos un hiperplano que separa las muestras por clase. La ecuación que obedecen los puntos en el hiperplano es

$$\vec{w} \cdot \vec{x} + b = 0, \tag{3.2}$$

donde $\vec{w}$ es el vector característico del hiperplano, normal a su superficie, por lo que la distancia perpendicular del hiperplano al origen estará dada por

$$\vec{w} \cdot \vec{x} + b = 0 \tag{3.3}$$
$$\|w\| \cdot \|x\| = -b \tag{3.4}$$
$$\|x\| = \frac{|b|}{\|\vec{w}\|}, \tag{3.5}$$

usando normas euclidianas. Llamaremos d al margen el cual es igual a la suma de las distancias del hiperplano a los puntos más cercanos de la clase positiva y negativa.

Ahora establecemos las condiciones que satisfacen los puntos en las dos clases

$$\vec{x_i} \cdot \vec{w} + b \geq +1 \quad \text{para } y_i = +1 \tag{3.6}$$
$$\vec{x_i} \cdot \vec{w} + b \leq -1 \quad \text{para } y_i = -1. \tag{3.7}$$

Se fusionan ambas condiciones en una sola ecuación

$$y_i(\vec{x_i} \cdot \vec{w} + b) - 1 \geq 0 \quad \forall i. \tag{3.8}$$

Podemos definir dos planos los cuales quedan descritos usando la igualdad en las ecuaciones anteriores (3.6) y (3.7).

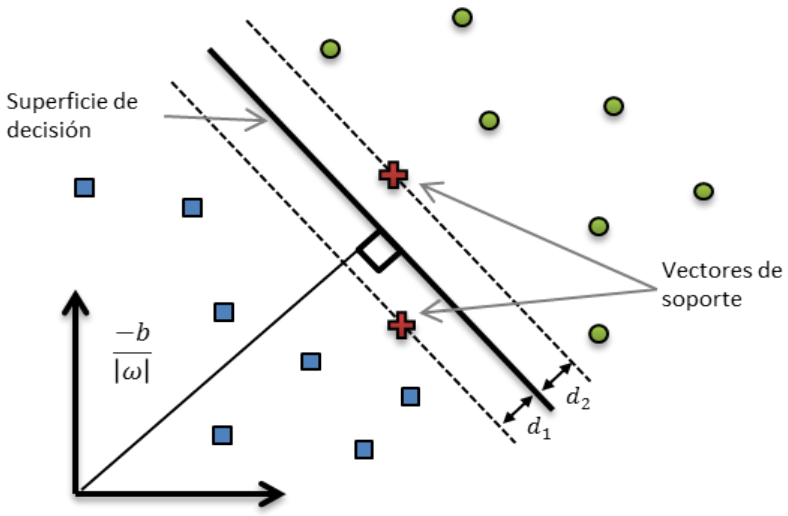


Figura 3.2: Se muestra el diagrama de la superficie de decisión dado un conjunto de puntos de dos clases distintas (cuadros azules:-1, círculos verdes:+1). Los vectores de soporte están indicados por cruces rojas, uno por cada clase.

Estos hiperplanos poseen una distancia al origen de $\frac{|1-b|}{\|\vec{w}\|}$ y $\frac{|-1-b|}{\|\vec{w}\|}$ respectivamente, los cuales son paralelos al hiperplano de decisión. Y su distancia al hiperplano de decisión está dada por, d_1 y d_2 (ver Figura 3.2) respectivamente es

$$d_1 = \frac{1-b}{\|\vec{w}\|} - \frac{-b}{\|\vec{w}\|} = \frac{1-b+b}{\|\vec{w}\|} = \frac{1}{\|\vec{w}\|} \tag{3.9}$$

$$d_2 = \frac{-b}{\|\vec{w}\|} - \frac{-1-b}{\|\vec{w}\|} = \frac{-b+1+b}{\|\vec{w}\|} = \frac{1}{\|\vec{w}\|}. \tag{3.10}$$

Por tanto el margen d está dado por

$$d = \frac{2}{\|\vec{w}\|}. \tag{3.11}$$

Así podemos encontrar un par de hiperplanos que nos den el máximo margen al minimizar $\|\vec{w}\|^2$ sujeto a las condiciones dadas en la ecuación (3.8). Notemos que hay un punto en cada uno de los hiperplanos de la figura 3.2, donde a las posiciones vectoriales dadas por estos puntos se les llama *vectores de soporte*. El nombre lo reciben ya que si los quitamos del conjunto de entrenamiento de alguna manera la solución queda alterada completamente. Por otro lado, si se retira cualquier otro punto que no sea vector de soporte, la solución es la misma. Se dice que estos vectores son el soporte de nuestra solución.

Después de planteado un problema de clasificación como el anterior se trata de extender el método para aplicar la máquina a problemas más complejos, de dimensiones mayores y no-separables linealmente. El siguiente paso es transformar nuestra formulación del problema a una Lagrangiana. Esto se hace por dos razones importantes.

- La primera es que las condiciones serán reemplazadas por las condiciones sobre los multiplicadores de Lagrange, lo cual facilita el problema.
- La segunda razón es porque los datos de entrenamiento en esta formulación aparecen en forma de productos internos de vectores. Esta característica especial es la que nos permite generalizar al caso no lineal, el cual se aborda más adelante.

Por lo tanto, se introducen los multiplicadores de Lagrange positivos α_i, $i = 1.\cdots, l$, uno para cada condición de desigualdad en la ecuación (3.8). Dado que las condiciones son de la forma $c_i \geq 0$, las condiciones son multiplicadas por multiplicadores positivos y sustraídos de la función objetivo obteniendo la función lagrangiana. Para las ecuaciones con igualdad, los multiplicadores de Lagrange son no-condicionados. Esto resulta en

$$L_p \equiv \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{l} \alpha_i y_i (\vec{x}_i \cdot \vec{w} + b) + \sum_{i=1}^{l} \alpha_i. \tag{3.12}$$

La solución al problema está determinada por el punto silla de la función lagrangiana que debe ser minimizada respecto a $\vec{w}$ y b, es decir,

$$\frac{\partial L_p(\vec{w}, b, \alpha)}{\partial \vec{w}} = 0 \tag{3.13}$$

$$\frac{\partial L_p(\vec{w}, b, \alpha)}{\partial b} = 0. \tag{3.14}$$

Aplicando la primera condición y reescribiendo los términos se obtiene

$$\vec{w} = \sum_{i=1}^{l} \alpha_i y_i \vec{x_i}. \tag{3.15}$$

Ahora aplicando la segunda condición, resulta

$$\sum_{i=1}^{l} \alpha_i y_i = 0. \tag{3.16}$$

Es posible demostrar que resolver este problema de optimización es equivalente a resolver el problema llamado "dual" [31]. Por lo que el problema del entrenamiento en las máquinas de soporte vectorial se reduce a maximizar L_D,

$$\sum_{i} \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j \vec{x_i} \cdot \vec{x_j}. \tag{3.17}$$

Se maximiza L_D respecto a los α_i, sujeto a las condición

$$\sum_{i} \alpha_i y_i = 0, \tag{3.18}$$

junto con la condición de que los α_i sean positivos. Notemos que se tiene el mismo número de multiplicadores que puntos de entrenamiento. Y además, en la solución, aquellos puntos tales que su multiplicador sea mayor a cero, corresponden a vectores de soporte, y caen sobre alguno de los hiperplanos descritos usando la igualdad en las ecuaciones (3.6) y (3.7). Cualquier otro punto tiene $\alpha_i = 0$ y pueden estar o no sobre los hiperplanos mencionados anteriormente, donde la igualdad o desigualdad en (3.8) permanece respectivamente.

Es importante hacer énfasis en que los vectores de soporte son los elementos decisivos del conjunto de entrenamiento. Estos se encuentran cerca de la frontera de decisión y si todos los puntos son removidos excepto los vectores de soporte y se repitiera el entrenamiento, se obtendría la misma solución, es decir, se obtendría el mismo hiperplano de separación.

Se propone resolver un ejemplo en el cual se tienen datos linealmente separables. Sea el conjunto de datos D (3.1) y su correspondiente clasificación C.

Por inspección geométrica podemos inferir que el problema se puede solucionar de manera trivial. A continuación se presentan los cálculos paso a paso y posteriormente se presenta la solución generada por código escrito desde cero en el lenguaje *python* para los casos separables linealmente.

Primero diferenciamos a las dos clases distintas. La clase positiva ($\{+1\}$)

$$\{(5, 1), (6, -1), (7, 3)\} \tag{3.19}$$

y la clase negativa ($\{-1\}$)

$$\{(1,7),(2,8),(3,8)\}\,. \tag{3.20}$$

Se desea encontrar una solución de MSV que separe los datos correctamente. Se tiene un caso lineal separable, por inspección en la figura 3.1, de todos los vectores $\vec{x_i}$ tenemos dos vectores de soporte vs_1 y vs_2, uno para cada clase,

$$\{vs_1 = (3,8), vs_2 = (5,1)\}\,. \tag{3.21}$$

Dada la ecuación (3.3) por conveniencia aumentamos una coordenada a cada punto (correspodiente al bias, el término constante) la cual será igual a 1. Por lo tanto $\tilde{vs}_1 = (3,8,1)$. Por lo que el problema se reduce a resolver el siguiente conjunto de ecuaciones dadas en términos de la ecuación (3.17),

$$\alpha_1 \tilde{vs}_1 \cdot \tilde{vs}_1 + \alpha_2 \tilde{vs}_2 \cdot \tilde{vs}_1 = -1 \tag{3.22}$$
$$\alpha_1 \tilde{vs}_1 \cdot \tilde{vs}_2 + \alpha_2 \tilde{vs}_2 \cdot \tilde{vs}_2 = +1. \tag{3.23}$$

Calculamos los productos internos de las ecuaciones anteriores, sustituyendo los valores correspondientes para encontrar los multiplicadores α_i

$$\alpha_1 \begin{pmatrix} 3 \\ 8 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 8 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 5 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 8 \\ 1 \end{pmatrix} = -1$$
$$\alpha_1 \begin{pmatrix} 3 \\ 8 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 5 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 1 \\ 1 \end{pmatrix} = +1,$$

de donde se obtiene

$$74\alpha_1 + 24\alpha_2 = -1 \tag{3.24}$$
$$24\alpha_1 + 27\alpha_2 = +1. \tag{3.25}$$

El sistema de ecuaciones anterior da como resultado los siguientes valores

$$\alpha_1 = \frac{-17}{474}, \quad \alpha_2 = \frac{49}{711}. \tag{3.26}$$

Luego, usamos la ecuación 3.15,

$$\begin{aligned}\vec{\tilde{w}} &= \sum_{i=1}^{l} \alpha_i y_i \vec{\tilde{x}}_i \\ &= \frac{-17}{474}\begin{pmatrix} 3 \\ 8 \\ 1 \end{pmatrix} + \frac{49}{711}\begin{pmatrix} 5 \\ 1 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} \frac{26,623}{112,338} \\ \\ \frac{-24,490}{112,338} \\ \\ \frac{3,713}{112,338} \end{pmatrix}.\end{aligned}$$

Como $\vec{\tilde{w}}$ contiene los valores de $\vec{w}$ y b, se produce la siguiente línea de decisión

$$\vec{w} \cdot \vec{x} + b = 0, \tag{3.27}$$

donde los componentes de los vectores $\vec{x}$ pueden escribirse de la siguiente forma, sustituyendo los valores de $\vec{w}$ y b

$$\begin{aligned} x_1 &= \text{libre}, \\ x_2 &= \frac{-b - x_1 w_1}{w_2}, \\ x_2 &= \frac{-\frac{3,713}{112,338} - x_1(\frac{26,623}{112,338})}{\frac{-24,490}{112,338}}. \end{aligned}$$

Escribiendo la ecuación en su forma ordenada al origen resulta

$$x_2 = \frac{26,623}{24,490} x_1 + \frac{3,713}{24,490} \tag{3.28}$$

La solución (3.28) se muestra en la figura 3.3, si buscamos clasificar los siguientes puntos: $(4.5, 7.5)$ y $(6, -5)$, los ingresamos en la ecuación para ver que condición se satisface

$$\begin{aligned}\vec{w} \cdot \vec{x} + b &= (\frac{26,623}{112,338}, \frac{-24,490}{112,338}) \cdot (4.5, 7.5) + \frac{3,713}{112,338} \\ &= -0.535513361, \end{aligned} \tag{3.29}$$

por lo tanto, el vector $(4.5, 7.5)$ es clasificado en los negativos (-1). Para el vector $(6, -5)$

$$\vec{w} \cdot \vec{x} + b = (\frac{26,623}{112,338}, \frac{-24,490}{112,338}) \cdot (6,-5) + \frac{3,713}{112,338} \tag{3.30}$$

$$= 2.545007032, \tag{3.31}$$

por lo tanto, el vector $(6,-5)$ es clasificado en los positivos $(+1)$. Ambas clasificaciones se muestran en la figura 3.3.

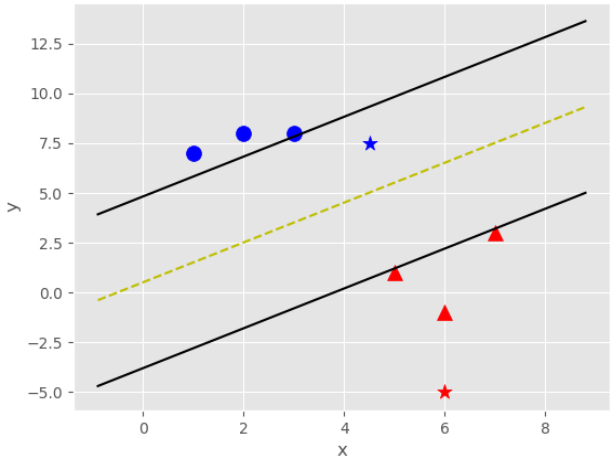


Figura 3.3: Se muestra la gráfica de la línea recta solución al problema propuesto, el margen, junto con los vectores de soporte y los datos restantes, así como dos datos nuevos (estrella) obteniendo su clasificación.

Se usó el lenguaje de python para implementar la optimización y encontrar los valores de b y de $\vec{w}$ que definan a la línea recta de decisión. Encontrando la misma solución, la gráfica generada por la implementación se muestra en la figura 3.3.

3.2. Caso No Separable Linealmente

La primer mejora que ataca la parte del caso no-separable linealmente es la tolerancia de errores. Esto ayuda a generalizar mejor, a cambio de permitir cierto número de errores en la clasificación sobre el conjunto de entrenamiento. Dicha tolerancia se describe mediante una modificación de las ecuaciones (3.6) y)3.7) [67].

$$\vec{x_i} \cdot \vec{w} + b \geq +1 - \zeta_i \quad \text{para} \quad y_i = +1, \tag{3.32}$$

$$\vec{x_i} \cdot \vec{w} + b \geq -1 + \zeta_i \quad \text{para} \quad y_i = -1, \tag{3.33}$$

$$\zeta_i \geq 0 \quad \forall i. \tag{3.34}$$

Por lo tanto para que un error pueda suceder, el valor del ζ_i correspondiente debe ser mayor a la unidad, y así la violación total $\sum_{i=1}^{l} \zeta_i$ es una cota superior del número de errores permitidos de entrenamiento. Reformulando el problema y la función a minimizar conviene establecerse como $\|\vec{w}\|^2/2 + c(\sum_i \zeta_i)$ [67], donde el parámetro c es elegido de forma arbitraria. Un valor "grande" de c corresponde a establecer una baja tolerancia a errores y un valor "pequeño" corresponde a una mayor tolerancia. (Para ver el desarrollo del análisis anterior y el problema Lagrangiano ver [67]).

Cuando se tienen datos no-separables linealmente el uso de mapeos a dimensiones mayores resulta muy útil. A continuación se da una descripción de lo anterior. Primero, notemos que las ecuaciones (3.12) y (3.16) están en la forma de productos internos $\vec{x} \cdot \vec{y}$. Supongamos que aplicamos un mapeo hacia otro espacio Euclideano, sin restricción en las dimensiones, $\mathcal{H}$, y a este mapeo lo llamamos f,

$$f : \mathbb{R}^d \mapsto \mathcal{H}. \tag{3.35}$$

Llamamos al producto interno de los vectores en el espacio $\mathcal{H}$ la *función kernel* K, tal que, $K(\vec{x}, \vec{y}) = f(\vec{x}) \cdot f(\vec{y})$.

Con esto en mente, procedamos a resolver un ejemplo e ilustrar la utilidad de lo que se acaba de definir.

Sea el vector $\vec{x} = (x_1, x_2)$ definido en $\mathbb{R}^2$ y el vector $\vec{X} = f(\vec{x})$ definido en $\mathbb{R}^3$ tras realizar un mapeo con la función f definida de la siguiente forma

$$f(\vec{x}) = (x_1^3, x_2^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2). \tag{3.36}$$

Contando el número de operaciones necesarias para calcular el producto interno entre dos vectores $\vec{x}$ y $\vec{y}$ pero en el espacio al cual se mapeó, es decir, el número de operaciones necesarias para calcular

$$\vec{X} \cdot \vec{Y} = X_1Y_1 + X_2Y_2 + X_3Y_3, \tag{3.37}$$

entonces se tienen que realizar cuatro operaciones para obtener $\vec{X}$, cuatro más para obtener a $\vec{Y}$ y cinco operaciones para obtener el producto interno, lo cual resulta en un total de 13 operaciones.

Ahora usemos la función kernel, $K(\vec{x}, \vec{y}) = (\vec{x} \cdot \vec{y})^3$, desarrollando

$$\begin{aligned}
K(\vec{x}, \vec{y}) &= (\vec{x} \cdot \vec{y})^3 && (3.38)\\
&= (x_1y_1 + x_2y_2)^3 && (3.39)\\
&= x_1^3y_1^3 + x_2^3y_2^3 + 3(x_1y_1)^2x_2y_2 + 3x_1y_1(x_2y_2)^2 && (3.40)\\
&= (x_1^3, x_2^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2) \cdot (y_1^3, y_2^3, \sqrt{3}y_1^2y_2, \sqrt{3}y_1y_2^2) && (3.41)\\
&= \vec{X} \cdot \vec{Y}. && (3.42)
\end{aligned}$$

Establecida la igualdad, nótese que en el paso dos para calcular el producto interno, ¡sólo se requieren cuatro operaciones! Es más rápido usar la función Kernel. También, cabe mencionar que, podemos realizar el cálculo del producto interno entre los vectores sin realizar el mapeo; es decir, podemos usar la función kernel en nuestro algoritmo de MSV sin conocer f. En la tabla 3.1 se muestran las distintas funciones kernel más utilizadas en el algoritmo de MSV.

Tabla 3.1: Se muestran las distintas funciones kernel utilizadas en problemas no-lineales de clasificación por medio de MSV. De las funciones presentadas, los parámetros kernel son γ, r y d [74].

Funciones kernel	
Lineal	$K_{lin}(\vec{x}, \vec{y}) = \sum_{i=1}^{d} x_i y_i$
Polinomial	$K_{pol}(\vec{x}, \vec{y}) = \left(\sum_{i=1}^{d} x_i y_i + l\right)^p$
RBF	$K_{rbf}(\vec{x}, \vec{y}) = \exp\left(-\gamma \sum_{i=1}^{d} (x_i - y_i)^2\right)$
Sigmoide	$K_{sig}(\vec{x}, \vec{y}) = \tanh(\gamma \sum_{i=1}^{d} x_i y_i + r)$

La generalización para multiclases puede obtenerse del siguiente hecho. El problema de clasificación binario puede resolverse también si se define un vector de peso $\vec{w}_i$ y un valor b_i para cada clase. Por lo que cada vez que se intente clasificar un nuevo dato, se evalúa sobre dos funciones y el punto a clasificar $\vec{x}$ se le asigna a la clase $+1$ si se cumple $\vec{w}_1 \cdot \vec{x} + b_1 \geq \vec{w}_{-1} \cdot \vec{x} + b_{-1}$, si no a la clase -1. Esta forma de clasificar es equivalente a la anterior con un solo hiperplano $(\vec{w}, b)$ si se sustituyen los valores $\vec{w} = \vec{w}_1 - \vec{w}_{-1}$ y $b = b_1 - b_{-1}$. Para un problema de multiclase los valores de salida son $y = \{1, 2, ..., N\}$. Y la generalización queda definida como se muestra a continuación.

A cada una de las N clases se le asocia un vector de peso y un valor bias $(\vec{w}_i, b_i)$, con $i \in \{1, ..., N\}$, y la función de decisión queda definida por

$$\text{clase}(\vec{x}) = \underset{1 \leq i \leq m}{\arg\max} \left(\vec{w}_i \cdot \vec{x} + b_i\right). \tag{3.43}$$

La interpretación geométrica de esta generalización es asociar un hiperplano a cada clase, y asignar al punto $\vec{x}$ a la clase cuyo hiperplano se encuentra más alejado de él [73].

En las siguientes subsecciones para implementar el método de SVM, se decidió utilizar la librería *LIBSVM* [34] la cual facilita la construcción de los códigos, ya que te permite cambiar de parámetros fácilmente, como el kernel utilizado y el valor de las constantes en los kernel, agilizando la búsqueda de un buen modelo [74]. Cabe mencionar que esta librería tiene una función llamada *find_parameters*, la cual realiza una validación cruzada (de 5 iteraciones por default) para encontrar los mejores parámetros en una región definida por el usuario. Esta función tiene como kernel la función *Radial Basis Function* (RBF) por default, por lo que los parámetros a variar son γ y c. Una vez definido los intervalos de búsqueda, se encuentran los mejores parámetros, se guardan y se utilizan para entrenar a la máquina con estos junto con el conjunto de entrenamiento en su totalidad. Además la función muestra la gráfica a tiempo real del espacio de parámetros explorado y al finalizar guarda una imagen de la gráfica en formato *png*.

3.3. Clasificando dígitos con SVM

Mencionado todo lo anterior, somos capaces de resolver el problema de la clasificación de dígitos una vez más. En esta ocasión es turno de las máquinas de soporte vectorial.

La librería *LIBSVM* incluye el conjunto de datos de Mnist en su base de datos de ejemplos. Se utilizó éste y no el extraído de la página de Mnist por simplicidad, ya que el conjunto de *LIBSVM* ya está en el formato necesario para su implementación [74]. Como el conjunto de datos de entrenamiento es de tamaño 60,000, se realiza la búsqueda sobre un subconjunto para ahorrar tiempo de cómputo. La selección del subconjunto es aleatoria. En caso de no funcionar se realizan varias pruebas para variar los subconjuntos y otros parámetros, como el rango de búsqueda de γ y c. El tamaño del subconjunto elegido fue de 10,000 (igual que el conjunto de predicción). El éxito del procedimiento depende del problema en cuestión por lo que en la práctica se suele intentar el procedimiento más sencillo y rápido, después se hacen ajustes o se recurren a otros procedimientos para la selección de parámetros son implementados.

Se realizó una búsqueda de los parámetros con la función *find_parameters()*. Se ajustan los valores tomando en cuenta el logaritmo del parámetro, por lo tanto se realiza la búsqueda en escala logarítmica. El intervalo de búsqueda se realizó en los siguientes intervalos

$$\gamma \in [-5, 5] \text{ con saltos de } 1, \tag{3.44}$$

$$c \in [-5, 5] \text{ con saltos de } 1. \tag{3.45}$$

Por lo que se genera una malla de 5×5. A continuación se presentan los resultados de la elección de parámetros por medio del entrenamiento con el subconjunto aleatorio.

Los valores de los mejores parámetros fueron los siguientes

Tabla 3.3: Se muestra el número de vectores de soporte, la precisión sobre los conjuntos de entrenamiento y predicción resultantes de implementar la clasificación del conjunto de datos de Mnist utilizanndo la librería *LIBSVM*.

Método	Precisón de predicción
K-vecinos más cercanos	97.51 %
Red Neuronal de propagación hacia adelante	96.45 %
Red Neuronal Convolucional	98.16 %
Máquina de soporte vectorial	98.56 %

$$\gamma = 0.03125, \tag{3.46}$$

$$c = 2.0. \tag{3.47}$$

Posteriormente, se realizó el entrenamiento con el valor de los parámetros encontrados sobre el conjunto de entrenamiento en su totalidad y luego se clasificó el conjunto de predicción. Los resultados se encuentran en la tabla 3.2.

Tabla 3.2: Se muestra el número de vectores de soporte, la precisión sobre los conjuntos de entrenamiento y predicción resultantes de implementar la clasificación del conjunto de datos de Mnist utilizanndo la librería *LIBSVM*.

Resultados	
No. de vectores de soporte	16, 353
Precisión de entrenamiento	99.9567 % (59974/60000)
Precisión de predicción	98.56 % (9856/10000)

Los resultados son más que satisfactorios, ya que con un procedimiento sencillo se logró clasificar el conjunto de predicción con una buena precisón. No fue necesario implementar alguna mejora o algún procedimiento sofisticado para clasificar correctamente el conjunto de Mnist. En la tabla 3.3 se muestra la precisón alcanzada por cada método presentado.

Los resultados de la tabla 3.3, muestran la superioridad de cada método sobre los otros, claro está, que puede depender del problema en cuestión. Finalmente, la MSV, resultó ser la mejor sobre el conjunto de Mnist. Es posible incrementar la precisión de cada uno implementando otras técnicas de búsqueda de parámetros o de los detalles matemáticos de cada algoritmo. El número de parámetros ajustables es muy grande por lo que no se aborda ninguno de ellos en este trabajo.

Presentadas ya algunas de las herramientas del aprendizaje de máquina, en el siguiente capítulo se aborda el problema de la clasificación de electrocardiogramas.

Capítulo 4

Electrocardiogramas

En esta sección se aborda el problema de la clasificación de ECG con los métodos de aprendizaje de máquina descritos en los capítulos anteriores. Se procede describiendo cada método con los preprocesamientos correspondientes y resultados.

La señal de ECG está compuesta por una serie de latidos del corazón manifestados como ondas eléctricas en secuencia caracterizadas por picos y valles. Los ECG proveen de dos tipos de información principalmente [79]. Uno es la duración de la onda eléctrica que pasa a través del corazón. El segundo es la amplitud del impulso eléctrico en el tiempo que atraviesa el músculo del corazón, esto ayuda a saber si alguna parte del corazón es muy grande o está esforzándose en exceso [79]. El rango de frecuencia de un ECG es 0.05-100 Hz y su rango voltaico es 1-10 mV. La señal ECG está caracterizada por cinco picos y valles representados por las letras P, Q, R, S, U y T. Una señal que abarca un pulso total sano se presenta en la figura 4.1.

Se tienen parámetros standard para la duración de cada sección así como la amplitud de cada una. Estos son los primeros datos analizados de un pulso cardiaco registrado en el ECG. El ritmo cardiaco usual es de 60 a 100 latidos por minuto [79]. Si la señal ECG tiene algo inusual entonces se indica que se tiene una Arritmia [75, 76].

La clasificación de ECG lleva consigo varios problemas, los cuales son descritos a continuación [13]:

- Falta de estandarización de las características utilizadas en el proceso del aprendizaje. Esto sucede gracias a que no se tienen límites o valores fijos en tiempo o amplitud en una señal de ECG. La precisión depende de las características elegidas por lo que una pequeña variación en estos valores puede resultar en una clasificación errónea cuando se trata de conjuntos de datos muy grandes.

- Variación en las características de los ECG. El ritmo cardiaco de un individuo cambia por el estado fisiológico y mental. Estrés, excitación, ejercicio físico y otras actividades que afectan al ritmo cardiaco.

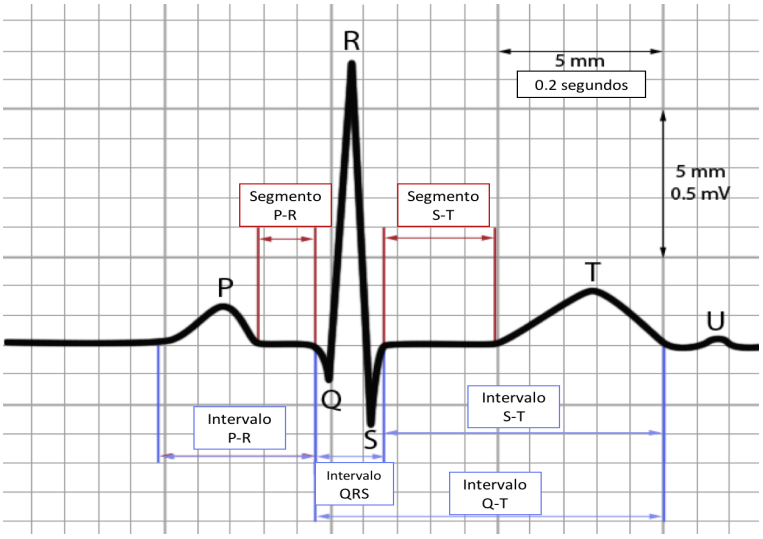


Figura 4.1: Diagrama de un pulso en forma de onda de un latido sano con sus secciones, picos y valles [80].

- Individualidad en los patrones de los ECG. Es decir, se pueden tener variaciones en las que los patrones en una señal sean parecidos a los de otra clase y diferentes a los patrones de otros ejemplos de la misma clase.
- La no-existencia de reglas de clasificación óptimas. (No es muy clara la manera en la que hay que clasificar las señales).
- Los pacientes pueden tener diferentes formas de onda en los ECG. Estos pueden tener distintas pendientes en la señal, distintos tiempos y amplitudes.
- La variación de los pulsos en un mismo ECG. Una señal de ECG puede contener miles de pulsos y pueden ser de diferente tipo (arritmias). Por lo tanto la clasificación tiene que llevarse a cabo con cuidado para minimizar errores.
- Encontrar el clasificador más apropiado. Encontrar tal clasificador es muy difícil debido a la miríada de parámetros de los cuales depende la clasificación, como el tipo de arritmia, la diferente manifestación de la misma, la base de datos con la cual se trabaja, la técnica de extracción de características usada, entre otros.

Existen dos formas de abordar el problema de la clasificación de ECG, una es por medio del análisis por pulsos. La segunda, es por medio del análisis de la señal de ECG. Donde se sabe que un ciclo cardiaco consiste de las ondas P, Q, R, S, T y U los cuales definen un latido o pulso. Por otro lado una señal de ECG contiene a miles de estos pulsos.

La clasificación de ECG está compuesta de cuatro pasos fundamentales: preprocesamiento, extracción de características, normalización y clasificación [13].

En el preprocesamiento se realizan usualmente los siguientes procesos:

- Limpieza de la señal, la cual consiste en eliminar el mayor ruido posible sin perder información genuina perteneciente a la señal de ECG. Por ejemplo, aplicando métodos como el low pass filter o linear phase high pass filter (ver [13]).

- Ajuste de la línea base, la cual se logra utilizando métodos como el median o mean median filter (ver [13]).

Extracción de características:
Para obtener la información que sirve de entrada para los métodos de clasificación se realizan distintas transformaciones para asignar valores a la duración de los intervalos de un pulso cardiaco, por ejemplo. Algunas de estas transformaciones son Discrete Wavelet transform, Continuous Wavelet transform y Discrete Fourier Transform (ver [13]).

Normalización de las características: Para llevar a cabo un entrenamiento adecuado se deben normalizar los valores de entrada, para ello se utilizan los métodos usuales como el z-score y unity standard deviation.

Clasificación: Para la clasificación es posible usar cualquier método de clasificación (máquinas de soporte vectorial, árboles de decisión, redes neuronales, etc.), de los cuales el de las redes neuronales ha sido el más popular en los últimos años. Donde se han realizado clasificaciones por latidos [77], detección de paros cardiacos [30], clasificación de electrocardiogramas usando redes convolucionales [78], entre otros.

El tipo de clasificación elegido fue el de la clasificación por señales y no por latidos. Esto basado en la propuesta del uso de una transformación de una serie de tiempo (señal ECG) a una imagen [25]. Lo cual se piensa que puede ser un atajo en los pasos a seguir para una clasificación de ECG. Ya que se tiene la hipótesis de que la transformación puede ser invariante en la presencia de cierto grado de ruido (no demasiado) y por la naturaleza de la transformación se tienen valores normalizados después de aplicarla. Más adelante se explica en qué consiste dicha transformación.

4.1. Conjunto de datos del Physionet CinC Challenge 2017

El conjunto sobre el cual trabajamos es el de la página de Physionet [15], en particular el lanzado en el concurso de *Computing in Cardiology* versión 2017. Dicho conjunto comprende de $8,528$ grabaciones de señales ECG para entrenamiento y 300 grabaciones para el conjunto de predicción. Las grabaciones son distintas de acuerdo a duración y fueron medidas con una frecuencia de 300 Hz. Se tienen 4 clases distintas en el conjunto, ritmo normal (5,076 señales), ritmo de fibrilación auricular (758 señales), otro ritmo (2,415 señales) y ruido (279 señales). En la figura 4.2 se muestra un ejemplo de cada clase de señal normalizada.

Se descargaron los datos en formato .mat, el cual era posible leer desde *unix* usando la paquetería *Waveform Database* (WFDB) y guardar los datos en formato de texto para leerlos con cualquier lenguaje de programación.

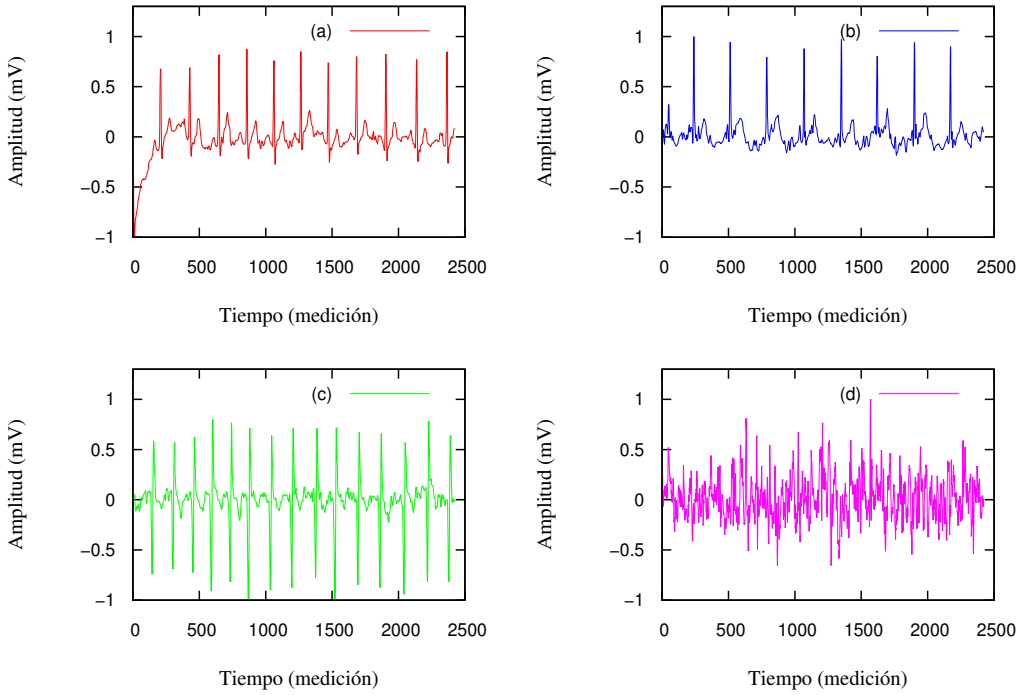


Figura 4.2: Primera mitad de las series de tiempo normalizadas. En (a) se muestra la señal correspondiente a un ECG de clase normal, en (b) se muestra la señal de un ECG perteneciente a la clase Otro, en (c) se muestra una señal ECG de tipo Fibrilación auricular y en (d) se muestra una señal de tipo ruido.

4.2. Preprocesamiento

Se aplica preprocesamiento para obtener señales normalizadas y para tener la misma longitud en cada muestra así como una transformación de series de tiempo a imágenes para ingresarlas a la red convolucional. Se consideran los primeros $1,212$ nodos de entrada por señal, esto nos ayuda a agilizar el entrenamiento. Además, se aplicó otro *downsampling* sobre las señales tomando cada 4 mediciones y deshechando el resto, resultando en una señal de 303 datos de longitud. En el caso de la red neuronal convolucional un paso extra se llevó a cabo, los elementos de entrada fueron convertidos en imágenes de 303×303

usando la matriz GASF [25] y reducidas a imágenes de tamaño final de 50×50 pixeles por medio de la función *imresize()* de Matlab.

Para llevar a cabo la normalización de las señales, se leyeron por medio de python. Cada señal era leída y guardada en una lista, guardando al mismo tiempo el valor máximo en valor absoluto de la señal, luego se dividía por cada valor de la señal. Finalmente se guardaban todas las señales en formato numpy.

4.2.1. Transformación GASF

Dada una serie de tiempo $V = \{v_1, v_2, v_3, ..., v_n\}$ con n valores reales de medición, se reescalan los valores de manera que dichos valores caigan en el intervalo $[0, 1]$ o $[-1, 1]$, obteniendo la serie de tiempo normalizada $\hat{V}$. Por lo tanto podemos representar la serie de tiempo en coordenadas polares con la transformación

$$\begin{aligned} \phi_i &= \arccos(\hat{v}_i), \quad -1 \leq \hat{v}_i \leq 1, \quad \text{con } \hat{v}_i \in \hat{V}, \\ r &= \frac{t_i}{N} \in \mathbb{N}, \end{aligned} \tag{4.1}$$

donde t_i corresponde a la etiqueta del tiempo y N es un factor constante para regularizar la extensión de la serie de tiempo en el plano polar. Esta forma polar de representación es una nueva forma de entender las series de tiempo [25]. Mientras avanza el tiempo los valores correspondientes se deforman en distintos puntos caracterizados por el ángulo. La transformación en la ecuación (4.1) tiene dos propiedades importantes. La primera es que es biyectiva ya que la función $\cos(\phi)$ es monótona en el intervalo $[0, \pi]$. Por lo tanto dada una serie de tiempo, el resultado de la transformación tendrá uno y sólo un mapeo al sistema polar. La segunda propiedad, en contraste a las coordenadas rectangulares, las coordenadas polares preservan relaciones temporales absolutas [25].

Nuestros datos reescalados caen sobre el intervalo $[-1, 1]$ y por lo tanto su tranformación angular cae sobre el intervalo $[0, \pi]$.

Una vez mapeada nuestra serie de tiempo a coordenadas polares es posible explotar la componente angular considerando la suma (o resta) trigonométrica entre cada punto para identificar correlación dentro de los distintos intervalos de tiempo en la serie. Se elige la operación de suma por lo que la matriz a calcular es *The Gramian Angular Summation Field* (GASF), Campo angular de suma Gramiano, por simplicidad.

La matriz GASF está definida de la siguiente forma

$$(GASF)_{ij} = \left[\cos(\phi_i + \phi_j)\right]. \tag{4.2}$$

Para ver más detalles de la matriz GASF y sus propiedades ver [25]. La matriz es grande, ya que tiene dimensiones de $n \times n$ cuando la longitud de las series de tiempo es de n. Para reducir el tamaño de la imagen, es posible usar métodos de reducción como la *Piecewise Aggregation Approximation* (PAA) [81]. Por simplicidad utilizamos la función

imresize() de Matlab, la cual produce nuevos pixeles tomando el promedio del vecindario más cercano de pixeles de tamaño 4×4.

Al aplicar esta transformación sobre las series de tiempo de los electrocardiogramas se obtuvieron imágenes de 303×303 y después se redujeron a 50×50 usando *imresize()*. En la figura 4.3 se muestra la imagen final de cada clase correspondiente a las series de tiempo. Estas imágenes fueron los datos de entrada de la red convolucional.

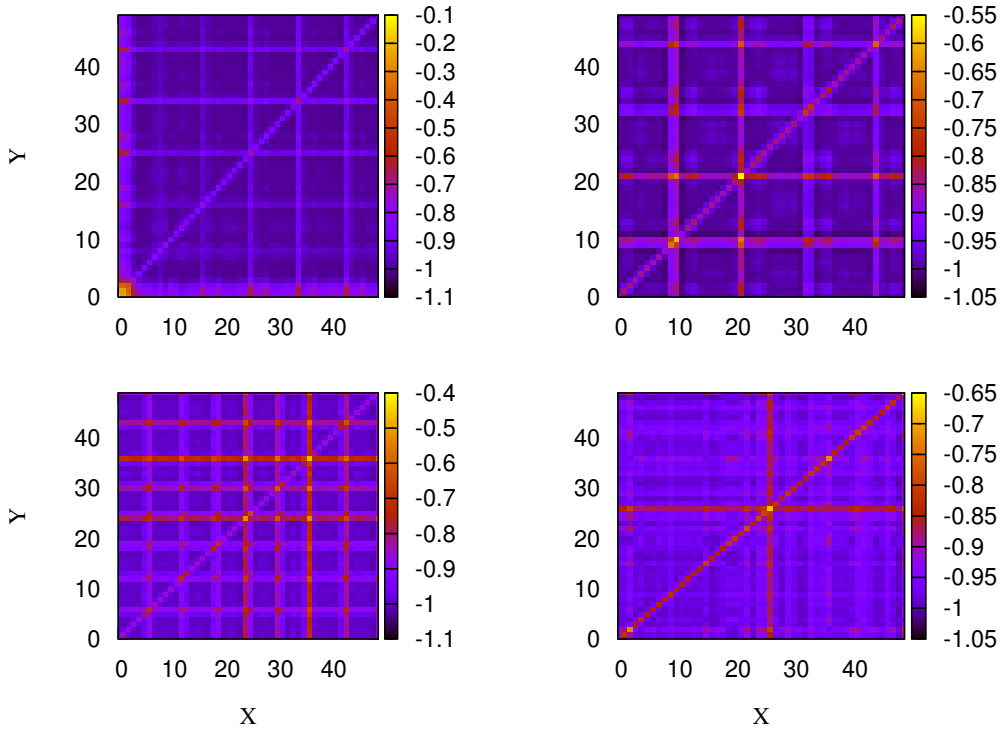


Figura 4.3: Se muestra la imagen final obtenida por medio de la transformación GASF y la reducción de tamaño con Matlab.

4.3. Clasificación

4.3.1. Red Neuronal Feed-Forward

Para este método se utiliza el mismo código que el implementado para resolver el caso del conjunto de Mnist. Las entradas de la red fueron las señales de longitud 303 normalizadas. En la primera prueba se utlizó el conjunto entero de entrenamiento sin grupo de validación y probando la red entrenada sobre el grupo de predicción. En la tabla 4.1 se muestran las características de la arquitectura de la red.

Tabla 4.1: Arquitectura de la red de la primera prueba sobre el conjunto de datos de Mnist.

No. Neuronas de entrada	300
Ciclos	220
Optimización	descenso gradiente
η	0.7
Lote	online batch
Función de activación	sigmoide

Para llegar a esta configuración de los parámetros utilizamos prueba y error. Se varían los parámetros desde valores usuales hasta encontrar la mejor solución aparente. Una vez encontrados los parámetros se procedió a correr el código 10 veces para obtener una precisión promedio ya que algunas soluciones pueden tener oscilaciones en su precisión dependiendo de la inicialización de los pesos. En la tabla 4.2 se muestran los resultados de cada corrida y la precisión promedio.

Se realizó el aumento de los datos de entrenamiento de la siguiente manera. Se tomaron los datos originales y al partir a la mitad las series de tiempo, se tomaron las segundas mitades de las clases con número de elementos más pequeño, es decir, las clases FA, Ruido y Otro. Por lo tanto se tenían los siguientes tamaños en las clases distintas: 5,076 Normal, 1,516 FA, 558 Ruido y 2,415 Otro. Esto con el fin de disminuir en cierto grado el sobreentrenamiento de la RNA sobre la clase Normal.

El número de neuronas se modificó a 350, mientras que el resto de parámetros se mantuvieron con el mismo valor. Se corrió el código 10 veces, de igual manera. Los resultados se muestran en la tabla 4.3.

La tabla 4.3 muestra que el valor de precisión para el conjunto de datos aumentado fue mayor, en promedio, aunque también es notorio que los resultados de predicción son más inestables. La gráfica del error contra ciclos de entrenamiento del mejor resultado de precisón se muestra en la figura 4.4.

4.3.2. Red Neuronal Convolucional

Para realizar el entrenamiento utilizando la RNC, se tomaron las imágenes generadas por la transformación GASF. Nuevamente se aprovechó la libreraía de *tflearn*. Las imágenes de entrenamiento utilizadas tienen una resolución de 50×50. Se entrenó la red sin grupo de validación y se obtiene la precisión de la red sobre el conjunto de predicción de 300 datos, al obtener una precisión suficiente en el conjunto de entrenamiento.

Los hiperparámetros iniciales fueron elegidos con base en los elegidos en el caso del entrenamiento para clasificar el conjunto de dígitos de Mnist, luego, realizando experimentos de prueba y error, analizando las gráficas de error, se ajustaron los parámetros hasta

Tabla 4.2: Precisión de la red después del entrenamiento sobre los conjuntos de entrenamiento y predicción en cada corrida junto con el promedio.

Experimento	Entrenamiento (%)	Predicción (%)
1	94.2	85.6
2	95.4	87.0
3	95.4	87.3
4	95.4	87.0
5	95.5	86.0
6	95.7	89.0
7	95.4	83.3
8	95.3	84.6
9	95.9	89.3
10	95.0	85.3
Promedio	95.3	86.4

Tabla 4.3: Precisión promedio del conjunto de validación en cada configuración.

Experimento	Entrenamiento (%)	Predicción (%)
1	96.6858	93.6667
2	95.8913	94.3333
3	96.069	93.6667
4	96.1317	94.3333
5	95.6194	92.3333
6	92.9012	87.0
7	96.1735	93.6667
8	92.6294	84.6667
9	92.7235	88.0
10	96.4036	96.0
Promedio	95.12284	91.76667

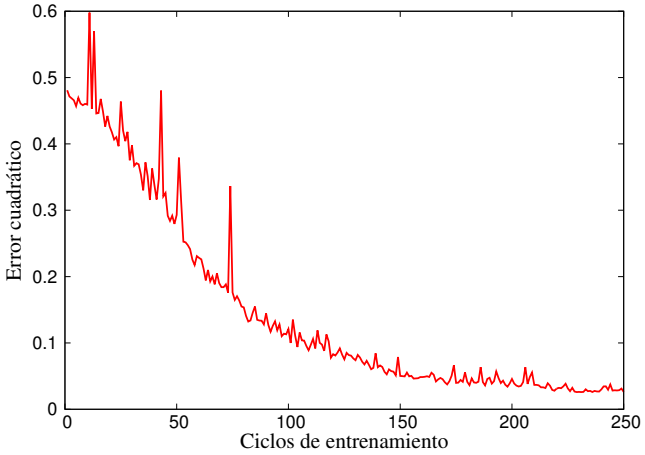


Figura 4.4: Error cuadrático en función de los ciclos de entrenamiento para la FFNN entrenada sobre el conjunto de CinC challenge 2017. Después del ciclo 250 la red no puede aprender más con los datos ingresados y se estabiliza el valor del error.

haber obtenido un resultado satisfactorio en el conjunto de predicción.

Los resultados de la clasificación se muestran en la tabla 4.4, donde se detuvo el entrenamiento una vez que la red dejaba de aprender. La ausencia del grupo de validación se debe a que en la partición aleatoria del conjunto en dos con una parte siendo la de validación, los resultados no fueron negativos, de alguna forma se manifiesta la idea que los datos son necesarios (muy representativos) para el entrenamiento en su totalidad (al menos para esta red). Es decir, los datos eran necesarios para un buen entrenamiento de la red. El experimento se llevó a cabo 10 veces para obtener un promedio, considerando la variación en la asignación de los pesos definidos de forma aleatoria.

En la figura 4.5 se muestra la precisión y el error contra ciclos de entrenamiento, es claro el comportamiento de un buen entrenamiento. Este entrenamiento se vio forzado a detener por las fluctuaciones crecientes que comenzó a tener sobrepasando el ciclo 140. Como la RNC comenzaba a olvidar lo aprendido se detiene su entrenamiento y se procede con la predicción de la RNC entrenada.

4.3.3. Máquina de Soporte Vectorial

Para entrenar la máquina de soporte vectorial se utilizó la librería *LIBSVM*. Se plantearon los experimentos de la siguiente forma.

Se realizó como primera prueba el entrenamiento con las series de tiempo normalizadas únicamente. Se clasificaron los datos usando los métodos de las funciones internas de la librería *LIBSVM*, dada la elección de los parámetros por la función *find_parameters()*. Tras haber realizado el entrenamiento se encontraron resultados deficientes en la precisión arrojada por la MSV sobre el conjunto de entrenamiento. Se llevaron a cabo una serie de

Tabla 4.4: Precisión del conjunto de entrenamiento.

Experimento	Entrenamiento (%)	Predicción (%)
1	92.7	97.0
2	96.1	98.3
3	93.9	98.3
4	96.2	99.0
5	96.3	99.0
6	96.1	97.6
7	93.9	97.6
8	92.6	96.3
9	94.2	96.3
10	95.1	97.3
Promedio	94.7	97.6

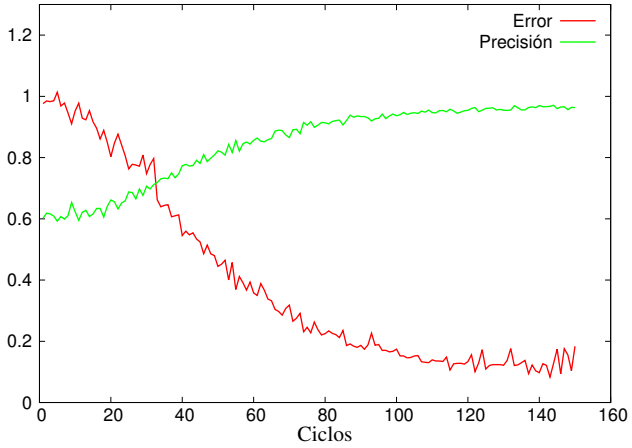


Figura 4.5: Error (Categorical Cross Entropy) y precisión contra ciclos de entrenamiento. Después del ciclo 140, la RNC dejó de aprender y el error empezó a fluctuar abruptamente por lo que obliga al entrenamiento a detenerse.

experimentos en los cuales se comenzaba a entrenar a la máquina por medio de un pequeño subconjunto del conjunto de entrenamiento tomado de forma aleatoria pero proporcional en cuanto al número de elementos de cada clase, posteriormente se puso a prueba clasificando el conjunto de predicción. Aumentando gradualmente el número de elementos del subconjunto de entrenamiento se observaron fluctuaciones en la predicción del conjunto de entrenamiento, sin tener éxito al momento de clasificar el conjunto de predicción, el cual resultaba en una precisión alrededor del 49 %. Esto sucedía posiblemente porque la MSV aprendía sobre el conjunto con mayor número de elementos, es decir, se tenía una preferencia, un tipo de sobre-entrenamiento. Se registraron cada uno de los experimentos realizados sin resultados favorables. Para resolver de esta manera el problema se necesitan otros algoritmos de búsqueda y/o preprocesamiento de las señales.

En la proesa de mejorar la clasificación por medio de las MSV se propuso utilizar las imágenes creadas de las transformaciones GASF (calculadas para alimentar a la RNC). Para ello se crearon distintas resoluciones para reducir el tiempo de entrenamiento al hacer uso de la MSV. Las resoluciones utilizadas se muestran en la tabla 4.5.

Tabla 4.5: Precisión del conjunto de entrenamiento.

Resolución
10×10
15×15
20×20
30×30
40×40
50×10

Se procedió utilizando la validación cruzada para la búsqueda de los mejores parámetros c (indicador de la tolerancia de errores) y γ (exponente de la función de base radial) y una vez definidos, se procede a entrenar la MSV con el conjunto de entrenamiento completo. Se realizaron los experimentos sobre cada resolución de la siguiente manera.

Se exploró el espacio de los parámetros con mallas de 3×3 de forma creciente (es decir a partir de ciertos valores, se incrementaban los parámetros) con prueba y error. La exploración se realizó sobre un subconjunto del conjunto de entrenamiento de tamaño 2,000 para ahorrar tiempo. Una vez relizada la búsqueda y elegidos los parámetros c y γ se procedió a entrenar la MSV, y probarla sobre el conjunto de predicción. Al comparar los resultados se elegían los parámetros de acuerdo a una dirección de crecimiento en la precisión del conjunto de predicción. Se encontró un comportamiento creciente en precisión de manera uniforme sobre todas las resoluciones.

La máquina de soporte vectorial fue capaz de clasificar correctamente la totalidad de

los datos presentados, tanto del conjunto de entrenamiento como del conjunto de predicción en cada una de las resoluciones, teniendo un 100 % de precisión. Los valores de los parámetros fueron los siguientes: $c = 1024$ y $\gamma = 4096$. Aunque esto parece ser el caso ideal, lamentablemente es muy probable que se tenga un sobre-entrenamiento ya que el número de vectores de soporte es muy grande, a pesar de la precisión obtenida en el conjunto de predicción. Este problema puede deberse principalmente al valor que le dimos a nuestros parámetros c y γ. También puede deberse al tipo de datos y al número de datos. Se pretende corregir este sobre-entrenamiento y reducir el número de vectores de soporte, sin perder tanta precisión.

Para minimizar el sobre-entrenamiento se realizó una búsqueda sistemática con prueba y error tratando de no elevar mucho ambos parámetros γ y c (al mismo tiempo). El espacio de parámetros explorado se muestra en la figura 4.6.

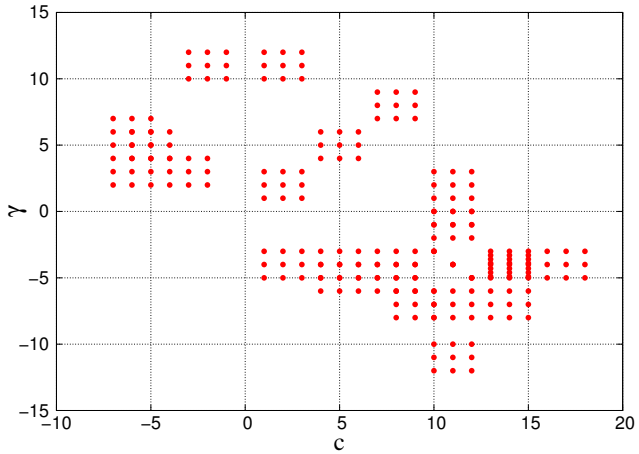


Figura 4.6: Se muestra el espacio de parámetros explorado para cada entrenamiento de cada resolución de las imágenes correspondientes a las señales.

Luego de llevar a cabo la exploración de esta manera, se registraron los resultados del número de vectores de soporte definidos en cada prueba de cada resolución. En la tabla 4.6 se muestran los resultados de cada experimento. Donde los resultados mejoraron significativamente en cuanto al número de vectores de soporte. Aún con el número de vectores de soporte obtenidos se tiene cierto grado de sobre-entrenamiento, ya que comparado con la solución para el conjunto de datos de Mnist, el número de vectores de soporte fue de tan solo la cuarta parte del número de ejemplos en el conjunto de entrenamiento aproximadamente. En este caso se tienen siete décimas partes.

Tabla 4.6: Se muestran algunos resultados de la clasificación, que son significativos, por medio de la MSV sobre el conjunto de imágenes de las señales ECG. Cada fila contiene un experimento donde se captura la resolución (Res) de las imágenes utilizadas, el valor del parámetro de tolerancia (c), el valor del parámetro (γ) de la función RBF, la precisión promedio de la validación cruzada (PVC), el número de vectores de soporte (# VS), la precisión del conjunto de entrenamiento (PE) y la precisión del conjunto de predicción (PP).

Res	c	γ	PVC (%)	# VS	PE (%)	PP (%)
10×10	128.0	512.0	59.4395	8,525	99.9883	100
10×10	32.0	64.0	59.2988	8,518	99.871	100
10×10	1,024.0	0.25	51.4	7,245	87.7111	84.3333
10×10	65,536.0	0.03125	51.1	7,064	85.0492	82.6667
15×15	65,536.0	0.03125	48.575	7,087	97.9362	97.3333
20×20	65,536.0	0.03125	46.5	6,854	99.3082	99.0
30×30	128.0	0.015625	53.725	7,695	78.7054	73
30×30	1,024.0	2^{-8}	53.6	7,467	72.9479	65.3333
30×30	1,024.0	0.125	48.7	7,562	99.5427	99.3333
30×30	32,768.0	0.125	46.925	6,804	99.9765	100
30×30	65,536.0	0.125	46.45	6,742	99.9765	100
30×30	16,384.0	$2^{-3.3}$	46.5	6,823	99.9062	100
40×40	128.0	0.015625	50.525	7,817	92.2022	89.3333
40×40	1,024.0	2^{-8}	49.75	7,658	87.9808	85.6667
40×40	65,536.0	0.125	47.625	7,137	100	100
50×50	128.0	0.0625	51.3	8,156	99.4371	99.0
50×50	1,024.0	0.015625	48.25	7,452	99.3551	98.6667

4.4. Metricas de evaluación

Hasta este punto se han presentado los resultados de cada clasificación implementada por medio de la precisión simple, la cual representa el porcentaje de aciertos o de clasificaciones correctas globales. Por ejemplo si se tienen 100 datos como ejemplo del conjunto de entrenamiento para n clases distintas y se entrena un sistema de aprendizaje, el cual clasifica a 85 ejemplos correctamente y a 15 incorrectamente, la precisión estaría dada por

$$\text{Precisión} = \frac{\text{Número de ejemplos clasificados correctamente}}{\text{Número total de ejemplos}}, \tag{4.3}$$

donde se toman en cuenta todas las clases.

A continuación se aborda un tema que trata sobre la mejora del reporte y evaluación de un clasificador o sistema de aprendizaje. Esto se debe a que en ciertos casos la precisión simple no nos da suficiente información o el resultado nos puede llevar a concluir cosas de forma equivocada. Se propone esclarecer y analizar más profundamente los resultados de la subsección previa introduciendo los métodos de evaluación del estado del arte.

El problema surge cuando se atacan problemas del mundo real, donde se manifiestan situaciones donde la teoría no aplica del todo o tiene excepciones. Primero, supongamos que tenemos un problema donde lo que queremos estudiar sucede muy esporádicamente, comparado con algún otro fenómeno. Se plantea, entonces, la clasificación de una clase positiva y una clase negativa. En el ámbito de la medicina, podemos identificar a la clase positiva con el hecho de padecer cierta enfermedad y por ende, la clase negativa al hecho de estar sano. Sea esta enfermedad en cierto grado no muy común. Digamos que se tiene un registro donde se plantea que una de cada cinco personas padecen la enfermedad. Y por lo tanto al recopilar las muestras de cada paciente sano y enfermo se consigue reunir un total de 1,200 ejemplos. Donde en este caso, dada la proporción, corresponde a tener 200 ejemplos de personas con el padecimiento (la clase positiva) y a 1,000 completamente sanas (la clase negativa).

Después de utilizar algún sistema de aprendizaje, se obtienen los resultados de la clasificación reportándose una precisión de 90 %. ¿Significa esto, que si ponemos a predecir a nuestro clasificador, este logra acertar si una persona está sana o enferma, 9 de cada 10 veces? La respuesta parace inclinarse a que sí, dado que asociamos el porcentaje inmediatamente de una manera uniforme y general. Sin embargo, no es el caso. Como se indicó, la clase con mayor número de ejemplos resulta ser seis veces más grande y por lo tanto podemos esperar que se aprenda menos de la clase minoritaria.

Calculemos entonces la precisión de una forma independiente. Primero traduzcamos el porcentaje de clasificaciones correctas, a número de ejemplos clasificados correctamente. Para ello, simplemente obtenemos el 90 % del total de ejemplos, es decir $1,080$, supongamos ahora que hay suficientes ejemplos para la clase mayoritaria (la de tamaño 1,000), y el sistema logra clasificar bien el 100 % de los ejemplos, consecuentemente el número de ejemplos correctamente clasificados de la clase minoritaria corresponde a 80 de los 200 de

esa clase ¡Esto corresponde a un 40 % correctamente clasificados! Y lo que podría estar indicando el modelo es que hay una probabilidad del 40 % de detectar la enfermedad en un paciente enfermo (en realidad es más complicado). Finalmente lo que parecía ser un buen modelo resulta no ser tan confiable, es por esto que se recurre a un objeto llamado matriz de confusión (se presenta en la subsección siguiente) para reportar diferentes aspectos evaluables en un sistema de aprendizaje y brindar una mayor confiabilidad del modelo.

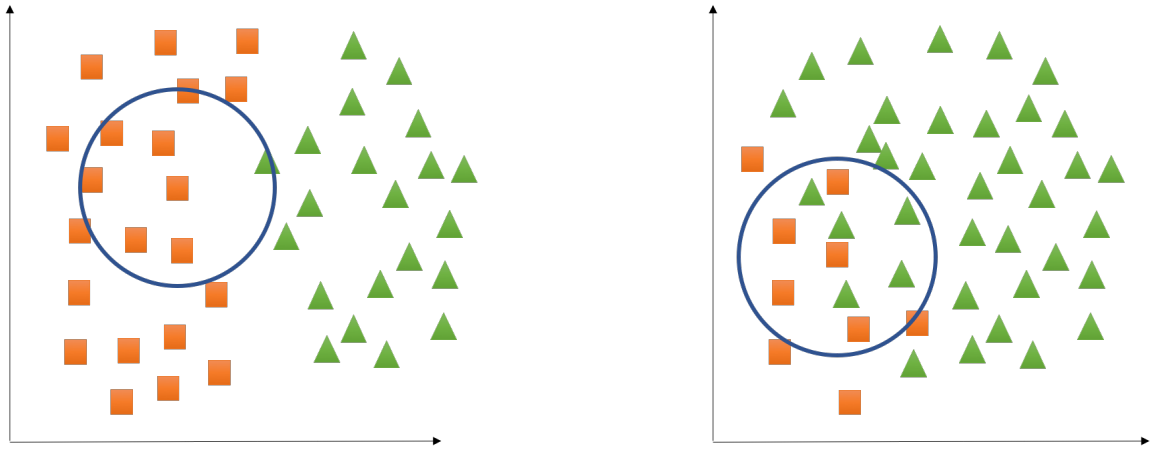
La situación mencionada anteriormente es un problema conocido como desbalanceo de clases. Donde los conjuntos de datos de las distintas clases difieren en número. En cuanto a los problemas resueltos en este trabajo, el conjunto de datos de Mnist corresponde a un conjunto de datos con 10 clases balanceadas, el conjunto de datos de CIFAR-10 también corresponde a un conjunto de clases balanceado con 6,000 imágenes por clase de un total de 10 clases, mientras que el conjunto de datos del reto CinC 2017 corresponde a un conjunto con clases desbalaceadas siendo la clase mayoritaria más del doble de datos de la segunda clase con más datos.

Hay muchos aspectos que afectan el desempeño de un sistema de aprendizaje. Existen antecedentes donde se afirma que uno de los aspectos está relacionado con este desbalanceo entre clases, donde se tiene alguna clase superando en número a cualquier otra de forma considerable. Esta situación se presenta frecuentemente en problemas del mundo real, como se mencionó anteriormente y los sistemas de aprendizaje tienen problemas para aprender las características de la(s) clase(s) con la minoría de los ejemplos de entrenamiento.

No obstante, se ha encontrado evidencia de que esta circunstancia no genera problemas en el desempeño por si sola de una forma sistemática, sino cuando se tienen problemas con pocos ejemplos de entrenamiento pero además donde se tienen ejemplos de distintas clases muy similares (indistinguibles), donde se dice que las clases se superponen [82].

Atacar un problema de clasificación donde las clases no son balanceadas es complejo. Supongamos que se tiene el siguiente dilema. Se requiere realizar la clasificación de datos en dos clases. La positiva y la negativa. El sistema de aprendizaje propuesto para la resolución del problema es el de los K vecinos más cercanos. Se representan los datos por medio de sus características en el plano como en la figura 4.7a, donde se tiene una situación ideal, las clases no se superponen y los grupos son distinguibles gráficamente. En este caso el método de los K vecinos más cercanos, resolvería el problema de forma directa. Por otro lado, en la figura 4.7b, una clase supera en número a la otra de forma considerable y en presencia de superposición de clases. El método de los K vecinos más cercanos ante esta situación cometerá errores seguros dado $K = 1$, por ejemplo, ya que existen datos donde el vecino más cercano pertenece a la clase contraria. Y si aumentamos el valor de K, entre más grande, más vecinos se consideran y finalmente la clasificación está totalmente cargada hacia la clase con mayor número de datos.

Considerando otro sistema de aprendizaje como la RNA, que se entrena usando *back-propagation*, al tener más ejemplos de cierta clase, es de esperarse que la red aprenda más sobre la clase mayoritaria. Ya que se corrigen los pesos en cada ciclo considerando cada



(a) (b)

Figura 4.7: (a) Representación gráfica de dos clases balanceadas y agrupaciones bien definidas. (b) Representación gráfica de dos clases donde la clase de triángulos verdes supera en número a la clase cuadro naranja. Además se tiene superposición de clases. El círculo trazado hace referencia a la distancia Euclidiana en $\mathbb{R}^2$.

ejemplo de entrenamiento. Esto se ha comprobado en la práctica y posiblemente se obtengan resultados aceptables, sin embargo, al reportar nuestros resultados usando la precisión simple (ecuación 4.3), no vemos reflejado específicamente el desempeño de la red en las clases minoritarias.

Las máquinas de soporte vectorial no están excentas de ser afectadas por el desbalanceo de clases. Ya que al buscar una solución en la que se busca la superficie de decisión óptima con el ajuste de los parámetros, como el de tolerancia a errores (c), pueden provocar que se descarten posibles vectores de soporte de la clase minoritaria. Y por lo tanto se clasifique erróneamente las clases minoritarias. Al trabajar con el conjunto de datos de los electrocardiogramas, dentro de los experimentos llevados a cabo en la exploración de parámetros adecuados para entrenar la MSV, resultaron clasificaciones donde casi todos los datos correspondientes a la clase Normal, eran clasificados correctamente y el resto de las clases de forma incorrecta. Signo de la preferencia de sobre la clase con el mayor número de ejemplos.

Dado el entrenamiento sobre el conjunto de los ECG, como se mencionó anteriormente, al realizar la partición de los datos en los nuevos grupos de entrenamiento y validación, los sistemas de aprendizaje mostraron tener problemas por generalizar y clasificar correctamente el grupo de validación. Por ejemplo, para la RNA de propagación hacia adelante, el error del grupo de validación fue creciente en todo momento a lo largo de los ciclos de entrenamiento. Intuitivamente se llegó a la conclusión de que los ejemplos de cada clase eran

muy representativos casi sin excepción por lo que al removerlos del conjunto de entrenamiento, la red perdía la capacidad de generalizar. Con el análisis previo, complementamos la razón de este fenómeno. El desbalanceo podría haber estado afectando el entrenamiento ya que las clases son considerablemente de menor número al comparar con la clase Normal del conjunto de ECG, y remover los ejemplos afectaba el entrenamiento de forma abrupta.

4.4.1. Matriz de confusión

Se introduce a continuación la herramienta que brinda toda una gamma de posibles características a evaluar en el desempeño de un sistema de aprendizaje para resolver problemas de clasificación. La matriz de confusión (o matriz de error), como su nombre lo dice, muestra cuales son las clases que el clasificador confunde, es decir clasifica de forma incorrecta. La tabla 4.7, muestra una matriz de confusión para el caso en el que se tienen dos clases, una positiva y una negativa. Se toma esta matriz como referencia para introducir las características evaluables.

Tabla 4.7: Matriz de confusión correspondiente al problema de clasificación en dos clases.

	Predicción Positivo	Predicción negativo
Clase positiva	Verdaderos positivos (VP)	Falsos negativos (FN)
Clase negativa	Falsos positivos (FP)	Verdaderos negativos (VN)

Por ejemplo, si queremos calcular la precisión simple (ecuación 4.3), se define como

$$\text{Precisión} = \frac{VP + VN}{VP + FN + FP + VN}. \tag{4.4}$$

Como se ilustró anteriormente, esta definición de la precisión nos puede llevar a realizar conclusiones equivocadas. Además, otra de las desventajas de utilizar esta definición, es que la misma no distingue entre errores graves y no tan graves.

Por ejemplo, se plantea el siguiente problema de clasificación. Se tienen tres clases, donde se analiza si una persona padece de la enfermedad A, de la enfermedad B, o es una persona sana. Supongamos que dos clasificadores después de ser entrenados, se ponen a prueba al clasificar un conjunto de predicción. El conjunto de predicción está formado por 10 personas sanas, 10 personas con la enfermedad A y 10 personas con la enfermedad B. Supongamos que estos son los resultados:

- El clasificador 1 arroja los siguientes resultados: 11 sanos de los cuales 1 es en realidad enfermo (con A o B), 17 con la enfermedad A de los cuales 9 tienen la enfermedad B y por último 2 con la enfermedad B, correctamente clasificados.

- El clasificador 2 arroja los siguientes resultados: 16 sanos de los cuales 6 son en realidad enfermos, 7 con la enfermedad A, correctamente clasificados y 7 con la enfermedad B, correctamente clasificados.

Calculamos la precisión simple del Clasificador 1, resulta

$$P_1 = \frac{10 + 8 + 2}{30} = 66\,\%. \tag{4.5}$$

La Precisión simple del Clasificador 2 es

$$P_2 = \frac{10 + 7 + 7}{30} = 80\,\%. \tag{4.6}$$

La intuición nos diría que como el Clasificador 2 resultó tener una precisón mayor, entonces es el clasificador más confiable, pero, ¡No tan rápido! Analicemos con mayor profundidad. Realizando el cálculo de la precisión de la clase sana de forma independiente y de las clases enfermas agrupadas; dicho de otro modo, se reduce el problema a dos clases, enfermo o no enfermo. Se obtiene lo siguiente

$$P_{1sano} = \frac{10}{10} = 100\,\% \tag{4.7}$$

$$P_{1enfermo} = \frac{19}{20} = 95\,\% \tag{4.8}$$

$$P_{2sano} = \frac{10}{10} = 100\,\% \tag{4.9}$$

$$P_{2enfermo} = \frac{14}{20} = 70\,\% \tag{4.10}$$

Esto quiere decir que el Clasificador 1 tiene 95 % de probabilidad de acertar si alguna persona padece de alguna enfermedad (A o B), mientras que el Clasificador 2 tienen 70 % de probabilidad de acertar si alguna persona padece de alguna enfermedad. Esto cambia las cosas, ya que lo anterior manifiesta que el Clasificador 1 es mejor detectando si una persona no es sana, aunque no pueda identificar exactamente cuál es la enfermedad. Por otra parte, el Clasificador 2 aunque si detecta una enfermedad, logra clasificar perfectamente si se trata de la enfermedad A o B, se equivoca más veces, confundiendo una persona enferma con una sana, lo cual es más grave. En tal caso, una posible solución al problema es utilizar el Clasificador 1 para detectar si existe alguna enfermedad y posteriormente, utilizar una modificación del Clasificador 2, en la que se realiza una clasificación binaria, donde se detecte el tipo de enfermedad que se padece (A o B).

Las cantidades calculadas en las ecuaciones (4.7-4.10) corresponden a métricas evaluables que la matriz de confusión permite calcular. He aquí la importancia de esta herramienta.

A continuación se presentan las definiciones de las métricas que brindan información más detallada de los clasificadores, de la tabla 4.7, se pueden definir 4 posibles métricas básicas que separan el desempeño de las clases negativa y positiva de forma independiente [82]:

- Razón de falsos negativos
$$RFN = \frac{FN}{VP + FN}, \tag{4.11}$$
corresponde al porcentaje de los casos positivos clasificados incorrectamente en la clase negativa.

- Razón de falsos positivos
$$RFP = \frac{FP}{FP + VN}, \tag{4.12}$$
corresponde al porcentaje de los casos negativos clasificados incorrectamente en la clase positiva.

- Especificidad
$$\text{Especificidad} = \frac{VN}{VN + FP}, \tag{4.13}$$
la especificidad o la razón de verdaderos negativos, corresponde al porcentaje de los casos negativos clasificados correctamente.

- Sensibilidad
$$\text{Sensibilidad} = \frac{VP}{VP + FN}, \tag{4.14}$$
corresponde al porcentaje de los casos positivos clasificados correctamente.

El objetivo sería minimizar las métricas RFP y RFN y maximizar la Especificidad y Sensibilidad. Desafortunadamente, la mayoría de los problemas del mundo real, al maximizar o minimizar alguno, se realiza a costa del otro, por lo que se ha analizado la relación entre las métricas de forma gráfica por medio de las gráficas ROC (Receiver Operating Characteristic). El análisis ROC va más allá del objetivo de esta tesis por lo que se recomienda ver [83].

Para poder contribuir al área de la medicina, se deben completar cierto requisitos para que los algoritmos computacionales y los sistemas automáticos puedan ser usados a nivel clínico. En el caso de los algoritmos de detección de arritmia se han propuesto nuevos requerimientos para la prueba y para el reportaje de resultados en el artículo *Proposed*

New Requirements for Testing and Reporting Performance Results of Arrythmia Detection Algorithms [84]. En estas nuevas propuestas se toman en cuenta las siguientes métricas importantes para el área médica:

- Valor de predicción positivo

$$VPP = \frac{VP}{VP + FP}, \tag{4.15}$$

  corresponde a la probabilidad de resultar ser positivo al haber sido clasificado positivo.

- Valor de predicción negativo

$$VPN = \frac{VN}{FN + VN}, \tag{4.16}$$

  corresponde a la probabilidad de resultar ser negativo al haber sido clasificado negativo.

- Prevalencia

$$\text{Prevalencia} = \frac{VP + FN}{VP + FN + FP + VN}, \tag{4.17}$$

  corresponde al porcentaje de pacientes con la característica en cuestión del total de individuos.

Muchas veces al obtener solamente la métrica de VPP o la sensibilidad no da suficiente información global por lo que se define la media armónica de estos dos valores, conocido como F-score. Se calcula mediante la siguiente fórmula

$$F_1 = 2 \cdot \frac{VPP \cdot \text{sensibilidad}}{VPP + \text{sensibilidad}}. \tag{4.18}$$

A continuación se realiza la generalización de la definición de la matriz de confusión para N clases así como las métricas correspondientes.

Supongamos en que en principio aumentamos el problema de clasificación en tres clases distintas. La matriz de confusión para este problema se muestra en la tabla 4.8. Donde la diagonal contiene a los verdaderos positivos de cada clase, en este caso corresponden a los valores VP_A, VP_B y VP_C. Cualquier otro valor fuera de la diagonal corresponde a los errores en la clasificación, donde en cada fila para un valor no diagonal se tiene una clasificación equivocada donde se confundió el valor de la clase de la fila con la clase de la columna correspondiente.

Tabla 4.8: Matriz de confusión correspondiente al problema de clasificación con multiclases.

		Predicción		
		A	B	C
Clase verdadera	A	VP_A	E_{AB}	E_{AC}
	B	E_{BA}	VP_B	E_{BC}
	C	E_{CA}	E_{CB}	VP_C

Para extender las métricas previas introducidas, se considera en cada caso que la clase en cuestión es la positiva y el resto la negativa. Características de la matriz de confusión multiclase:

- El número total de ejemplos de cualquier clase es la suma de la fila correspondiente a esa clase.
- El total de FN para una clase es la suma de los valores de la fila correspondiente a esa clase sin sumar el valor de VP de esa clase (el valor de la diagonal de esa fila).
- El total de FP para una clase es la suma de los valores de la columna correspondiente a esa clase sin sumar el valor de VP de esa clase (el valor de la diagonal de esa columna).
- El número total de VN para una clase es la suma de todas las filas y columnas exceptuando a la fila y columna correspondientes a esa misma clase.

Calculamos las métricas de algunas clases. Por ejemplo, para calcular el valor de predicción positivo para la clase B, debemos calcular

$$VPP_B = \frac{VP_B}{VP_B + FP_B} = \frac{VP_B}{VP_B + (E_{AB} + E_{CB})}. \tag{4.19}$$

Para calcular la sensibilidad (razón de verdaderos positivos) de la clase A, debemos calcular

$$\text{Sensibilidad A} = \frac{VP_A}{VP_A + FN_A} = \frac{VP_A}{VP_A + (E_{AB} + E_{AC})}. \tag{4.20}$$

Para calcular la especificidad (razón de verdaderos negativos) de la clase C, debemos calcular

$$\text{Especificidad C} = \frac{VN_C}{VN_C + FP_C} = \frac{VN_C}{VN_C + (E_{AC} + E_{BC})}. \tag{4.21}$$

Para calcular F_1 Score de la clase A, debemos calcular

$$F_1 = 2 \cdot \frac{VPP_A \cdot \text{sensibilidad A}}{VPP_A + \text{sensibilidad A}}. \tag{4.22}$$

Finalmente, para calcular F_1 Score promedio de las clases A,B y C, se calcula la media aritmética de los F_1 Score de cada clase.

Para la generalización a N clases, usaremos la notación de suma donde los índices i, j corren sobre las distintas clases, por lo tanto las características de la matriz de confusión (MC) quedan expresadas de la siguiente forma, para el número total de ejemplos de la clase i,

$$\text{total}_i = \sum_{j}^{N} (MC)_{ij}. \tag{4.23}$$

Para número de FN de la clase i,

$$\text{FN}_i = \sum_{j \neq i}^{N} (MC)_{ij}. \tag{4.24}$$

Para número de FP de la clase i,

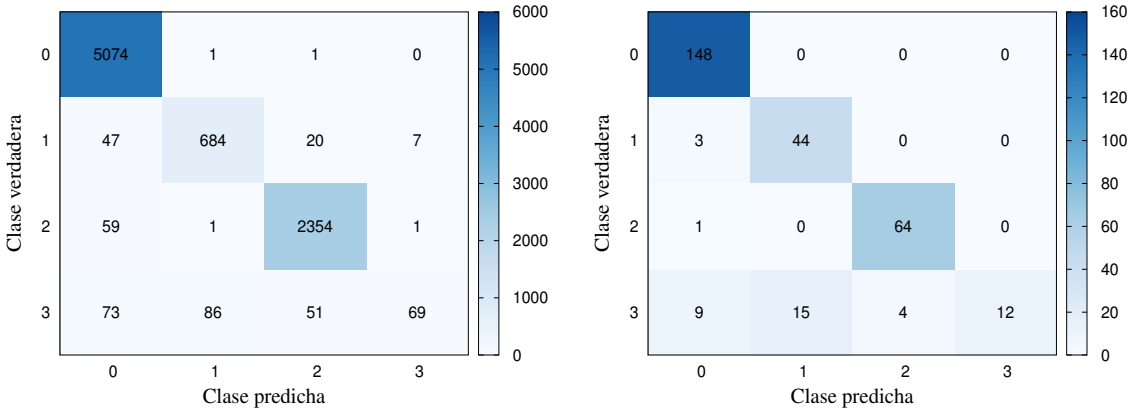
$$\text{FP}_i = \sum_{j \neq i}^{N} (MC)_{ji}. \tag{4.25}$$

Para número de VN de la clase i,

$$\text{FP}_i = \sum_{k \neq i}^{N} \sum_{j \neq i}^{N} (MC)_{kj}. \tag{4.26}$$

A continuación se presentan las matrices de confusión de las clasificaciones realizadas sobre el conjunto de ECG de la base de datos de CinC challenge 2017, ya que es la único conjunto de datos que tienen clases desbalanceadas. Los otros conjuntos como el Mnist y CIFAR-10 son conjuntos balanceados.

Las matrices de confusión resultantes generadas por la FFNN sobre el conjunto de entrenamiento y de predicción se muestran en la figura 4.8a y 4.8b, respectivamente. Las matrices de confusión resultantes generadas por la FFNN sobre el conjunto de entrenamiento aumentado y de predicción de la red entrenada con el conjunto de entrenamiento aumentado se muestran en la figura 4.9a y 4.9b, respectivamente. Las matrices de confusión resultantes generadas por la RNC sobre el conjunto de entrenamiento y de predicción se muestran en la figura 4.10a y 4.10b, respectivamente. Las matrices de confusión resultantes

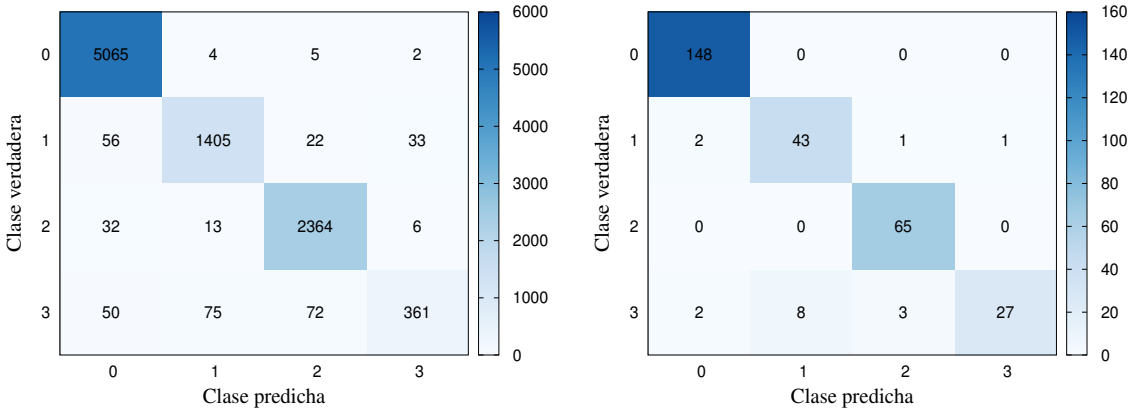


(a) (b)

Figura 4.8: Matrices de confusión del experimento con el valor de los parámetros que arrojó la mejor precisión sobre el conjunto de predicción generadas por la FFNN sobre el conjunto de señales de ECGs, las clases 1, 2, 3 y 4, corresponden a las clases N, AF, O y R, respectivamente. (a) Matriz de confusión generada por la FFNN sobre el conjunto de entrenamiento. (b) Matriz de confusión sobre el conjunto de predicción.

de la MSV definida por uno de los experimentos de la tabla 4.6 se muestran en la figura 4.11a y 4.11b.

En la tablas 4.9-4.12, se muestran los resultados del cálculo de las métricas definidas anteriormente. Se eligen calcular solo algunas ya que el análisis se realiza con base en el objetivo de la clasificación y en el número de datos de cada clase. Se calcula el Score F_1, no sólo por su resumen y utilidad, sino porque el reto del concurso CinC challenge 2017, lo usa como parámetro de evaluación.



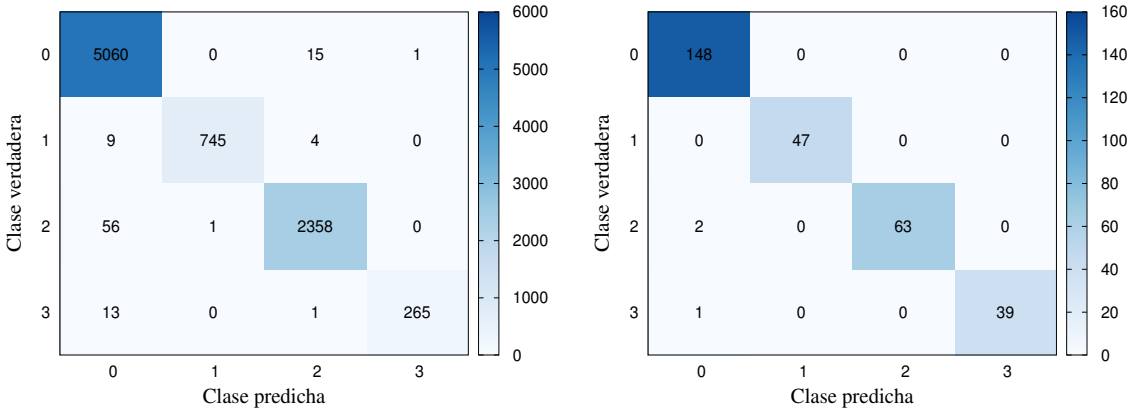
(a) (b)

Figura 4.9: Matrices de confusión del experimento con el valor de los parámetros que arrojó la mejor precisión sobre el conjunto de predicción generadas por la FFNN sobre el conjunto de señales de ECG, las clases 1, 2, 3 y 4, corresponden a las clases N, AF, O y R, respectivamente. (a) Matriz de confusión generada por la FFNN sobre el conjunto de entrenamiento con aumento en el número de datos. (b) Matriz de confusión sobre el conjunto de predicción generada por la FFNN entrenada con aumento de datos.

Tabla 4.9: Se muestran los valores de las métricas (en %) de acuerdo al método implementado con base en la matriz de confusión.

Método	Métrica	N	AF	O	R
FFNN	VPP	91.9255	74.5763	94.1176	100
	Esp.	91.4474	94.0711	98.2979	100
	Sens.	100	93.617	98.4615	30
	F_1	95.7929	83.0189	96.2406	46.1538

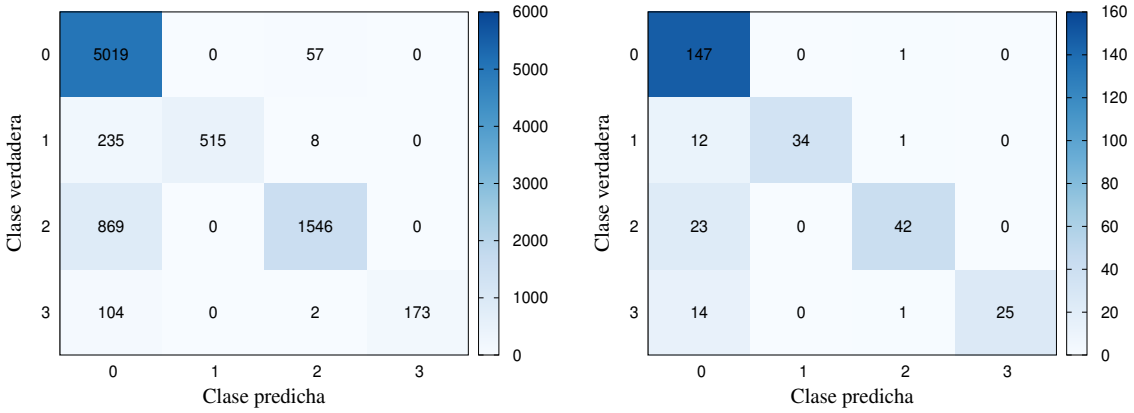
Tabla 4.10: Se muestran los valores de las métricas (en %) de acuerdo al método implementado con base en la matriz de confusión.



(a) (b)

Figura 4.10: Matrices de confusión del experimento con resolución de 50×50 pixeles con el valor de los parámetros que arrojó la mejor precisión sobre el conjunto de predicción generada por la RNC sobre el conjunto de señales de ECG, las clases 1, 2, 3 y 4, corresponden a las clases N, AF, O y R, respectivamente. (a) Matriz de confusión generada por la RNC sobre el conjunto de entrenamiento. (b) Matriz de confusión sobre el conjunto de predicción.

Método	Métrica	N	AF	O	R
FFNN (Aumento)	VPP	97.3684	84.3137	94.2029	96.4286
	Esp.	97.3684	96.8379	98.2979	99.6154
	Sens.	100	91.4894	100	67.5
	F_1	98.6667	87.7551	97.0149	79.4118



(a) (b)

Figura 4.11: Matrices de confusión del experimento con resolución de 10×10 pixeles, $c = 1,024.0$ y $\gamma = 0.25$ de la MSV sobre el conjunto de señales en forma de imagen de ECG, las clases 1, 2, 3 y 4, corresponden a las clases N, AF, O y R, respectivamente. (a) Matriz de confusión generada por la MSV sobre el conjunto de entrenamiento. (b) Matriz de confusión sobre el conjunto de predicción.

Tabla 4.11: Se muestran los valores de las métricas (en %) de acuerdo al método implementado con base en la matriz de confusión.

Método	Métrica	N	AF	O	R
RNC	VPP	98.0132	100	100	100
	Esp.	98.0263	100	100	100
	Sens.	100	100	96.9231	97.5
	F_1	98.6667	100	98.4375	98.7342

Tabla 4.12: Se muestran los valores de las métricas (en %) de acuerdo al método implementado con base en la matriz de confusión.

Método	Métrica	N	AF	O	R
MSV	VPP	76.9634	100	93.75	100
	Esp.	71.0526	100	98.7234	100
	Sens.	99.3243	76.5957	69.2308	62.5
	F_1	86.7257	86.747	79.646	76.9231

Tabla 4.13: Se muestra la métrica de F_1-Score promedio del conjunto de predicción de cada uno de los métodos de clasificación implementados de las matrices de confusión de las figuras 4.8-4.11.

Método	F_1 promedio (%)
FFNN	80.3015
FNNN (Aumento)	90.7121
RNC	98.9596
MSV	82.5104

Analizando los valores resultantes de la tabla 4.13, podemos notar que la RNA simple sin el aumento de datos obtuvo un valor de F_1-Score inferior al de los demás métodos, como era de esperarse. Cabe mencionar que el aumento de datos realizado, mejoró significativamente la clasificación, con base en los resultados mostrados en las tablas 4.9 y 4.10, donde para la clase Ruido, con el menor número de ejemplos en el conjunto de entrenamiento, tuvo una sensibilidad del 30 % aumentando a 67.5 % después de aumentar el número de ejemplos de las clases minoritarias. Nótese que la precisión máxima simple obtenida para la FFNN fue de 89.3 %, sin embargo, la matriz y las métricas demuestran que en realidad se clasificaron correctamente menos de la mitad de la clase Ruido, por lo que la red tuvo problemas para clasificar esta clase en específico. Otra conclusión que podemos obtener de la matriz de confusión y de las métricas de la tabla 4.9, es que al haber obtenido un valor de predicción positivo de 100 %, esto nos dice que si se clasifica un ejemplo en la clase Ruido, es seguro (casi) que la clasificación es correcta. Y al haber obtenido una especificidad del 100 %, esto nos dice que ninguna clase ajena a la de Ruido, se clasificó como Ruido, en otras palabras, ninguna de las otras clases se confunde con la clase Ruido. Sin embargo, el valor del 30 % en sensibilidad, nos dice que no todo lo que es ruido se pudo clasificar correctamente; es decir, la red confunde la clase Ruido con otras clases. De la tabla 4.10, también podemos concluir que al obtener 100 % en Sensibilidad sobre las clases Normal y Otra, se pudieron clasificar correctamente el total de los ejemplos correspondientes a dichas clases. Confundiendo poco otras clases con estas, información que obtenemos del VPP con un valor de 97.3684 % y 94.2029 % para las clases Normal y Otra, respectivamente.

En el caso de las MSVs, no se muestra la matriz de confusión de la clasificación con 100 % de precisión simple, ya que no nos daría mucha información. Desafortunadamente, no es tan confiable el método, gracias a que se tienen demasiados vectores de soporte. Se esperaría que la MSV estuviera sobreentranada y la clasificación sobre el conjunto de predicción sería un fracaso, pero no sucedió así, esto puede deberse a la naturaleza de los datos. Para asegurar una mayor confiabilidad es necesario disminuir el número de vectores de soporte y probar la MSV con otro conjunto de predicción (El oculto por Phyisionet y CinC challenge 2017).

En el caso de los resultados obtenidos con la RNC, es importante mencionar que corresponde al mejor método, no sólo por su precisión simple de 99.0 % sino porque de la tabla 4.11, podemos observar que el valor de las cuatro métricas sobre la clase de Fibrilación Auricular es del 100 %. Esto quiere decir que si se clasifica una señal como FA, seguro (casi) corresponde a esta enfermedad. También quiere decir que la red puede diferenciar perfectamente la enfermedad y no confunde esta clase con ninguna otra ni las otras clases con esta. Este es un resultado ideal para la detección de enfermedades y uno de los objetivos buscados en este ámbito. En contraste con las MSV, en este caso no se tienen indicios de sobre-entrenamiento. Para averiguarlo, será necesario hacer más pruebas (también sobre el conjunto de predicción oculto de Phyisionet de CinC challenge 2017).

En el caso de la MSV, mostrado en la tabla 4.12, donde se tiene que la MSV no confunde a las otras clases con la clase de FA (pero si a la clase FA con otras). Esto indica que podemos tener cierto grado de confiabilidad en el diagnóstico de esta enfermedad, incluso teniendo una precisión simple de menos del 90 %, cosa que sin el análisis de la matriz de confusión y las métricas no se podría saber.

Nótese que en todos los métodos, las matrices demuestran que se tienen cierta preferencia sobre la clase de ritmo Normal, gracias a el número tan grande que se tiene de ejemplos, comparado con las demás clases.

Capítulo 5

Conclusiones y trabajo futuro

Después de los resultados obtenidos se tiene una mejor visión del problema abordado así como sus dificultades en cuanto a la cantidad de parámetros implicados, las posibilidades son enormes. En todo caso se puede extender este trabajo de muchas maneras: Aumentando las bases de datos a la cual se aplican los métodos propuestos, la forma en la que se eligió la clasificación y la búsqueda de un entrenamiento óptimo por medio de algoritmos genéticos y la paralelización.

El mejor método resultó ser el de las redes neuronales convolucionales, tal vez por la naturaleza de los datos ingresados, ya que estas fueron hechas para analizar imágenes. La mejora en el caso de las redes de propagación hacia adelante cuando se realizó el aumento de los datos fue notoria a pesar de la inestabilidad de los resultados en cada experimento del entrenamiento y predicción. Es difícil encontrar un procedimiento general en el ámbito de la clasificación por medio de métodos de inteligencia artificial, esto se debe a la inmensa variedad de datos y formas de analizarlos. En cuanto a la base de datos de CinC challenge 2017, fue difícil extraer algo útil del grupo de validación al llevar a cabo la partición de los datos sobre el conjunto de entrenamiento. La opción a futuro será poner a prueba los sistemas de aprendizaje ingresando el conjunto oculto de datos del CinC challenge 2017, el cual no ha sido publicado a la fecha.

Considerando el reporte de resultados y evaluación de los sistemas de aprendizaje cuando se abordan problemas con clases debalanceadas, se demostró que recurrir a las métricas como F_1-Score, PPV, Sensibilidad y Especificidad por medio de la matriz de confusión es crucial para no caer en conclusiones erróneas. A su vez para demostrar con mejor detalle la confiabilidad del sistema de aprendizaje en cuestión, así como los problemas que pueda tener en particular cierto sistema y/o dificultades que se tengan con cierta clase.

Uno de los objetivos futuros a cumplir es el de aplicar la transformación GASF en la clasificación de ECG por medio de pulsos individuales. Ya que esta transformación resultó ser muy útil, ahorrando tiempo en implementar filtros de ruido y normalización de los datos. Los resultados muestran que esta transformación ayuda a los sistemas de aprendizaje de tal manera, que las imágenes tienen la propiedad de filtrar los ruidos de las señales.

Recientemente se formó una nueva área llamada Medicina el cerebro y el corazón la cual sostiene que el cerebro es el que controla el funcionamiento del corazón de donde se puede asumir que el problema yace en el cerebro y por lo tanto un estudio conjunto corazón-cerebro puede llegar a obtener mejores resultados en el diagnóstico y tratamiento de las enferemedades cardiacas, abriendo otra puerta de posible solución al problema.

Bibliografía

[1] Luna, A. B. (2007). *Basic electrocardiography: Normal and abnormal ECG patterns*. Malden, MA: Blackwell Pub.

[2] Wolf, P. A., Benhamin, E. J., Belanger, A. J., Kannel, W. B., Levy, D., & Dagostino, R. B. (1996). Secular trends in the prevalence of atrial fibrillation: The Framingham study. *American Heart Journal, 131*(4), 790-795. doi:10.1016/s0002-8703(96)90288-4

[3] Camm, A. J., Kirchhof, P., Lip, G. Y., Schotten, U., Savelieva, I., Ernst, S., . . . Rutten, F. H. (2010). Guidelines for the management of atrial fibrillation: the task force for the management of atrial fibrillation of the european society of cardiology (ESC). *European Heart Journal*, 31 23692429. doi:10.1093/eurheartj/ehq278

[4] Go, A. S., Hylek, E. M., Phillips, K. A., Chang, Y., Henault, L. E., Selby, J. V., & Singer, D. E. (2001). Prevalence of Diagnosed Atrial Fibrillation in Adults. *Jama*, 285(18), 2370. doi:10.1001/jama.285.18.2370

[5] Kannel, W., Wolf, P., Benjamin, E., & Levy, D. (1998). Prevalence, incidence, prognosis, and predisposing conditions for atrial fibrillation: Population-based estimates 11Reprints are not available. *The American Journal of Cardiology, 82*(7). doi:10.1016/s0002-9149(98)00583-9

[6] Wolf, P. A. (1987). Atrial Fibrillation: A Major Contributor to Stroke in the Elderly. *Archives of Internal Medicine, 147*(9), 1561. doi:10.1001/archinte.1987.00370090041008

[7] Wang, T. J., Larson, M. G., Levy, D., Vasan, R. S., Leip, E. P., Wolf, P. A., . . . Benjamin, E. J. (2003). Temporal Relations of Atrial Fibrillation and Congestive Heart Failure and Their Joint Influence on Mortality. *Circulation, 107*(23), 2920-2925. doi:10.1161/01.cir.0000072767.89944.6e

[8] Krahn, A. D., Manfreda, J., Tate, R. B., Mathewson, F. A., & Cuddy, T. E. (1995). The natural history of atrial fibrillation: Incidence, risk factors, and prognosis in

the manitoba follow-up study. *The American Journal of Medicine,98*(5), 476-484. doi:10.1016/s0002-9343(99)80348-9

[9] Stewart, S., Hart, C. L., Hole, D. J., & Mcmurray, J. J. (2002). A population-based study of the long-term risks associated with atrial fibrillation: 20-year follow-up of the Renfrew/Paisley study. *The American Journal of Medicine, 113*(5), 359-364. doi:10.1016/s0002-9343(02)01236-6

[10] Ott, A., Breteler, M. M., Bruyne, M. C., Harskamp, F. V., Grobbee, D. E., & Hofman, A. (1997). Atrial Fibrillation and Dementia in a Population-Based Study. *Stroke, 28*(2), 316-321. doi:10.1161/01.str.28.2.316

[11] Lara Vaca, S., Cordero Cabra A., Martínez-Flores E., & Iturralde Torres P. (2014). Registro Mexicano de Fibrilación Auricular. *Gaceta Médica de México 2014, 150* Suppl 1:48-59

[12] January, C. T., Wann, L. S., Alpert, J. S., Calkins, H., Cigarroa, J. E., Cleveland, J. C., . . . Yancy, C. W. (2014). 2014 AHA/ACC/HRS Guideline for the Management of Patients With Atrial Fibrillation: Executive Summary. *Circulation, 130*(23), 2071-2104. doi:10.1161/cir.0000000000000040

[13] Jambukia, S. H., Dabhi, V. K., & Prajapati, H. B. (2015). Classification of ECG signals using machine learning techniques: A survey. *2015 International Conference on Advances in Computer Engineering and Applications*. doi:10.1109/icacea.2015.7164783

[14] K, C. (2017). A Survey on various Machine Learning Approaches for ECG Analysis. *International Journal of Computer Applications*. 163. 25-33. doi:10.5120/ijca2017913737.

[15] Goldberger, A. L., Amaral, L. A., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., . . . Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet. *Circulation, 101*(23). doi:10.1161/01.cir.101.23.e215

[16] Rich, E. (1983). *Artificial intelligence*. New York: McGraw-Hill.

[17] Mitchell, T. M. (2017). *Machine learning*. New York: McGraw Hill.

[18] Ertel, W., Black, N., & Mast, F. (2017). *Introduction to artificial intelligence*. Cham, Switzerland: Springer.

[19] Atlas. (n.d.). Retrieved October, 2018, from https://www.bostondynamics.com/atlas

[20] Ertel, W. (n.d.). Assistive Robot for people with physical disabilities. Retrieved October, 2018 from http://asrobe.hs-weingarten.de/?lang=eng

[21] Jain, A., Mao, J., & Mohiuddin, K. (1996). Artificial neural networks: A tutorial. *Computer, 29*(3), 31-44. doi:10.1109/2.485891

[22] Parallel computing (n.d.). Retrieved October, 2018, http://www.cse.unt.edu/ tarau/teaching/parpro/papers/Parallel %20computing.pdf

[23] Parallel computing (n.d.). Retrieved October, 2018, http://www.cs.toronto.edu/ bonner/courses/2014s/csc321/lectures/lec5.pdf

[24] Gonzalez, O., Shrikumar, H., Stankovic, J., & Ramamritham, K. (1997). Adaptive fault tolerance and graceful degradation under dynamic hard real-time scheduling. *Proceedings Real-Time Systems Symposium*. doi:10.1109/real.1997.641271

[25] Wang, Z., & Oates, T. (2015). Imaging Time-Series to Improve Classification and Imputation Preprint arXiv:1506.00327 [cs.LG]

[26] Sermanet, P., Chintala, C., & LeCun, Y. (2012) Convolutional neural networks applied to house numbers digit classification. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, 3288-3291.

[27] Street View House Number Recognition using Deep ... (n.d.). Retrieved October, 2018, from http://www.iitp.ac.in/ arijit/dokuwiki/lib/exe/fetch.php?media=courses:2017:cs551:03_presentation.pdf

[28] Li, K., Mao, S., Li, X., Wu, Z., & Meng, H. (2018). Automatic lexical stress and pitch accent detection for L2 English speech using multi-distribution deep neural networks. *Speech Communication, 96*, 28-36. doi:10.1016/j.specom.2017.11.003

[29] Yu, D., & Li, J. (2017). Recent progresses in deep learning based acoustic models. *IEEE/CAA Journal of Automatica Sinica, 4*(3), 396-409. doi:10.1109/jas.2017.7510508

[30] Dohare, A. K., Kumar, V., & Kumar, R. (2018). Detection of myocardial infarction in 12 lead ECG using support vector machine. *Applied Soft Computing, 64*, 138-147. doi:10.1016/j.asoc.2017.12.001

[31] Burges, C. J. C. (1998). A tutorial on Support Vector Machines for Pattern Recognition Data Mining and Knowledge Discovery. *Kluwer Academic Publishers, 2*, 121-167.

[32] T. (2018, September 24). Tflearn/tflearn. Retrieved May, 2018, from https://github.com/tflearn/tflearn

[33] TensorFlow: Large-Scale Machine Learning on Heterogeneous ... (2015). Retrieved May, 2018, from http://download.tensorflow.org/paper/whitepaper2015.pdf

[34] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm

[35] Oliphant, T. E. (n.d.). Guide to NumPy: 2nd Edition. Retrieved May, 2018, from https://www.barnesandnoble.com/w/guide-to-numpy-travis-e-oliphant-phd/1122853007

[36] Esquema resumen del sistema nervioso del hombre. (n.d.). Retrieved May/June, 2018, from https://www.educaycrea.com/2015/11/esquema-resumen-del-sistema-nervioso-del-hombre/

[37] Mcculloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics, 5*(4), 115-133. doi:10.1007/bf02478259

[38] Khan, S., Rahmani, H., Shah, S. A., & Bennamoun, M. (2018). *A guide to convolutional neural networks for computer vision*. San Rafael, CA: Morgan & Claypool.

[39] Hebb, D. O. (1952). *The organisation of behaviour: A neuropsychological theory*. New York: Wiley.

[40] Vojt, J. (2016). Deep neural networks and their implementation (Unpublished master's thesis). Univ. Charles University in Prague, Faculty of Mathematics and Physics.

[41] Hecht-Nielsen, R. (1994). *Neurocomputing*. Reading, Mass: Addison-Wesley.

[42] Specht, D. (1991). A general regression neural network. I*EEE Transactions on Neural Networks,2*(6), 568-576. doi:10.1109/72.97934

[43] Salehinejad, H., Sankar, S., Barfett, J., Colak, E., & Valaee, S. (2018) Recent Advances in Recurrent Neural Networks arXiv:1801.01078v3 [cs.NE]

[44] Hertz, J., Krogh, A., & Palmer, R. G. (1999). *Introduction to the theory of neural computation*. Reading: Addison-Wesley.

[45] Simon, H. (2008). *Neural networks: A comprehensive foundation*. New Delhi: Prentice-Hall of India.

[46] Hoffmann, M. (2017, November 16). Exploring Stochastic Gradient Descent with Restarts (SGDR). Retrieved May, 2018, from https://medium.com/38th-street-studios/exploring-stochastic-gradient-descent-with-restarts-sgdr-fa206c38a74e

[47] Wilson, D., & Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. *Neural Networks, 16*(10), 1429-1451. doi:10.1016/s0893-6080(03)00138-2

[48] J. S. (2017, December 14). File:MnistExamples.png. Retrieved May, 2018, from https://commons.wikimedia.org/w/index.php?curid=64810040

[49] Y. Lecun and C. Cortes. The MNIST database of handwritten digits. (n.d.). Retrieved May, 2018, from http://yann.lecun.com/exdb/mnist/

[50] Zupan, J., & Gasteiger, J. (1993). *Neural networks for chemists: An introduction*. Weinheim: VCH.

[51] Aghdam, H. H., & Heravi, E. J. (2017). Convolutional Neural Networks. *Guide to Convolutional Neural Networks*, 85-130. doi:10.1007/978-3-319-57550-6_3

[52] Fayed, H., & Atiya, A. (2009). A Novel Template Reduction Approach for the *K*-Nearest Neighbor Method. *IEEE Transactions on Neural Networks, 20*(5), 890-896. doi:10.1109/tnn.2009.2018547

[53] Arlot, S., & Celisse, A. (2010). A survey of cross-validation procedures for model selection. *Statistics Surveys, 4*(0), 40-79. doi:10.1214/09-ss054

[54] Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. Retrieved from https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf

[55] Krizhevsky, A. (n.d.). The CIFAR-10 dataset. Retrieved May/June, 2018, from https://www.cs.toronto.edu/ kriz/cifar.html

[56] å ml design. (n.d.). Retrieved May/June, 2018, from http://blog.otoro.net/2016/04/06/the-frog-of-cifar-10/

[57] Scherer, D., Mller, A., & Behnke, S. (2010). Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. *Artificial Neural Networks ICANN 2010 Lecture Notes in Computer Science*, 92-101. doi:10.1007/978-3-642-15825-4_10

[58] Springenberg, J.T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2015). Striving for simplicity: the all convolutional net. *In: ICLR-2015 workshop track*, 1-14. arXiv:1412.6806

[59] U. (2017, May 29). An Intuitive Explanation of Convolutional Neural Networks. Retrieved May/June, 2018, from https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[60] S. (2017, June 17). Convolutional Neural Network with TensorFlow implementation. Retrieved May/June, 2018, from https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117

[61] Zhang, Z. (2016). Derivation of Back Propagation in Convolutional Neural Network. University of Tennessee. Retrieved May/June, 2018, from https://pdfs.semanticscholar.org/5d79/11c93ddcb34cac088d99bd0cae9124e5dcd1.pdf

[62] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research 15*.

[63] Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278-2324. doi:10.1109/5.726791

[64] Janocha, K., & Czarnecki, W. M. (2017). On Loss Functions for Deep Neural Networks in Classification. arXiv:1702.05659v1

[65] G. (2018, July 04). Gwding/draw_convnet. Retrieved July, 2018, from https://github.com/gwding/draw_convnet

[66] Lee, C., Gallagher, P., & Tu, Z. (2018). Generalizing Pooling Functions in CNNs: Mixed, Gated, and Tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 40*(4), 863-875. doi:10.1109/tpami.2017.2703082

[67] Vapnik, V., & Cortes, C. (1995). Support-Vector Networks *Kluwer Academic Publishers 20*, 273-297.

[68] Blanz, V., Schlkopf, B., Blthoff, H., Burges, C., Vapnik, V., & Vetter, T. (1996). Comparison of view-based object recognition algorithms using realistic 3D models. *Artificial Neural Networks ICANN 96 Lecture Notes in Computer Science*, 251-256. doi:10.1007/3-540-61510-5_45

[69] Schmidt, M. (1996). Identifying speaker with support vector networks. *In Interface 96 Proceedings*.

[70] Osuna, E., Freund, R., & Girosit, F. (1995). Training support vector machines: An application to face detection. Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. doi:10.1109/cvpr.1997.609310

[71] Dastidar, J. G., Basak, P., Hota, S., & Athar, A. (2018). SVM Based Method for Identification and Recognition of Faces by Using Feature Distances. *Advances in Intelligent Systems and Computing Intelligent Engineering Informatics*, 29-37. doi:10.1007/978-981-10-7566-7_4

[72] Silva, S., Pereira, R., & Ribeiro, R. (2018). Machine learning in incident categorization automation. *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*. doi:10.23919/cisti.2018.8399244

[73] Cristianini, N., & Shawe-Taylor, J. (2006). *An introduction to support vector machines: And other kernel-based learning methods*. Cambridge: Cambridge University Press.

[74] Chih-Wei, H., Chih-Chung, C., & Chih-Jen, L. (2016). A Practical Guide to Support Vector Classification Department of Computer Science. National Taiwan University, Taipei 106. Retreived from http://www.csie.ntu.edu.tw/ cjlin

[75] Saritha, C., Sukanya, V., & Murthy, Y. M. ECG Signal Analysis Using Wavelet Transforms. (2008) *Bulg. J. Phys. Chapter 35*, 68-77.

[76] Ranjan R., & Giri, V. K. (2012). A Unified Approach of ECG Signal Analysis. *International Journal of Soft Computing and Engineering (IJSCE) 2*(3), 5-10.

[77] Jiang, C., Song, S., & Meng, M. Q. (2017). Heartbeat classification system based on modified stacked denoising autoencoders and neural networks. 2017 IEEE International Conference on Information and Automation (ICIA). doi:10.1109/icinfa.2017.8078961

[78] Zihlmann, M., Perekrestenko, D., & Tschannen, M. (2017). Convolutional Recurrent Neural Networks for Electrocardiogram Classification arXiv:1710.06122v1

[79] Rajni, R., & Kaur, I. (2013). Electrocardiogram Signal Analysis - An Overview. *International Journal of Computer Applications, 84*(7), 22-25. doi:10.5120/14590-2826

[80] Basics of ECG- Interpretation of waves and intervals. (2016, June 12). Retrieved May/June, 2018, from http://epomedicine.com/medical-students/ecg-interpretation-waves-intervals/

[81] Keogh, E. J., & Pazzani, M. J. (2000). Scaling up dynamic time warping for datamining applications. *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 00*. doi:10.1145/347090.347153

[82] Gustavo E. A. P. A. Batista, Prati, R. C., & Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter, 6*(1), 20. doi:10.1145/1007730.1007735

[83] Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters, 27*(8), 861-874. doi:10.1016/j.patrec.2005.10.010

[84] Wang, J. (2013). Proposed new requirements for testing and reporting performance results of arrhythmia detection algorithms *Computing in Cardiology 2013* Zaragoza, 967-970.