UNIVERSIDAD TECNOLÓGICA CENTROAMERICANA

FACULTAD DE INGENIERÍA

PROYECTO DE INVESTIGACIÓN

IMPLEMENTACIÓN DE UNA RED NEURONAL CONVOLUCIONAL PARA LA AUTONOMÍA DE UN ROBOT

PREVIO A LA OBTENCIÓN DEL TÍTULO:

INGENIERO EN MECATRÓNICA

PRESENTADO POR:

21541272 DARWIN VALENTÍN LUQUE MADRID

ASESOR: ING. ALBERTO MAX CARRASCO

CAMPUS: SAN PEDRO SULA; JULIO, 2020

DEDICATORIA

El presente proyecto es dedicado a todas las personas que confiaron en mí, pero principalmente a mi madre y a mi padre quienes me dieron todo lo que estuvo a su alcance para que me pudiera preparar como un profesional, y quienes me han guiado e inspirado a ser la persona que soy.

AGRADECIMIENTO

Sobre todas las cosas le doy gracias a Dios quien me ha guiado e iluminado a lo largo de mi trayectoria como estudiante, pero sobre todo como persona. A mi madre que a pesar de todas las circunstancias ha estado siempre ahí para apoyarme, guiarme y para sacrificarse para verme convertido en la persona que soy. A mi padre que siempre me dio todo lo que estuvo a su alcance para que me pudiera superar todos los días y que ahora desde el cielo sé que me cuida.

A mis tíos, Karla Madrid, Allan Madrid y Marvin Madrid, quienes estuvieron ahí para mí para apoyarme tanto moral como económicamente. A mis abuelos Valentín Madrid e Isabel Bautista quienes a través de su experiencia y cariño me han enseñado a respetar, ser leal, ser responsable y, sobre todo, a ser un buen ser humano.

A Bessy Alcerro quien me ha apoyado, comprendido y estado conmigo en los momentos de celebración, pero más que todo en los momentos más complicados. A Eduardo Luque quien sin su apoyo y consejos este proyecto no pudo haber sido posible.

A mis compañeros y amigos que estuvieron conmigo durante esta etapa como estudiantes con quienes estudiamos, aprendimos y nos apoyamos para poder sobre llevar nuestra carrera de la mejor forma posible, Marlon Delcid, Marvin Lorenzana, Lester Torres, Emerson Isaula, Luis Pineda, Cristian Cisneros, Marcelo Rodríguez, Josué Vargas y Nayra Valle.

EPÍGRAFE

"Si algo es lo suficientemente importante, incluso si las probabilidades están en tu contra, debes seguir intentándolo."

- ***Elon Musk***

RESUMEN EJECUTIVO

Honduras, hasta estos momentos, tiene muy pocas investigaciones para realizar robots que se conduzcan de manera autónoma, por lo que esta investigación le apunta a ser un punto de arranque para este tema y una innovación para la industria logística. El propósito es realizar un robot que se conduzca el mismo haciendo uso de equipo estándar de cámara y un software. Por medio de la librería de Open CV, un paquete de recurso abierto de Python, y una red neuronal convolucional, fue modelada una red neuronal que fuera capaz de clasificar los cinco posibles movimientos que tiene el robot en uso VPR, el robot de función logística utilizado en esta investigación. La red fue hecha utilizando la librería Keras y optimizada utilizando el algoritmo de Adam. El modelo logro alcanzar una precisión de predicción de movimiento del 98.7% debido a la imagen proveída por la cámara. Esta alta precisión le permite al usuario poder confiar en la habilidad de la red de tener al robot moviéndose autónomamente en un trayecto especifico con poca supervisión necesaria.

Palabras Claves: autónoma, Open CV, Python, red neuronal convolucional, Keras, Adam.[AMCB1]

ABSTRACT

As of this day, Honduras has few investigations for making autonomous driving robots, so this paper aims to be a kick-starter for the topic and innovation in the logistics industry. Its purpose is to make a self-driving robot using standard camera equipment and computer-generated software. By means of the CV library, a Python open-source package, and a convolutional neural network (CNN), the authors were able to model a CNN that is capable of classifying the 5 different movements the VPR robot, the logistic-based robot used in this research, can do. The network was built upon the Keras library and optimized with the Adam algorithm. The model was able to achieve 98.7% movement prediction accuracy based on the camera's input. This high accuracy rate allows the user to rely on the network's ability to have the robot autonomously go through a determined trajectory with minimum supervision required.

Keywords: autonomous, Open CV, Python, convolutional neural network, Keras, Adam.

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

I. INTRODUCCIÓN

Con el tiempo se han hecho grandes esfuerzos que han sido dedicados a generar técnicas para la identificación de sistemas lineales no variantes en el tiempo. La identificación lineal está basada en la medida de los valores de las entradas y salidas del sistema, (Specht, 1991). Narendra & Parthasarathy (1990)[AMCB2] demostraron que se puede utilizar redes neuronales para identificación y control de sistemas no lineales dinámicos. ¿Pero que son las redes neuronales? Las redes neuronales artificiales son técnicas de aprendizaje automático populares que simulan los mecanismo de aprendizaje de un organismo biológico, (Aggarwal, 2018). Estas redes neuronales aprovechan diferentes tipos de algoritmos, como el de propagación hacia atrás, el de propagación hacia adelante y diferentes tipos de algoritmos de optimización, para poder formar un sistema con distintas capas para poder, por medio de un entrenamiento, predecir sus salidas con respecto a un set de entradas.

Las redes neuronales forman parte de una rama del aprendizaje automático que son los algoritmos de aprendizaje supervisado. Estos se basan en un entrenamiento previo por el cual se obtiene un sistema que luego, dependiendo de su eficiencia, puede predecir posibilidades que por medio de un valor limite estos valores pueden definirse como ceros o unos en las salidas. En cuanto este concepto de aprendizaje automático, este es el arena de estudio que le da a las computadoras la habilidad de poder aprender a solucionar problemas o a realizar operaciones sin ser explícitamente programadas, (Samuel, 1959). Estos aprendizajes automáticos se pueden entender mejor hay que analizarlos desde su rol en un campo más amplio de la computación. Sin embargo, debido al amplio rango de paradigmas que existe en el área de la computación, ¿cuál es el nicho a ser llenado por este aprendizaje automático? Se puede decir que este aprendizaje automático es un sistema que hace que una computadora aprenda por una experiencia E con respecto a una tarea T sobre un rendimiento P, todo esto para mejor la experiencia E, (Mitchell, 1997). En el presente proyecto se sugiere usar una red neuronal para que un robot de logística puede movilizarse desde un punto A, que puede ser un punto cualquiera, hacia otro punto B, que será un punto fijo y señalizado de alguna forma por definir.

En la presente investigación se buscará poder generar un modelo de red neuronal de manera de poder hacer que el robot VPR se pueda mover de manera autónoma a través de un trayecto cerrado. También se generará un software para la recolección de datos del robot VPR haciendo uso de un Windows Form. Para conocer si la red neuronal está funcionando debidamente se medirá la precisión que esta tiene sobre un conjunto de prueba.[AMCB3] Se le estará haciendo un chequeo a VPR, de manera de saber si está funcionando debidamente y si habrá adaptaciones necesarias a realizar. Con el robot y el software listo se procederá a entrenar el robot. Recolectando todos los datos necesarios para que la red pueda ser entrenada debidamente.

II. PLANTEAMIENTO DEL PROBLEMA

En este capítulo se estará describiendo el origen de la problemática, el problema como tal y la razón por la cual es importante resolver esta problemática. Así mismo se discutirán las preguntas que se estarán planteando a través de la investigación y en base a estas definir los objetivos del proyecto.

2.1 PRECEDENTES DEL PROBLEMA

En los últimos 50 años ha habido extensas investigaciones en cuanto a la robótica y la inteligencia de los sistemas. Sin embargo, muchas de estas investigaciones han atacado problemas técnicos bien específicos, mejoras y avances en esta área que lleva a los sistemas y soluciones a un profundo impacto en la sociedad. Fundamentalmente la mayoría de los avances se dan por una mejora exponencial sin precedentes de la tecnología informática, (Reddy, 2006). Carros que se conducen solos, agentes digitales inteligentes capaces de interactuar con un usuario y los robots en general están avanzando rápidamente, (Smith & Anderson, 2014).

Actualmente la sociedad está pasando por una situación muy difícil, como lo es la pandemia COVID-19. Esta pandemia ha venido a demostrar lo valioso que es un apretón de mano y lo importante que puede llegar a ser un distanciamiento social. Hoy más que nunca las personas mantienen una distancia debido a una situación muy complicada como lo es la pandemia, ya que claro nadie quiere contraer el virus. La situación emergente que se generó debido a la pandemia COVID-19 se puede observar la gran necesidad que pueden tener las soluciones robóticas para muchos problemas, desde aplicaciones medica hasta las aplicaciones logísticas. Según las fábricas, bodegas y almacenes van generando nuevas estrategias y protocolos necesarios para mantener a sus empleados trabajando de una manera segura, los robots salen más a flote en áreas como lo es la manufactura, (Logística, 2020).

2.2 DEFINICIÓN DEL PROBLEMA

En Honduras existe una falta de información y de trabajos de investigación que se direccionen a la inteligencia artificial y aprendizaje automáticos. Por lo que uno de los propósitos de esta investigación es poder dar un arranque a este tipo de investigaciones. Esta

falta de información hace que Honduras este atrasado, tecnológicamente hablando. Por lo que poder generar un modelo de red neuronal que logre generar la autonomía de un robot puede llegar a ser un gran paso. Especialmente para la sociedad hondureña que no tienen un camino claro en esta temática de redes neuronales y visión computacional.

2.3 JUSTIFICACIÓN DEL PROBLEMA[AMCB4]

Particularmente en el sector industrial y en países de alta demanda laboral, la automatización y el uso de productos robóticos están liderando a las empresas a ahorrarse mucho dinero en cuanto a costo de mano de obra y productos. Mientras a una empresa alemana de la industria automotriz le cuesta 40 dólares la hora de producción, el uso de un robot le cuesta a una empresa con robótica integrada entre 5 y 8 dólares la hora, (Wisskrichen *et al.*, 2017)[AMCB5]. Según datos del IDC los gastos que invierten las empresas en robótica para lo que será el 2021 alcanzaran la cifra aproximada de 230 millones de dólares para el 2021. Dato que fue obtenido previo a la pandemia, esto sugiere que la inversión debido al necesario distanciamiento social va a incrementar. Igualmente se busca poder llenar un vacío de información que existe alrededor de la sociedad hondureña sobre el aprendizaje automático, redes neuronales y visión computacional. Todos estos temas unidos tienen un mismo fin y es poder un día obtener inteligencia artificial. Cabe recalcar que la presente investigación no busca aclarar tema aun de alta complejidad en potencias mundial. Si no más bien buscar en caminar a la sociedad hondureña a animarse a invertir en este tema y poder dar un gran paso tecnológico.

2.4 PREGUNTAS DE INVESTIGACIÓN

1. ¿Se puede aplicar una red neuronal a un robot de logística para el movimiento autónomo de este por medio visión computacional?
2. ¿Es necesario diseñar un software para poder conseguir la información necesaria para entrenar la red neuronal?
3. ¿Qué adaptaciones son necesarias para que el robot logre funcionar con la red neuronal a diseñar?
4. ¿Qué modelado o arquitectura mejor se adaptaría al movimiento del robot por medio de la visión computacional?

5. ¿Qué algoritmo de optimización se puede utilizar para poder definir un buen modelo que logre predecir los valores de la manera más precisa?

2.5 Objetivos

Los objetivos son la parte más crucial a definir en una investigación científica, ya que estos moldean el camino a seguir para cumplir con las expectativas esperadas del proyecto. Estos se basan en las preguntas de investigación planteadas en la sección anterior. Se plantearan los objetivos específicos los cuales al cumplirse deben definir el camino para cumplir con el objetivo general, y se podría decir, principal de la investigación.

2.5.1 Objetivo General

Implementar una red neuronal para el movimiento autónomo por medio de visión computacional de un robot logística.

2.5.2 Objetivos Específicos

1. Realizar un programa para poder recopilar toda la información necesaria para entrenar la red neuronal, pudiendo este programa adaptarse al robot en uso.
2. Realizar las adaptaciones necesarias al robot en uso de manera de poder utilizarlo con la red neuronal a diseñar.
3. Definir la arquitectura de la red neuronal para el procesamiento de imágenes y la clasificación de las posibles salidas
4. Establecer el algoritmo de optimización y sus parámetros para el entrenamiento de la red neuronal.

III. MARCO TEÓRICO

En el presente capítulo se estará describiendo la teoría de sustento necesaria para la realización del proyecto de investigación. Se tocará el tema sobre el aprendizaje automático, se hablará sobre las dos principales ramas de estos, el aprendizaje supervisado y el no supervisado. Se analizarán los conceptos matemáticos de los algunos algoritmos de aprendizaje supervisado necesarios para la comprensión concreta de una red neuronal. También se describirá a fondo el robot al cuál se le estará aplicando esta red neuronal y, finalmente, se explicará a fondo la metodología que se estará utilizando para llevar a cabo el presente informe.

La investigación sobre el aprendizaje automático se dio a través del sueño de poder llegar algún día generar inteligencia artificial. Turing (1950) propuso una pregunta, ¿pueden las maquinas pensar? Esto debería de iniciar con la definición de los conceptos "maquina" y "pensar", (Turing, 1950). Ahora por un tiempo el concepto de aprendizaje automático e inteligencia artificial tomaron caminos distintos. Debido a que el aprendizaje automático toma valores y datos reales para entrenar sus circuitos y funcionamientos; este era un concepto que de alguna manera contradecía la idea de inteligencia artificial y del hecho de que esta área proponía hacer absolutamente todo de manera artificial. Ahora la temática que vino a unir estos dos conceptos es la de redes neuronales. Estas a pesar de ser entrenadas con datos reales genera una arquitectura totalmente artificial y que se entrena para que un set de entradas, o como se le nombrará después, un nuevo ejemplo produzca una o varias salidas que es de tipo discreta.

Irónicamente la primera red neuronal se dio antes que Turing se hiciera la pregunta descrita en el párrafo anterior. Fue en 1943 cuando Warren S. McCulloch y Walter H. Pitts decidieran crear un sistema basado en el sistema neuronal biológico a base de circuitos eléctricos, de esta manera entonces naciendo las redes neuronales. Debido al característica de todo o nada de la actividad nerviosa, los eventos neuronales y la relación entre estas pueden ser tratadas con un base de lógica proposicional, (McCulloch & Pitts, 1943). Sin embargo, esta investigación fue más una moderna descripción matemática en aquel momento. La primera red neuronal artificial fue Perceptrón diseñada por Frank Rosenblatt en 1958. Un modelo del cerebro puede en realidad ser construido, de una manera física, como una ayuda a determinar

sus potenciales lógicos y rendimiento; esto como sea no es una característica esencial del acercamiento del modelado. La esencia de un modelado teórico es que es un sistema con propiedades conocidas, fácilmente susceptible a análisis, los cuales son hipotetizados para incorporar las características esenciales del sistema con propiedades ambiguas o desconocidas; en el presente caso el cerebro biológico, (Rosenblatt, 1961).

3.1 Aprendizaje Automático

Como ya se había mencionado, los sistemas de un aprendizaje automático se pueden ver como un programa que aprende por una experiencia E con respecto a una tarea T y se le realizada una medición de rendimiento P, si su rendimiento en T, medido por P, mejora su experiencia E. Debido a como se presente la información de entrada digamos los algoritmos de aprendizaje tiene dos grandes ramas, los algoritmos de aprendizaje supervisado y no supervisado.

Pero previo a explicar los tipos de algoritmos de aprendizajes que hay es importante saber cómo funciona en síntesis este tipo de sistemas. Este tipo de mecanismo de software lo que hace es primero conseguir un set de entrenamiento, basado en alguna fuente fidedigna, y alimentar al algoritmo de aprendizaje con este set. Luego el algoritmo va a generar una hipótesis o modelado, el cual al recibir nuevos ejemplos debe estimar valores definidos como salidas, que dependiendo la clase puede ser regresiva o clasificatoria, (Flach, 2012)

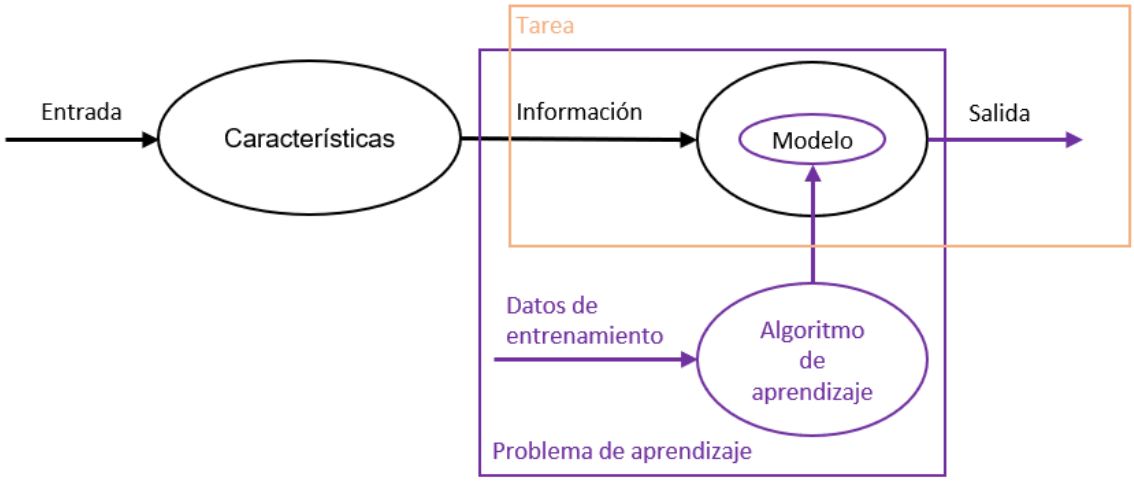


Ilustración 1- Modelado de como un algoritmo de aprendizaje automático ataca una tarea dada.

Fuente de la Imagen: (Flach, 2012).

Es importante saber la forma en la que se representará las entradas y salidas en un sistema de aprendizaje automático. Por lo que la representación matemática para un set de entrenamiento es la siguiente.

$$\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \dots, (x^{(m)}, y^{(m)})$$

Ecuación #1. Representación matemática de un set de entrenamiento.

Donde $(x^{(i)}, y^{(i)})$ es un set en específico, $x^{(i)}$ es un vector de dimensión $n + 1$ de la cantidad de características de las variables de entradas que hay, donde n es esa cantidad de características, y m es la cantidad de ejemplos tomados para el set de entrenamiento, también se puede ver como la longitud del set de entrenamiento. Este modelado, predicción o hipótesis se representa como una gráfica continua en el espacio de las variables que permite asignar un valor a cualquier variable desconocida, eso es en cuanto al análisis de la información. Esto puede predecir el valor del precio al cual se venderá algo en el futuro, o, alternativamente, puede significar seleccionar que tipo de documento es un documento, valga la redundancia, (Kelleher et al., 2015).

3.1.1 Aprendizaje Supervisado

El aprendizaje supervisado es un rama del aprendizaje automático que busca poder predecir una salida con respecto a un set de variables de entradas y que se entrena por medio de un set de entrenamiento que se dice esta etiquetado, ya que se conoce tanto la entrada como la salida de este set de entrenamiento, (Mohri et al., 2012). Este tipo de aprendizaje se puede dividir en dos clases, si algoritmo tiene salidas regresivas o clasificatorias, (Russell & Norvig, 2010). Lo más importante a tomar en cuenta es que todos los sets de entrenamiento tienen que ser valores dados con su respuesta correcta, ósea que la salida y concuerde con los datos dados por el conjunto de variable x.

Algunas aplicaciones de este tipo de aprendizaje automático son, como ya se había mencionado, la predicción de precios de algún objeto, en la bioinformática, en reconocimiento de voz, detección de spam, estructuras cuánticas, etc. Algunos algoritmos son el algoritmo de

regresión lineal, el algoritmo de regresión logística, redes neuronales, algoritmo de propagación hacia atrás, estimación de núcleos, máquinas de soporte de vectores, etc.

3.1.1.1 Regresión

Cuando un aprendizaje automático es regresivo se dice que su valor de salida es continuo sobre la medida de las entradas, (NG, 2011) Para poner un ejemplo sobre a un aprendizaje supervisado que cabe dentro de una clase regresiva es crear un sistema que prediga el costo de un carro, de una marca y edición en específico. Ahora para esto se tiene que plantear un set de entrenamiento donde el precio varía según un set de variables, por sentido de visualización se dice que el precio del carro nada más varía según la cantidad de millaje que tiene el carro recorrido. Los precios como tales se dice que se tomaron de diferentes establecimientos de venta de carros y evaluadores profesionales. De manera entonces que se obtiene la siguiente grafica.

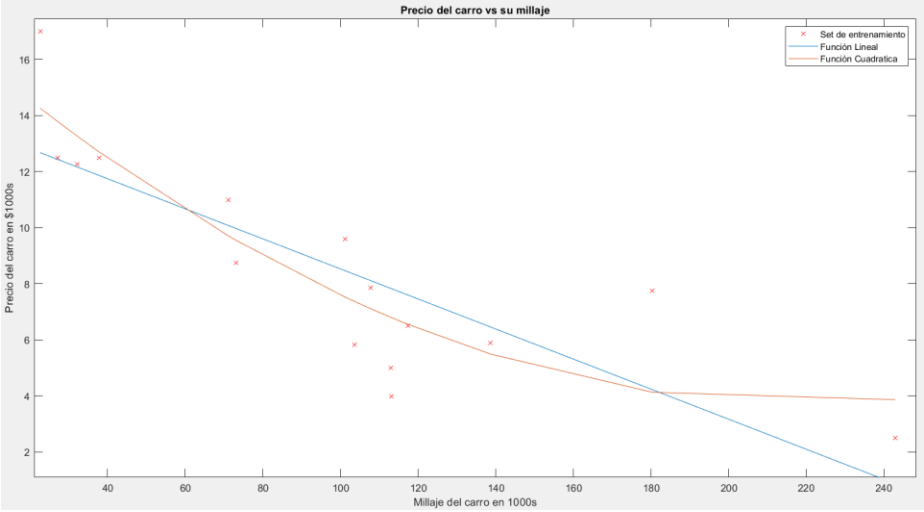


Ilustración 2- Gráfica de la predicción de precio de un carro con respecto a su millaje

Fuente de la imagen: Elaboración propia.

En la ilustración 2 entonces se pueden observar diferentes valores de precios de carros con respecto a la cantidad de millaje que estos tiene. Por lo que se observa diferentes valores con respecto a distintos valores de millaje porque claro a pesar de considerar la misma marca y modelo hay factores, digamos, humanos que afectan este tipo de linealidad. Lo importante de

observar es la variedad de valores que se pueden presentar y que la predicción es un valor regresivo o continuo en el espacio de las características de las variables de entrada.

Sabiendo entonces sobre la continuidad se plantea una hipótesis que se ajuste de la mejor manera al set de entrenamiento. Se puede plantear una gráfica lineal, como se observa, en la imagen, pero van a haber muchos casos donde este tipo de función seguramente no sea el mejor ajuste. Entonces se puede aplicar una función cuadrática, como se observa en rojo. Sin embargo, el grado de la función puede aumentar según la necesidad del entrenamiento. Pero se va a analizar más adelante si usar funciones de grados muy altos pueda afectar de algún modo la precisión al momento de tomar un set de prueba.

3.1.1.2 Clasificación

Ahora entonces cuando un algoritmo de aprendizaje supervisado es considerado de clasificación es porque este maneja valores entre 0 y 1 dependiendo de sus características de entrada, (NG, 2011). Un ejemplo para poder visualizar un aprendizaje supervisado de clase clasificatorio se puede plantear uno de querer saber si un tumor es maligno o benigno dependiendo del tamaño que este tenga. Suponiendo que se tienen varios valores y que estos se tomaron de una fuente fidedigna, se puede realizar una gráfica de la siguiente manera. Cabe recalcar que por motivo de visualización solo se tomara en cuenta la característica ya descrita, pero en realidad hay más factores que afectan esta clasificación sobre este tema.

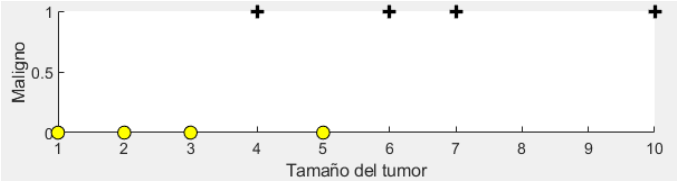


Ilustración 3- Gráfica de malignidad vs. Tamaño del tumor

Fuente de la imagen: Elaboración propia.

En esta ilustración 3 se puede observar que la información solo cabe dentro de dos parámetros 1 o 0. Por lo que entonces la hipótesis a medir tiene que devolver valores dentro de este sistema. Sin embargo, las funciones que se pueden utilizar no dejan de ser regresivas por lo que se ocupa una función que maneje el rango de valores entre 0 y 1 y se pueda medir posibilidades. Así regir si un valor cabe en un lado o en el otro debido a un valor limite. Este

concepto es muy importante al momento de hablar de redes neuronales ya que estas manejan este mismo concepto en sus salidas. Ya que las redes neuronales son también de tipo clasificatorias. Se estará explicando más a profundidad eso más adelante.

4.1.2 Aprendizaje No Supervisado

La gran diferencia que hay entre un algoritmo de aprendizaje supervisado y uno de no supervisado es que el set de entrenamiento que recibe un algoritmo de aprendizaje no supervisado no está etiquetado, (Flach, 2012). El hecho de que no esté etiquetado se puede plantear como el hecho de que el set de entrenamiento solo conoce las variables de entrada y no sabe que son o como debe de actuar en la salida. La representación matemática de estos sets de entrenamiento es la siguiente.

$$x^{(1)},\ x^{(2)},\cdots,\ x^{(m)}$$

Ecuación #2. Representación matemática de un set de entrenamiento para un aprendizaje no supervisado.

Lo que este tipo de algoritmos busca hacer es tomar esta información y dividirla en grupos con la información que tiene una relación o parecido. También es conocido como ser un auto organizador permitiendo modelar una densidad de probabilidades sobre las entradas del entrenamiento, (Hinton & Sejnowski, 1999). Las representaciones matemáticas de este tipo de algoritmos se pueden visualizar de la siguiente manera, que por motivos de lo mismo solo van a plantear dos variables de entradas.

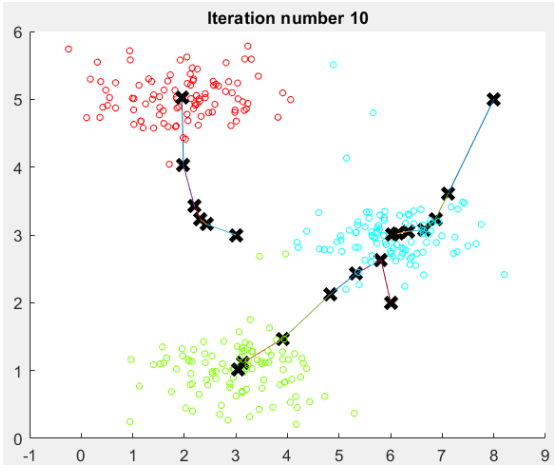


Ilustración 4- Visualización de un algoritmo de aprendizaje no supervisado

Fuente de la imagen: (NG, 2011)

En esta ilustración 4 se puede ver que la información, la cual son casos de dos variables, se distribuye en diferentes grupos que se pueden distinguir por los tres distintos colores. El algoritmo lo que hace es aleatoriamente iniciar 3 señalizaciones en el plano y luego comienza a buscar un punto medio de la información que más se parezca a la respectiva señalización, (Hinton & Sejnowski, 1999). Este algoritmo en específico se llama K-medias.

Las mayores aplicaciones para este tipo de algoritmos son la organización de grupos de computación, análisis de redes sociales, segmentación de mercados, análisis de información astronómica, etc. Donde la información que le está entrando no necesariamente se sabe que es, pero se puede separar en diferentes grupos por similitud en sus características. Otros algoritmos de este tipo de aprendizaje automático son el algoritmo de agrupación jerárquica, el algoritmo OPTICS, el algoritmo de detección de anomalías, incluso algunos tipos de redes neuronales, como los autoencoders, mapas auto organizadores, etc.

3.2 Regresión Lineal

La regresión lineal es un algoritmo de aprendizaje supervisado que se utiliza para medir tendencia debido a un set de entrenamiento y de características, (Mohri et al., 2012). El primer paso importante de analizar es como se da la información en este tipo de algoritmos. La información de entrada para el entrenamiento se da etiquetado entonces se sabe el valor correcto de salida con respecto a esas entradas. Se puede entonces tomar el ejemplo que se utilizó en la ilustración 2, de manera de ejemplificar como se presentan los datos.

Tabla 1. Tabla de un set de entrenamiento

Millaje en 1000s (x)	Precio en \$1000s (y)
37.8	12.5
22.6	17
32.2	12.25
180.3	4.75
113.1	3.99
⋮	⋮

Fuente de la tabla: Elaboración propia

Esta información es necesario que sea proveniente de una fuente fidedigna de manera que el algoritmo como tal se adapte a valores reales que de manera que la aplicación del sistema sea para un problema real. Cabe destacar que los datos de la tabla 1 son valores inventando por motivo de ejemplificación. La siguiente cuestión importante a considerar es que hacer con esta información. Esta información sirve para generar una función que pueda predecir valores futuros aproximada. Para esto entonces se tiene que plantear la hipótesis que se va a estar utilizando. Para un caso lineal de una variable la hipótesis se puede plantear de la siguiente manera.

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Ecuación #3. Hipótesis para regresión lineal

Otra forma de poder plantear esta función es vectorizada. Ya que la variable x y los parámetros θ se pueden plantear como un vector, luego se explicará a fondo esto. Por lo que se puede reescribir la función de la siguiente manera.

$$h_\theta(x) = \theta^T x$$

Ecuación #4. Hipótesis para regresión lineal vectorizada

Donde en esta ecuación la función a representar son los valores aproximados de la predicción debido a un sistema entrenado. Los valores θ_0 y θ_1 son parámetros que dictan la apertura y pendiente de la gráfica, (Seber & Lee, 2003). Para observar el tipo de impacto que tienen estos parámetros sobre la gráfica se puede observar las siguientes graficas.

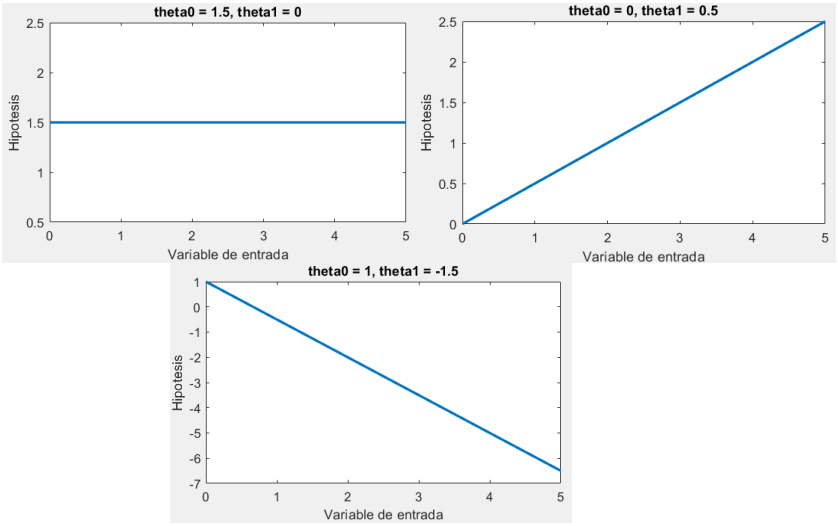


Ilustración 5- Representación gráfica de algunas posibles hipótesis.

Fuente de la imagen: Elaboración propia

La ilustración 5 ejemplifica 3 posibilidades de hipótesis debido a los parámetros de manera de observar cual es el efecto que estos tienen sobre la hipótesis como tal. Si θ_1 es o tiende a 0 pues se obtiene un valor constante en cualquier valor recibido, si θ_0 es o tiende 0 es porque el punto de origen de la gráfica o el ejemplo $(0\,,0)$ es parte del set de entrenamiento, por ejemplo, al estimar el precio de una casa en base a su área, claramente, una casa con 0 metros cuadrados tiene un precio de 0 dólares, y finalmente cuando el valor de θ_1 es negativo la tendencia de la gráfica es empezar en un punto alto debido a valores pequeños de entrada y terminar en algún punto bajo en valores altos en la entrada, un ejemplo podría ser el ejemplo, valga la redundancia, planteado en la ilustración 1, donde entre menor sea el millaje mayor es el precio del carro y entre mayor el millaje el carro se devalúa más.

3.2.1 Algoritmo de Descenso por Gradiente

El siguiente paso es considerar el error que esta grafica va a tener con respecto a los diferentes puntos del set de entrenamiento. Para calcular el error de la función de la hipótesis se utiliza una función llamada función de perdida. Esta lo que hace es tomar la diferencia entre el valor calculado por la hipótesis en cada una de las entradas del set de entrenamiento con las salidas respectivas del mismo set de entrenamiento, luego esta diferencia se eleva al cuadrado, y, finalmente se hace una suma de todos los valores obtenidos, (Taguchi, 1974). De tal forma que la ecuación se ve de la siguiente manera.

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}[h_\theta(x^{(i)}) - y^{(i)}]^2$$

Ecuación #5. Función de perdida

Planteada entonces la función de perdidas el objetivo que tiene el algoritmo como tal es conseguir un set de parámetros que reduzca lo más que pueda la función de perdida, (NG, 2011). Por lo que el objetivo de que busca el algoritmo es poder optimizar esta función de perdida, por lo que matemáticamente hablando este concepto se puede plantear de la siguiente manera.

$$\min_{\theta_0\theta_1} J(\theta_0, \theta_1)$$

Ecuación #6. Idea de optimización

Por lo que entonces se tiene que poder optimizar de alguna manera esta función de perdida con respecto a los parámetros de la hipótesis. Para eso se puede utilizar un algoritmo llamada descenso por gradiente. Este es un algoritmo de optimización de primer orden que busca punto local mínimo de una función diferencial. Este por medio de una cierta cantidad de iteraciones busca reducir el valor de la función a reducir en base al valor de los parámetros, tomando paso que son negativamente proporcionales a la derivada de primer orden de la función, (Cauchy, 1847). Aplicando entonces el algoritmo de optimización a la función de perdida de la hipótesis se obtiene el siguiente algoritmo.

$$repetir\ hasta\ convergencia\{\ \theta_j \coloneqq \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1), \qquad (para\ toda\ j)\ \}$$

Ecuación #7. Algoritmo de descenso por gradiente

Para poder visualizar el algoritmo en funcionamiento se va considerar un caso cualquiera de para una función de perdida para diferentes valores de θ_0 y θ_1. Obteniendo entonces la siguiente grafica.

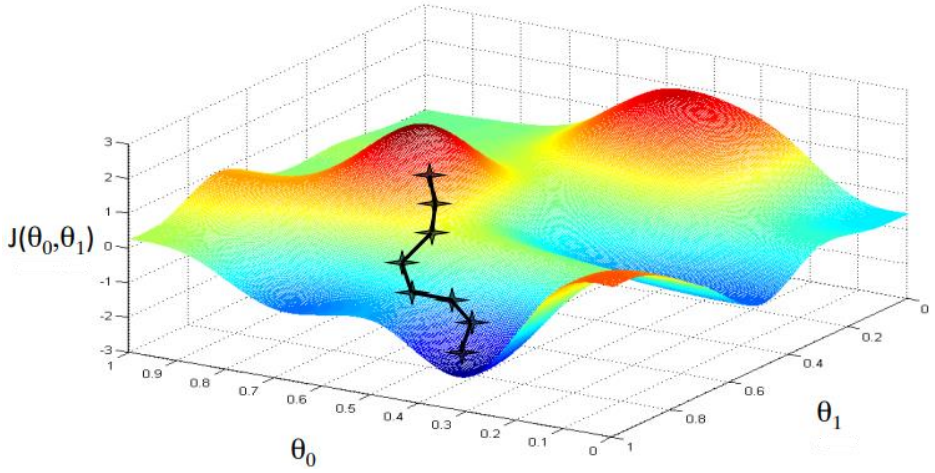


Ilustración 6- Representación gráfica del trabajo del algoritmo de descenso por gradiente

Fuente de la imagen: (NG, 2011)

Como se puede observar en la imagen al empezar en punto cualquiera para valores para θ_0 y θ_1, en este caso el primer valor es la X en la posición más alta sobre el área en rojo, el algoritmo busca ir cambiando los valores de θ_0 y θ_1 de manera de encontrar el menor valor para $J(\theta_0, \theta_1)$. Sin embargo, aquí hay algunos otros parámetros que también afectan a la forma de encontrar el mínimo global, como lo es la variable α, esta es la razón de aprendizaje. Este valor es una constante que define que tan grande son los pasos que va a dar el algoritmo. Como se puede observar en la ilustración 6 da varios pasos antes de llegar al punto global mínimo. La longitud de estos es definida por este coeficiente. Estos pasos van reduciendo según va avanzando el algoritmo, el cambio es algo insignificante para la vista y por eso no se observa muy bien en la gráfica. Esta reducción se da debido al termino diferencial. La derivada es la pendiente de la recta tangencial que pasa por el punto en específico, y esta evaluación como tal va reduciendo según el algoritmo se va acercando al global mínimo. Esto en específico es una ventaja al momento de llegar global mínimo, ya que en ese punto la recta tangencial que pasa por ahí es una línea horizontal cuya pendiente es igual a 0, (Jr., 2014).

El mayor problema que presenta el algoritmo es que al encontrarse con un local mínimo el algoritmo al llegar a un punto donde la evaluación de la derivada, debido a que la recta tangencial es una línea recta, el valor no puede buscar irse al mínimo global, como se puede ver en la siguiente grafica.

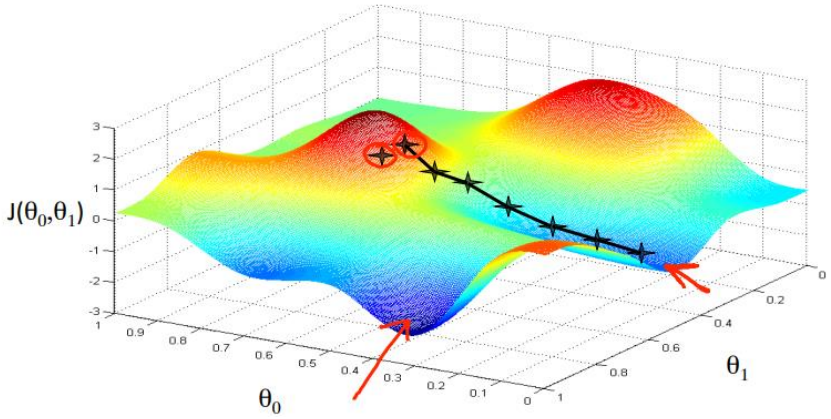


Ilustración 7- Descenso por gradiente trabado en un mínimo local.

Fuente de la imagen: (NG, 2011)

La inicialización de las variables empezó, hablando en términos de la gráfica, un poco más al fondo a la derecha. Por lo que al buscar el mínimo global el algoritmo en realidad se topó con un mínimo local. Sin embargo, muchas veces este problema no representa un error muy abrumador. Pero puede darse un caso en el que si lo sea por lo que esta problemática puede ser algo grave. Otra problemática del algoritmo es con el coeficiente de la razón de aprendizaje, cuando este se empieza en un valor muy grande lo que algoritmo va a hacer es divergir en vez de convergir. Sin embargo, empezar este valor en valores muy pequeños el programa se puede tardar mucho, en el mejor de los casos, a llegar al mínimo global y esto puede ser computacionalmente muy costoso. Por lo que es importante manejar el valor correcto de este. Lamentablemente no hay una formula o algoritmo que devuelva el valor correcto para este coeficiente y no queda más que realizar prueba y error. Se estará analizando un poco más a fondo este parámetro.

Al aplicar el algoritmo de descenso por gradiente parte del algoritmo es poder plantear la primera derivada de esta función de perdida. Ya que generar la pendiente de la recta tangente es la parte crucial de este algoritmo. La derivada, sin embargo, es parcial y dependiendo del parámetro que se busque conseguir, se tiene que derivar la función con respecto a ese parámetro en específico,(Kelleher et al., 2015). Por lo que se pondrá la función de perdida en función, valga la redundancia, de los parámetros de la hipótesis.

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}[\theta_0 + \theta_1 x^{(i)} - y^{(i)}]^2$$

Por lo que, derivando la función de perdida, se puede decir, modificada para ambos parámetros se obtienen las siguientes derivadas, (Jr., 2014).

$$\frac{\partial}{\partial\theta_0}J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x^{(i)} - y^{(i)})$$

$$\frac{\partial}{\partial\theta_1}J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x^{(i)} - y^{(i)}) \cdot x^{(i)}$$

Ecuación #8. Derivadas parciales con respecto a cada parámetro de la función de perdida.

Aplicando entonces estas derivadas parciales al algoritmo como tal, se obtiene la siguiente forma para el algoritmo de descenso por gradiente para regresión lineal de una variable queda de la siguiente manera.

$$repetir\ hasta\ convergencia\{$$

$$\theta_0 \coloneqq \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 \coloneqq \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \}$$

Ecuación #9. Algoritmo de descenso por gradiente para regresión lineal de una variable.

Una forma de poder potenciar estos algoritmos al momento de potenciar su computarización es vectorizarlos por lo que previamente se va aplicar un concepto más a la hipótesis y es agregar una variable más, esta variable será una x_0 y se renombrará la variable x como x_1, por lo que la hipótesis queda de la siguiente modificación.

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1$$

Ecuación #10. Hipótesis de regresión lineal modificada.

Esta nueva variable x_0 se dice que siempre tiene valor de uno, es también conocida como una variable parcial, (Delgado-Rodríguez & Llorca, 2020) lo que esta variable va a brindar a la ecuación es poder generar una matriz de dimensión $m \times 2$, al generar una matriz para la variable de entrada también es necesario poder generar un vector, en este caso, para θ, el cual tendría una dimensionalidad de 2, (NG, 2011). Por lo que la matriz de la variable, la cual se representara con una X y el vector de θ, el cual se representara con el mismo símbolo, se ven de la siguiente manera.

$$X = \begin{bmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ \vdots & \vdots \\ 1 & x^{(m)} \end{bmatrix}, \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

Ecuación #11. Vector X y matriz θ para regresión lineal de una variable

Por lo que al querer conseguir la hipótesis a base de esta matriz y vector se puede obtener al hacer una simple multiplicación de matrices entre estos dos. Ya que la cantidad de columnas de

la matriz X es igual al número de líneas del vector θ. Por razón de comprobación solo se va a considerar un solo ejemplo, y se va a plantear como la variable pura.

$$h_\theta(x) = [1 \quad x] \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \theta_0 + \theta_1 x$$

3.2.2 Regresión Lineal con Varias Variables

En la mayoría de casos presentes, se puede decir, en la vida real son con más de una variable; varias veces con mucho más de una tan sola variable. Por lo que se tiene que poder plantear la regresión lineal para varias variables. Se puede comenzar explicando las modificaciones planteando un simple ejemplo. En este caso se puede plantear el caso en el que se quiera desarrollar un sistema que pueda predecir el valor de una casa.

Tabla 2. Set de entrenamiento de precios de casas con varias variables

x_1 Tamaño (m^2)	x_2 No. de cuartos	x_3 No. de plantas	x_4 Edad de la casa	y Precio en $1000s
195	5	2	10	455
132	3	2	7	250
142	3	1	30	280
79	2	1	15	185
⋮	⋮	⋮	⋮	⋮

Fuente de la tabla: Elaboración propia.

De esta tabla se pueden sacar varias nuevas notaciones. Para señalizar un ejemplo en específico seguirá utilizando la misma notación, $(x^{(i)}, y^{(i)})$, donde m sigue siendo la cantidad de ejemplos en el set de entrenamiento, n, que es una nueva variable, es igual a la cantidad de variables, o características, en el sistema, θ_j es el parámetro de la variable o característica j, y $x^{(i)}{}_j$ es igual a la característica j en el ejemplo número i, (Kelleher et al., 2015). Donde estas notaciones tienen las siguientes características y formas.

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}, \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

Ecuación #12. Vectores de características y parámetros para varias variables

Por lo que para conseguir la ecuación de la hipótesis para regresión lineal con múltiples variables se tiene que hacer una multiplicación entre estos dos vectores. Pero para poder realizar dicha multiplicación hay que aplicarle la transpuesta a uno de estos dos vectores. Por lo que entonces la hipótesis se ve de la siguiente manera.

$$h_\theta(x) = \theta^T * x = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

Ecuación #13. Hipótesis para regresión lineal con múltiples variables.

Debido a ciertas modificaciones en la hipótesis es esencial ver cómo tanto la función de perdida como el algoritmo de descenso por gradiente se ven afectados por este cambio. En cuanto a la función de perdida el cambio es mínimo, es más que todo en la referencia a las variables, se toma en cuenta todo el vector de parámetros.

$$J(\theta_0, \theta_1, \dots, \theta_n) \rightarrow J(\theta)$$

En cuanto al algoritmo de descenso por gradiente lo primero que se puede considerar para que algoritmo funcione para varas variables es la derivada. Como se puede observar en la ecuación 7 en la derivada con respecto al primer parámetro, debido a que este no está acompañado por una variable, la sumatoria no se multiplica por una de las características. Ahora en el segundo parámetro si se multiplica la variable que lo acompaña. Sin embargo, si se plantea la variable que lo acompaña como la variable parcial, se puede decir que la sumatoria se multiplica por la variable del parámetro para el que se está derivando la función de perdida.

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot {x^{(i)}}_j$$

Ecuación #14. Derivada parcial para el parámetro j de la función de perdida.

Aplicando entonces la ecuación 14 al algoritmo de descenso por gradiente se obtiene la siguiente modificación para el algoritmo para varias variables.

$$repetir\ hasta\ convergencia\{\ \theta_j \coloneqq \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot {x^{(i)}}_j\,, \qquad (para\ toda\ j)\ \}$$

Ecuación #15. Algoritmo de descenso gradiente para varias variables.

Una pregunta muy interesante es, ¿cómo elegir las constate de razón de aprendizaje? Como ya se había mencionado no hay un procedimiento o algoritmo que devuelva el valor

óptimo de esta constante. No queda más que hacer prueba error y tomar el valor con las mejores estadísticas, (NG, 2011). Una buena forma de saber si el valor que se tomó para esta constante es graficar el error después de cada iteración y ver que el valor tiende a cero luego de cada iteración. Se puede decir que cada caso contrario indica que el valor elegido para esta constante no es el correcto. Probablemente haya algún caso donde luego de un buen número de iteraciones el número después de algunos, se puede decir, movimientos raros en la gráfica termine convergiendo. Sin embargo, se trata de mantener el sistema lo más sencillo posible entonces lo más indicado es controlar este valor de manera que este en su valor más óptimo.

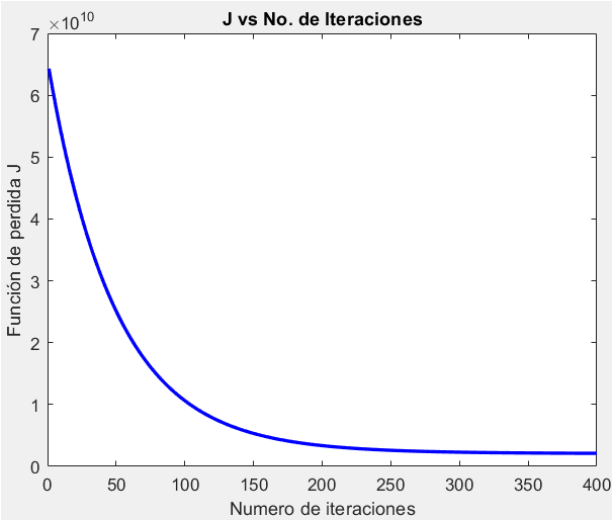


Ilustración 8- Gráfica de la función de perdida vs. el número de iteraciones que se le realicen, ejemplo óptimo.

Fuente de la imagen: (NG, 2011)

En la ilustración 7 se puede observar el caso óptimo de cómo se debe de comportar el algoritmo en cada iteración. En esta grafica también se puede observar que llegado a un valor entre 200 y 250 iteraciones es suficiente, ya que el cambio que se da a partir no es muy notable y este algoritmo cuando m y/o n son números grandes puede ser computacionalmente costos por lo que encontrar el valor optimo también para el numero de iteraciones es importante.

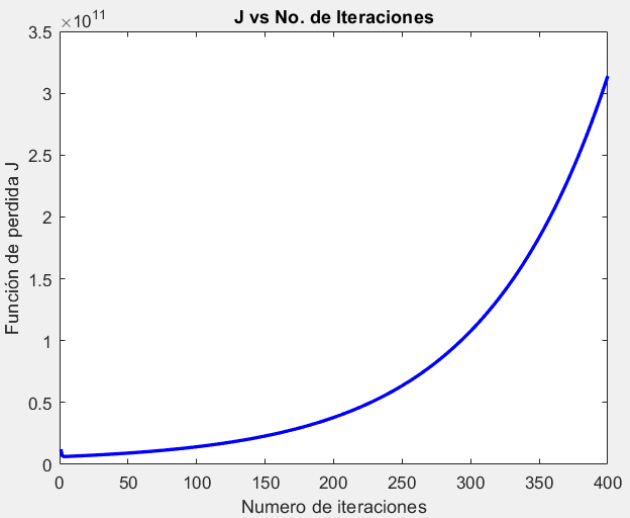


Ilustración 9- Gráfica de la función de perdida vs. el número de iteraciones que se le realicen, mal ejemplo.

Fuente de la imagen: Elaboración propia.

En esta ilustración 8 se puede observar un ejemplo de cómo una mala decisión de razón de aprendizaje puede hacer que el algoritmo termine divergiendo. Incluso pareciera que al principio el algoritmo en realidad está entregando un buen valor, pero estos valores para la función de perdida están en una escala de 10^{11} por lo que el algoritmo nunca en realidad entrega un buen valor y por ende lo más conveniente es elegir un mejor valor para la razón de aprendizaje.

3.2.3 Regresión Polinomial

Muchas veces la mejor forma de ajustar una función a un set de entrenamiento no es una línea recta, sino más bien una curva, (Seber & Lee, 2003). Por lo que entonces se pueden obtener muchas combinaciones de hipótesis con diferentes grados de funciones. Por ejemplo, en la gráfica de la ilustración 2 la gráfica en azul es una función cuadrática, en el caso de esas funciones la cuadrática tiene un menor error. Algunas hipótesis para este tipo de funciones pueden ser como las siguientes, que por motivos de ejemplificación solo se tomara un problema en función a una sola variable.

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2\ (a), \qquad h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 (b),$$
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 \sqrt{x}\ (c)$$

Ecuación #16. Ejemplos de hipótesis polinomiales

La forma de poder abordar este tipo de funciones exponenciales y radicales, y en realidad de cualquier tipo es igualarlas a otra variable de grado 1.

$$(a)\ x_1 = x, \qquad x_1 = x^2$$

$$(b)\ x_1 = x, \qquad x_2 = x^2, \qquad x_3 = x^3$$

$$(c)\ x_1 = x, \qquad x_2 = \sqrt{x}$$

Este procedimiento para poder atacar este tipo de hipótesis es procesar la información clasificándola en distintas variables a priori a cualquier procedimiento como la función de perdida, y el algoritmo de descenso por gradiente o cualquier otro algoritmo de optimización, (NG, 2011).

3.2.4 Ecuación Normal

Una forma intuitiva de generar el set de parámetros en un solo paso es a través de la ecuación normal, (NG, 2011). Esta es un acercamiento vectorial, se debe de manera el set de parámetros como un vector, el set de entrenamiento debe ser una matriz donde se debe de agregar la columna de los términos parciales, ósea una columna de unos como primera variable, y las salidas del mismo entrenamiento como un vector. La intuición es que para encontrar el punto menor de una función, se debe encontrar, dentro del dominio de una función, donde la derivada parcial de cada variable que conforma la función sea igual a 0, (Jr., 2014).

$$\frac{\partial}{\partial \theta_j} J(\theta) = 0$$

Ecuación #17. Intuición para obtener el mínimo global de una función

Por lo que vectorizando la ecuación 14 y aplicando la intuición planteada en la ecuación 17 se puede entonces plantear la siguiente igualación.

$$X^T X\theta - X^T y = 0$$

Al despejar esta igualación para el vector de parámetro θ se obtiene entonces así la ecuación normal para la solución de valores mínimos locales para la función de perdida para la regresión lineal.

$$\theta = (X^T X)^{-1} \cdot (X^T y)$$

Ecuación #18. Ecuación normal

Una buena pregunta es, ¿porque utilizar el algoritmo de descenso por gradiente, que en casos muy complicados puede ser computacionalmente costoso, y no utilizar la ecuación normal? El problema que puede tener la ecuación normal es que el termino $X^T X$ sea singular y por ende no ser invertible. Lo que causaría que el sistema quede en algún ciclo del que no va a salir o simplemente no corra la función. También si el número de características o variables de entradas y el número de set de entrenamiento son muy grandes la ecuación normal puede llegar a ser computacionalmente muy costoso, (Flach, 2012).

Tabla 3. Comparación entre el algoritmo de descenso por gradiente y la ecuación normal

Descenso por gradiente	**Ecuación Normal**
Desventajas	*Ventajas*
Se necesita elegir una constante α.	No se necesita elegir una constante α.
Necesita varias iteraciones.	Solo se necesita un paso.
Ventajas	*Desventajas*
Funciona bien con número grandes para la cantidad de características.	Se necesita computar $X^T X$, por lo que cantidades de características vuelven lenta la operación
No se debe preocupar por funciones no invertibles.	Se necesita computar $X^T X$, el cual si es singular la ecuación no se puede invertir.

Fuente de la imagen: (NG, 2011)

Una posible solución para poder solucionar el problema de no poder invertir el producto $X^T X$ puede ser reducir la cantidad de características. Revisar si no hay características redundantes o que en realidad puede ser omitidas. De esta manera cambiar la dimensión del sistema y poder conseguir una matriz que si se pueda invertir. Este problema no es común sin embargo es una gran problemática al toparse con él. Ya que este vuelve, se puede decir, inútil a la ecuación.

3.3 Regresión Logística

La regresión logística es un modelado de un acercamiento matemático que puede ser utilizado para describir una relación entre varias variables de entradas para obtener una variable dependiente, o una variable de salida, que es discontinua, o discreta, (Kleinbaum & Klein, 2002).

La representación matemática de la salida en un tipo de sistema de clasificación es el de un sistema binario.

$$y \in \{0,1\}$$

Ecuación #19. Definición de una salida para un modelo de regresión logística.

Cuando la variable es 0 se dice que está en la clase negativa, por ejemplo, si un correo no es spam, y se dice que la clase es positiva si la variable es 1, por ejemplo, que un correo sea spam. Cabe destacar que hay casos donde el sistema puede tener múltiples casos, se describirá con más profundidad eso luego. Ahora que tal si se intentara aplicar una regresión lineal a un tipo de problema como este. Lo que se podría hacer es aplicar el sistema y definir un valor como un punto limite y que ese límite sea nuestro límite de decisión, valga la redundancia. Sin embargo, esta línea recta en realidad no queda dentro de un rango aceptable de valores que varían entre 0 y 1, por lo que entonces aplicar este modelo de regresión lineal más bien arruina el sistema como tal. Y es por eso que se aplica el modelo de regresión logística definido al principio de la sección. Por lo que entonces se debe replantear ciertas características de la hipótesis para este nuevo modelo.

$$0 \leq h_\theta(x) \leq 1$$

Ecuación #20. Rango de valores para la hipótesis del modelo de regresión lineal

Hay una función matemática que hace que cualquier dominio de valores al ser evaluados queden dentro del rango definido en la ecuación 20, esta es la función de sigmoide. A esta función se le conoce como una función logística, y debido a esto es el nombre del modelo en general, (Tommiska, 2003). Siendo entonces la función de sigmoide la siguiente.

$$g(z) = \frac{1}{1 + e^{-z}}$$

Ecuación #21. Función de sigmoide

Una de las características más ventajosas que tiene esta función es la forma de su derivada, (Tommiska, 2003).

$$\frac{d}{dz} g(z) = g(z) \cdot (1 - g(z))$$

Ecuación #22. Primera derivada de la función de sigmoide

La grafica que esta función devuelve es una con una forma de "S" y también se le conoce a esta curva como la curva de sigmoide.

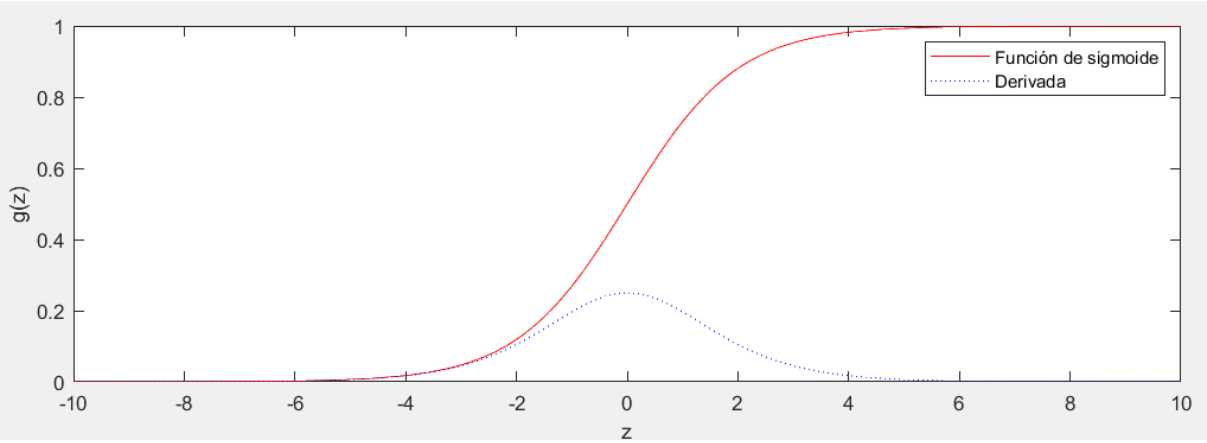


Ilustración 10- Gráfica de la función de sigmoide y su derivada

Algo muy interesante de esta función es que hay un punto de simetría cuando la variable z vale 0, la función en este punto está justo a la mitad de su rango, ósea 0.5. Por lo que replanteando la hipótesis para regresión logística utilizando la ecuación 21 y tomando la función vectorizada para la hipótesis de regresión lineal planteada en la ecuación 4, se obtiene la siguiente hipótesis.

$$h_\theta(x) = g(\theta^T x)$$

Ecuación #23. Hipótesis para regresión logística

Donde también se tiene un set de parámetros y se necesita un set de entrenamiento etiquetado. La intuición de dicha hipótesis es que esta estima una posibilidad de que una de las clases ocurra. Por ejemplo, tomando la gráfica de la ilustración 3, si la variable para el tamaño del tumor del paciente es un numero cualquiera que devuelve, debido a la ecuación 22, un valor de 0.7, entonces se le debe de decir al paciente que tiene un 70% de probabilidad de que su tumor sea maligno. Dicho esto, se puede plantear, matemáticamente, la intuición de esta hipótesis como la probabilidad de que la variable de salida sea 1 debido a las características del vector x parametrizado por el vector θ, (NG, 2011).

$$h_\theta(x) = P(y = 1|x; \theta)$$

Ecuación #24. Intuición de la hipótesis para regresión lineal

Partiendo entonces de la característica de simetría que tiene esta función en el punto $(0, 0.5)$, normalmente se puede decir, incluso aplicando leyes de redondeo, que si la función es 0.5 el valor se debe de redondear al valor más alto que le sigue, por ejemplo 0.26 se redondea a 0.3. En este caso solo se busca redondear a 0 o 1 en base a si la probabilidad se encuentra bajo este punto, o sobre o más arriba de este punto, respectivamente para cada valor. Por lo que entonces la función que se inserta en la función de sigmoide, que es $\theta^T x$, es la que define si un valor esta en un extremo o el otro. Entonces siendo esta función nuestra z para la función de sigmoide y definiendo la simetría como el límite de decisión, se puede plantear la siguiente característica.

$$\theta^T x \geq 0, \qquad \theta^T x < 0$$

Ecuación #25. Límite de decisión estándar para la hipótesis para regresión logística

Cabe destacar que este valor en realidad puede variar pero un estándar un valor que siempre va a ser una decisión razonable es este, (NG, 2011).

3.3.1 Limite de Decisión

El límite de decisión es el punto de medida el cual al estar en ese punto o haberlo pasado se dice que la salida es igual a 1, cualquier caso contrario indica que el valor de salida es 0. Por motivo de visualización se puede plantear el siguiente ejemplo, donde solo se tienen dos variables de entrada, y si una representa una clase positivo se marcará con una equis roja y si representa una clase negativa se marcará con un círculo azul.

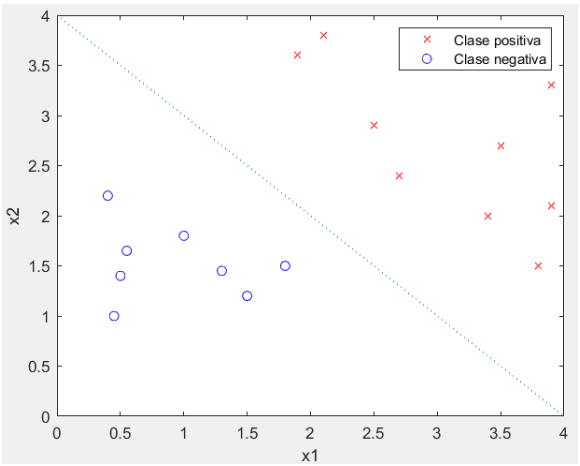


Ilustración 11- Grafica con ejemplo de visualización de un límite de decisión lineal

Fuente de la imagen: Elaboración propia

Planteando la ilustración 10 con solo los puntos de los ejemplos se puede notar que se puede separar fácilmente la decisión por una línea recta. Por lo que la función, que va dentro de la función de sigmoide, como una relación lineal de ambas variables.

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Por lo que entonces, suponiendo que se corrieron los algoritmos necesarios y ya se obtuvieron los valores de los parámetros y se obtuvo el siguiente vector de parámetros.

$$\theta = \begin{bmatrix} -4 \\ 1 \\ 1 \end{bmatrix}$$

Debido a este vector de parámetros y definiendo el valor límite de la función como el estándar, se obtienen las siguientes desigualdades.

$$y = 1;\ -4 + x_1 + x_2 \geq 0 \rightarrow x_1 + x_2 \geq 3$$

$$y = 0;\ -4 + x_1 + x_2 < 0 \rightarrow x_1 + x_2 < 3$$

Definiendo la curva, en función de la variable sobre el eje vertical, en este caso x_2.

$$x_2 = -x_1 + 3$$

Convirtiendo dicha desigualdad en una igualdad se puede obtener la recta que divide el punto de decisión del algoritmo. Obteniendo así la recta del límite de decisión. Sin embargo, las decisiones normalmente no se dan por una línea recta, de hecho con datos de la vida real y por

cómo se presentan tener una caso de este tipo lineal seria demasiada suerte, (Kleinbaum & Klein, 2002). Para decisiones no lineales se tienen que plantear funciones $z(x)$, que es la función que va en la función de sigmoide, como una función de diferentes grados. Por ejemplo, se puede plantear el siguiente caso, bajo las mismas condiciones de señalización que el anterior. En este caso se puede plantear una hipótesis cuadrática de la siguiente manera.

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 {x_1}^2 + \theta_4 {x_2}^2)$$

Basando siempre en que el límite de decisión es en 0.5, ósea cuando la función $z(x)$ es igual a 0. Dicho eso se puede plantear la siguiente curva, en base a la desigualdad planteada por el límite de decisión, como la función para el límite de decisión. Sabiendo también el set de parámetros.

$$\theta = \begin{bmatrix} -1.1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \qquad y = 1;\ -1 + {x_1}^2 + {x_2}^2 \geq 0 \rightarrow {x_1}^2 + {x_2}^2 \geq 1$$

Definiendo la curva, en función de la variable sobre el eje vertical, en este caso x_2.

$$x_2 = \sqrt{1 - {x_1}^2}$$

Planteando entonces la gráfica, con sus ejemplos y la curva para el límite de decisión.

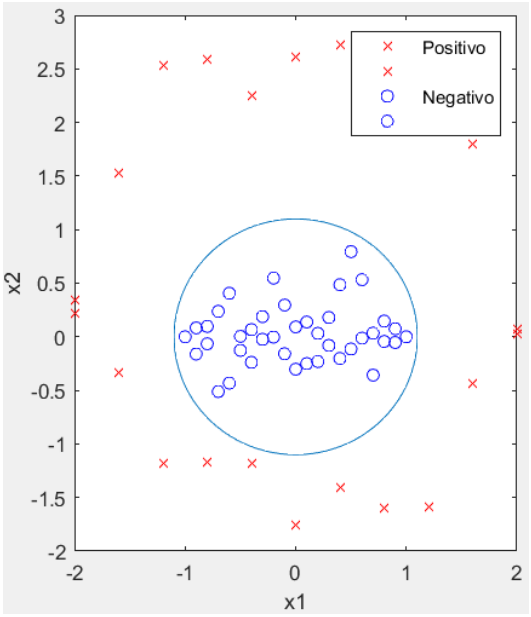


Ilustración 12- Grafica de visualización de límite de decisión no lineal

Pudiendo entonces adaptar el límite de decisión a distintos problemas y claro se puede aplicar la misma idea problemas más glorificados en cuanto al tamaño de n.

3.3.2 Función de Perdida para Regresión Logístico

Entonces, a pesar de incluir la función de sigmoide y de haber modificado ciertas formas de adaptar la hipótesis, siempre es crucial poder elegir el mejor set de parámetros para la hipótesis para la función $z(x)$. Por lo que para poder hacer un análisis se va a tomar la siguiente función de perdida.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(h_\theta(x^{(i)}), y^{(i)})$$

Ecuación #26. Intuición de la función de perdida para regresión logística

Ahora, por ejemplo, para la regresión lineal este nuevo termino $cost(h_\theta(x^{(i)}), y^{(i)})$ es igual al producto dentro de la sumatoria di dicha función, $[h_\theta(x^{(i)}) - y^{(i)}]^2$. Esta función sin embargo es una parábola que no necesariamente puede ser convexo. Y se logro demostrar el gran problema que puede causar que tal función no sea convexa.

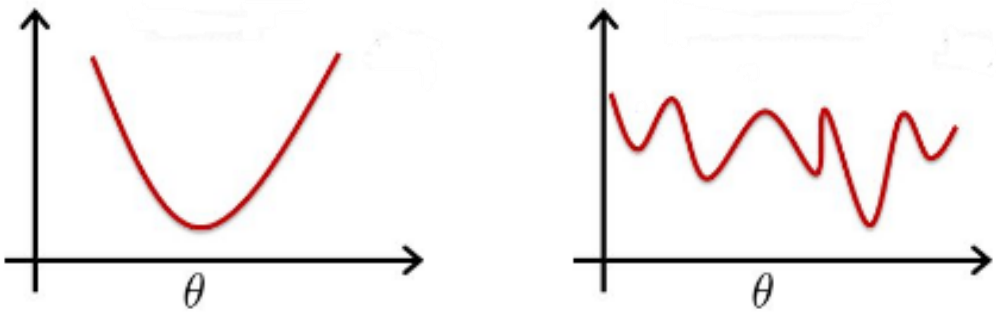


Ilustración 13- Graficas convexa y no convexa

Fuente de la imagen: (Genesis, 2018).

Esta función evaluada para la regresión logística es siempre una función no convexa, por lo que aplicar el mismo concepto de la función de perdida no es para nada efectivo. Sin embargo, encontrar una función adecuada para este sistema aplicando el mismo algoritmo de descenso por gradiente puede ser muy efectivo, (Kleinbaum & Klein, 2002). Por lo que se puede

aplicar el siguiente conjunto de funciones de perdida para los casos de las variables de salida y, (Hosmer et al., 2013).

$$y = 1; cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = -\log\left(h_\theta\left(x^{(i)}\right)\right), \qquad y = 0; cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = -\log\left(1 - h_\theta\left(x^{(i)}\right)\right)$$

Ecuación #27. Función de perdida seccionada para la regresión lineal

Planteadas estas ecuaciones se debe ver que efecto tienen estas y por qué funcionan. Una función logarítmica es convexa para cualquier rango entre 0 y 1, que en realidad es lo que interesa porque eso devuelve el valor de la hipótesis, $h_\theta\left(x^{(i)}\right)$.

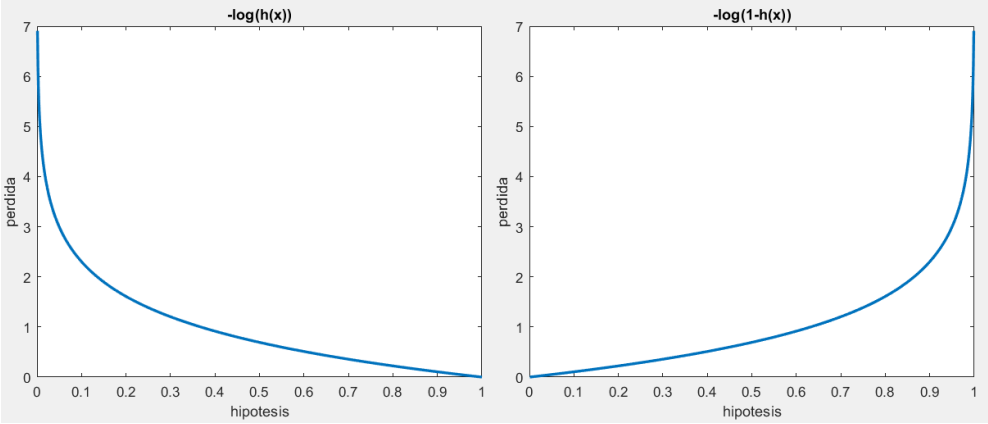


Ilustración 14- Gráfica de la pérdida para ambos casos de la salida y

Fuente de la imagen: Elaboración propia

La intuición que se capta de la ilustración 13, por ejemplo, para la primera gráfica que es para cuándo $y = 1$, es que si el sistema devuelve la hipótesis igual a 0 y el valor real es $y = 1$ es que penalice al sistema con un valor de la función de perdida alto. Ahora lo importante es poder definir una única función para la función de perdida, ya que el algoritmo de descenso por gradiente ocupa la primera derivada de dicha función, (Kleinbaum & Klein, 2002). Para esto se puede utilizar la característica de este sistema del rango binario que tiene el modelo. Por lo que cuando $y = 1$ se busca que se refleje nada mas $-\log\left(h_\theta\left(x^{(i)}\right)\right)$por lo que se le puede multiplicar el mismo valor de y así también cuando este valga 0 la función se cancele, y cuando $y = 0$ se busca reflejar $-\log\left(1 - h_\theta\left(x^{(i)}\right)\right)$ por qué se puede multiplicarle el valor restándole a 1, así cuando y sea 0 la función se multiplique por 1 y cuando valga 1 la resta de 0 y la función se cancele. Aplicando esto entonces se puede obtener la siguiente función de perdida, cabe

destacar que la función es un logaritmo natural, pero computacionalmente como este tipo de logaritmo es más utilizado se generaliza con el termino log.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]$$

Ecuación #28. Función de perdida para regresión logística

3.3.3 Algoritmo de Descenso por Gradiente

Teniendo en cuenta siempre la idea de optimización planteada en la ecuación 5 se busca poder conseguir el set de parámetros que mejor cumpla esa idea. Por lo que para eso se aplicara el algoritmo de descenso por gradiente. El cual en realidad tiene la misma intuición y la misma secuencia que tiene con la regresión lineal. Con la diferencia ahora del factor derivativo de la nueva función de perdida para la regresión logística. Por lo que lo esencial es conseguir ese factor derivando entonces la función de perdida se obtiene la siguiente ecuación, aplicando la ecuación 22 que es la derivada de la función de sigmoide y la derivada para una función logarítmica natural.

$$\frac{\partial}{\partial\theta}J(\theta) =$$

$$\frac{1}{m}\sum_{i=1}^{m}\left[\frac{y^{(i)}}{h_\theta(x^{(i)})}h_\theta\left(x^{(i)}\right)\left(1-h_\theta\left(x^{(i)}\right)\right)\cdot x^{(i)} + \frac{1-y^{(i)}}{1-h_\theta(x^{(i)})}\left(-h_\theta\left(x^{(i)}\right)\left(1-h_\theta\left(x^{(i)}\right)\right)\right)\cdot x^{(i)}\right]$$

Luego de algo de algebra la precedente derivada planteada se puede reducir a la siguiente función.

$$\frac{\partial}{\partial\theta}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[h_\theta\left(x^{(i)}\right) - y^{(i)}]\cdot x^{(i)}$$

Ecuación #29. Primera derivada de la función de perdida para regresión logística

Por lo que el algoritmo de descenso por gradiente para regresión logística queda de la misma manera que se planteó en la ecuación 15 para la regresión lineal, con la diferencia claro que la hipótesis es en realidad una función de sigmoide evaluada en la función dada por la suma de los productos de las variables y sus respectivos parámetros, que vectorizado como se sabe es igual a $\theta^T x$, o en forma de matriz aplicada a todo el set de entrenamiento es igual a $X\theta$.

3.3.4 Clasificación Multiclasista

Qué pasaría si se tiene un problema en el que se tienen varias elecciones, sin embargo, la variancia no es de tipo regresiva señalando que no se puede usar regresión lineal. Para esto se debe plantear una regresión logística que sea multiclasista. Por ejemplo, si se quiere realizar un software que plantee la probabilidad de que un día sea soleado, nublado, que llueva o que nevé. En este caso se tienen cuatro claras elecciones. Principalmente lo que se puede realizar es asignarle una variable de salida a cada una de estas opciones. Y luego aplicar el algoritmo uno-contra-todos, (Bishop, 2006).

El algoritmo de uno-contra-todos es tomar una clase en específico y aplicar una clasificación binaria a esta, ósea medir la posibilidad de que esta ocurra con respecto al resto. Luego aplicar este procedimiento para el resto de las clases.

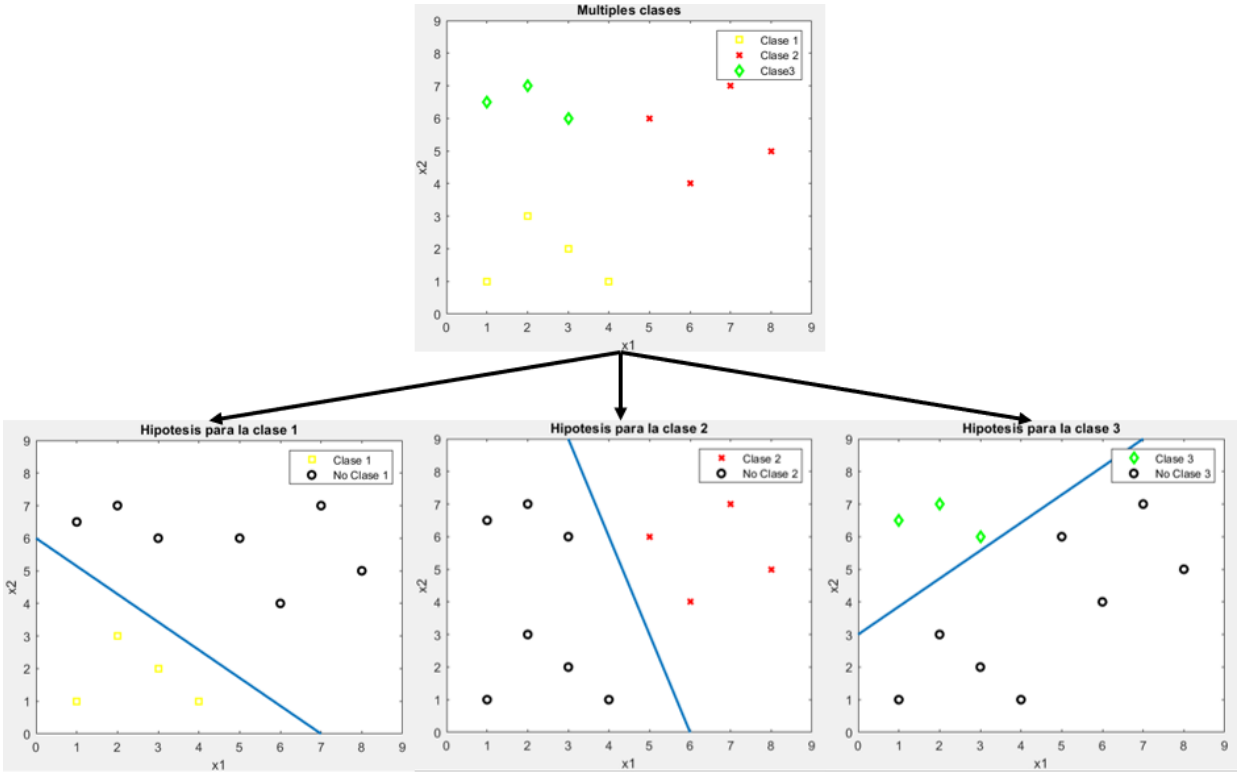


Ilustración 15- Visualización del algoritmo de uno-contra-todos.

Fuente de la imagen: Elaboración propia

Entonces el algoritmo secciona cada clase y luego le aplica individualmente el modelo ya analizado para la regresión logística. Sin embargo, la hipótesis tiene una modificación en cuanto a que está midiendo la probabilidad.

$$h_\theta{}^{(i)}(x) = P(y = i|x;\theta), \qquad para\ toda\ i$$

Ecuación #30. Hipótesis para regresión lineal con clasificación multiclases

La intuición o la manera de plantear como funciona este clasificador es tomar una nueva entrada x cualquiera y clasificarla para la i que devuelva la hipótesis con el valor más grande.

$$\max_i h_\theta{}^{(i)}(x)$$

Ecuación #31. Intuición de regresión logística con múltiples clases

3.4 El Problema del Sobreajuste

En pasadas secciones se ha planteado que para que la hipótesis de algunos de los algoritmos se puede plantear la función entre las características y parámetros de diferentes grados y combinaciones entre las mismas características. Sin embargo, elegir grados muy altos y combinaciones muy complejas puede no ser la mejor opción y esto causa un sobreajuste. Esto ocurre cuando la hipótesis se ajusta perfectamente al set de entrenamiento, pero falla a generalizar nuevas entradas, (Kelleher et al., 2015). A este problema también se le conoce como tener una alta varianza. Pero al tomar en cuenta funciones con grados muy bajos se produce que un error relativamente alto en ambos el set de entrenamiento y nuevas entradas, a esto se conoce como subajuste. Ha este problema se le conoce también como tener una alta parcialidad, (Delgado-Rodríguez & Llorca, 2020).

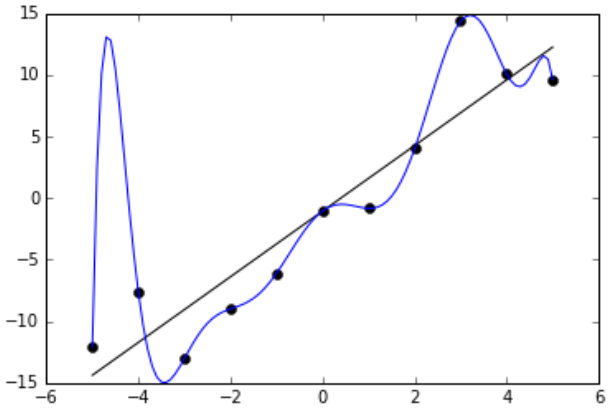


Ilustración 16- Visualización de sobreajuste y subajuste en un modelo de regresión lineal

Fuente de la imagen: (Wikipedia, 2020)

En la ilustración 15 se puede observar como la función en color azul en realidad pasa por todos los puntos de lo que sería el set de entrenamiento. Por lo que entonces el error medido por la función de perdida seria 0. Esto daría entender que se consiguió la fórmula perfecta. Pero en realidad observando tal gráfica se puede observar que no es así. Esto debido a que la función tiene curvas que al medir valores distintos a los brindados por el set de entrenamiento no da los valores reales para tal punto y pues fallo al medir valores reales. Sin embargo, la línea negra, que incluso parece tener una mejor aproximación al problema que la gráfica en azul, no logra capturar varios valores y tiene bastante distancia entre los valores lo que sugiere que los valores a devolver no van a hacer muy precisos y además la función de perdida evaluada para el set de entrenamiento también devolvería un valor algo alto sugiriendo un error algo alto.

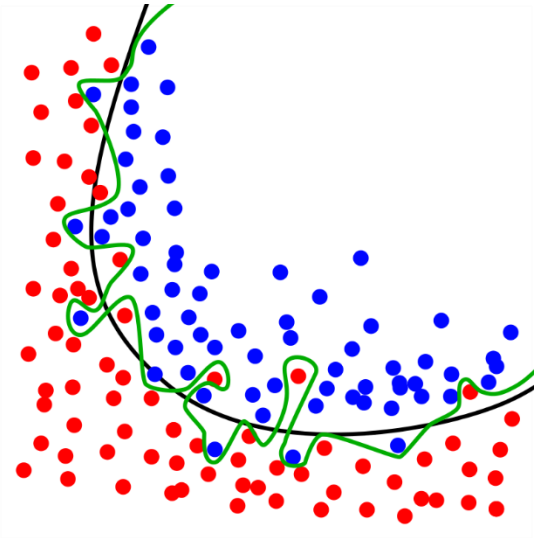


Ilustración 17- Visualización de sobreajuste y subajuste en un modelo de regresión logística

Fuente de la imagen: (Wikipedia, 2020)

En esta ilustración 16 se presenta un caso de sobreajuste en verde debido a una función $z(x)$ de grado muy alto y en negro se tiene una función regularizada, termino del cual ya se estará tocando a fondo.

Una forma de poder atacar el problema de sobreajuste es reducir la cantidad de características del sistema. Esto significa manualmente seleccionar que variables se deben de quedar y cual no, o aplicar algún algoritmo de selección por modelo. La otra forma de atacar este problema es por medio de la regularización. Este procedimiento ayuda a que el sistema conserve todas sus características. Lo que este procedimiento busca es reducir la magnitud de los parámetros θ_j. Esto funciona muy bien con una n de cualquier tamaño, permitiendo así a cada característica poder contribuir un poco más en la predicción de los valores de salida del sistema como tal, (NG, 2011).

3.4.1 Regularizando la Función de Perdida

La regularización busca es penalizar los valores dados por los parámetros. Esto lo busca hacer por medio de la función de perdida la cual es la que se busca minimizar lo más que se pueda. Y si se penalizan los valores dados para los parámetros, es decir en la función luego de la sumatoria de los errores, sumar el valores de los parámetros a penalizar por algún valor alto, (Bühlmann & Geer, 2011). Entonces lo que busca la regularización es reducir el valor de los parámetros de la función de manera de conseguir una hipótesis más sencilla y con menos tendencia a tener un sobreajuste. Por lo que la nueva función de perdida regularizada buscara reducir el valor de los parámetros poniendo estos al cuadrado y multiplicarlos por una constante regularizadora.

$$\frac{\lambda}{2m}\sum_{j=1}^{n}{\theta_j}^2$$

Ecuación #32. Termino regularizador

No se busca regularizar el valor de θ_0, sin embargo, regularizarlo o penalizarlo no hace gran diferencia, (NG, 2011). Pero ahora, ¿qué pasaría si el valor de λ es bien alto, como por decir 10^{10}? Aplicando entonces el concepto de optimización, y aplicando algún algoritmo de este tipo, lo que el algoritmo como tal va a devolver para el vector de parámetros es lo siguiente.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots, dado\ que\ \theta_1, \theta_2, \theta_3, \ldots \approx 0 \rightarrow h_\theta(x) \approx \theta_0$$

Dado de tener esta situación se sabe que una función igual a una constante es una línea vertical que está a la magnitud de tal constante del eje x. Esto claramente es un problema de subajuste por lo que se debe de tener cuidado con el valor que se elige para λ, porque también elegir un valor pequeño pues no causaría una regularización notable por lo que igual se mantiene el problema de sobreajuste.

3.4.2 Algoritmo de Descenso por Gradiente Regularizado

Este algoritmo como ya se sabe utilizada la primera derivada de la función de perdida que debido a la regularización se ve modificada por el termino planteado en la ecuación 32. También es importante tomar en cuenta que esta función no es la misma para los problemas regresivos y clasificatorios por lo que es importante realizar el análisis de ambos casos. Previo a dicho análisis es bueno saber entonces cual es la función de la primera derivada de este término regularizador.

$$\frac{\partial}{\partial\theta}\left(\frac{\lambda}{2m}\sum_{j=1}^{n}{\theta_j}^2\right)=\frac{\lambda}{m}\theta_j$$

Ecuación #33. Primera derivada del término regularizador

3.4.2.1 Regresión Lineal

Aplicando el termino planteado en la ecuación 32 y tomando la función de perdida planteada en la ecuación 5, para cualquier tamaño de θ, se obtiene entonces la siguiente función.

$$J(\theta)=\frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2+\lambda\sum_{j=1}^{n}{\theta_j}^2\right]$$

Ecuación #34. Función de perdida para regresión lineal regularizada

Ahora sabiendo entonces el cambio que tiene la función de perdida, se debe conocer el termino derivativo de esta que afecta directamente al algoritmo de descenso por gradiente. De manera entonces que la primera derivada de la ecuación 34 es una fusión de la ecuación 14 y la ecuación 33.

$$\frac{\partial}{\partial\theta}J(\theta) = \frac{1}{m}\left[\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) + \lambda\theta_j\right]$$

Ecuación #35. Primera derivada de la función de perdida para regresión lineal regularizada

Aplicando entonces este término al algoritmo de descenso por gradiente para la regresión lineal se obtiene lo siguiente, importante antes de plantear el algoritmo es saber que no se regulariza el primer parámetro por lo que no se debe aplicar la ecuación 35 en esta instancia.

$$repetir\ hasta\ convergencia\{$$

$$\theta_0 \coloneqq \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right),$$

$$\theta_j \coloneqq \theta_j\left(1 - \frac{\alpha\lambda}{m}\right) - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right), \qquad para\ j \neq 0\}$$

Ecuación #36. Algoritmo de descenso por gradiente regularizado

El término que está multiplicando al θ_j a lado izquierdo de la actualización es un término que en teoría debe ser menor que uno por lo que entonces el producto de α y λ dividido por m debe ser un término mayor a cero.

3.4.2.2 Regresión Logística

Al igual que con la regresión lineal se debe analizar el efecto que este término regularizador tiene sobre la función de perdida aplicada a la regresión logística. Por lo que aplicando el termino planteado en la ecuación 32 y la función de perdida de la ecuación 28 se obtiene la siguiente función.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}{\theta_j}^2$$

Ecuación #37. Función de perdida para la regresión logística regularizada

Teniendo ya planteada la función de perdida regularizada para este modelo se debe conseguir la primera derivada, debido al termino derivativo necesario de esta función para

aplicar el algoritmo de descenso por gradiente. Para esto se pueden fusionar las ecuaciones 29 y 33.

$$\frac{\partial}{\partial\theta}J(\theta) = \frac{1}{m}\left[\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) + \lambda\theta_j\right]$$

Ecuación #38. Primera derivada de la función de perdida para la regresión logística regularizada.

Viendo entonces que el termino derivativo de la función de perdida para la regresión logística, planteado en la ecuación 38, es igual al de la regresión lineal, planteado en la ecuación 35. Ayudando esto a deducir que el algoritmo de descenso por gradiente se aplica igual para ambos casos, regresión lineal y regresión logística.

3.4.3 Ecuación Normal Regularizada

Como se vio en secciones anteriores una forma de conseguir lo parámetros en un paso para la regresión lineal es a través de la ecuación normal. Que es una aplicación vectorizada de los términos. Por lo que al término X^TX es al termino que se le va a agregar la regularización, ya que este es el que maneja la dimensionalidad de los parámetros, que es una matriz cuadrada con dimensiones $n+1$. En este caso lo que se debe realizar es tomar el termino regularizador λ, multiplicarlo por una matriz, matriz cuya tiene la misma dimensionalidad que el término X^TX, se puede decir, identidad con el primer termino de todos siendo cero, esto debido a que no se quiere regularizar el parámetro θ_0, y luego este producto sumárselo al termino X^TX, (Byliskii, 2015).

$$\theta = \left(X^TX + \lambda\begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & \ddots & \vdots\\ 0 & 0 & \dots & 1\end{bmatrix}\right)^{-1} * X^T * y$$

Ecuación #39. Ecuación normal regularizada

Lo asombroso de esta regularización es que soluciona el problema de que el término a invertirse sea singular, (NG, 2011). Sin embargo, al agregar una operación más para valores de n

torna ser una operación computacionalmente costosa. Por lo que para estos casos sigue siendo mejor usar el algoritmo de descenso por gradiente o algún otro algoritmo de optimización.

3.5 REDES NEURONALES

Qué pasaría si se tiene una clasificación que es no lineal, como lo son la mayoría de los casos de la vida real, y se tienen cien características de entrada, y se le asigna un tiempo de complejidad de $O(n^2)$ entonces se obtienen aproximadamente cinco mil características. Y si en busca de precisión se aumenta el grado de complejidad a $O(n^3)$ entonces se obtienen ciento setenta mil características.

Si se quiere saber si en una imagen, por ejemplo, sale un perro. Se podría tomar dos pixeles en cualquier punto de varias imágenes y clasificar, según esos dos pixeles, si la imagen tiene o no un perro. Ahora, que, si se busca analizar cada pixel de la imagen, se puede plantear que la imagen es de 50x50 pixeles. Entonces se tienen dos mil quinientos diferentes valores de intensidad de pixeles, siendo ese el número de características del sistema. Seguramente el set de entrenamiento seria no lineal, por lo que el tiempo de complejidad es polinomial, el cual en el mejor de los casos seria $O(n^2)$, por lo que la cantidad de características son un poco mas de tres millones. Eso definitivamente es computacionalmente costoso.

Por lo que se puede plantear una pregunta, ¿cómo hace el cerebro humano? Se sabe que el cerebro humano puede trabajar a velocidades muy altas, puede generar nuevas neuronas y crear nuevas conexiones hasta el día en que la persona muere, (Caine & Caine, 2011). Por ejemplo, la corteza auditiva del cerebro humano, siguiendo lecciones especificas mientras se desarrolla puede aceptar y procesar información visual que entra por medio de la retina. Esta transformación visual que procesa la corteza auditiva es muy similar a la que procesa la corteza visual, (Roe et al., 1992). La corteza somatosensorial y la corteza visual trabajan de manera similar, y esta al igual que la corteza auditiva, puede aprender a saber que está tocando, por lo que se puede ver con el tacto, (Métin & Frost, 1989). Todo esto el cerebro lo logra hacer por un sistema de neuronas, por lo que poder entenderlas y poder simular sus procedimientos puede llegar a ser una herramienta poderosa.

3.5.1 Neuronas en el Cerebro

El cerebro humano tiene un estimado de entre diez a veinte miles de millones de neuronas en la corteza cerebral y entre cincuenta y cinco a setenta miles de millones de neuronas en el cerebelo, (von Bartheld et al., 2016). Las neuronas consisten de tres partes el axón, las dendritas y el cuerpo celular, las partes esenciales a analizar son el axón y las dendritas. Un impulso nervioso es cuando una neurona empieza a secretar un químico entre la sinapsis que se forma entre su axón y las dendritas de otra neurona, (Stone, 2004). Pudiendo entonces plantear las dendritas como cables de entrada y el axón como cables de salida, y estos se comunican por ese impulso, que podría ser un impulso eléctrico también.

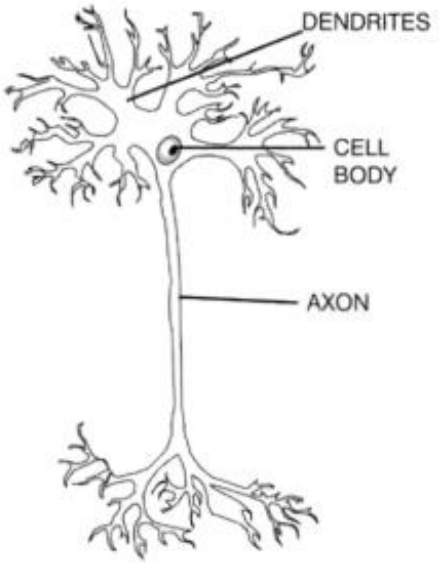


Ilustración 18- Neurona.

Fuente de la imagen: (Stone, 2004)

3.5.2 Modelo Neuronal para unidades Logísticas

El modelado neuronal es sencillo de entender al saber el simple funcionamiento de las neuronas. Se ocupa un set de cables de entradas y de salidas. Estos van a ser procesados por alguna función activadora que va a estar parametrizada al igual que las funciones de regresión logística y lineal. Para poder medir la hipótesis devuelta por dicha red se ocuparía la función de sigmoide por el hecho de estar trabajando en sistema logístico.

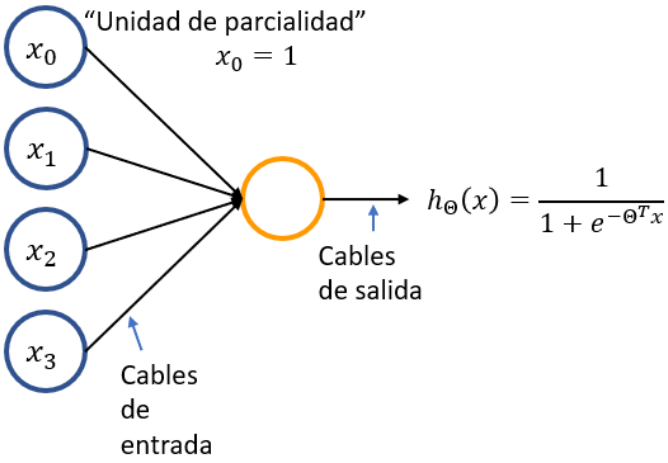


Ilustración 19- Modelo neuronal

Fuente de la imagen: (NG, 2011)

Teniendo siempre el vector de características de entrada, que al meter el set de ejemplos este se manipula como una matriz, y los parámetros, estos se cambia la forma de señalizarlos, ahora para redes neuronales los parámetros se señalizan con una theta mayúscula. La cantidad de características, como siempre, dependen del sistema como tal. Sin embargo, se pueden manipular diferentes arquitecturas, (Du & Swamy, 2013) Estas dependen de la cantidad de capas que se le deseen agregar a la red neuronal, ya se tocará más a fondo ese tema. Para poder comprender la definición matemática se va a plantear una arquitectura sencilla, con una capa de entradas, una capa oculta, y una capa de salidas, que en este caso va a ser un sistema de una sola clase.

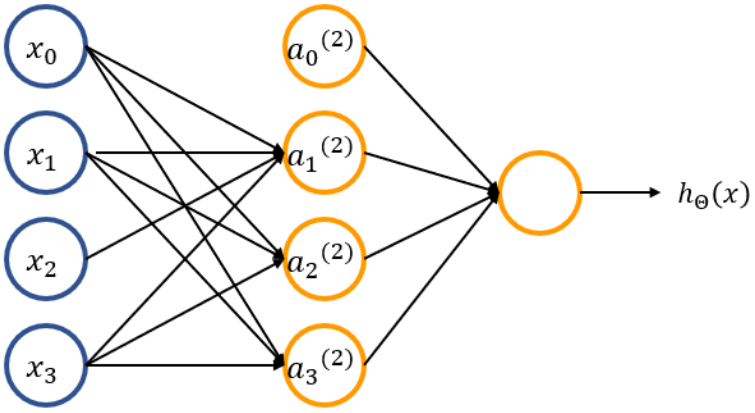


Ilustración 20- Esquema de una red neuronal.

Fuente de la imagen: Elaboración propia

En esta red neuronal se tiene una nueva variable que es ${a_j}^{(l)}$, que es la función de activación de la unidad número j en la capa número l. Y dado que va a haber un parámetro en específico para cada característica en cada unidad de activación, estos parámetros ahora son una matriz, y se va a tener una matriz por cada capa. Por lo que cada matriz se distingue por la variable ${\Theta_{ji}}^{(l)}$, que es el parámetro de la unidad número j sobre la entrada número i de la capa número l. También se puede plantear nada más como $\Theta^{(l)}$, que es la matriz de parámetros que mapea la capa l a la capa $l+1$, (Aggarwal, 2018). La dimensionalidad de esta matriz de parámetros se define según si la red neuronal tiene s_l unidades en la capa l, y tiene s_{l+1} unidades en la capa $l+1$, entonces, $\Theta^{(l)}$ tendrá dimensionalidad de $s_{l+1} \times (s_l + 1)$. Por ejemplo, la dimensionalidad de la matriz de parámetro que mapea la primera capa de la red en la ilustración 19 a la segunda seria de 4×3.

3.5.3 Propagación hacia Adelante

La propagación hacia adelante es el algoritmo que se utiliza para conocer las funciones de cada capa. Se dice que es hacia adelante debido a que para conocer las funciones de la capa l se necesita saber las funciones de la capa $l-1$, (NG, 2011). También se debe mantener en cuenta la ecuación 21 de la función de sigmoide. La primera capa, la de entradas se puede ver como una capa de unidades activadores. Donde cada nivel le corresponde a una variable de entrada.

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, \qquad a^{(1)} = \begin{bmatrix} {a_0}^{(1)} \\ {a_1}^{(1)} \\ \vdots \\ {a_{s_1}}^{(1)} \end{bmatrix}, \qquad a^{(1)} = x, \qquad s_1 = n$$

Ecuación #40. Funciones de activación de la capa de entradas

Por lo que entonces $a^{(1)}$ se puede visualizar como un vector de dimensionalidad n donde cada elemento es igual a cada característica del vector de entradas. Para obtener las funciones de activaciones de las siguientes capas se aplicarán todos los conceptos planteados anteriormente. Cabe destacar que se tomará como referencia la red planteada en la ilustración 19, donde entonces solo se debe plantear la única capa oculta que hay, que es la segunda capa.

En el caso de las capas ocultas, estas también tienen una función parcial las cuales son iguales a 1 siempre.

$$a_0{}^{(2)} = 1$$

$$a_1{}^{(2)} = g\left(z_1{}^{(2)}\right) \rightarrow z_1{}^{(2)} = \Theta_{10}{}^{(1)} x_0 + \Theta_{11}{}^{(1)} x_1 + \cdots + \Theta_{1s_1}{}^{(1)} x_{s_1}$$

$$a_2{}^{(2)} = g\left(z_2{}^{(2)}\right) \rightarrow z_2{}^{(2)} = \Theta_{20}{}^{(1)} x_0 + \Theta_{21}{}^{(1)} x_1 + \cdots + \Theta_{2s_1}{}^{(1)} x_{s_1}$$

$$\vdots$$

$$a_{s_2}{}^{(2)} = g\left(z_{s_2}{}^{(2)}\right) \rightarrow z_{s_2}{}^{(2)} = \Theta_{s_2 0}{}^{(1)} x_0 + \Theta_{s_2 1}{}^{(1)} x_1 + \cdots + \Theta_{s_2 s_1}{}^{(1)} x_{s_1}$$

Ecuación #41. Funciones de activación de la primera capa oculta

Por lo que entonces $a^{(2)}$ se puede definir como un vector de dimensión arbitraria que depende de la función sigmoide definida en el vector $z^{(2)}$, el cual es un vector con la misma dimensión que $a^{(2)}$. Definidas las funciones de activación de la capa oculta, que en el caso de la ilustración 19 es la única, se puede obtener las funciones de activación de la última capa que al ser de clase singular solo es una sola función de activación. Esta función de activación es igual la hipótesis de todo el sistema.

$$h_\Theta(x) = a_1{}^{(3)} = g\left(z_1{}^{(3)}\right) \rightarrow z_1{}^{(2)} = \Theta_{10}{}^{(2)} a_0{}^{(2)} + \Theta_{11}{}^{(2)} a_1{}^{(2)} + \cdots + \Theta_{1s_2}{}^{(2)} a_{s_2}{}^{(2)}$$

Ecuación #42. Hipótesis de una red neuronal de una sola clase

Al igual que en regresión logística esta función devuelve la probabilidad de que la salida y se a igual 1 debido a las entradas x parametrizadas, como la que afecta directamente a la hipótesis es el último set de parámetros, por $\Theta^{(2)}$. Por lo que, aplicando las funciones vectorizadas de cada una de las capas de la red neuronal, se puede plantear el algoritmo de propagación hacia delante vectorizado, lo cual le brinda una mejoría computacional al algoritmo.

$$a^{(1)} = x \rightarrow z^{(2)} = \Theta^{(1)} a^{(1)} \rightarrow a^{(2)} = g\left(z^{(2)}\right) \rightarrow a_0{}^{(2)} = 1 \rightarrow z^{(3)} = \Theta^{(2)} a^{(2)} \rightarrow a^{(3)} = g\left(z^{(3)}\right)$$
$$\rightarrow h_\Theta(x) = a^{(3)}$$

Ecuación #43. Algoritmo de propagación hacia adelante vectorizado

3.5.4 Clasificación Multiclasista

Muchos de los casos, tales como el presente proyecto, requiere clasificar la salida y en varias distintas clases. Por ejemplo, si se quiere saber si en una fotografía hay un perro, un gato, un ratón o un caballo. Se debe de asignar una vector de 0s y 1s distinto a cada una de estas clases.

$$y^{(perro)} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \qquad y^{(gato)} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \qquad y^{(ratón)} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \qquad y^{(caballo)} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

Por lo que entonces $y^{(i)}$ se vuelve un vector de dimensionalidad igual a 4 y dependiendo que valor en específico es 1 clasifica al sistema a esa clase. Ahora la hipótesis medida en un ejemplo especifico también es vuelve en vector de la misma dimensionalidad que la salida. En este caso la hipótesis va a devolver un valor entre 0 y 1 en los cuatro casos. Y debido a un valor limite estos valores probabilísticos luego se mandan a ser 0 o 1. Es la misma idea que regresión logística. Ahora la última capa va a tener un cambio ya que ahora se requiere obtener más valores debido a que van a haber más hipótesis. Es importante recordar que la cantidad de clases se definen con la variable K.

$$\begin{aligned}
h_\Theta{}^{(1)}(x) = a_1{}^{(L)} &= g\left(z_1{}^{(L)}\right) \rightarrow z_1{}^{(L)} \\
&= \Theta_{10}{}^{(L-1)} a_0{}^{(L-1)} + \Theta_{11}{}^{(L-1)} a_1{}^{(L-1)} + \cdots + \Theta_{1 s_{L-1}}{}^{(L-1)} a_{s_{L-1}}{}^{(L-1)}
\end{aligned}$$

$$\begin{aligned}
h_\Theta{}^{(2)}(x) = a_2{}^{(L)} &= g\left(z_2{}^{(L)}\right) \rightarrow z_2{}^{(L)} \\
&= \Theta_{20}{}^{(L-1)} a_0{}^{(L-1)} + \Theta_{21}{}^{(L-1)} a_1{}^{(L-1)} + \cdots + \Theta_{2 s_{L-1}}{}^{(L-1)} a_{s_{L-1}}{}^{(L-1)}
\end{aligned}$$

$$\vdots$$

$$\begin{aligned}
h_\Theta{}^{(k)}(x) = a_{s_L}{}^{(L)} &= g\left(z_{s_L}{}^{(L)}\right) \rightarrow z_{s_L}{}^{(L)} \\
&= \Theta_{s_L 0}{}^{(L-1)} a_0{}^{(L-1)} + \Theta_{s_L 1}{}^{(L-1)} a_1{}^{(L-1)} + \cdots + \Theta_{s_L s_{L-1}}{}^{(L-1)} a_{s_{L-1}}{}^{(L-1)}
\end{aligned}$$

Ecuación #44. Hipótesis de una red neuronal de clasificación multiclasista

La ecuación 44 se planteó de una manera general, de manera de observar a la capa de salida como es la última. Entonces se plantea L como es el numero de capas que hay, s_L es la

cantidad de unidades que tiene esta última capa que es igual a K, y s_{L-1} que es igual al número de unidades que tiene la capa anterior a esta, que es la última capa de las capas ocultas.

3.5.5 Función de Perdida

Debido al modelo logístico que tienen las redes neuronales la función de perdida que mejor se adapta es la que se utiliza para la regresión logística en la ecuación 32, claro con sus modificaciones. Para las redes neuronales hay que tomar en cuenta la cantidad de clases, K, al igual que en regresión logística, la cantidad de capas que tiene la arquitectura de la red, L, la cantidad de unidades que tiene la capa sobre la cual se está pasando el parámetro l, s_l, y la cantidad de unidades que tiene la siguiente capa a esa, s_{l+1}. Ósea que en la sumatoria estos últimos dos valores van a depender del número de capa por el que se esté pasando.

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=0}^{K} {y_k}^{(i)} \log\left(h_\Theta\left(x^{(i)}\right)_k\right) + (1 - {y_k}^{(i)}) \log\left(1 - h_\Theta\left(x^{(i)}\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}} {\Theta_{ji}}^{(l)}$$

Ecuación #45. Función de perdida para redes neuronales regularizada

Claro siempre se busca los mejores sets de parámetros para que la función de perdida sea un valor lo mínimo posible. Por lo que se debe de computar la función de perdida y la función derivativa de esta. Sin embargo, estos extra parámetros hace que se deba de conseguir un parámetro por matriz de parámetro, ósea por cada capa. Para eso se debe aplicar un nuevo algoritmo.

3.5.6 Propagación hacia Atrás

Para poder obtener el termino derivativo de la función de perdida se debe conseguir el "error" en el nodo j de la capa l. Primero se puede conseguir el error que tienen los nodos de la última capa, este error se puede definir como la diferencia que hay entre los valores de $a^{(L)}$ y las salidas obtenidas a través del set de entrenamiento. Y a partir de este valor obtener los errores de las capaz hasta llegar a la capa 2. No se llega hasta la capa 1 porque esa es la capa de entradas y, en teoría, ahí no hay errores, (Narendra & Parthasarathy, 1990). Y debido a que para conseguir el error de una de las capas se tiene que saber el error de la siguiente es por el

motivo que se dice que es propagación hacia atrás. Se empieza en la última capa y se termina en la segunda capa, se va de enfrente hacia atrás.

$$\delta_j{}^{(L)} = a_j{}^{(L)} - y_j, \qquad \delta^{(L)} = a^{(L)} - y$$

Ecuación #46. Error de la última capa de una red neuronal para cada nodo de la capa y vectorizado.

A partir entonces de este error se debe de conseguir el resto de valores. Para poder conseguir el siguiente error, y esto se aplicaría en todas las capas a partir de saber todos los errores de la capa que le sigue, se debe de transformar el error obtenido en la fórmula 46 a la dimensionalidad de $a^{(L-1)}$. Eso se puede obtener multiplicando la matriz de parámetros de esa capa por el error $\delta^{(L)}$, y cada uno de esos elementos obtenidos, que son parte de un vector de dimensión $s_{L-1} + 1$, se van a multiplicar por la derivada de la función de activación, $a^{(L-1)}$, de cada respectivo nodo. Esa derivada de la función de activación también se puede ver como la derivada de la función de sigmoide definida en la función $z^{(L-1)}$. De manera de generalizar se va a definir el error para la capa l.

$$\delta^{(l)} = \left(\Theta^{(l)}\right)^T \delta^{(l)} .* g'\left(z^{(l)}\right), \qquad para\ l = 2, \dots, L-1$$

Ecuación #47. Error para el resto de capas de una red neuronal vectorizado

Entonces dado un set de entrenamiento con m cantidad de ejemplos, y dado un set de parámetros $\Delta_{ij}{}^{(l)} = 0, para\ toda\ i, j, l$. Dado esto entonces se puede plantear de la siguiente manera el algoritmo de propagación hacia atrás.

$$Para\ i = 1\ hasta\ m\{$$

$$Propagación\ hacia\ adelante\ y\ calcular\ a^{(l)}, \qquad para\ l = 2, \dots, L$$

$$Utilizar\ y^{(i)}\ para\ calcular\ \delta^{(L)} = a^{(L)} - y^{(i)}$$

$$Calcular, en\ base\ a\ \delta^{(L)}, \delta^{(l)}, \qquad para\ l = L-1, \dots, 2\}$$

$$\Delta_{ij}{}^{(l)} \coloneqq \Delta_{ij}{}^{(l)} + \mathrm{a}_j{}^{(l)} \delta_i{}^{(l+1)} \rightarrow \Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} \left(a^{(l)}\right)^T, \qquad para\ toda\ j, l$$

$$D_{ij}{}^{(l)} = \frac{1}{m} \Delta_{ij}{}^{(l)}, \qquad para\ j \neq 0\ y\ toda\ l$$

$$D_{ij}{}^{(l)} = \frac{1}{m} \Delta_{ij}{}^{(l)} + \lambda \Theta_{ji}{}^{(l)}, \qquad para\ j = 0\ y\ toda\ l$$

Ecuación #48. Algoritmo de propagación hacia atrás.

Se define cada elemento en las diferentes matrices que devuelve el algoritmo de propagación hacia atrás como la derivada con respecto a los parámetros de la función de perdida para redes neuronales.

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}, \qquad para\ toda\ i, j, l$$

Ecuación #49. Definición de las derivadas parciales de la función de perdida para redes neuronales.

Estos valores se pueden entonces computarizar junto con la función de perdida por cualquier algoritmo de optimización, como por ejemplo el algoritmo de descenso por gradiente que se ha analizado en la presente tesis. Cabe destacar también la inicialización de todos los parámetros de las diferentes matrices de Θ. Si estos valores se inician todos iguales, se podría decir inicializar todos en 0, estos van a permanecer igual luego de aplicar los respectivos algoritmos, (NG, 2011). Por lo que es importante inicializar estos valores como valores aleatorios entre un rango de $[-\epsilon, \epsilon]$ donde ϵ es un numero pequeño, podría ser por ejemplo 10^{-4}, valor que funciona muy bien, (Aggarwal, 2018).

3.6 Set de Entrenamiento/ Validación/ Prueba

En el presente marco teórico se ha analizado el problema de parcialidad y de variancia, y se sabe que estos dos son grandes problemas para un algoritmo de aprendizaje automático. Por lo que es importante ir analizando el algoritmo obtenido luego del entrenamiento. Para eso se secciona todo el set de datos conseguidos para entrenar el sistema en tres partes. El set de entrenamiento, que se utiliza pues como su nombre indica para entrenar la red, el set de validación que es como un filtro, y el set de prueba que es la prueba final para dar un reporte generalizado del sistema elegido debido a lo devuelto por el set de validación, (Hosmer et al., 2013). La idea inicial era trabajar nada más con el set de entrenamiento y de prueba. Sin embargo, al utilizar este sistema se puede dar el caso en el que el error en el set de prueba sea optimista ante un error en el set de entrenamiento bastante alto. Por lo que la solución es agregar un tipo de filtro en medio de estos dos sets. Que es el set de validación.

Todos estos sets tienen que ser datos fidedignos de manera de obtener valores basados en la vida real. Normalmente se busca que el set de entrenamiento sea el más grande de estos tres sets. Debido a que para obtener el mejor algoritmo posible no se debe tanto al mejor código sino a la mejor cantidad de información con la que se entrene el sistema. Y el set de validación y de prueba es recomendable que tengan la misma cantidad, (NG, 2011).

Tabla 4. Asignación porcentual de todo el set de data a los sets de entrenamiento/ validación/ prueba

	Data set:		
	Tamaño	Precio	
60%	2032	396	Set de entrenamiento
	1623	335	
	2421	375	
	1400	227	
	3003	541	
	1963	482	
20%	1602	323	Set de Validación
	1415	186	
20%	1395	235	Set de Prueba
	1486	240	

Fuente de la tabla: Elaboración propia

Como se puede apreciar en la tabla 4, porcentualmente hablando, una buena distribución dejar el 60% de toda la data para el set de entrenamiento, a esta cantidad de información se representa con la variable m, un 20% para el set de validación, esta cantidad de información se representa con la variable m_{cv}, y el restante 20% para el set de prueba, esta cantidad de información se representa con la variable m_{test}. Cabe destacar que la división de información se debe de dar aleatoriamente, así se evita algún tipo de redundancia debido a la forma en la que se pudo haber conseguido la información. Esto normalmente no debería de ser problema pero es una buena práctica dividir la información aleatoriamente, (Mohri et al., 2012). Los errores de estos sets se medirán con el error del tipo de algoritmo que se esté utilizando sobre los ejemplos que están dentro de la respectiva división.

3.7 Precisión, Recordación y Puntaje F1

Cuando se entrena un algoritmo de aprendizaje automático de tipo clasificatorio y, por el momento, de una sola clase pues solo se pueden dar dos casos, que la salida sea 1 o 0. Se entrenó entonces un algoritmo de este tipo y se obtiene un error de 1% en algún set de pruebas, ósea que se acertó el 99% de los casos y eso suena a un algoritmo de aprendizaje automático bastante preciso. Pero que tal que este es un sistema para predecir si un paciente tiene alguna enfermedad, y de todos los pacientes que se sometieron al software en realidad solo 0.5% estaban enfermos. Entonces ya no suena tan fabuloso el sistema. Ya que las pocas veces que le tocaba predecir un 1 devolvía un 0 y más de una vez que le tocaba devolver 0 devolvió 1. A este tipo de clase se le conoce como clases sesgadas. Para este tipo de clases donde el 1 es tan poco común se puede generar un código donde la función del algoritmo devuelva un $y = 0$ y en caso que solo el 0.5% de los pacientes estén enfermos ese va a ser el error, (Ricamato et al., 2008). Sin embargo, generar una función que devuelve nada más ceros no es un algoritmo de aprendizaje. Por lo que se debe aplicar otras técnicas para poder medir la eficacia del sistema.

En un problema de decisión binaria, un clasificador etiqueta los ejemplos como positivo o negativo. La decisión realizada por el clasificador puede ser representada en una matriz de confusión o tabla de contingencia, (Davis & Goadrich, 2006). Esta tabla de contingencia tiene cuatro categorías que se dividen en base a la clase predicha y la clase de la vida real. Las categorías son: verdadero positivo, false negativo, false positivo y verdadero negativo.

Tabla 5. Tabla de contingencia

	Positivo Real	Negativo Real
Positivo Predicho	VP	FP
Negativo Predicho	FN	VN

Fuente de la imagen: (Davis & Goadrich, 2006)

La recordación o sensibilidad es la proporción entre los verdaderos positivos, que son los positivos que se acertaron, y los positivos reales, que son todos los valores que si son positivos. Es una característica útil porque refleja los valores relevantes para los verdaderos positivos. La precisión o confianza denota la proporción entre los valores predichos a ser positivos, que son

todos los 1 que devolvió el sistema, con respecto a los verdaderos positivos. Esta es una medición de exactitud para saber qué porcentaje de lo predicho es correcto, (Powers, 2011).

$$Recordación = \frac{VP}{\#positivos\ predichos} = \frac{VP}{VP + FP}, \qquad Precisión = \frac{VP}{\#positivos\ reales} = \frac{VP}{VP + FN}$$

Ecuación #50. Recordación y precisión

La ventaja que brindan estas dos nuevas variables sobre las clases sesgadas es de que su medida se basa en la cantidad de verdaderos positivos, valor que es igual a 0 para algún sistema que devuelva nada más 0 como predicción, valga la redundancia. Por lo que si la cantidad de verdaderos positivos es 0 tanto la recordación como la precisión van a ser igual a 0.

Por ejemplo, se tiene un sistema de regresión logística i se plantea un valor límite de 0.5, llegado o pasado este número se predice un 1 cualquier caso contrario se predice un 0. Este modelo con ese valor limite tiene entonces un valor de recordación y de precisión. Entonces que pasa si el sistema solo quiere predecir que un paciente cáncer si el sistema está totalmente seguro. Entonces el valor limite se sube a 0.9, y se plantea que estas predicciones tienden a ser una clase sesgada, pero no lo son. Entonces al subir el valor la precisión aumenta, pero la recordación disminuye, esto debido a que el valor de falsos positivos disminuye, pero el valor de falsos negativos aumenta. Pero que tal que ahora se desea hacer un sistema que evite pasar por alto algún caso que tuvo que ser predicho con cáncer, entonces se disminuye el valor limite a 0.3. En este caso la recordación aumenta, pero la precisión disminuye, esto se da porque ahora pasa el caso contrario que en el otro sistema.

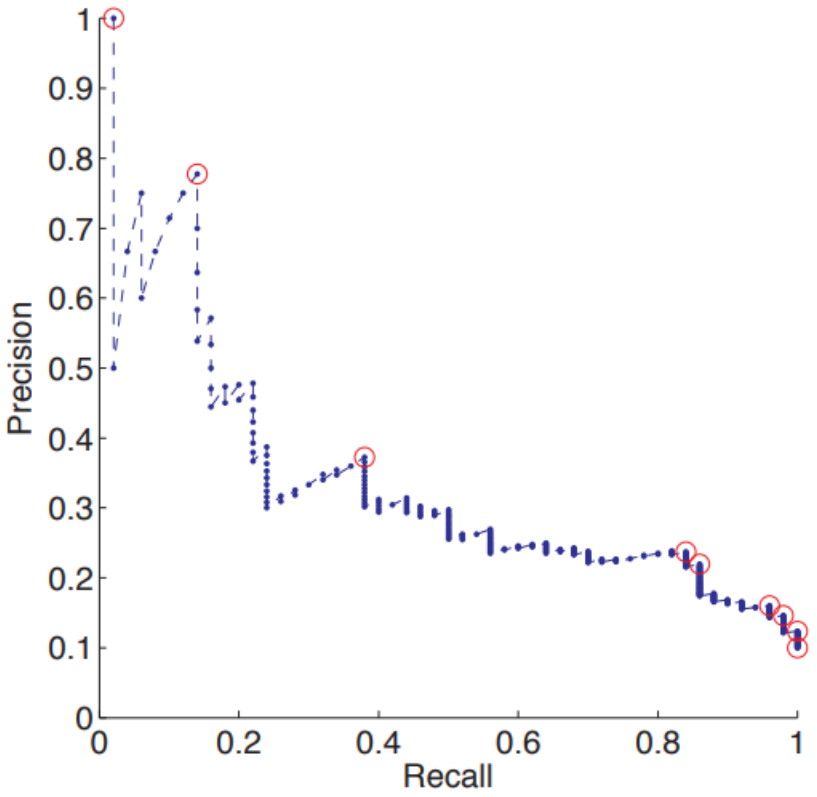


Ilustración 21- Grafica de recordación vs. Precisión

Fuente de la imagen: (Flach & Kull, 2015)

En la ilustración 20 se aprecia una gráfica, debido a un sistema en específico, de cómo según hay una mayor magnitud en una de las características se obtiene una menor magnitud en la otra. Es una tendencia que tiene cualquier sistema pero con diferentes curvas, (Flach & Kull, 2015). Pero, ¿cómo comparar estos dos valores? Como saber, en base a estas características, que sistema o que algoritmo es mejor.

Tabla 6. Comparación de algoritmos en base a su precisión y recordación

	Precisión (P)	Recordación (R)
Algoritmo 1	0.46	0.42
Algoritmo 2	0.76	0.05
Algoritmo 3	0.05	1

Fuente de la tabla: Elaboración propia

Sin embargo, como saber en base a la tabla 6 que algoritmo es el mejor, probablemente un experto en análisis de data sabría por experiencia. La forma más eficiente de poder comparar

estos valores es a través del puntaje F1. Este es una simple formula que devuelve un valor entre 0 y 1 en base a los valores obtenidos para la precisión y recordación, (Beitzel, 2006).

$$F_1 = \frac{2PR}{P + R}$$

Ecuación #51. Puntaje F1

La idea de optimización de esta ecuación 51 es aumentar el valor del puntaje F1 en base a la recordación y precisión obtenida.

3.8 Modelo en Espiral

Parte muy importante de una investigación científica es la metodología a utilizar para atacar las preguntas de investigación y la progresión del proyecto como tal. Utilizar la metodología indicada para cualquier proyecto es esencial ya que esta puede permitirle mayor fluidez a las formas e ideas de desarrollar la investigación, brindando las herramientas necesarias para tal fluidez. La importancia de poder plantear diferentes ideas es poder tener un mayor conocimiento de estas, y poder reflexionar a si estas se adaptan al concepto base del proyecto, (Dávila & Dávila, 2000). Para poder desarrollar un software es necesario poder plantear un modelo que brinde una serie de actividades para el diseño y desarrollo del software, este mismo modelo debe de organizar bien las etapas y los criterios a considerar, basados en las etapas, que están afectando al sistema, (Fariño, 2011).

El modelo en espiral es un modelo que evoluciona basado en los riesgos a los que se enfrenta un marco de referencia que guía al proceso de un software y sus aplicaciones, (Boehm, 1988). El modelo en espiral permite desarrollar un buen manejo de un proyecto de software. El manejo de los proyectos de software es una habilidad de integración de la tecnología de software, la economía que rodea el proyecto y las relaciones humanas en el especifico contexto del proyecto, como tal, (Boehm & Ross, 1989).

En cuanto a la progresión del modelo, el modelo en espiral se basa en cuatro ciclos, teniendo un total de cuatro fases, donde uno vas tras otro hasta volver al inicio, en forma de espiral, (Fariño, 2011).

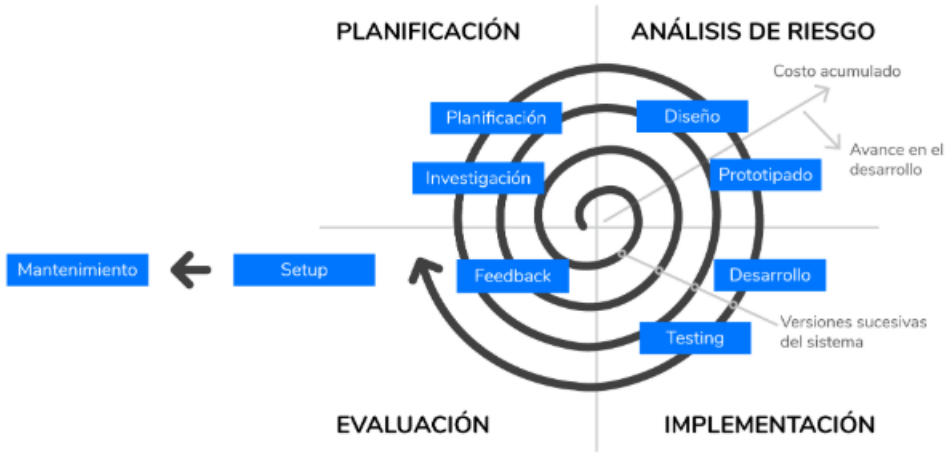


Ilustración 22- Modelo en espiral

Fuente de la imagen: (*El modelo de desarrollo en espiral como mezcla de cascada e iterativo*, 2019).

Las etapas del modelo espiral se pueden describir de la siguiente manera:

Tabla 7. Etapas del modelo en espiral

Etapas	Aspectos a lograr	Actividades
Planificación	• Determinación de objetivos, límites y condiciones de contorno (condiciones que limitan de alguna manera el desarrollo, económicas, de tiempo, etc.) y alternativas.	○ Predecir la duración de las actividades y tareas de nivel individual, Recursos requeridos, concurrencia y solapamiento de tareas para el desarrollo en paralelo y camino crítico a través de la red de actividades. ○ Estimar recursos: Predicción de personal esfuerzo y costo que se requerirán para terminar las actividades y productos conocidos asociados con el proyecto. ○ Planificar tareas y solapamiento de actividades y tareas. ○ Definir y desarrollar los requerimientos de software. ○ Definir los requisitos de interfaz. ○ Priorizar e integrar los requisitos de software.
Análisis de Riesgo	• Desarrollo de un plan para descubrir los riesgos más importantes y resolver los mismos. • Eliminación de aspectos no compatibles con las condiciones, o condiciones de contorno o	○ Identificar ideas o necesidades. ○ Formular soluciones potenciales. ○ Conducir estudios de viabilidad. ○ Planificar la transición del sistema. ○ Refinar y finalizar la idea o necesidad. ○ Analizar las funciones del sistema.

	límites. (Véase la tabla 8)	○ Desarrollar la arquitectura del sistema. ○ Descomponer los requisitos del sistema. ○ Planificación de contingencias.
Implementación	• Desarrollo del producto o prototipado según las condiciones de la etapa anterior.	○ Realizar el diseño arquitectónico. ○ Analizar el flujo de información. ○ Diseñar la base de datos. ○ Seleccionar o desarrollar algoritmos. ○ Realizar el diseño detallado de la etapa. ○ Crear el código fuente.
Evaluación	• Evaluar los resultados del prototipo obtenido, verificar y validar.	○ Crear los datos de prueba de código fuente. ○ Ejecutar las tareas de verificación y validación. ○ Recoger y analizar los datos de la métrica. ○ Planificar las pruebas. ○ Desarrollar las especificaciones de las pruebas. ○ Ejecutar las pruebas. ○ Generación de los aspectos de mejora, errores, defectos, ampliaciones.
Toma de decisiones	• Se determina si se pasa al ciclo exterior o se realiza una nueva iteración.	○ Evaluación de resultados. ○ Contraste con los hitos fijados o condiciones predeterminadas. ○ Fijación de las Condiciones de pasaje a los ciclos externos de no haberlas fijado. ○ Repetición del ciclo o pasaje a otro modelo. ○ Técnicas de toma de decisiones (arboles de decisión, decisiones en condiciones de incertidumbre, etc.).
Refinamiento	• Si se toma la decisión de continuar en los ciclos internos se sofistican las condiciones a tomar en cuenta en el planteamiento del nuevo ciclo, en los ciclos exteriores es una etapa que no se utiliza.	○ No hay actividades específicas, sino aquellas que generan las posibilidades de sofisticar indicadas anteriormente.

Fuente de la tabla: (Corcos, 2011)

Al momento de analizar los riesgos que tiene el proyecto a ser desarrollado hay que tener en cuenta los tipos de riesgos que rodean al proyecto. Al saber estos problemas el objetivo es saber cómo enfrentar estos problemas.

Tabla 8. Tipos de riesgos y cómo afrontarlos

Riesgos	**Formas de tratarlos**
Identificación de riesgos	o Lista de chequeo o Taxonomía basada en riesgos o Análisis de manejo de decisiones
Análisis de riesgos	o Árbol de decisiones o Modelo de nodos (PERT)
Costo y Calendario	o Trabajo de rotura de estructuras (WBS) o Modelo de Putman o Cocomo
Presupuesto	o Método del Factor de Riesgo
Desarrollo, control del planteamiento	o Reducción o Protección o Transferencia
Resolución del riesgo	o Prototipado o Modelado o Simulación o Benchmarking
Desarrollo de la implementación	o Lista de chequeo
Recursos humanos	o Acciones de proactividad
Monitoreo del riesgo	o Monitoreo de los 10 riesgos principales o Revisión e inspección o Índice de madurez del software
Aspectos colaterales al riesgo	o Métrica o Entrenamiento o Comunicación

Fuente de la tabla: (Corcos, 2011)

IV. METODOLOGÍA

La metodología como ya se había mencionado es una fuente de herramientas y pasos a seguir para la realización más efectiva y eficaz del proyecto a realizar. Se estará definiendo el enfoque que se le dará al proyecto, las variables de investigación, las técnicas a utilizar, los materiales necesarios para el desarrollo del proyecto, la manera en la que se aplicara la metodología como tal, que es el modelo en espiral, y el cronograma de actividades para la realización del proyecto.

5.1 ENFOQUE

El enfoque planteado para el presente proyecto investigativo es de tipo cuantitativo. Debido a que se van a utilizar diferentes intensidades de pixeles para hacer un análisis de visión computacional, se analizaran distancias en cada extremo del robot de manera de brindarle profundidad al análisis visual y estos datos se pasaran por varios algoritmos que, como se analizó en el capítulo anterior, son fórmulas matemáticas que buscan parametrizar estos valores para devolver estimaciones probabilísticas, que por un valor limite, se van a clasificar a unos o ceros.

De la misma manera, se plantea al proyecto de ser de tipo experimental. Ya que se realizarán diferentes experimentos y procedimientos, tanto como para obtener la información de los sets de entrenamiento, validación y prueba, el entrenamiento de la red neuronal utilizando esta información, y, finalmente, aplicar la red obtenida para poder hacer que el robot, a tiempo real, se mueva en un trayecto especifico.

5.2 VARIABLES DE INVESTIGACIÓN

Las variables de investigación son una de las partes más importantes a definir, ya que estas ayudan a moldear la investigación de manera de darle un camino a la misma de seguir de manera de cumplir con los objetivos. Se manejan dos tipos de variables: dependientes e independientes. Las dependientes son las variables que demuestran un resultado final que definirá si un objetivo se cumplió o no, estas dependen de una o varias variables independientes. Las variables independientes son por otro lado un tipo de variable que vendrían

a ser los casos o entradas necesarias para poder medir el cumplimiento o reacción que tienen las variables dependientes.

4.2.1 Variables Independientes

La red neuronal va a depender enteramente en sus entradas de la imagen devuelta por la cámara del robot. Por lo que los pixeles de la imagen van a dictar el estado inicial escalando de aquí la red por medio de otros métodos logrará determinar el movimiento que el robot requiere. Estos movimientos predichos definirán la precisión, que valdrá como filtro para saber si la red esta lista es la del conjunto de prueba. Por otro lado, también se busca armar un software que logre guardar la información que provee la cámara y el control del robot, por comunicación serial. Por lo que la conexión con estas dos comunicaciones también serán una variable crucial para considerar.

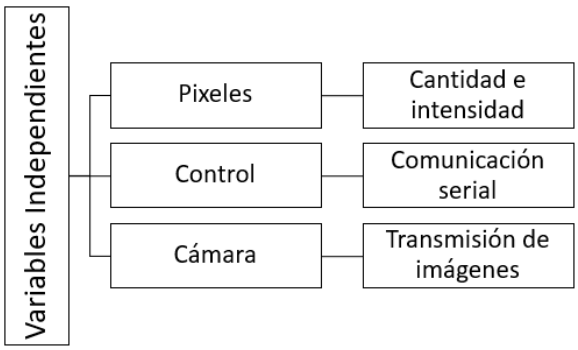


Ilustración 23- Variables independientes.

Fuente de la imagen: Elaboración propia.

4.2.2 Variables Dependientes

Como resultado del análisis hecho a las variables independiente se obtienen las variables independientes. En el caso de la red neuronal, la red predice que movimiento debe realizar el robot y esta se va definir como el modelo definitivo debido a la precisión obtenida. Para saber que tal está funcionando la red es necesario saber la precisión que esta tiene, la precisión se mediría sobre el conjunto de entrenamiento, validación y prueba. Esta precisión será planteada entonces como variable dependiente. Esta precisión dependerá de los movimientos predichos, ya que la predicción de la red es la que hace que este valor varié. Estos movimientos dependen

de los pixeles de entrada de la red. Y finalmente, en cuanto al software entrenador, se definirá como variable dependiente si los archivos se logran almacenar en el formato y directorio indicado.

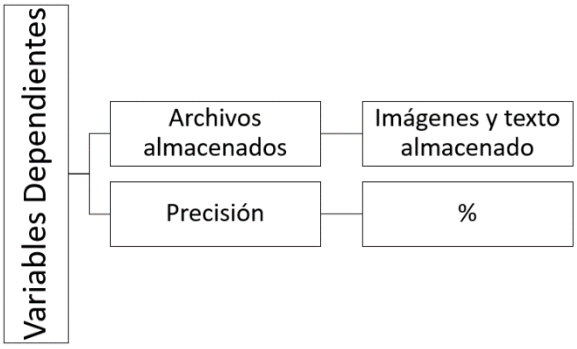


Ilustración 24- Variables dependientes.

Fuente de la imagen: Elaboración propia

5.3 TÉCNICAS E INSTRUMENTOS APLICADOS

Las técnicas e instrumentos aplicados son un parte fundamental para la realización de un proyecto. Ya que a través de estos se realiza el proyecto como tal, y están diseñadas para poder desarrollar proyectos con mayor facilidad.

Parte crucial de este proyecto fue la investigación, ya que a través de esta se logró recopilar la base teórica del proyecto. Para realizar la investigación se utilizaron varios libros, videos educativos, revistas académicas, otros proyectos investigativos, etc. Los temas de todos estos girando alrededor de redes neuronales artificiales, aprendizaje automático y la metodología que se utilizó, el modelo en espiral.

Para el desarrollo en software se utilizaron las siguientes herramientas:

- Visual Studio: un IDE para el desarrollo de aplicaciones, que es compatible con varios tipos de lenguaje de programación.
- Windows Form: Es una interfaz de tipo .NET para Windows.
- C#: Es un lenguaje de programación, que en el presente proyecto fue utilizado para la codificación de la aplicación entrenadora.
- Aforge: Es una librería de visión computacional que se utilizó para la aplicación, se utilizó esta librería para la aplicación por la muy buena respuesta que tiene esta con el lenguaje C#

- Python: Es un tipo de lenguaje de programación, muy utilizado para el análisis de datos, que en el caso del presente proyecto se utilizó para la creación de la red neuronal, y la conexión entre la red y el control del robot.
- Open CV: Es una librería de visión computacional, que se utilizó para la red neuronal, en este caso se utilizó esta librería por la buen eficiencia y respuesta que esta tiene con el lenguaje Python.
- Keras: Es una librería de aprendizaje automático para Python, la cual se utilizó para realizar el modelado de la red neuronal y para optimizar los parámetros del modelo.
- PySerial: Es una librería de comunicación serial para Python, la cual se utilizó para generar una conexión entre el cmd y el control y poder mandar los valores predichos por la red neuronal.

5.4 MATERIALES

Lo que respecta a los materiales para el desarrollo en hardware para el presente proyecto gira en torno al robot utilizado para el desarrollo de la red neuronal, robot VPR. Para lo que entonces se utilizaron los siguientes materiales:

- Robot VPR: Este es un robot desarrollado en la Universidad Tecnológica Centroamericana por los alumnos que en su momento cursaban la clase de Diseño Mecatrónico. Este robot fue desarrollado para aplicaciones logísticas, transportar objetos de un lado al otro, y este es teleoperado por medio de comunicación de radiofrecuencia entre el robot y el control, el cual es un control de PlayStation configurado pues para manipular el robot.
- Arduino MEGA: Para controlar los motores y sensores presentes en el robot se utilizó este microcontrolador que tiene varias entradas y salidas. Esta buena cantidad de pines permite poder controlar en una sola placa todas las características de este robot. Cabe destacar que la implementación de este dispositivo es una modificación para el presente proyecto. El robot inicialmente se manipulaba con dos Arduinos UNO.
- Arduino UNO: Debido a la complicada situación que se pasa en el momento en el que se desarrolla este proyecto, debido a la pandemia COVID-19, el control del robot se encontraba

fuera del alcance. Por lo que se implementó un Arduino UNO, el cual se comunica por bluetooth al Arduino MEGA, para el control del robot.

- Modulo Bluetooth HC-01: Se utilizo un módulo de bluetooth por ambos microcontroladores, el MEGA y UNO, para comunicar al control y el robot.

5.5 METODOLOGÍA DE ESTUDIO

En esta sección se definirá el uso que se le dará al modelo en espiral para la realización del proyecto. Pasando por las 4 etapas del modelo de manera de poder ir progresando en el proyecto como tal de manera de ir cumpliendo con los objetivos planteados.

4.5.1 PRIMERA CICLO: SOFTWARE DE ENTRENAMIENTO

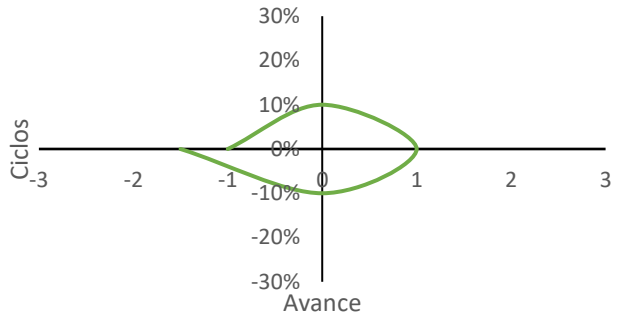


Ilustración 25- Primer ciclo basado en el software de entrenamiento del robot

Fuente de la imagen: (Jimenez, 2020).

4.5.1.1 Primera Etapa: Planificación

Para la primera etapa se debe de planificar como se va a resolver el problema como tal para el primer ciclo. Lo primero a considerar es como recoger las variables de manera de poder almacenar en algún tipo de documento. También se debe de plantear la pista por la que se va a implementar el software. Considerar la mayor cantidad de escenarios.

4.5.1.2 Segunda Etapa: Análisis de Riesgo

En la segunda etapa se debe de plantear los riesgos a considerar al momento de aplicar un software de este tipo. La cantidad de información que se está procesando puede ser de

mucha capacidad, debido a que la dimensión de la imagen tiene que ser de la mayor cantidad posible. El retraso que puede haber en la información para pasar desde los microcontroladores en el robot al microcontrolador del control, y de este control al sistema.

4.5.1.3 Tercera Etapa: Implementación

En la tercera etapa se busca tener terminado el prototipo del software basado en los objetivos específicos de este y en los riesgos a considerar. El software se va a realizar con el programa Visual Studio 2019 en el lenguaje C# haciendo uso de Windows Form. A priori al prototipo finalizado se procede a realizar pruebas al software.

Para poder recopilar la información se hará uso de la librería Aforge, la cual permitirá al sistema tener visión computacional, y se usará comunicación serial para comunicarse con el control.

4.5.1.4 Cuarta Etapa: Evaluación

Llegada a la etapa final del ciclo se realiza una evaluación del software bajo cualquier tipo de error que pueda tener el sistema. Exponerlo a otros usuarios para verificar que el sistema funcione de una manera amigable.

Obtenida esta retroalimentación se procede a realizar un proyecto de setup en Visual Studio de manera de realizar una aplicación ejecutable para cualquier ordenador con sistema operativo de Windows.

Luego, aplicar el software en el robot para obtener la información necesaria para poder alimentar la red neuronal con la mejor y mayor información posible.

4.5.2 Segundo Ciclo: Red Neuronal

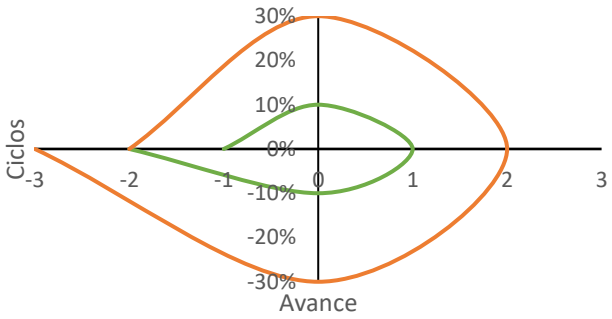


Ilustración 26- Segundo ciclo basado en la red neuronal.

Fuente de la imagen: (Jimenez, 2020)

4.5.2.1 Primera Etapa: Planificación

Para esta etapa se debe planificar las características de la red neuronal para la autonomía del robot. Se debe de considerar la mejor arquitectura, la mejor forma de separar las salidas y la mejor forma de elegir los pixeles.

4.5.2.2 Segunda Etapa: Análisis de Riesgo

Sabido los objetivos de este ciclo se debe considerar los riesgos que puede tener este sistema. Como puede ser una alta parcialidad o una alta variancia. También es importante tener en cuenta que tan computacionalmente costos puede ser este sistema. Debido a la gran cantidad de variables de entrada que puede llegar a tener el sistema.

4.5.2.3 Tercera Etapa: Implementación

Entrenar la red neuronal utilizando la información obtenida utilizando el software de entrenamiento obtenido en el primer ciclo. La programación del sistema se realizará con el programa Visual Studio 2019 utilizando el lenguaje Python. Hacer pruebas del sistema utilizando el set de validación, para analizar a profundidad los riesgos planteados en la etapa anterior.

4.5.2.4 Cuarta Etapa: Evaluación

Poner a prueba el sistema con el set de prueba, valga la redundancia, para medir el error final del sistema. Medidos lo errores hacer distintas pruebas para realizar la retroalimentación necesaria.

De no recibir datos prometedores, se buscaría repetir el ciclo de manera de mejorar el sistema, para obtener los menores errores posibles o al menos llegar a un rango aceptable.

4.5.3 Tercer Ciclo: Implementación de la Red Neuronal en el Robot

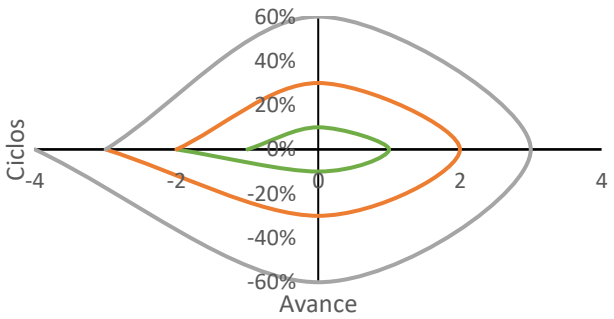


Ilustración 27- Tercer ciclo basado en la implementación de la red neuronal en el robot

Fuente de la imagen: (Jimenez, 2020)

4.5.3.1 Primera Etapa: Planificación

En esta etapa del último ciclo se debe tener en cuenta varios escenarios, que podrían ser los mismos que el primer ciclo, pero para una mayor amplitud de casos se debería de implementar distintos circuitos.

4.5.3.2 Segunda Etapa: Análisis de Riesgos

Se debe de considerar la seguridad del robot a lo largo de las pruebas en tiempo real de la red neuronal implementada. Ya que debido a la situación complicada que se pasa en el momento en el que se desarrolla el presente proyecto, la pandemia del COVID-19, se torna muy complicado buscar repuestos para el robot.

4.5.3.3 Tercera Etapa: Implementación

Considerando el objetivo y los riesgos de este ciclo se busca poder conectar al robot con la red neuronal de manera que, en tiempo real, esta pueda manipular al robot para llegar desde el punto en el que se encuentra hasta un punto fijo definido.

4.5.3.4 Cuarta Etapa: Evaluación

En esta etapa se debe de medir la eficiencia del algoritmo obtenido, haciendo usos de herramientas como la precisión y recordación, o el puntaje F1.

5.6 CRONOGRAMA DE ACTIVIDADES

Como su nombre lo indica en esta sección se estará definiendo las actividades a hacer en el transcurso de tiempo en el que se llevará a cabo el proyecto. Definiéndose en cinco pilares claves: informe, software entrenador, robot, red neuronal e implementación de la red neuronal.

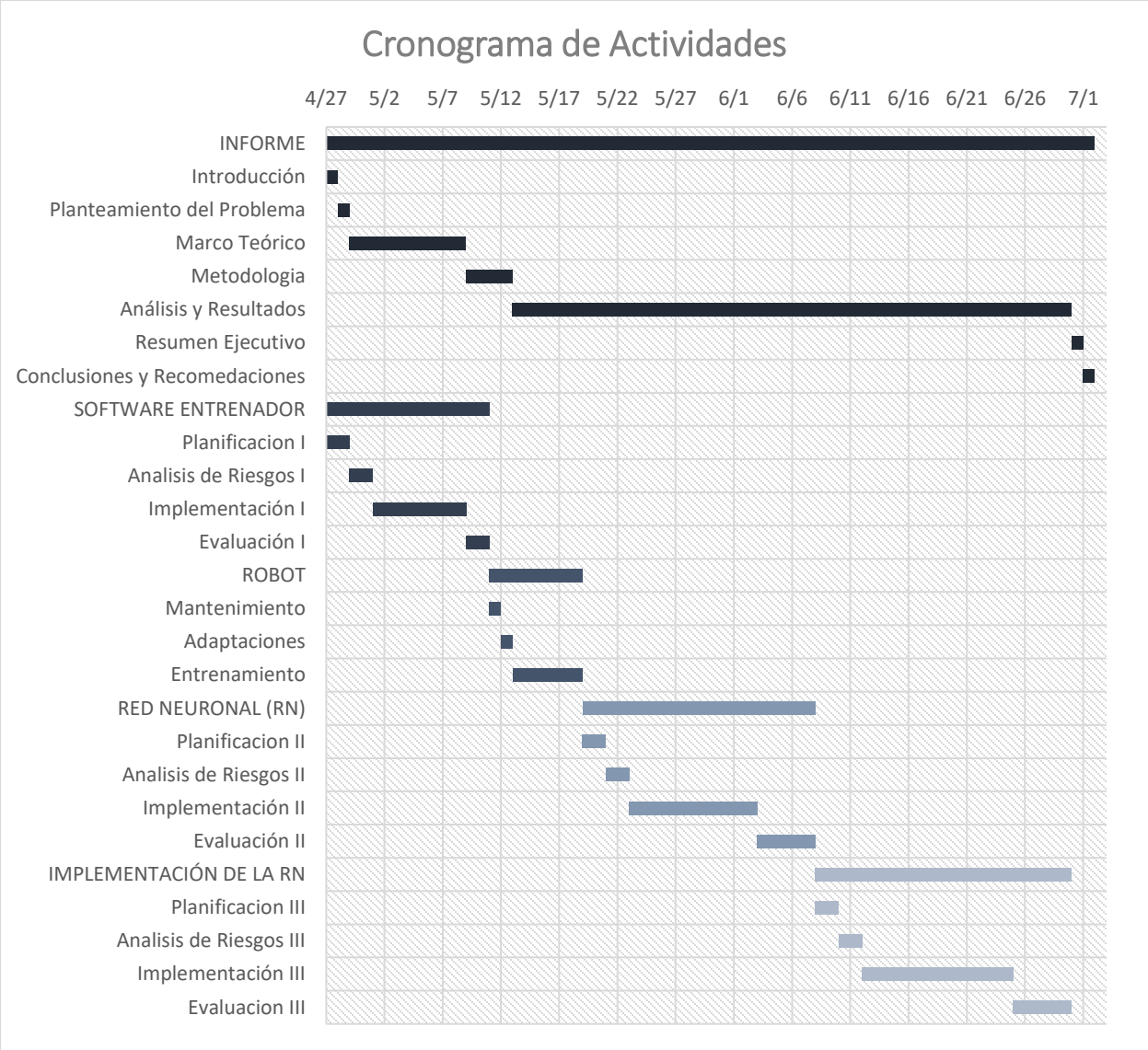


Ilustración 28- Cronograma de actividades

Fuente de la imagen: Elaboración propia.

V. ANÁLISIS Y RESULTADOS

En el presente capítulo se estará planteando la manera en la que se desarrolló el proyecto y la investigación. Aplicando la teoría de sustento planteada en el marco teórico, y también la forma en la que se utilizó la metodología planteada en el capítulo anterior para cumplir con los objetivos y responder las preguntas de investigación planteadas en el planteamiento del problema.

5.1 PRIMER CICLO: DESARROLLO DEL SOFTWARE ENTRENADOR

En el primer ciclo se buscó desarrollar un software que permitiera reunir los datos necesarios para el entrenamiento de la red neuronal.

5.1.1 PLANIFICACIÓN: PRIMER CICLO

Lo más crucial es saber cuál es el objetivo de esta aplicación y si se va a resolver algún problema como tal. Entonces saber este problema a priori a generar la aplicación va ayudar a no tener que estar variando la aplicación y permite trabajar unidireccionalmente hacia cumplir el objetivo. Por lo que entonces, ¿cuál es el problema a resolver? La manera en la que se observó en el marco teórico que se obtiene un buen algoritmo de aprendizaje automático es a través de la mejor y más fiable data posible. Para eso se debe de generar un software que permita recolectar la información necesaria y en tiempo real. Pero para eso es importante también saber cómo funciona el robot que se estará utilizando y ver si es necesario generar una adaptación de parte del robot. Hay una sección del presente capitulo que cubre esas adaptaciones en específico.

El robot funciona de una manera sencilla se podría decir el robot es teleoperado e interpreta distintos caracteres, como ser la A, B, C, etc., en que movimiento debe de realizar. Esos caracteres son enviados por un control. Por lo que se puede obtener el movimiento del robot sabiendo que carácter está enviando el control. Esta información se puede obtener del control vía comunicación serial. Este también cuenta con 5 ultrasónicos colocados en lugares estratégicos en los lados del robot. Estos envían sus señales a un microcontrolador distinto al de los motores. La placa controladora de los motores es un Arduino Mega, y la de los sensores es

un Arduino UNO. Estas dos placas se comunican por comunicación serial, de manera que el Arduino Mega, que vendría a ser la placa maestra, tenga toda la información del robot en cuanto a sensores y movimientos. Los sensores fueron puestos para brindarle profundidad al robot y a un software a parte que se generó para el monitoreo del mismo. Sin embargo, más adelante se verá que la información que brindan los sensores no es muy útil para el modelo y tipo de red neuronal que se estará aplicando. El robot también cuenta con una cámara de manera de saber su posición al momento de manejarlo a largas distancias. Esta cámara es la parte más importante para la red neuronal, y por ende para el proyecto. Ya que esta es la que provee la información de donde está el robot y que movimientos tiene que realizar según lo que está viendo. La cámara funciona de cierta manera independiente al resto del sistema. Esta tiene su propia antena y se conecta con su propio receptor también por comunicación de radiofrecuencia.

Sabiendo lo anterior entonces es claro que el objetivo de la aplicación es conseguir los movimientos que está realizando el robot, que vendrían a ser la salida del robot, y las imágenes que está captando la cámara. Estas dos cosas las debe hacer simultáneamente y en tiempo real, de manera de poder conseguir los datos más fiables posibles. Otra cuestión importante a considerar es poder generar esta aplicación de una manera más generalizada. De forma que cualquier usuario que quiera obtener información de algún sistema que utiliza comunicación serial y una cámara pueda perfectamente utilizar este software propuesto.

5.1.2 Análisis de Riesgo: Primer Ciclo

Sabiendo entonces los objetivos se debe de plantear que riesgos o trabas se pueden tener en el camino. El detalle probablemente más delicado de esta etapa es poder conseguir la información desde dos distintos canales. Otro concepto que se debe de mantener en cuenta es generalizar, ósea crear una aplicación que pueda ser utilizada por cualquiera y que sea amigable para el usuario. Basados entonces en la tabla 8 el tipo de riesgo al cual se enfrenta este ciclo es de desarrollo de implementación. Ya que los túmulos que se puede llegar a encontrar la aplicación son tecnicidades y son riesgos que se pueden solucionar mejorando e implementando diferentes formas del código.

Entonces para afrontar estos riesgos es importante mantener el sistema lo más sencillo posible. Como se vio en la planificación del presente ciclo. Se necesita poder comunicar con un puerto serial y con una cámara. El objetivo de la aplicación es ir capturando la información brindada por estas dos comunicaciones, dado cierto tiempo. Por lo que la aplicación debe hacer dos preguntas importantes: ¿Cuántos ejemplos? Y ¿Cada cuanto tomo cada ejemplo? Por lo que se estará siguiendo el siguiente diagrama de flujo.

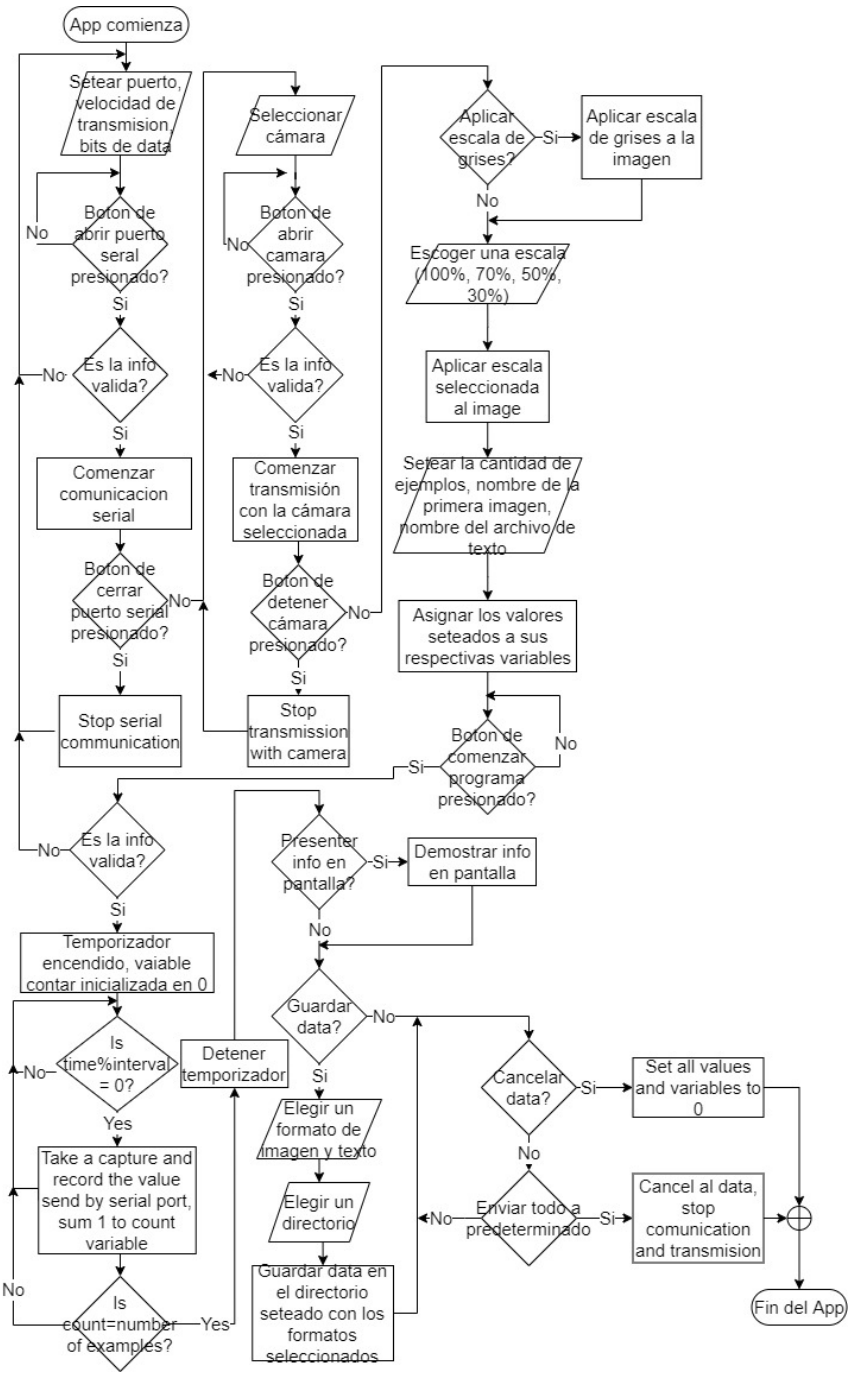


Ilustración 29- Diagrama de flujo del software entrenador[AMCB6]

Fuente de la imagen: Elaboración propia

5.1.3 Implementación: Primer Ciclo

Sabiendo entonces el flujo de código planteado en la sección anterior se procedió a trabajar en el código. El código se realizó en el lenguaje de programación C# en el editor de

código Visual Studio. Para la interfaz del programa se utilizó un Windows Form facilitado también por el software Visual Studio.

Para la comunicación serial se utilizará la librería disponible en Visual Studio que permite abrir la comunicación y seleccionar los valores de rapidez de transmisión, los bits de transferencia, los bits de detención y los bits de paridad. La rapidez de transmisión que se usará como predeterminada es de 9600, los bits de transferencia serán 8, los bits de detención se manejaran en nada más 1, los bits de paridad se manejaran en cualquier caso posible, ya que estos pueden ser pares o impares.

En cuanto a la cámara Visual Studio en su set de librerías no tiene una librería de comunicación visual, solo manejo de mapa de bits, por simplicidad se utilizará el termino bitmaps que es la traducción en ingles de mapa de bits. Dado esto se tiene que exportar una librería externa para comunicación visual. Una de las mejores librerías para manejo de comunicación visual para la aplicación en inteligencia artificial, como lo es el aprendizaje automático y las redes neuronales, es la librería Aforge. Esta librería ofrece varios métodos para manipulación de imágenes, videos y comunicación visual, al igual maneja varias librerías para redes neuronales y aprendizaje automático, pero eso será para otra sección. De esta librería se utilizó la extensión Direct Show, esta lo que permite es realizar una conexión con una cámara disponible en la computadora e ir actualizando marco por marco la imagen que se presenta en aluna caja de imagen. Esta misma librería tiene una extensión, llamada Grayscale, que permite convertir una imagen en una imagen en escala de grises.

Para poder establecer los valores de numero de ejemplos, intervalo de tiempo, nombre del archivo y nombre de la primera imagen. Para esto se utilizará un botón y una caja de texto. Cualquier valor escrito en la caja de texto, al presionar el botón, ese valor se le asigna a la respectiva variable. Cabe destacar que todos estos tienen que ser valores enteros positivos mayores a cero. El motivo por el cual los nombres de los archivos son números es porque al momento de volver a utilizar estos archivos en un programa o código a parte su declaración siendo tan sencilla como un número entero hace pues que el código para llamarlos sea pues igual sencillo. El nombre que se asigna para las imágenes es solo para la primera, ya que el resto se llamara un número mayor a la imagen anterior. Por ejemplo, si se establecen 10 ejemplos y

que el nombre de la primera imagen es 10, la segunda imagen se llamaría 11, la siguiente 12, así sucesivamente hasta la última imagen que se llamaría 20. Por lo que la interfaz de la aplicación se ve de la siguiente manera.

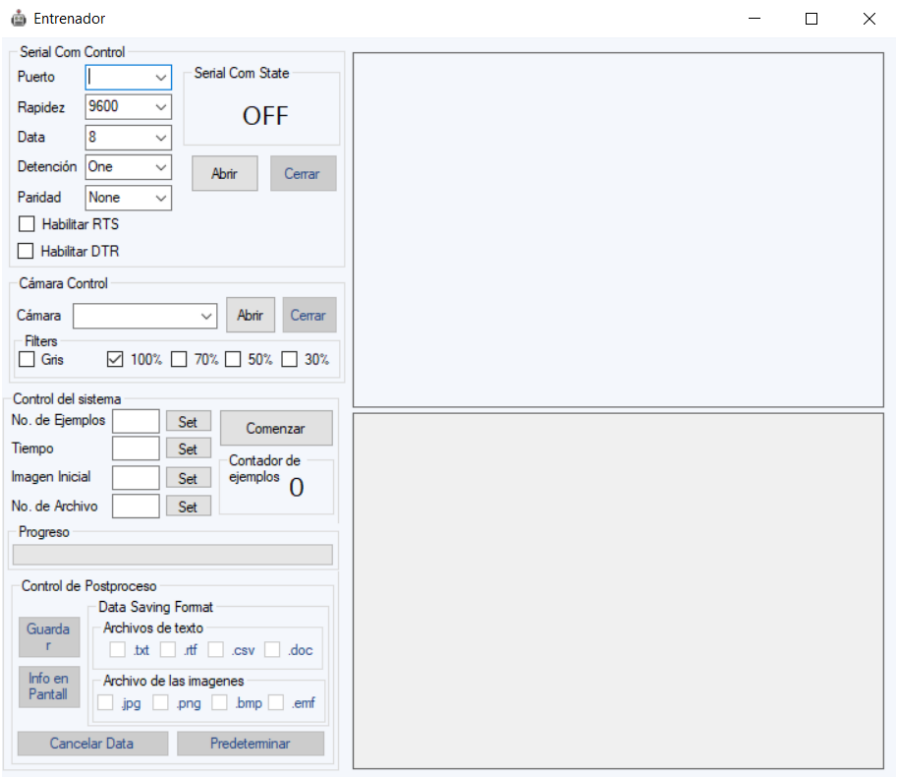


Ilustración 30- Interfaz del software entrenador

Elaboración de la imagen: Elaboración propia

Como se puede ver en la interfaz al terminar el programa de capturar toda la información establecida por el usuario se necesita varios procesos al terminar el programa. El más esencial es el botón de guardar. Pero previo a presionar este botón se tiene que seleccionar el formato deseado para guardar los dos tipos de archivos que se manejan. Seleccionado los formatos y presionado el botón de guardar el software pide la selección de un directorio para guardar la información. Al igual se puede presentar la información en pantalla, de manera de poder monitorear previo al almacenar la información si esa es la información necesaria para lo que se necesita. También se puede cancelar la información, que lo que hace es mandar las variables a 0 y olvidar la información guardada en el sistema, o mandar todo a predeterminado que aparte de hacer lo mismo que el protocolo de cancelar la información también apaga la comunicación serial y la transmisión con la cámara.

5.1.3.1 Pruebas

Para esto se pondrá a prueba el software bajo los diferentes objetivos en el que se realizó. Primer objetivo será ver si este captura la cantidad de información que se le pide, luego si almacena toda la información en el formato establecido en el directorio seleccionado. También se pondrá a prueba los diferentes postprocesos y los filtros de las imágenes, en los cuales las escalas solo se podrán ver reflejado el cambio una vez el programa termine. Esto debido que la información que se presenta en pantalla va a ser la dimensionalidad de la imagen y el valor leído del puerto serial.

Para la comunicación serial se utilizará como herramientas el software Configure Virtual Serial Port Driver, que es un software que permite generar pares de puertos virtuales, y Tera Term, que permite mandar o recibir información a través de un puerto serial. Por lo que a priori a las pruebas se debe de encender los puertos virtuales y configurarlos para que se comuniquen entre ellos. El software lo que generará son dos puertos virtuales llamados COM1 y COM2. La aplicación Tera Term se configurará para recibir o enviar su información a través del COM1. Para la cámara se utilizará tanto la cámara del robot y la cámara de la computadora, de manera de ver que tal funciona en ambas situaciones el software.

Prueba #1: Conexiones: puerto serial y cámara externa

Para la primera prueba se estará probando las conexiones de tanto el puerto serial como la cámara. Para el puerto serial se estará utilizando los puertos virtuales generados con el software Configure Virtual Serial Port Driver y se estará mandando los mensajes por medio del software Tera Term. Para la cámara se estará utilizando directamente la cámara del robot.

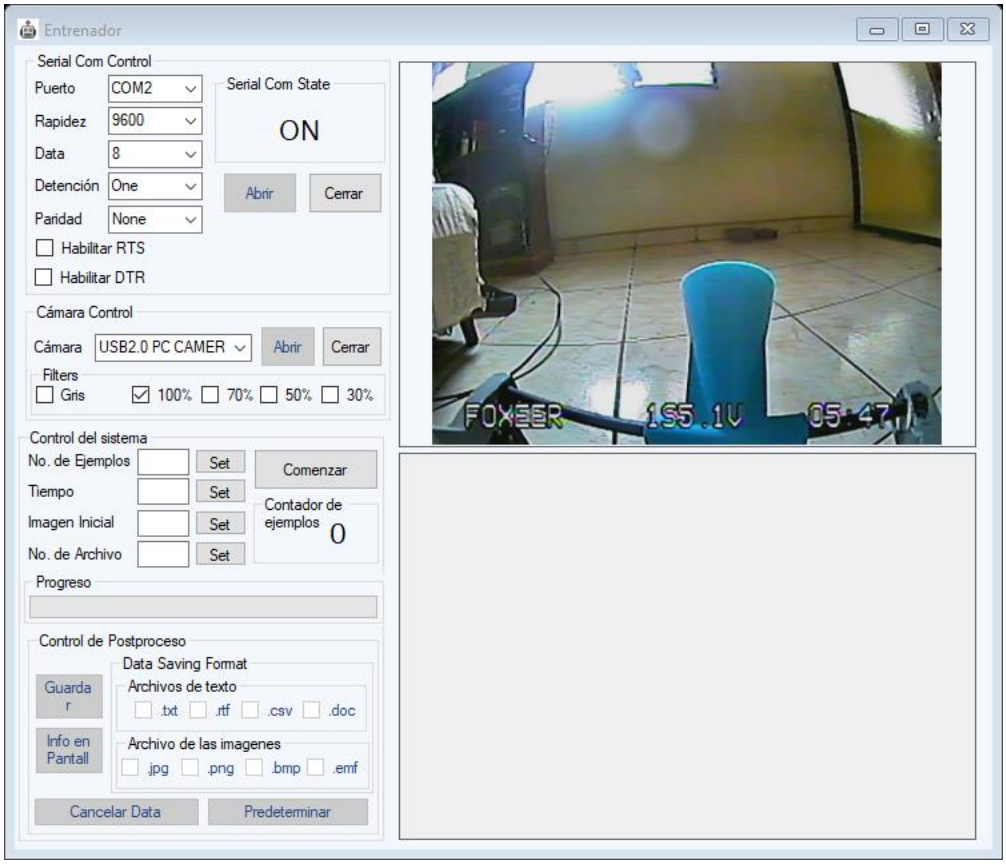


Ilustración 31- Prueba #1 del software entrenador

Fuente de la imagen: Elaboración propia

Como se puede observar en la imagen el software responde como se esperaba. La conexión con la cámara es estable al igual que con el puerto virtual.

Prueba #2: Cámara interna y filtros

Para esta segunda prueba se estará probando la cámara integrada de la computadora, de manera de probar diferentes conexiones, el filtro de escala de grises que tiene el software, y se aplicará alguna de las escalas disponibles. Siempre se estará encendiendo el puerto serial, pero en la prueba anterior se logra observar una conexión exitosa.

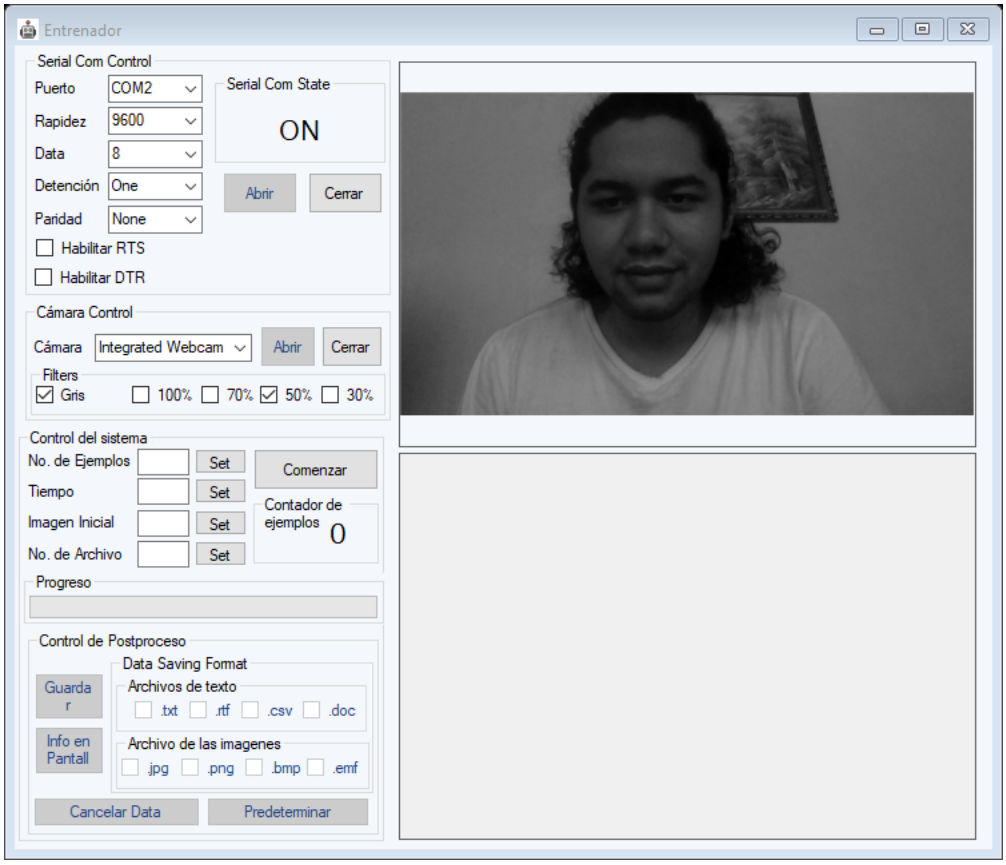


Ilustración 32- Prueba #2 del software entrenador

Fuente de la imagen: Elaboración propia

Como se puede observar en la imagen la cámara funciona perfectamente con el software y el filtro de escala de grises se aplica a tiempo real en cada marco que brinda la cámara. Sin embargo, la escala de 50% no parece haber tenido un gran impacto en la imagen. Se puede observar que la imagen se ve del mismo tamaño. Esto es debido a que la caja de imagen que se utilizó en la interfaz está programada con un parámetro llamado "zoomed", este filtro lo que realiza es ajustar la imagen al tamaño de la caja de imagen. Por lo que entonces las escalas solo se podrán ver, se podría decir, en acción ya al momento de ver la dimensionalidad de imagen al terminar el programa. Esta información se programa que se presente cuando se utiliza el botón de presentar la información en pantalla.

Cabe destacar que el botón de postproceso cancelar data llevaría a la aplicación al punto en el que se encuentra tanto la prueba 1 y la prueba 2. Ya que está destinado a conservar las conexiones, pero eliminar la información recaudada por el software. Y ahí es donde esta

exactamente el software en ambas pruebas, sin información recaudada, pero con las conexiones encendidas y estables. Al igual que presionar el botón de cancelar data al terminar el protocolo de salvar la información en el ordenador queda en la misma posición al igual que presionar dicho botón.

Prueba #3: Recaudar información y presentar la información en pantalla

Para esta etapa se recaudará la información obtenida según los parámetros definidos, se presentará la información para revisar si se realizó la cantidad de ejemplos establecidos y se podrá observar el efecto que tiene el filtro de escalar la imagen a diferentes porcentajes. Por motivo de pruebas se configurarán valores cortos. Para la cantidad de ejemplos se dejará en 10, el intervalo de tiempo en 20ms de momento no interesa el parámetro de imagen inicial y numero de archivo, pero debido a que el programa no podría arrancar sin estos dos valores configurados, se van a establecer en 1 ambos. La escala se pondrá esta vez en un 30%.

A priori a la ejecución del programa se definirá cuál debería ser la dimensionalidad que debe de devolver el sistema aplicando un 30%. Se utilizará la cámara integrada de la computadora. Esta devuelve predeterminadamente imágenes de dimensionalidad de 1280×720, aplicando una escala de 30% debería entonces de dejar una dimensionalidad de 384×216. En cuanto a la información a recaudada del puerto serial se utilizará el software Tera Term para mandar los caracteres.

Como se puede observar en la ilustración 32 el software respondió como lo esperado. Logro recaudar la cantidad de ejemplos establecidos. Se comunico perfectamente con Tera Term, software el que solo estaba presionando teclas de manera aleatoria. Y en cuanto al filtro de escala se puede observar que la dimensionalidad es exactamente el valor establecido de 384×216.

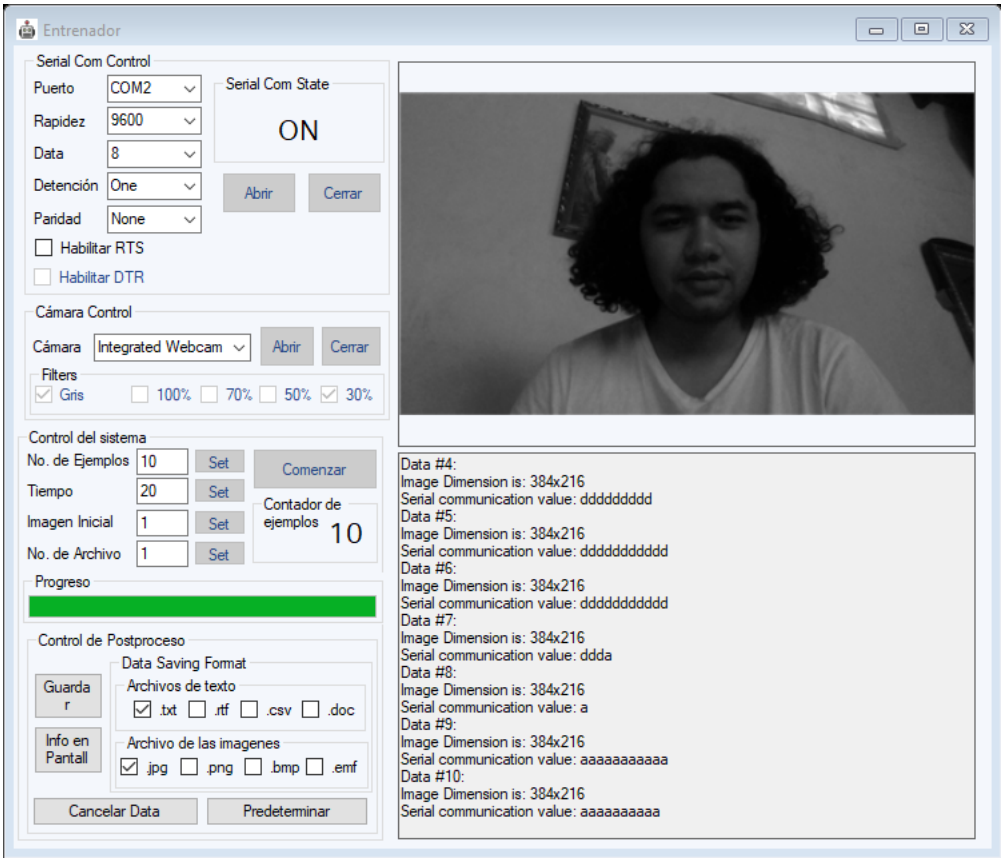


Ilustración 33- Prueba #3 del software entrenador

Fuente de la imagen: Elaboración propia

Prueba #4: Guarda la información

Para esta última prueba se buscará guardar la información en algún directorio disponible en la computadora en el que se utiliza el software. Como ya se obtuvo un conjunto de información en la prueba anterior se utilizará esa información para probar el protocolo de guardar la información. Los formatos que se estarán utilizando son: para la imagen JPG y para el archivo de texto CSV. El formato de imagen no importa mucho para una red neuronal, simplemente fue una decisión personal utilizar el formato JPG. El formato de texto CSV es uno de los formatos más utilizados en la rama de análisis de data. Lo interesante de cómo se configuro la información en este tipo de archivo es de que al generarse este va a clasificar cada dato obtenido de la comunicación serial con la dirección en la computadora de la respectiva imagen con la que se tomó tal dato. Ósea que el primer dato obtenido se va a direccionar, en texto, con la primera imagen y así sucesivamente con todas las imágenes. Este tipo de archivo se

puede abrir con Microsoft Excel. Cabe destacar que esta unión solo se da con este tipo de archivo, con los demás el software solo generará una línea de texto por dato obtenido, cada línea claro tendrá los caracteres obtenidos por la comunicación serial.

Para guardar la información se generó una carpeta en el escritorio de la computadora llamada "Info Recaudada". Por lo que se configuro ese directorio para guardar la información.

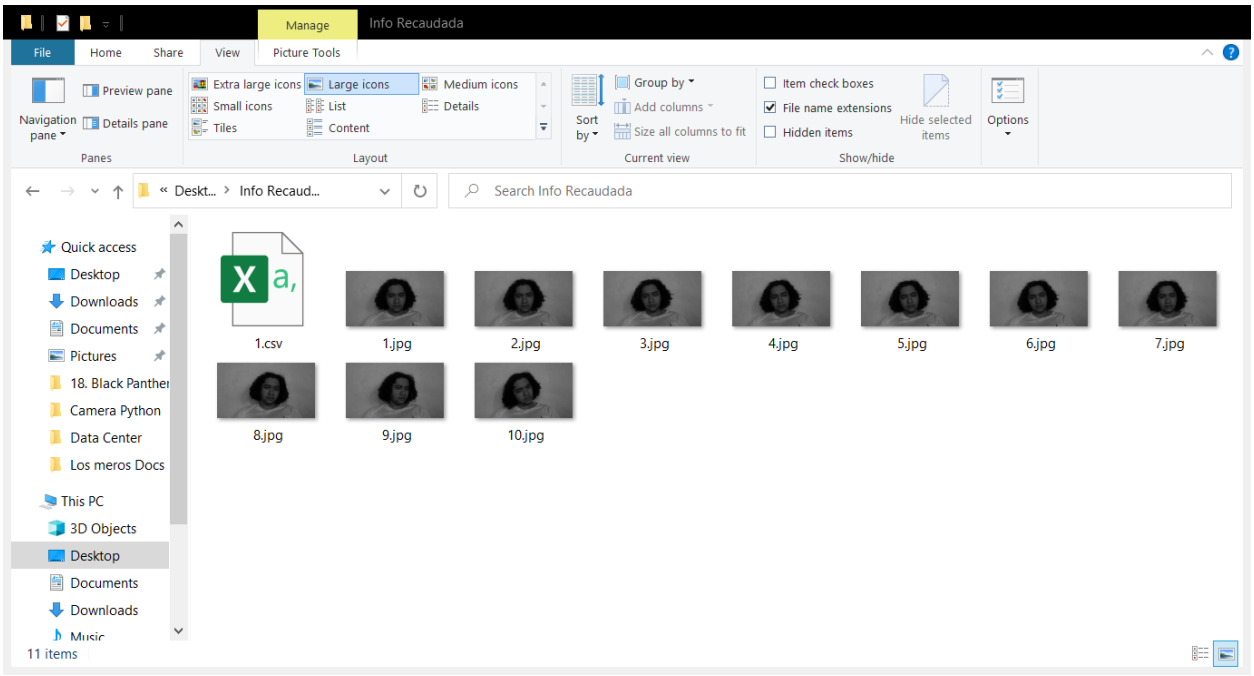
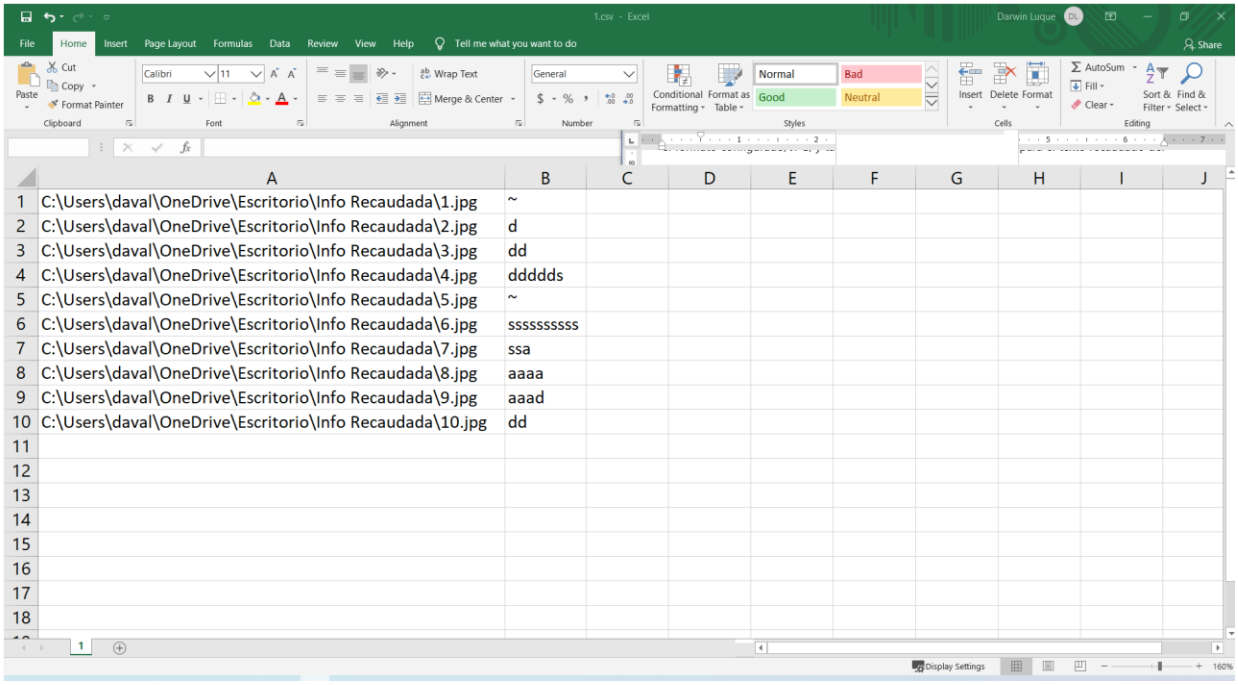


Ilustración 34- Prueba final del software entrenador

Fuente de la imagen: Elaboración propia

Como se puede observar en la imagen guardo exitosamente las 10 capturas tomadas con el formato configurado, JPG, y también se almaceno el archivo CSV para el texto recaudado del puerto serial.

Como se predijo el archivo direcciona cada texto recaudado desde el puerto serial con cada imagen almacenada (véase la ilustración 36). Un último detalle a analizar es el símbolo "~" que se ve en algunas casillas. Este símbolo se da cuando el software no detecta un texto vacío viniendo del puerto serial. O en palabras más sencillas el símbolo se da cuando no se recibe información del puerto serial.}



	A	B
1	C:\Users\daval\OneDrive\Escritorio\Info Recaudada\1.jpg	~
2	C:\Users\daval\OneDrive\Escritorio\Info Recaudada\2.jpg	d
3	C:\Users\daval\OneDrive\Escritorio\Info Recaudada\3.jpg	dd
4	C:\Users\daval\OneDrive\Escritorio\Info Recaudada\4.jpg	dddddds
5	C:\Users\daval\OneDrive\Escritorio\Info Recaudada\5.jpg	~
6	C:\Users\daval\OneDrive\Escritorio\Info Recaudada\6.jpg	sssssssss
7	C:\Users\daval\OneDrive\Escritorio\Info Recaudada\7.jpg	ssa
8	C:\Users\daval\OneDrive\Escritorio\Info Recaudada\8.jpg	aaaa
9	C:\Users\daval\OneDrive\Escritorio\Info Recaudada\9.jpg	aaad
10	C:\Users\daval\OneDrive\Escritorio\Info Recaudada\10.jpg	dd

Ilustración 35- Archivo CSV generado por el software entrenador

Fuente de la imagen: Elaboración propia

El archivo logra direccionar cada texto recaudado desde el puerto serial con cada imagen almacenada. Un último detalle a analizar es el símbolo "~" que se ve en algunas casillas. Este símbolo se da cuando el software no detecta un texto vacío viniendo del puerto serial. O en palabras más sencillas el símbolo se da cuando no se recibe información del puerto serial.}

5.1.4 Evaluación: Primer Ciclo

En esta parte final del ciclo lo que se busca es tener una retroalimentación de manera de poder confirmar que el software funciona debidamente y está bien generalizado. Para esto se pondrá a prueba en una diferente aplicación con un diferente usuario de manera de ver si la aplicación le es de uso y si la aplicación es amigable en cuanto a la interfaz.

El presente software se logro aplicar en un proyecto que busca recopilar información sobre una generación de agua. Haciendo uso de la comunicación serial para obtener los datos devueltos por un Arduino y las capturas de imagen se utilizaron para mantener un seguimiento del nivel de agua y ver si la cantidad de agua que se mira concuerda con lo devuelto por el Arduino.

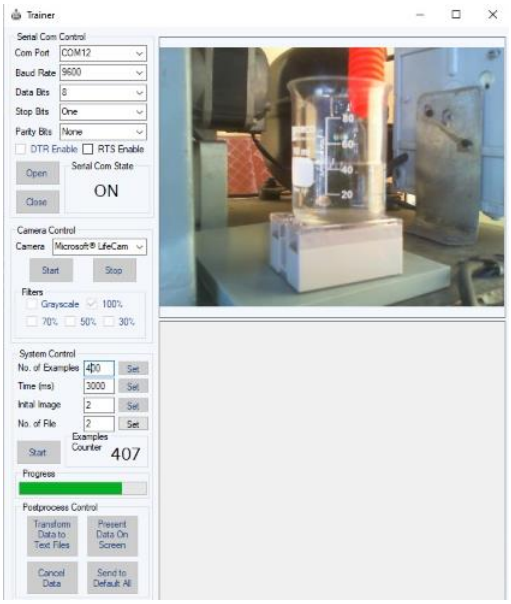


Ilustración 36- Retroalimentación del Software Entrenador

Fuente de la imagen: (Trejo, 2020)

5.2 VPR

En esta sección del proyecto se le pone un paréntesis a la metodología y se busca atacar de una manera más directa al mantenimiento necesario que puede llegar a necesitar el robot, las adaptaciones que necesita para funcionar con la red neuronal y el software diseñado en el primer ciclo de la metodología. Pero previo al mantenimiento y adaptación del robot es importante saber cómo funciona el robot como tal.

Para empezar el robot fue diseñado en la Universidad Tecnológica Centroamericana (UNITEC) por alumnos de la carrera de ingeniería en Mecatrónica, como parte de un proyecto para la clase de Diseño Mecatrónico. El robot es teleoperado y funciona con dos Arduinos, un Mega y un UNO ambos teniendo distintas aplicaciones, y una cámara. El control del robot funciona con un Arduino UNO que está configurado con un control de consola de videojuegos y cuenta con una pantalla. Este se comunica con el Arduino Mega por medio de comunicación RF. La señal que este control envía son caracteres que dictan que movimiento debe realizar el robot, al igual recibe una señal del Arduino Mega que es una señal codificada para los sensores ultrasónicos que tiene el robot. El Arduino Mega del robot recibe la señal del control y también le envía una señal devuelta al control. Este Arduino es también el que manipula la señal de los

motores. El Arduino UNO del robot recibe la señal de los ultrasónicos y este por medio de comunicación serial le envía la señal codificada al Arduino Mega y este solo reenvía dicha señal.

La cámara del robot por otro lado funciona, se podría decir, como un sistema aparte. Debido a que esta tiene su propio transmisor y receptor de imagen. Por lo que está también maneja su propia señal, individual a los sensores y movimientos del robot. Estas dos señales individuales, como ya se habían mencionado, son el motivo por el cual el software se tuvo que realizar de manera individual para ambos la transmisión de la cámara y las señales de la comunicación serial.

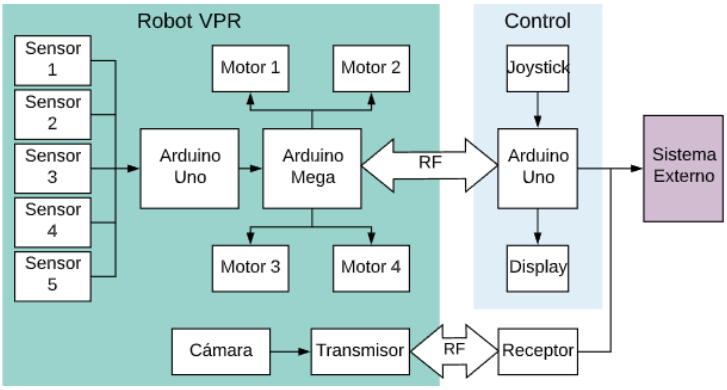


Ilustración 37- Esquema de funcionamiento del robot VPR

Fuente de la imagen: Elaboración propia

Sin embargo, el robot necesita un mantenimiento y una pequeña adaptación. Ambas cosas, el mantenimiento y adaptación, giran en torno al control. Ya que en primer lugar el control original no estaba al alcance. Ya que este se quedó en la universidad y debido a la situación surgida debido a la pandemia COVID-19 el acceso a la universidad es prácticamente imposible. Teniendo entonces que hacer uno desde cero.

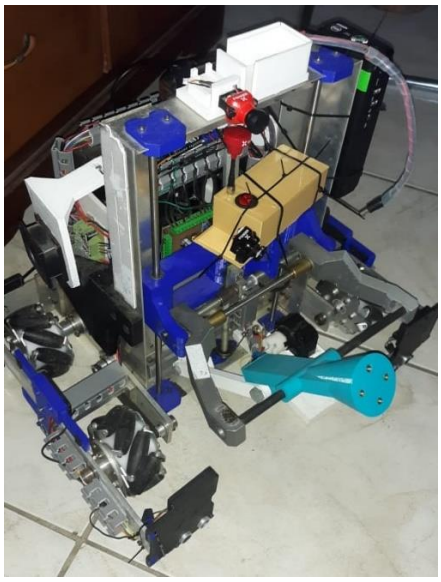


Ilustración 38- VPR

Fuente de la imagen: Elaboración propio

5.2.1 MANTENIMIENTO

Como se mencionó anteriormente el principal objetivo del mantenimiento hacia el robot es el control. Debido a la pandemia se hizo muy complicado poder conseguir un módulo de radio frecuencia que funcione con dos canales. Por lo que se utilizó los materiales que se tenían al alcance. En este caso para una comunicación inalámbrica se utilizó un módulo de bluetooth. La ventaja de utilizar este módulo es que la lógica detrás del control es la misma. Siempre se puede mandar una señal con caracteres y recibir el mensaje codificado de los sensores de manera de no tener que modificar la programación de los Arduinos del robot.

Para la manipulación del robot a través del control se utilizó una palanca de mando. Esta funciona con dos potenciómetros los cuales varían su señal entre 0 y 1023. Sin embargo, solo interesa el movimiento como tal, no la velocidad, por lo que pasado un punto de esa señal solo se busca que el robot vaya hacia ese lado en su única velocidad. Otro detalle es que el robot no puede hacer movimientos diagonales debido a que sus motores solo van hacia adelante o hacia atrás. Lo que se puede hacer al querer doblar es hacer que el robot haga un movimiento rotativo ya sea horario o antihorario. Por lo que el robot se limita a 5 movimientos en total:

frontal, retroceso, giro horario, giro antihorario y detenerse. Utilizando entonces el módulo de bluetooth y la palanca de mando, el esquema del Arduino se vería de la siguiente manera.

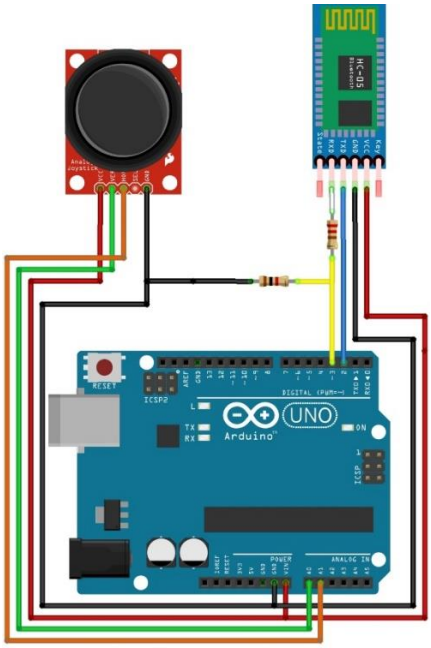


Ilustración 39- Esquema del control del robot

Fuente de la imagen: Elaboración propia

El Arduino tiene que convertir la señal de la palanca de mando en los caracteres que necesita el robot para saber hacia dónde moverse.

Tabla 9. Caracteres según el movimiento

Carácter	Movimiento
A	Hacia Adelante
B	Hacia Atrás
C	Giro Horario
D	Giro Antihorario
.	Detenerse

Fuente de la tabla: Elaboración propia

La palanca de mando funciona con dos distintos potenciómetros, uno para el eje X y uno para el eje Y. Ambos valores que devuelve la palanca están en un rango de 0 a 1023, donde en el eje X si el valor es 0 la palanca esta hacia la izquierda y en 1023 el caso contrario, en el caso del eje Y si la palanca esta hacia abajo el valor es 0 y 1023 en el caso contrario. Por lo que

cuando la palanca está en su estado natural los valores que devuelve son de 512 para ambos ejes.

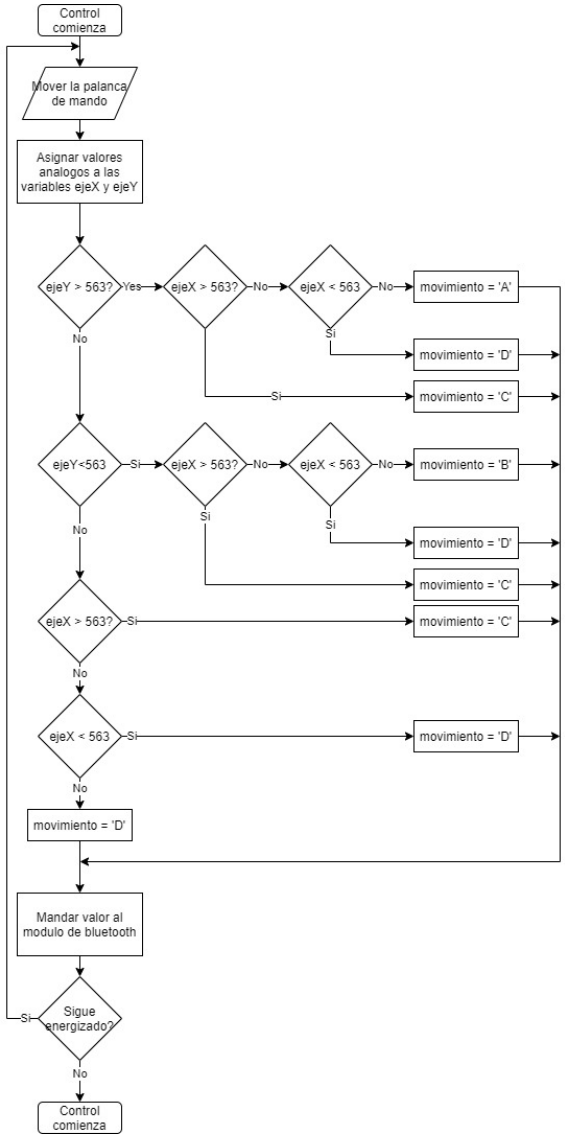


Ilustración 40- Diagrama de flujo del control

Fuente de la imagen: Elaboración propia

La interpretación que se le dará a estos valores entonces será que pasado cierto punto el sistema va a devolver un carácter, ya que se maneja una sola velocidad. Se va ajustar una condición que pasado un ±10% del punto natural del mando se interpreta como derecha, izquierda, arriba o debajo para el eje X y el eje Y respetivamente. Otro punto a considerar es que la palanca se puede poner en forma diagonal, pero el robot no tiene este movimiento. Lo que se hará para manejar esta situación es darle prioridad al giro, ya sea horario o antihorario, debido a

que este es un movimiento más crítico que los movimientos rectilíneos. Por lo que el Arduino UNO del Robot seguirá el algoritmo planteado en la ilustración 38.

5.2.2 Adaptaciones

En esta sección se presentará las adaptaciones que se le hicieron al robot para poder adaptarlo a la red neuronal. Estas adaptaciones, al igual que el mantenimiento, se enfocarán en el control. Como se vio en el diagrama de flujo del código del control los caracteres del movimiento dependen directamente de los valores procesados de la palanca de mando. Sin embargo, la red neuronal se va a entrenar de manera de devolver la señal ya codificada como el respectivo carácter que prediga.

El esquema de las conexiones del Arduino para las adaptaciones va a ser el mismo que el de mantenimiento, con la pequeña diferencia de no tomar en cuenta la palanca de mando. Y el código se basará en leer la información que devuelve el puerto serial y pasar esa misma información al módulo de bluetooth. El Arduino del control ahora se podría ver nada más como un intermediario entre la computadora, que estará ejecutando la red neuronal, y el Arduino Mega del robot.

5.2.3 Entrenamiento

En esta parte se estará haciendo uso del software entrenador. Este entrenamiento es en realidad la recolección de toda la información necesaria para el entrenamiento como tal de la red neuronal. La recolección de la información es probablemente la parte más importante de un algoritmo de aprendizaje automático en general. Se debe de recolectar la suficiente información para poder obtener los mejores resultados para la red neuronal, de manera de conseguir un conjunto de datos de entrenamiento, validación y prueba, dividiendo el total de datos en 70%, 15% y 15%, respectivamente. El robot se entrenará para estar recorriendo en una casa, pero con técnicas de regularización y variaciones en las imágenes de manera de generalizar lo mejor que se pueda el modelo y así poder poner al robot a funcionar en diferentes lugares o pistas.

Para poder saber la frecuencia de captura del software es importante saber cuánto le tiempo le toma al robot dar una vuelta entera en la zona en la que se entrenará. De manera de

entrenar a la red con un aproximado de 2 vueltas completas. Por lo que se hará análisis de cuánto tiempo le toma llegar de un punto A, que sería el punto de inicio, hasta un punto B, que sería la parte más lejana de la pista.

Tabla 10. Análisis del tiempo de recorrido del robot a través de la pista de entrenamiento

No.	Tiempo por vuelta en segundos			
	Numero de vuelta	Tiempo (A→B)	Tiempo (B→A)	Tiempo (Total)
1.	Vuelta 1	132	169	301
2.	Vuelta 2	143	165	308
3.	Vuelta	125	170	295

Fuente de la tabla: Elaboración propia

Dado los valores obtenidos en la tabla 10, la cantidad de tiempo que se tarda, en total, el robot en dar una vuelta es de 300 segundos, a groso modo, ósea cinco minutos por vuelta. Suponiendo que se quiere dar dos vueltas el tiempo total de entrenamiento es de 10 min del robot caminando. Se va a configurar el software para que este tenga una frecuencia de captura de 20 imágenes por segundo, que en términos de tiempo es igual a intervalo de 50 milisegundos.

Dado entonces las 20 imágenes por segundo y los 600 segundos aproximados que se tarda el robot en dar dos vueltas la cantidad de información que se estará recolectando es de 12000 ejemplos. De manera de poder entrenar, validar y probar la red neuronal apropiadamente. El software sin embargo tuvo un problema, y es que debido a la gran cantidad de ejemplos se quedó sin memoria disponible. Por lo que entonces se recolecto la información en 12 partes iguales, cada parte siendo almacenada en 12 carpetas distintas, estas carpetas localizadas en la misma carpeta. También se configuro, como ya se ha mencionado, los archivos de manera de que las imágenes se guarden como un archivo JPG y el texto de la comunicación serial como un archivo CSV.

Obtenidos los 12000 ejemplos, se procedió configurar los archivos para que en el caso de las imágenes estén nombradas desde el 1 hasta el 12000, y en el caso de los archivos de texto tengan corregido el directorio de la imagen y luego unir todos los archivos CSV en uno solo.

Ilustración 41- Ejemplos de las imágenes

Fuente de la imagen: Elaboración propia

Para tener un la data como fuente abierta y fácil manejo en línea de la misma se creó un repositorio en la plataforma de GitHub. Esto va a permitir a poder utilizar herramientas como Google Colaboratory Notebooks para manipular y descargar la información en el código de la red neuronal. El repositorio se creó con el nombre de HomeTrack, en ingles debido a que este es el idioma que predomina en esta plataforma.

5.3 Segundo Ciclo: Red Neuronal

En esta sección se estará analizando el tipo de red neuronal que se utilizará, la arquitectura de esta misma, y el entrenamiento que se le dará a la red neuronal como tal.

5.3.1 Planificación: Segundo Ciclo

El primer paso para la parte principal de este proyecto, la red neuronal, es decidir qué tipo de red neuronal se estará utilizando y que modelo o arquitectura se usará de esta red neuronal. Los datos de entrada serán los pixeles de la imagen devuelta por la cámara y las salidas serán los cinco posibles movimientos que puede hacer el robot. El mejor tipo de red neuronal para el procesamiento de imagen y patrones son las redes neuronales convolucionales, (Valueva et al., 2020).

Definido entonces el tipo de red neuronal que se utilizará se definirá el modelo a usar. El modelo usado para la presente tesis es uno basado en la investigación presentada por Bojarski et al. (2016), titulada "End to end learning for the self-driving car." Este modelo fue diseñado para mapear pixeles en bruto desde una cámara para predecir un ángulo de dirección, (Bojarski et al., 2016). Sin embargo, este modelo se utiliza para medir valores regresivos, y el tipo de información a medir en esta investigación es de tipo discreta, o clasificatoria.

El modelo consta de 1 capa de entrada, 5 capas convolucionales, 3 capas completamente conectadas y una capa de salida. La capa de entrada se configurará para recibir imágenes con dimensionalidad de 66×200 con un color de pixeles YUV, por lo que la forma de entrada de la red neuronal es de $66 \times 200 \times 3$. Las capas convolucionales se dividen en dos partes debido al tamaño del núcleo, las primeras tres capas que tendrán un núcleo de 5×5 con una profundidad de 24, 26 y 48, respectivamente, y las últimas dos capas tendrán un núcleo de 3×3 con una profundidad de 64 para ambas capas.

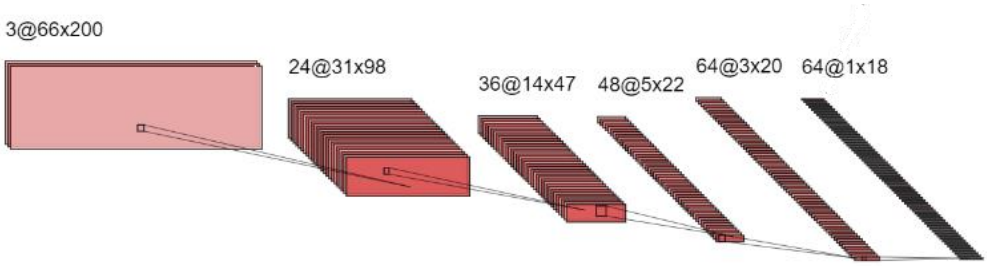


Ilustración 42- Arquitectura de las capas convolucionales

Fuente de la imagen: Elaboración propia

Cuando toda la información haya sido procesada por las capas convolucionales la última capa se deberá de pasar por un proceso de aplanamiento, de manera de juntar todos los parámetros y hacer una sola columna de neuronas con estos. Debido a los parámetros puestos esta capa sería de 1152 parámetros. Las tres capas conectadas completamente tendrán 800, 400 y 100 parámetros, respectivamente.

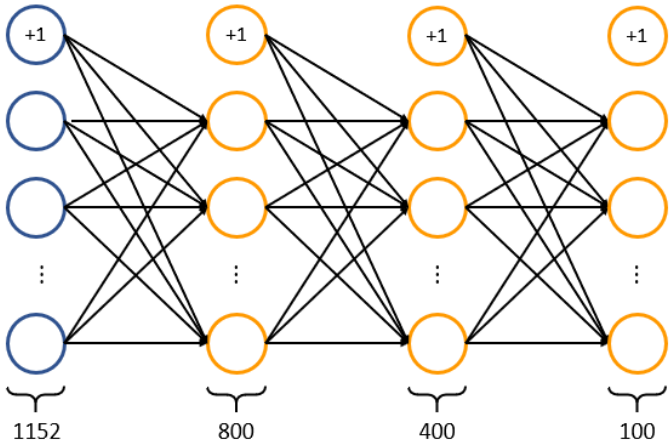


Ilustración 43- Arquitectura de las capas completamente conectadas

Fuente de la imagen: Elaboración propia

Una vez este proceso termine las capas completamente conectadas deben de devolver el vector predicho por todo el modelo como tal.

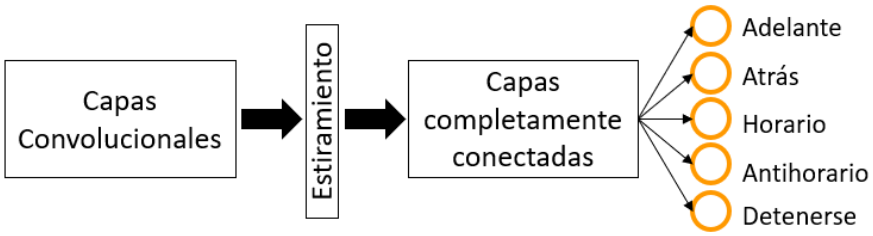


Ilustración 44- Diseño final del modelo

Fuente de la imagen: Elaboración propia

Tabla 11. Descripción del modelo

Capa	**Forma de salida**	**Parámetros**
Entrada	(66, 200, 3)	0
Convolucional	(31, 98, 24)	1824
Convolucional	(14, 47, 48)	21636
Convolucional	(5, 22, 48)	43248
Convolucional	(3, 20, 64)	27712
Convolucional	(1, 18, 64)	36928
Estiramiento	1552	0
Completamente Conectada	800	922400
Completamente Conectada	400	320400
Completamente Conectada	100	40100
Output	5	505
Total de Parámetros	1,414,753	

Fuente de la tabla: Elaboración propia

También se puede hacer un análisis numérico a este modelo de manera de poder analizar la cantidad de parámetros que se van a estar manejando y las figuras que tendrán cada una de las capas (véase la tabla 11).

5.3.2 Análisis de Riesgo: Segundo ciclo

Los datos brindados por el Arduino UNO, como se ha observado, son caracteres que están definidos en la tabla 9. Sin embargo, la clasificación de los valores funciona con valores enteros. Por lo esté vendría ser un riesgo al no tratarse. Más sin embargo la solución es sencilla, se le asignará un valor entre 0 al 4 a cada uno de los caracteres planteados en la tabla 9, la asignación será 1, 2, 3, 4 y 0, respectivamente.

El principal riesgo que se enfrenta cualquier algoritmo de aprendizaje automático es el de sobreajustar o subajustar la información. En el caso de la cantidad de información que se logró almacenar debería ser suficiente para no sobreajustar la información. Sin embargo, como cualquier tipo de vehículo la mayoría de tiempo pasa manejando de manera rectilínea, y son pocos los casos en el que el vehículo gira. Análisis que se puede observar en el siguiente histograma.

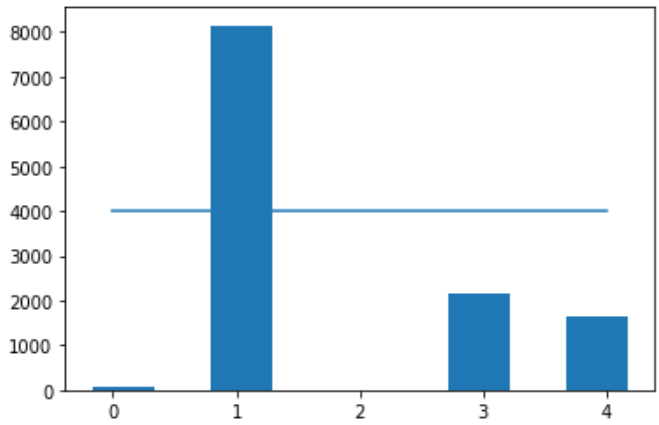


Ilustración 45- Histograma del total de datos

Como se puede observar en el histograma la mayoría de los valores son imágenes en la que el robot se conducía hacia el centro. Por lo que sistema puede llegar a predecir unos y ser uno de estos sistemas sesgados mencionados en el capítulo 3.

Otro detalle a tomar en cuenta son las modificaciones que se le necesita hacer a la imagen para poder generar diferentes casos al momento de estar entrenando la red. Al igual se necesita analizar que parte de la imagen es necesaria para la red. Y también que preprocesamiento se necesita para poder devolver el formato y dimensión que necesita en la entrada el modelo.

5.3.3 Implementación: Segundo Ciclo

Primero se debe de encarar el problema puesto en la sección anterior sobre los datos tendiendo siempre hacia el centro. La solución para esto será poner un límite. Ese límite será 4000 ejemplos por clase. Por lo que se pasara a la información por un sistema que aleatoriamente escoja valores de clases que estén por arriba del límite, y luego almacene esos índices para después quitar esos índices de los datos completos. Obteniendo entonces el siguiente histograma

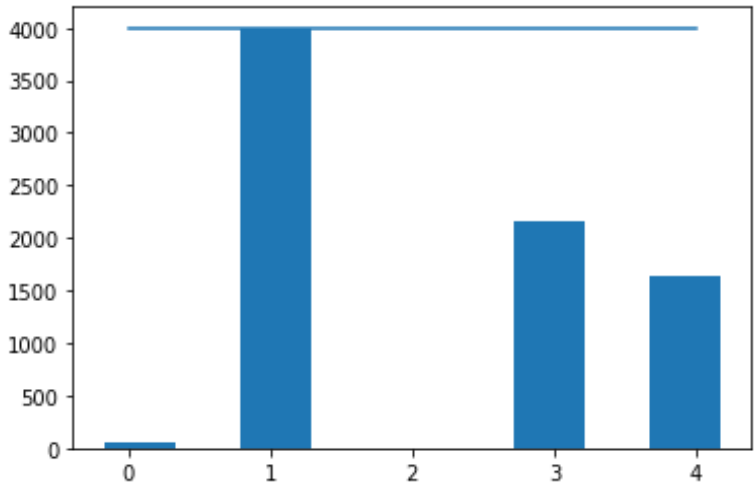


Ilustración 46- Histograma de los datos modificados

Fuente de la imagen: Elaboración propia

La cantidad de valores que se removieron para la clase de movimiento hacia adelante fueron de 4145, dejando así un total de data de 7855. A partir de esta información modificada se dividirá los conjuntos de entrenamiento, validación y prueba, con la proporción 70:15:15. Obteniendo el siguiente histograma para la división de los tres conjuntos.

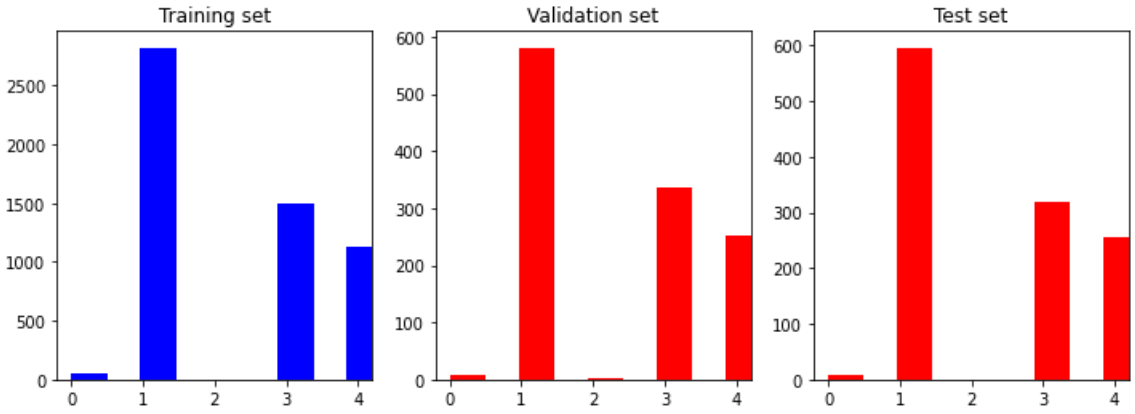


Ilustración 47- Histograma del conjunto de entrenamiento, validación y prueba

Fuente de la imagen: Elaboración propia

Como se planteó la forma de la entrada del modelo es de $66 \times 200 \times 3$, por lo que se necesita generar un sistema que convierta la imagen en esta dimensión y color. En la imagen se encuentran varios detalles que no son necesarios para el modelo. Como, por ejemplo, el techo u objetos que están muy lejos para aun ser analizados.

Ilustración 48- Imagen graficada

Fuente de la imagen: Elaboración propia

Como se puede observar en la ilustración de arriba la cámara captura el techo y algunos objetos a una relativa larga distancia. Por lo que se necesita reducir la visión en el eje horizontal.

Un corte desde el punto 150 al 380 parece ser una división congruente. Ya que en esta sección se encuentra lo que está próximo y al alcance del robot en el momento que se toma la captura. En el eje vertical, sin embargo, se necesita toda la información que se ve ya que provee objetos que se encuentran a relativa corta distancia.

Haciendo uso de la librería de OpenCV se cambiará el formato de la imagen pasando de un tipo de color RGB a uno YUV. Usando la misma librería se reformará la imagen para que tenga la forma 66×200. Después de esto se normalizarán los valores de los pixeles. De manera de pasar de tener valores entre 0 y 255 a tener valores entre 0 y 1. Esto va a ayudar mucho al algoritmo optimizador a trabajar más rápido.

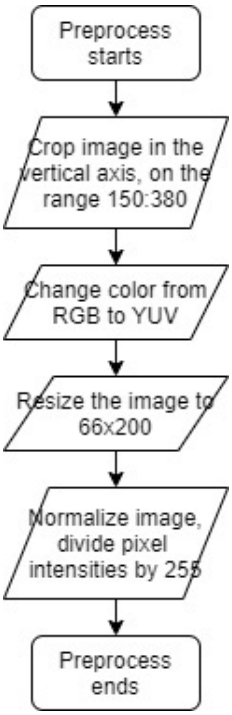


Ilustración 49- Diagrama de flujo de la función de preprocesamiento

Fuente de la imagen: Elaboración propia

Para poder generalizar el modelo se de poder generar, se podría decir, nuevos casos de la información ya obtenida. Para eso se necesita tener una función que aumente la información y varié ciertos parámetros, como el brillo, dar diferentes panoramas, girar la imagen y aplicar un acercamiento. Para estos procesos se estará haciendo uso de la librería Image Augmentation, imgaug, y la ya usada OpenCV.

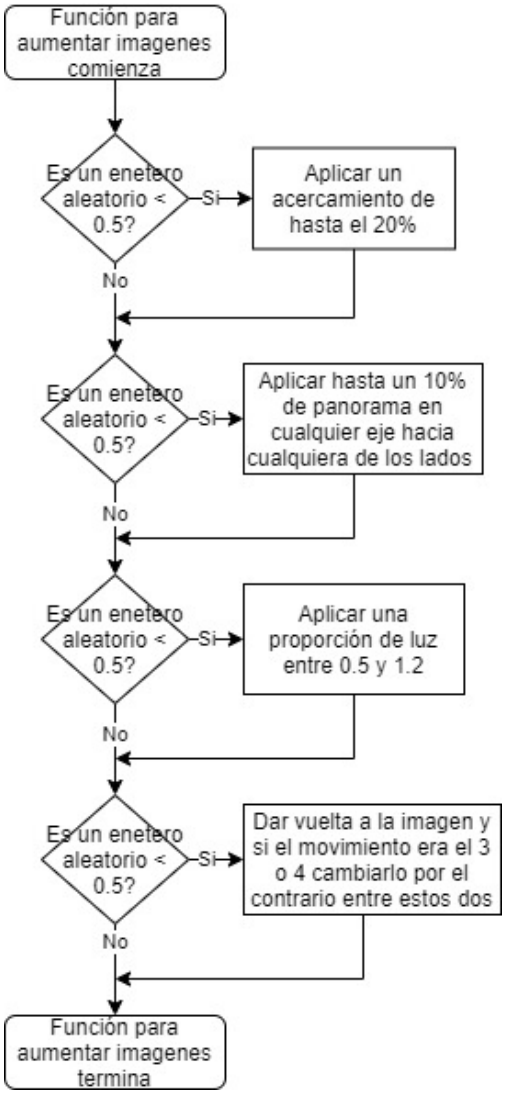


Ilustración 50- Diagrama de flujo de la función para aumentar imágenes

Finalmente, para aplicar estas dos funciones se necesita hacer una función que genere varios lotes debido a los directorios de las imágenes y los movimientos. Para esto se va a hacer uso de la librería de Matplotlib, una librería para graficar y procesar imágenes en Python. Esta librería permitirá utilizar la imagen según un el directorio proveído. También se hará uso de las últimas dos funciones ya planteadas. También para generar las categorías de las clases, ósea el vector de ceros y unos debido a la respectiva clase, se utilizará la librería de TensorFlow, Keras. Para esta función se debe de proveer la lista de los directorios de las imágenes, las clases de los movimientos, el tamaño deseado para el lote y un valor binario que representa si la información es del conjunto de entrenamiento o no.

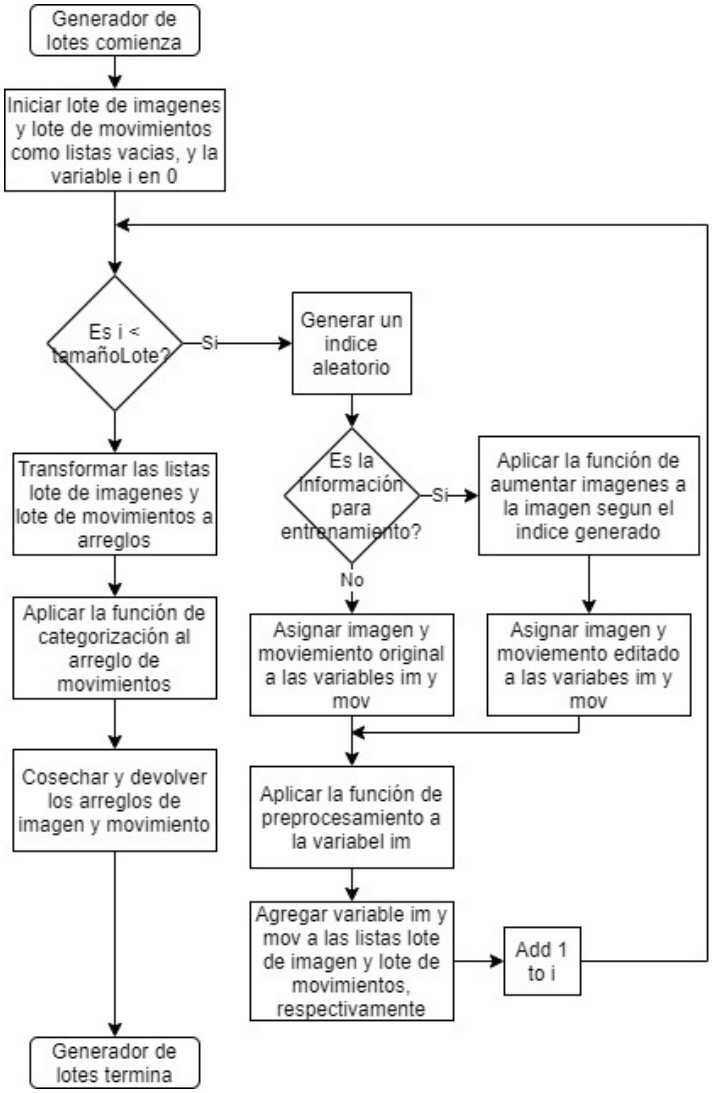


Ilustración 51- Diagrama de flujo del generador de lotes

Fuente de la imagen: Elaboración propia

Es crucial que esta función coseche la información. Esto lo que significa es que la función recuerda el ultimo valor que devolvió, permitiendo que cuando vuelva a correr la optimización a través del lote se tenga la última función que se generó. Ya que de lo contrario el optimizador estaría volviendo a empezar en cada iteración.

El algoritmo que se utilizó para optimizar la red neuronal es un algoritmo basado en el descenso por gradiente estocástico, Adam. El algoritmo se corrió con una razón de aprendizaje de 0.001, ya que esta razón es la que presenta mejores estadísticas al momento de optimizar conjuntos de datos en general, (Kingma & Ba, 2017). Se realizaron un total de 300 iteraciones

para poder optimizar el algoritmo de la mejor manera. Ya que pequeños cambios en su precisión pueden provocar tomas de decisiones en ciertos instantes que pueden causar un choque del, y esos sería muy caro. Por lo que se quiere evitar estos errores lo más que se pueda reduciendo la perdida y aumentando la precisión.

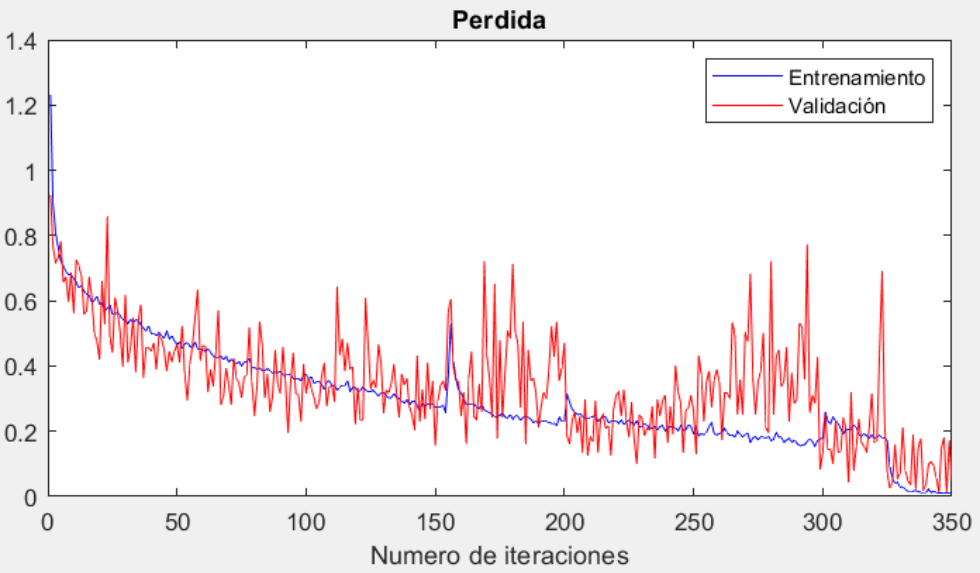


Ilustración 52- Perdida vs. numero de iteraciones del entrenamiento de la red neuronal

Fuente de la imagen: Elaboración propia

La perdida en el conjunto de entrenamiento tiene claramente una tendencia inversamente exponencial. Sin embargo, a mitad del entrenamiento ocurrió algo interesante y algo inusual. Fue un salto repentino desde un punto aproximadamente en 0.35 a uno aproximadamente en 0.55. Este salto ocurrió debido a que se encontró con un caso totalmente nuevo en ese punto. Y el punto de aumentar las imágenes es claro precisamente eso. Pero en ese punto se encontró una imagen seguramente modificada en los puntos más altos de cada variación, provocando una imagen totalmente nueva. Más sin embargo el algoritmo se logró adaptar rápidamente a este brusco cambio. La validación por otro lado parece tener variaciones muy pronunciadas entre iteraciones y por momentos diferencias muy altas con respecto a la perdida en el conjunto de entrenamiento. Este es una tendencia un poco alarmante, y para verificar si se necesita reentrenar el modelo se puede ver la gráfica de la precisión en cada iteración.

5.3.4 Evaluación: Segundo Ciclo

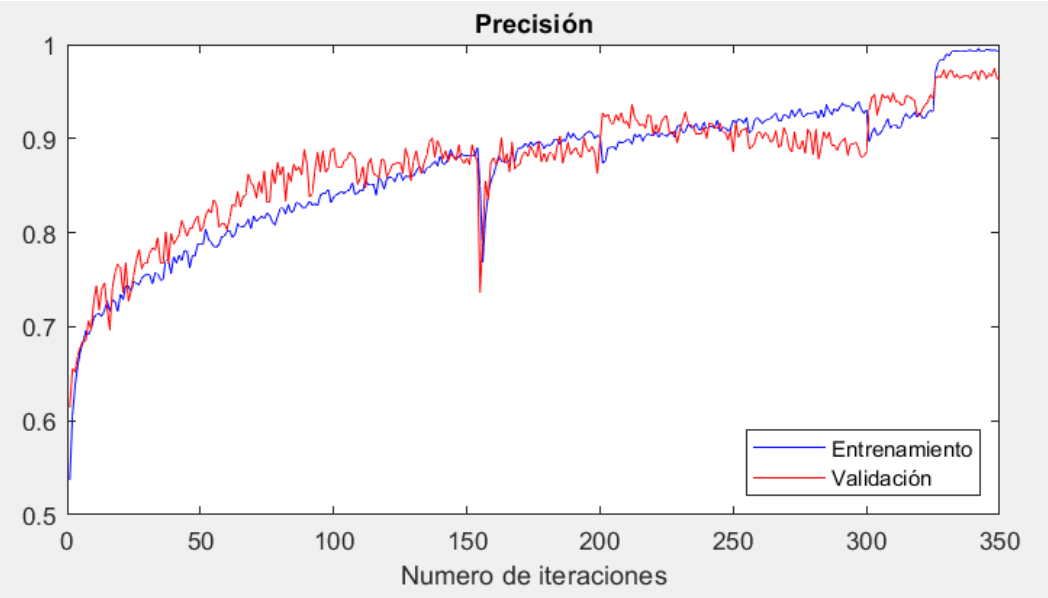


Ilustración 53- Precisión vs número de iteraciones de la red neuronal

Como se puede ver en esta grafica la precisión del conjunto de validación tiene una forma menos variada y más, se podría decir, centrada en un subir de manera exponencial, al igual que la precisión en el conjunto de entrenamiento. Esta pronunciada variación que tuvo el conjunto de validación en la perdida es debido al factor de lo que depende una perdida. El concepto de perdida es qué diferencia hay entre el valor original y el valor predicho. Y el valor original son arreglos como los que se presentan en la sección 3.5.4, que se basan en ceros y unos. El algoritmo sin embargo predice valores precisamente que sean ceros y unos, sino que predice valores entre 0 y 1 y si el numero predicho se acerca más a uno que al otro se predice al que más se acerca. En cambio, la precisión va por todo el conjunto y solo le importa que clase como tal eligió. Por lo que en este caso el algoritmo puede predecir valores relativamente altos pero que, al momento de transformarlos a una clase, son la clase que se tuvo que predecir. Para ver esto en valores numéricos se presentará un ejemplo. Se planteará que la clase a predecir es el siguiente vector.

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Ese vector representaría la clase 1, que en términos de la red utilizada en este proyecto sería la orden de moverse hacia adelante. Ahora se planteará que el valor que predice el algoritmo en algún punto intermedio del proceso de la optimización es el siguiente vector.

$$\begin{bmatrix} 0.1 \\ 0.85 \\ 0.23 \\ 0.1 \\ 0.001 \end{bmatrix}$$

Utilizando entonces como un punto límite para la decisión de la clase al 0.5, el 0.1 se consideraría un 0, el 0.85 un 1, el 0.23 un 0, el 0.1 un 0 y el 0.001 también un 0. Prediciendo uno en el segundo término, por ende, prediciendo la clase correcta. Sin embargo, la diferencia entre estos dos vectores es bastante pronunciada.

$$\begin{bmatrix} 0.1 \\ 0.85 \\ 0.23 \\ 0.1 \\ 0.001 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.1 \\ -0.15 \\ 0.23 \\ 0.1 \\ 0.001 \end{bmatrix}$$

La máxima precisión a la que se consigue llegar para el conjunto de entrenamiento es del 0.995 y para el conjunto de validación es de 0.984.

5.4 Tercer Ciclo: Implementación de la Red Neuronal en VPR

En este ciclo se pondrá a trabajar el modelo diseñado en la sección anterior para hacer que VPR se mueva de manera autónoma a través de una pista.

5.4.1 Planificación: Tercer Ciclo

El robot se estará probando en la pista en la que se entrenó con ciertas variancias en los obstáculos. De manera de que, a pesar de estar moviéndose en el mismo lugar, lo pueda identificar como una pista relativamente nueva. Así poder observar que tal reacciona el robot. Se espera que este pueda moverse bastante bien a través de la casa, pero de vez en cuando ocupando ayuda en vueltas muy pronunciadas.

5.4.2 Análisis de Riesgos: Tercer Ciclo

En esta sección el mayor cuidado que se debe de tener es evitar a toda costa que el robot choque. Ya que al dañarse alguna parte del robot puede llegar a ser muy delicado y complicado, debido a la situación global, conseguir reemplazos. Otro tema importante a considerar es como se hará que la cámara este actualizando a la red neuronal y que luego esta le devuelva el valor del movimiento que tiene que realizar.

5.4.3 Implementación: Tercer Ciclo

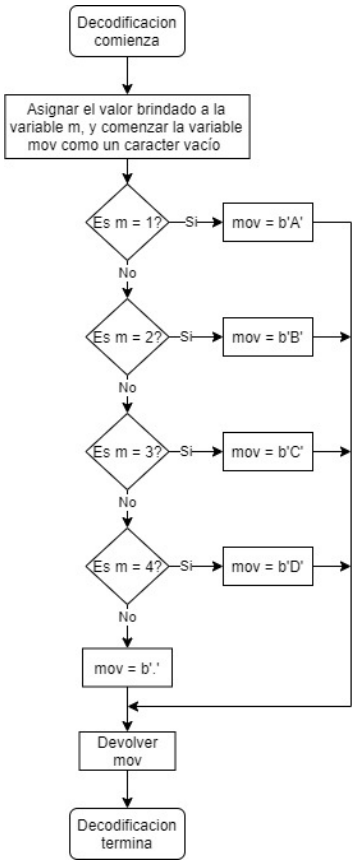


Ilustración 54- Diagrama de flujo del decodificador

Fuente de la imagen: Elaboración propia

Obtenido el archivo del modelo de la red neuronal es necesario ahora hacer un sistema que pueda recibir la señal de la cámara y que pueda enviar los caracteres al Arduino. En este punto el Arduino se tiene que regir por la lógica propuesta en la sección de adaptaciones de VPR. Tal programa se realizara con Python y se utilizará el símbolo del sistema, cmd, para correr

el programa. Pero previo a definir la lógica a seguir para poder implementar la red neuronal en el robot se debe definir la transformación de los valores devueltos por la red neuronal. Ya que esta devuelve enteros entre el 0 y 4, y hay que hacer una codificación inversa a la utilizada en la sección 5.3.2 para transformar estos valores a los caracteres 'A', 'B', 'C', 'D' y '.'. La comunicación serial funciona con formato Unicode, por lo que se tiene que transformar estos caracteres a este formato igual. En Python transformar un carácter a este Unicode es sencillo, solo se debe de poner una b al inicio. De esta manera el sistema sabe que debe leer esta variable como el conjunto de bits que le corresponden a la letra entre comillas.

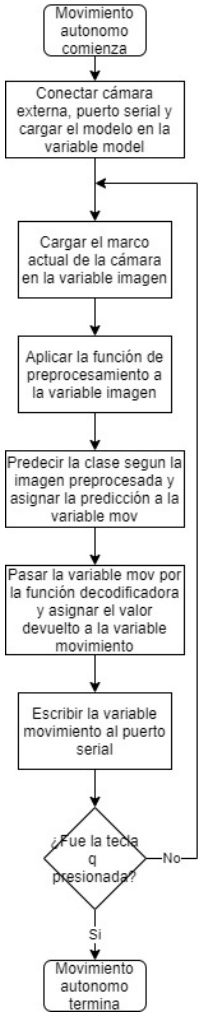


Ilustración 55- Diagrama de flujo de la función de movimiento autónomo

Fuente de la imagen: Elaboración propia

Con esta última herramienta definida (véase la ilustración 54), ya se tienen todas las herramientas definidas para poder hacer el código para el movimiento autónomo de VPR. Para poder hacer este código es necesario utilizar las librerías de comunicación serial, PySerial, de visión computacional, OpenCV, de aprendizaje automático, Keras, y la de manejo matemático, Numpy. Por lo que el código seguirá la lógica planteada en la ilustración 55.

Aplicado entonces el código el robot logro moverse de manera autónomo haciendo uso de la red neuronal. Logro darle la vuelta a la casa con los nuevos obstáculos puestos en su camino. Sin embargo, en algunas vueltas se necesitó regresar el robot debido a que estaba tendiendo a chocar. Pero al ponerlo un poco más atrás y someterlo a la misma situación logro predecir el movimiento necesario para no chocar.

5.4.4 Evaluación: Tercer Ciclo

Para evaluar el sistema se utilizara el error y precisión medido sobre el conjunto de entrenamiento. Para esto claro que hay que pasar la imagen por la función de preprocesamiento. Se puede hacer uso del generador de lotes, poniendo la variable que necesita saber si el lote es de entrenamiento como falso, ya que claro no lo es, y el tamaño del lote seria el arreglo completo. Dando un promedio de error del 0.0435 y una precisión de 98.732%.

VI. CONCLUSIONES

Debido a los análisis y resultados obtenidos en el capítulo anterior se logró contestar a los objetivos planteados en la sección 2.5. Pudiendo concluir que se logró realizar una red neuronal de tipo convolucional para obtener la autonomía del robot VPR. Logrando este modelo llegar a una precisión del 98.7%, permitiéndole al operador del robot tenerle una alta fiabilidad a este. De una manera más concreta y especifica se pueden plantear las siguientes conclusiones:

1. Fue necesario y se realizó un software que logrará adaptarse al robot para poder reunir la información necesaria para el entrenamiento de la red, esta información se consiguió desde una cámara y un puerto serial.
2. Se logro aplicar una configuración al control de manera de poder manipular al robot con la red neuronal, utilizando a este mismo como un intermediario entre la computadora y VPR.[AMCB7]
3. Se definió un modelo que se adaptará a la visión computacional como entrada para la predicción de valores de manera discreta.
4. Se utilizó el algoritmo de optimización de Adam, basado en el algoritmo de descenso por gradiente estocástico, con una razón de aprendizaje del 0.001.

VII. RECOMENDACIONES

En la presente investigación se logró efectivamente realizar una red neuronal para la autonomía de un robot. Haciendo uso de la teoría de sustento se logró aplicar los conceptos explicados en este capítulo. Lo más importante es que estas teorías nos permiten entender cómo funcionan estos algoritmos de aprendizaje automático, en especial las redes neuronales. Haciendo uso de regresiones logísticas se logró generar un sistema para la clasificación de varias clases, las cuales tienen un mismo fin, en el caso de este proyecto predecir el movimiento del robot VPR. Entendiendo también como funciona la perdida que hay entre las salidas predichas y las salidas reales. También se logró comprender el funcionamiento que tienen los algoritmos de optimización y el efecto que tienen estos sobre los parámetros que rigen cada una de las neuronas que hay dentro de estas redes.

Una forma de poder escalar este proyecto o de poder seguir mejorándolo seria probar distintos hiperparametros, como ser el numero de capas, las unidades por capas, la razón de aprendizaje etc. Esto de manera de encontrar si hay valores que puedan mejorar el rendimiento. Aplicar técnicas de aprendizaje transferido, de manera de obtener resultados más rápido en base a modelos ya funcionales, aunque sean para distintas tareas. Otra forma de mejorar es que en teoría las redes neuronales entre mas profundas mejor resultados proveen, pero en practica tiende a no ser así. Por lo que se puede aplicar redes residuales de manera de poder utilizar una red bastante profunda y poder, probablemente, observar mejores resultados.

Para futuros trabajos se recomienda poder utilizar esta red neuronal para ver si se puede diseñar una red que prediga valores regresivos. Con el propósito de predecir ángulos de giro mientras el robot está en movimiento, ósea movimientos diagonales, para diferentes robots. En la presente investigación no se utilizó este concepto debido a que el robot VPR se tiene que detener para poder hacer un giro y lo hace sobre su propio eje no sobre la línea de desplazamiento en la que va. Sería muy interesante también poder aplicarla para poder predecir diferentes velocidades y ver si esto puede ser una alternativa más precisa que un PID, o si estos dos conceptos, PID y redes neuronales, se pueden aplicar mutuamente para generar un sistema más completo.

VIII. REFERENCIAS

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning*. Springer.

Beitzel, S. M. (2006). *On Understanding and Classifying Web Queries*.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning* (1a ed.). Springer.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, *21*(5), 61–72. https://doi.org/10.1109/2.59

Boehm, B. W., & Ross, R. (1989). Theory-W software project management principles and examples. *IEEE Transactions on Software Engineering*, *15*(7), 902–916. https://doi.org/10.1109/32.29489

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). End to End Learning for Self-Driving Cars. *ArXiv:1604.07316 [Cs]*. http://arxiv.org/abs/1604.07316

Bühlmann, P., & Geer, S. van de. (2011). *Statistic for High-Dimensional Data: Methods, Theory and Applications*. Springer.

Byliskii, Z. (2015). *Generalized Linear Regression with Regularization*. MIT Press.

Caine, R. N., & Caine, G. (2011). *Natural Learning for a Connected World: Education, Technology, and the Human Brain*. Teachers College Press. https://books.google.hn/books?id=mhxhAgAAQBAJ

Cauchy, A. L. (1847). M´ethode g´en´erale pour la r´esolution des syst`emes d’´equations simultan´ees. *C. R. Acad. Sci.*, 536–538.

Cicco, M. (2020, mayo). *Oportunidad en los modelos de operaciones outsourcing* [Logísitica Énfasis].

Corcos, D. (2011). *El modelo esprial*. Instituto Tecnologico de Buenos Aires.

Dávila, G. G., & Dávila, M. C. G. (2000). *Metodología de la Investigación*. Grupo Editorial Patria.

Davis, J., & Goadrich, M. (2006). The Relationship between Precision-Recall and ROC Curves. *Proceedings of the 23rd International Conference on Machine Learning*, 233–240. https://doi.org/10.1145/1143844.1143874

Delgado-Rodríguez, M., & Llorca, J. (2020). *Bias*. *10*, 635–641.

Du, K. L., & Swamy, M. N. S. (2013). *Neural Networks and Statistical Learning*. Springer London. https://books.google.hn/books?id=wzK8BAAAQBAJ

El modelo de desarrollo en espiral como mezcla de cascada e iterativo. (2019, abril 5). ASPgems. https://aspgems.com/metodologia-de-desarrollo-de-software-iii-modelo-en-espiral/

Fariño, G. (2011). *Modelo Espiral de un proyecto de desarrollo de software*. 9.

Flach, P. (2012). *Machine Learning: The Art and Science of Algorithms that Make Sense of Data* (1a ed.). Cambridge University Press.

Flach, P., & Kull, M. (2015). Precision-Recall-Gain Curves: PR Analysis Done Right. En C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 28* (pp. 838–846). Curran Associates, Inc. http://papers.nips.cc/paper/5867-precision-recall-gain-curves-pr-analysis-done-right.pdf

Genesis. (2018). *Stochastic Gradient Descent*. *7*, 1.

Hinton, G., & Sejnowski, T. J. (1999). *Unsupervised Learning: Foundations of Neural Computation* (1a ed., Vol. 1). MIT Press.

Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied Logistic Regression* (3a ed.). Wiley-Interscience.

Jimenez, H. (2020). *Sistema electrónico para el Control de un robot teleoperado dedicado a la monitorización de las fincas de café* [Proyecto de Investigación]. Universidad Tecnologica Centroamericana.

Jr., G. B. T. (2014). *Thomas' Calculus: Multivariable* (13a ed., Vol. 2). Pearson.

Kelleher, J. D., Namee, B. M., & D'Arcy, A. (2015). *Machine Learning for Predictive Data Analytics* (1a ed., Vol. 1). MIT Press.

Kingma, D. P., & Ba, J. (2017). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*. http://arxiv.org/abs/1412.6980

Kleinbaum, D. G., & Klein, M. (2002). *Logistic Regression: A Self-Learning Text* (2a ed.). Springer.

Livingston, G. (2014). *Digital Life in 2025* [Comunicación personal].

Logística, É. (2020, mayo 6). *Covid-19: La importancia de la robótica en la cadena de suministros*. 2.

McCulloch, W. S., & Pitts, W. H. (1943). *A Logical Calculus of the Ideas Immanent in Nervous Activity*. *5*, 115–133.

Métin, C., & Frost, D. O. (1989). Visual responses of neurons in somatosensory cortex of hamsters with experimentally induced retinal projections to somatosensory thalamus. *Proceedings of the National Academy of Sciences*, *86*(1), 357. https://doi.org/10.1073/pnas.86.1.357

Mitchell, T. M. (1997). Does Machine Learning Really Work? *AI Magazine*, *18*(3), 11–20.

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). *Foundations of Machine Learning*. MIT Press.

Narendra, K. S., & Parthasarathy, K. (1990). Identification and Control of Dynamical Systems Using Neural Networks. *IEEE Transaction on Neural Networks*, *1*, 4–27.

NG, A. (2011, agosto 15). *Machine Learning* [Teaching a skill]. Machine Learning. https://www.coursera.org/learn/machine-learning

Powers, D. M. (2011). *Evaluation: From Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation*. *2*(1), 63.

Reddy, R. (2006, mayo). *Robotics and Intelligent Systems in Support of Society*. *21*(3), 24–31.

Ricamato, M. T., Marrocco, C., & Tortorella, F. (2008). MCS-based balancing techniques for skewed classes: An empirical comparison. *2008 19th International Conference on Pattern Recognition*, 1–4. https://doi.org/10.1109/ICPR.2008.4761359

Roe, A., Pallas, S., Kwon, Y., & Sur, M. (1992). Visual projections routed to the auditory pathway in ferrets: Receptive fields of visual neurons in primary auditory cortex. *The Journal of Neuroscience*, *12*(9), 3651. https://doi.org/10.1523/JNEUROSCI.12-09-03651.1992

Rosenblatt, F. (1961). *Principle of Neurodynimcs*. Cornell Aeronautical Laboratory, Inc.

Russell, S. J., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3a ed.). Prentice Hall.

Samuel, A. L. (1959). *Some Studies in Machine Learning Using the Game of Checkers*. 210–299.

Seber, G., & Lee, A. J. (2003). *Linear Regression Analysis* (2a ed.). Wiley-Interscience.

Smith, A., & Anderson, J. (2014). *AI, Robotics, and the Future Jobs*.

Specht, D. F. (1991). A General Gregression Neural Network. *IEEE Transaction on Neural Networks*, *2*, 568–576.

Stone, C. L. (2004). *The Basics of Biology*. Greenwood Press. https://books.google.hn/books?id=p8r7rfsmWNUC

Taguchi, G. (1974). *A New Statistical Analysis Method for Clinical Data, the Accumulation Analysis, in Contrast with the Chisquare Test. 29*, 806–813.

Tommiska, M. T. (2003). Efficient digital implementation of the sigmoid function for reprogrammable logic. *IEE Proceedings*, *150*, 403–411.

Turing, A. M. (1950). *Computing Machinery and Intelligence*. *49*, 433–460.

Valueva, M. V., Nagornov, N. N., Lyakhov, P. A., Valuev, G. V., & Chervyakov, N. I. (2020). Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, *177*, 232–243. https://doi.org/10.1016/j.matcom.2020.04.031

von Bartheld, C. S., Bahney, J., & Herculano-Houzel, S. (2016). The Search for True Numbers of Neurons and Glial Cells in the Human Brain: A Review of 150 Years of Cell

Counting. *The Journal of comparative neurology*, *524*(18), 3865–3895. https://doi.org/10.1002/cne.24040

Wikipedia. (2020, de abril de). *Overfitting* [Enciclopedia]. Wikipedia. https://en.wikipedia.org/wiki/Overfitting#cite_note-1

Wisskrichen, G., Biacabe, B., Bormann, U., Muntz, A., Niehaus, G., Soler, G., & von Brauchitsch, B. (2017). *Artificial Intelligence and Robotics and Their Impact on the Workplace*. International Bar Association.