



Semáforo Inteligente utilizando el Perceptrón de Rosenblatt

Sistemas Basados en Redes Neuronales

**Sánchez Patiño Natalia
Rosas Otero Mario**

Profesor: Dr David Tinoco Varela

Licenciatura en Tecnología

Facultad de Estudios Superiores Cuautitlán
Universidad Nacional Autónoma de México

19 - 11 - 2020

-Proyecto para Sistemas Basados en Redes Neuronales: Aplicación a un Perceptrón.-

Resumen

Semáforo Inteligente utilizando el Perceptrón de Rosenblatt. Este proyecto se realizó para construir desde su forma más básica, la unidad de cálculo elemental de las redes neuronales, conocida como perceptrón. Realizando una implementación que involucra tanto hardware como software, así como la comunicación entre elementos que conforman al sistema, con el objetivo de crear una solución a pequeña escala pero que pueda ser transformada a la solución de una situación cotidiana. Es así, como se decidió crear un sistema capaz de distinguir entre personas, en nuestro figuras humanas de plástico, y autos a escala de juguete, pero con el detalle suficiente para poder representar la forma real de estas entidades. En el proyecto se utilizaron métodos de pre-procesamiento de imágenes para poder obtener información que fuese clasificable para un perceptrón, obteniendo la respuesta dada por el mismo, se envía la señal a un microcontrolador, para poder ejercer el control de un semáforo para vehículos y uno para peatones. Demostrando la importancia del análisis de datos para poder comprender de forma computacional la variación de magnitudes físicas en un entorno, el como algunos algoritmos pertenecientes al área de inteligencia artificial no pueden ayudar a almacenar conocimiento de como dar una respuesta ante dichas magnitudes, por último, como mediante el enlace de los elementos mencionados con componentes electrónicos, nos permite la eficiente obtención de datos, así como reflejar el procesamiento, ejerciendo control sobre un sistema físico. Es posible obtener el código total del proyecto en: <https://github.com/NM-Labs/Semaforo-Inteligente-Con-Perceptron>

Índice

1	Introducción	1
1.1	Motivación	1
1.2	Objetivos	1
1.3	Solución Propuesta	2
2	Marco teórico	3
2.1	Perceptrón de Rosenblatt	3
2.2	Funcionamiento del perceptrón	4
3	Elementos Utilizados y Proceso de Construcción	6
3.1	Hardware	6
3.2	Software	6
3.3	Funcionamiento del Sistema	7
3.3.1	Adquisición y Almacenamiento de Datos	7
3.3.2	Pre-Procesamiento de Datos	8
3.3.3	Entrenamiento de la Red	8
3.3.4	Validación de Entrenamiento	9
3.3.5	Pruebas de Funcionamiento	10
3.3.6	Integración MATLAB - C++ - Arduino	10
3.3.7	Simulación del sistema	10
4	Resultados obtenidos	12
4.1	Binarización de imágenes	12
4.1.1	Experimentos de Clasificación	12
4.1.2	Experimento 1	13

4.1.3	Experimento 2	14
4.2	Experimentación en tiempo real	15
5	Conclusiones	18
5.1	Posibles extensiones del proyecto	19
5.2	Comentarios Finales	19
Appendix A	Código de Pre-procesamiento	21
Appendix B	Código del Perceptrón	26
Appendix C	Integración Matlab - C++ - Arduino	36
Referencias		39

Introducción

El problema escogido implica construir un semáforo inteligente, que sea capaz de ceder el paso a humanos o coche conforme una cámara los vea, para que la imagen sea preprocesada, evaluada por el perceptrón y finalmente se produzca una respuesta que se reflejara en el cambio de luz del semáforo.

1.1 | Motivación

Con el crecimiento de áreas como Inteligencia Artificial con Redes Neuronales, Big Data, Internet de las Cosas, entre otras; se ha popularizado el uso de sistemas computacionales para control automático de espacios comunes, tal como los semáforos en las calles. Si se consigue, trasladar las nociones básicas planteadas en este proyecto, a un sistema utilizable en calles reales, donde sera necesario tomar en cuenta mas parámetros de variación dados en las condiciones de dicha situación, podríamos optimizar los tiempos de paso de acuerdo a la afluencia entre carros y personas, mejorando así, la eficiencia de movilidad en la ciudad.

1.2 | Objetivos

■ Objetivo general:

Construir de forma teórica y práctica un sistema completo, hardware-software, para la creación de un semáforo inteligente utilizando como unidad de procesamiento un perceptrón.

- Realizar una revisión acerca de la conexión entre software y hardware en un sistema.

- Buscar la aplicabilidad de un perceptrón, como mecanismo de toma de decisión a partir de lectura de información en tiempo real.
- Comprender de manera teórica y práctica, mediante un lenguaje de programación, el funcionamiento del perceptrón.
- Desarrollar habilidades de programación en el lenguaje C++, así como la interconexión de funcionamiento entre lenguajes de programación.
- Comprobar fuera del ámbito virtual el correcto funcionamiento del sistema creado, bajo condiciones controladas y su replicabilidad.
- Comprobar de manera aplicada los conceptos vistos en clase.

1.3 | Solución Propuesta

Para poder ejercer control sobre un semáforo de forma automática, se propone la adquisición de imágenes en tiempo real desde la ubicación del semáforo, que será analizada cada cierto periodo de tiempo, el análisis de dicha imagen, consiste en la transformación a un distinto espacio de color, para obtener una imagen en escala de grises con los objetos a clasificar resaltados, dicha imagen en escala de grises es analizada mediante un algoritmo para poder obtener un umbral con el cual realizar una binarización de dicha imagen, es decir, mandar el objeto de interés a blanco, y todo lo demás a negro, hasta aquí todo el procedimiento es realizado a través de MATLAB. La imagen binarizada es analizada por el perceptrón creado en C++, el cual emite una respuesta respuesta, dando la clasificación obtenida, es decir, va a indicar si es humano o auto, dicha respuesta es mandada a la placa de arduino para ejercer un cambio de luces, que son las pertenecientes al semáforo.

Marco teórico

Un perceptrón es la unidad básica de una red neuronal y consiste en una neurona con pesos sinápticos y un bias que son ajustables y puede ser usado para la clasificación en problemas linealmente separables.

2.1 | Perceptrón de Rosenblatt

El algoritmo usado para ajustar los parámetros libres de esta red neuronal apareció por primera vez en un procedimiento de aprendizaje desarrollado por Rosenblatt (1958) para su modelo de perceptrón del cerebro. Dicho perceptrón está limitado a realizar clasificación de patrones con sólo dos clases. Expandiendo la capa de salida del perceptrón para incluir más que una neurona, podemos realizar dicha clasificación con más de dos clases. Sin embargo, las clases tendrían que ser linealmente separables para que el perceptrón trabaje correctamente.

En la figura 2.1 los pesos sinápticos se denotan por W_1, W_2, \dots, W_n . y la información que entra al perceptrón son los píxeles de la imagen representados como P_1, P_2, \dots, P_n , y donde b es el sesgo o umbral que se añade al proceso. Después de realizar la operación de suma de la multiplicación entre pesos sinápticos y entradas, para posteriormente sumar el sesgo y obtener una salida, dicha salida sera evaluada por una función de activación, en este caso una función escalón de Heaviside, el cual establece un límite, para determinar si la imagen contiene una determinada clase.

El aprendizaje de este algoritmo se basa en el cálculo del error calculado de la diferencia entre la salida esperada y_e con respecto a la salida obtenida y_i , entonces el error es

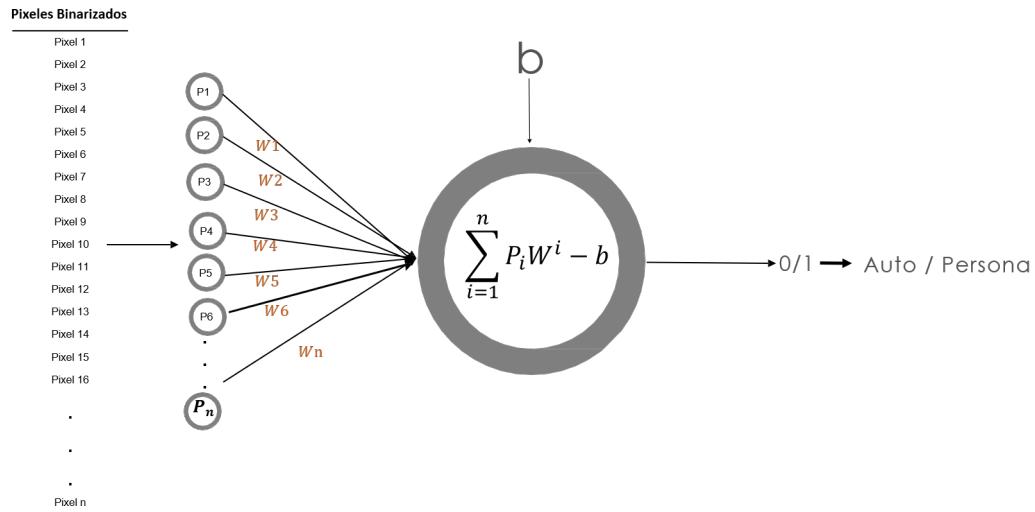


Figura 2.1: Representación de un perceptrón para clasificación de imágenes

calculado como

$$y_e - y_i$$

A partir de este se van a actualizar los pesos sinápticos y el sesgo, intentando adaptarse a las entradas ingresadas e intentando buscar una configuración de los mismo con los que se puede obtener una salida esperada para todos los elementos, dentro de las operaciones de actualización interviene también un valor conocido como factor de aprendizaje, o tasa de aprendizaje denotado como λ , este factor regula la cantidad de corrección de error para no divergir en la respuesta al intentar corregir. [Stephen \(1990\)](#) A continuación se muestran las ecuaciones utilizadas para actualización de sesgos y pesos.

$$W_n[t + 1] = W_n[t] + \lambda(y_e - y_i)P_n$$

$$b[t + 1] = b[t] + \lambda(y_e - y_i)$$

2.2 | Funcionamiento del perceptrón

■ Parámetros:

- Vector de entradas: $p = [p_1, p_2, \dots, p_n]$
- Vector de pesos: $W = [W_1, W_2, \dots, W_n]$
- Umbral/bias: b
- Respuesta actual: y

- Respuesta deseada: d
- Tasa de aprendizaje: λ

- **Inicialización:** Hacemos $W(0) = 0$ y se llevan a cabo los siguientes cálculos en los instantes de tiempo $t = 1, 2, \dots$
- **Activación:** En el instante de tiempo n se activa el perceptrón aplicando el vector de entrada p y la respuesta deseada d .
- **Cálculo de la respuesta actual:** Se calcula la respuesta actual del perceptrón mediante la función de activación

$$y(t) = \text{activ}[W^T p - b]$$

en nuestro caso activ es la función escalón de Heaviside.

- **Adaptación del vector de pesos y bias:** Se actualiza el vector de pesos del perceptrón:

$$W(t) = W(t) + \lambda[d(t) - y(t)]p(t)$$

Mientras que para el bias:

$$b(t) = b(t) - \lambda[d(t) - y(t)]$$

- Este proceso se repite con todos los pesos y entradas del problema.

Elementos Utilizados y Proceso de Construcción

Para la creación de nuestro proyecto se utilizó un conjunto de componentes tanto de software como de hardware, lo que nos permite hacer una simulación del funcionamiento del sistema de forma general. Es decir parte de los elementos utilizados son componentes del funcionamiento propio del sistema, y la otra parte, sirve solo para realizar una simulación de funcionamiento, y puesta a prueba de experimentación.

3.1 | Hardware

- Placa de desarrollo arduino.
- Componentes electrónicos: LEDs, cables y resistencias.
- Cámara web o cámara de un celular.
- Figuras de prueba, autos y persona. (Juguetes)
- Escenario o fondo blanco/liso.

3.2 | Software

- Sistema Operativo Microsoft Windows 10
- MATLAB R2020b - academic use
- CLion 2020.2 por JetBrains con cuenta de uso académico

- Compilador MinGW
- Aplicación IP Webcam

3.3 | Funcionamiento del Sistema

3.3.1 | Adquisición y Almacenamiento de Datos

Para comenzar la construcción del sistema, después de tener todos los elementos necesarios tanto de software como de hardware listos y configurados para utilizarse, aclarando que el proyecto es perfectamente realizable con algunas otras versiones del software especificado. Se requiere obtener un conjunto de datos representativo del problema, por lo cual se procede a colocar un fondo blanco y los elementos a utilizar en el, por separado, para poder tomar fotografías a los objetos a clasificar, cuidando que no existan tantas variaciones de ángulo, distancia e iluminación entre las fotografías tomadas, para mantener controladas las condiciones del problema. Siendo además importante tomar la misma cantidad de imágenes para el total de categorías a clasificar, en este caso misma cantidad de imágenes de autos y personas.



Figura 3.1: Representación de coches y personas.

Del total de imágenes tomadas a los elementos, se toma un porcentaje entre el 15% – 30% que serán tomadas como imágenes para validación, y el resto serán imágenes para entrenamiento, verificando que haya variedad de las mismas respecto a la variación de posición entre los elementos utilizados para ambas clases. Al momento de realizar el almacenamiento es recomendable hacer carpetas distintas para entrenamiento y prueba, así como dentro de estas, almacenar en carpetas distintas las imágenes de categorías a clasificar distintas.

3.3.2 | Pre-Procesamiento de Datos

Con el fin de simplificar la información contenida en una imagen y que pueda ser clasificable por un perceptrón se realizan métodos de pre-procesamiento digital de imágenes, en este procedimiento lo primero que se realiza es cambiar las imágenes a otro espacio de color, en este caso la imágenes cuando se ingresan están dadas construidas en el espacio de color RGB, pero estas como primer paso sufren una transformación hacia el espacio CMYK, que tiene cuatro capas. Despues de realizar la transformación se toma solo la capa Y , que resalta un rango de tonalidades en específico. Posteriormente dicha imagen en la capa Y , se pasa por una función de contraste que resalta aun mas ciertos rangos de valor en la imagen, en esta caso se utiliza una función cúbica para realizar la mejora de contraste. Los píxeles P_1, P_2, \dots, P_n son utilizados para la operación

$$\frac{P_n^3}{256^2}$$

Esto dado que le rango de valores de píxeles esta en una escala de 0 a 255. Posterior a esto se la misma imagen ya con mejoras de contraste se pasan los píxeles de la imagen por el algoritmo de Otsu, el cual busca en los valores de píxeles el valor de umbral que minimice la varianza σ^2 entre los mismos, el cual va a dar un umbral para realizar la binarización de la imagen. El proceso del algoritmo requiere de obtener el histograma de la imagen, y obtener las probabilidades de cada nivel de intensidad de los píxeles, se establecen pesos iniciales $W_n(0)$ o probabilidades, y se obtiene así mismo las medias entre clases $\mu_i(0)$, con estos datos para cada nivel de intensidad se itera para obtener las varianzas entre clase, y despues de tener todas las varianzas, toma la mayor varianza entre dos clases para establecer el umbral de binarización. Posterior a este procedimiento se utilizan algunos Momentos de Hu, que describen la manera en que se distribuyen los píxeles de un objeto sobre el plano de una imagen, estos momentos son invariantes a las transformaciones geométricas como traslación, rotación y escalamiento. Estos momentos se calculan para eliminar ruido en las imágenes binarizadas, dado que se discriminan objetos que hayan superado el umbral establecido por el algoritmo de Otsu, discriminándolos por algunas propiedades como el área. El código de pre-procesamiento

3.3.3 | Entrenamiento de la Red

Para poder ingresar las imágenes al perceptrón estas tienen que ser re-dimensionadas, ya que se tienen como matrices, pero deben ser convertidas a vectores, esto se hace poniendo cada fila de la matriz una tras otra. Realizado el pre-procesamiento a todas las imágenes de entrenamiento se forma una matriz de características, de decir se forma

una matriz con todas las imágenes de entrenamiento ya re-dimensionadas como vectores , es decir imágenes donde la cada fila sea una imagen binarizada, y se construye el vector de salidas esperadas conforme se acumularon las imágenes en la matriz de características. Ahora se tiene todo para realizar el entrenamiento, la construcción del perceptrón y su algoritmo de entrenamiento se divide en dos partes principales. La primera parte es con la que se realiza el entrenamiento, y que fue estructurada acorde a lo descrito en el marco teórico, donde se lee la matriz de características de un archivo externo, y se realiza el mismo procedimiento para la lectura del vector de valores esperados, así como se realiza la inicialización de pesos y sesgo. Cada una de las imágenes es pasada por el perceptrón y se busca ajustar los pesos y el sesgo, hasta se obtengan los valores correctos para simular dicha imagen en particular y así se puede pasar a la siguiente imagen, esto se realiza hasta que se encuentren los pesos capaces de simular todas las imágenes en el conjunto de entrenamiento, cuando esto se alcanza, se guarda el vector de pesos en un archivo así como el sesgo, para que sean posteriormente utilizados al momento de realizar simulación o simplemente implantar el sistema. Con el conocimiento ya adquirido, es decir, los pesos y el sesgo ajustados. Para que este procedimiento se lleve a cabo intervienen algunas funciones creadas por aparte con las que se lee el archivo matriz de características y el archivo de vector de target. Adicional a esto se realizan impresiones para conocer el avance del proceso, y la verificación de salidas esperadas con salidas obtenidas, y se define un número máximo de iteraciones. La segunda parte del código es una función para realizar la simulación del sistema en tiempo real.

3.3.4 | Validación de Entrenamiento

Posteriormente una vez que se obtuvieron pesos que ajustaran a todas las muestras dadas al perceptrón en el entrenamiento, se tomó el conjunto de datos de validación para medir la capacidad de generalización de la red, que es una sub-rama del conjunto original de imágenes. Se realizó el mismo procedimiento de pre-procesamiento de imágenes, pero para este efecto se realizó la construcción de la segunda parte del perceptrón, donde lee el vector de pesos y el sesgo del archivo donde fueron guardados en la parte de entrenamiento, y se programó el proceso de cálculo del perceptrón para generar una respuesta, y realizar la clasificación, de igual forma se construyeron funciones externas a la función de cálculo, para obtención e imágenes de validación, el vector de pesos y sesgo.

3.3.5 | Pruebas de Funcionamiento

De manera similar a la validación, se hizo otro conjunto de prueba, pero esta vez las imágenes fueron obtenidas por medio de la aplicación IP Webcam, se pre-procesaron a través de MATLAB y se guardaron en un csv, de manera similar a lo realizado en la validación. Después fueron alimentadas una por una al programa verificando en cada imagen la respuesta de la neurona, así como un estimado numérico de la certeza de esa respuesta.

3.3.6 | Integración MATLAB - C++ - Arduino

Una vez que nuestro sistema fue entrenado, validado y probado con nuevas imágenes, podemos continuar con la implementación completa del sistema. Para ello tenemos que integrar todas las partes. En primer lugar, es necesario realizar la conexión entre la cámara que leerá la información del entorno, para ello construimos el script de MATLAB llamado *ArduinoCamera.m* que se puede encontrar en el apéndice C.

Para comenzar, usamos una función que guarde la información de un url; en este caso el url es el dado por la aplicación de IP Webcam, este contendrá la imagen que es vista por el celular en tiempo real. A continuación vamos a llamar a las funciones de binarización que ya se explicaron anteriormente y estas pre-procesaran la imagen contenida en el url. Una vez que nuestra imagen es vectorizada, mediante un archivo .csv, esta se transmitirá al programa escrito en C++ (apéndice B) donde será procesada por el perceptrón y devolverá su respuesta a MATLAB a través de un archivo .txt. Finalmente, MATLAB leerá la información contenida en el archivo de texto y generara la respuesta acorde al contenido, que se verá reflejada en el encendido de los LEDS en la placa arduino.

Es importante aclarar que este ciclo se realizará de forma predeterminada 10 veces dejando un espacio de lectura entre cada nueva imagen de entre 30 y 60 segundos. Esto depende fuertemente de la capacidad de procesamiento de la computadora pues el rendimiento puede variar de una a otra. Ambos scripts, *ArduinoCamara.m* y *main.cpp*, han sido programados para que esperen la respuesta entre ellos, logrando comunicarse y sincronizarse, aunque esto pudiera alterarse si el pre-processamiento de las imágenes tarda más tiempo de lo esperado.

3.3.7 | Simulación del sistema

Una vez que tenemos ya todo los elementos necesarios construidos y armados, hacemos la implementación final del sistema, que es la que se haría por 'usuarios finales'. Es necesario abrir el archivo *ArduinoCamara.m* en MATLAB, el archivo *main.cpp* del proyecto

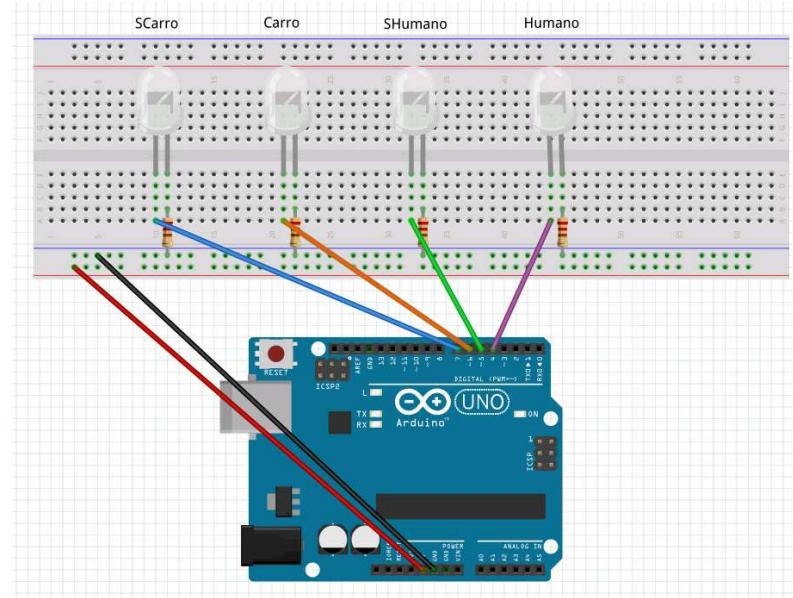


Figura 3.2: Esquema de conexión del arduino.

Neurona en el compilador de C++, el escenario listo y la aplicación IP Webcam corriendo y conectada al servidor, así como escribir esta dirección del servidor dentro del campo url en el archivo de MATLAB.

Una cosa importante a tomar en cuenta es que puede ser necesario cambiar las direcciones donde se van a guardar los archivos que se crean. Además para que se pueda observar la salida física del sistema es necesario conectar la placa arduino a la computadora y armar el circuito que se muestra en la figura 3.2. El LED con el nombre "SHumano" representa el alto para peatones mientras que "Humano" representa el siga. De manera similar pasa con los nombres de "SCarro" y "Carro".

Una vez que se tienen todos los componentes listos se procede a correr el script de MATLAB y el de C++, intercambiando los objetos de la escena y observando las respuestas de salida, tanto en terminal como en los LEDs.

Resultados obtenidos

4.1 | Binarización de imágenes

Al inicio se realizaron pruebas con imágenes y fotografías tomadas de internet donde se probó aplicó el binarizado a dichas imágenes, con esto se verificó que tan bien se realizaba la segmentación de los objetos de interés. Como podemos ver en la figura 4.1 las imágenes al tener una gran cantidad de elementos pueden resultar muy ruidosas. Las mejores binarizaciones de imágenes fueron las personas que tenían un fondo blanco o transparente, así como autos en un entorno claro y sin demasiadas cosas alrededor. Este hecho deja en claro que el limpiar imágenes en entornos con muchos objetos además del objeto de interés requiere de un pre-procesamiento más robusto.

Conforme a los resultados obtenidos de la binarización de imágenes de internet, y debido a que se requería realizar un experimento midiendo variables físicas reales, teniendo la limitante de que en los entornos de circulación de tráfico real, la obtención de datos hubiera representado imágenes con mucho ruido, difíciles de binarizar y clasificar solo con un perceptrón, se decidió utilizar imágenes de un entorno controlado por nosotros, donde las condiciones fueran óptimas para la binarización y detección de dichas imágenes.

4.1.1 | Experimentos de Clasificación

De manera formal, y para clasificación de imágenes de autos y personas, se realizaron dos experimentos. Las diferencias principales entre los mismos, están dadas por el numero de imágenes utilizadas para entrenamiento y validación, así como diferencias en el control de las condiciones del ambiente donde se adquirieron las imágenes.



Figura 4.1: Ejemplos de imágenes binarizadas.

4.1.2 | Experimento 1

Para este primer experimento se utilizaron 48 imágenes en total de entrenamiento, es decir 24 por clase, y 12 para validación por clase. Las imágenes fueron tomadas por nosotros mediante la cámara dos celulares distintos, ambos en un escenario blanco, como se puede ver en la figura 3.3.1. Posteriormente se almacenaron, organizaron y binarizaron dichas imágenes, el resultado de binarización de algunas de ellas se puede ver en las figuras 4.2 y 4.3 .

Como se puede notar en la figura 4.2 también hubo ciertas imágenes en las que no se

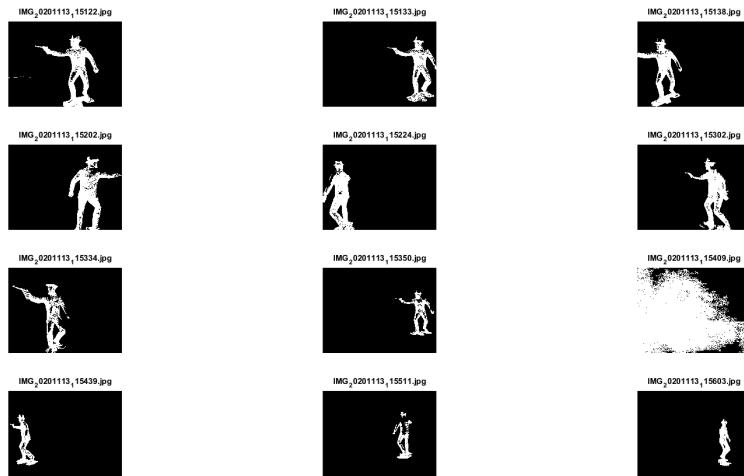


Figura 4.2: Exp1-. Ejemplos de humanos binarizados.

alcanzó a segmentar el objeto de interés de forma correcta, esto debido a falta de control en condiciones del entorno como la iluminación y sombras creadas por otros objetos. A pesar de esto, el perceptrón logró hacer una adaptación exitosa a los datos de entrenamiento, pero una clasificación relativamente aceptable en los datos de validación, presentando algunos errores al momento de clasificar personas. Al momento de realizar pruebas en tiempo real se pudo notar un ligero sesgo debido a las condiciones de iluminación de la escena donde se tomaba la imagen. Así mismo, notamos que en la clasificación se producía un mejor resultado de clasificación cuando el objeto se orientaba a cierta parte de la imagen.

4.1.3 | Experimento 2

En el experimento dos, se utilizó un conjunto nuevo de imágenes más grande que el utilizado en el experimento 1 (este consistió en 100 imágenes en total de entrenamiento, es decir 50 por clase, y 15 para validación por clase), donde se cuidaron más las condiciones del entorno al momento de tomar las imágenes buscando un escenario más uniforme en texturas e iluminación. Esto ayudó a que sobretodo las imágenes de humanos salieran más limpias al momento de realizar la binarización. Algunos ejemplos son:

Debido al cuidado en el entorno las imágenes utilizadas en este experimento arrojaron



Figura 4.3: Exp1-. Ejemplos de coches binarizados.

mejores resultados de binarización, lo cual permitió al perceptrón tener una convergencia más rápida al conjunto de imágenes de entrenamiento, para esto también influyó el aumento del número de datos ingresados. Algunos de los sesgos encontrados en el experimento 1 fueron suprimidos, pero esto también provocó que el modelo sobreajustara más a los datos de entrenamiento ingresado. Volviéndose un poco más sensible a las variaciones en las condiciones el entorno. Sin embargo en la prueba de validación realizada se obtuvo un buen resultado con únicamente una clasificación errónea.

4.2 | Experimentación en tiempo real

La implementación en tiempo real del sistema fue exitosa, ya que se logró hacer una buena integración de los programas corriendo en paralelo, así como de la lectura de datos y de emisión de respuesta, es decir el cambio de luces en el semáforo.

Conforme se fueron analizando los resultados de las pruebas en tiempo real, tanto para el modelo obtenido con en el experimento uno, como en el experimento dos, con el sistema completamente armado se pudo notar que al cambiar las condiciones de luz, los resultados se veían afectados. Con una buena iluminación, es decir midiendo de día y sin crear muchas sombras en el escenario las predicciones, en su mayoría eran correctas, pero al hacer experimentos en la tarde y con poca iluminación el rendimiento de el sistema bajaba, cometiendo errores más frecuentemente cuando se trataba de coches.

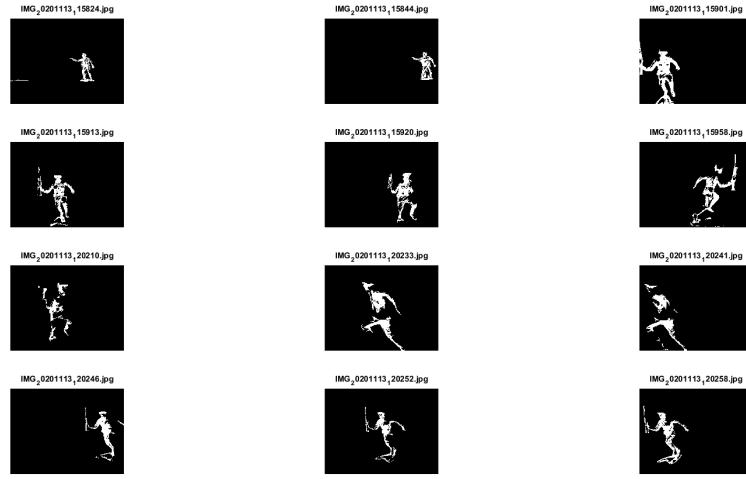


Figura 4.4: Exp2-. Ejemplos de humanos binarizados.

De acuerdo a lo observado en las imágenes binarizadas de estos ejemplos, se podían observar algunas manchas de sobras creadas por los objetos, lo que seguramente causó que los resultados se afectaran.

En las pruebas del modelo obtenido con el primer experimento, se pudieron seguir observando los sesgos de posición de los objetos en la imagen, sin embargo realizaba una muy buena clasificación si las condiciones de sesgos de posición se satisfacían. Con el modelo del segundo experimento como ya se había anticipado, hubo clasificación aceptable, pero no perfecta, debido a que el modelo se volvió más sensible a variaciones presentadas en el entorno, y que diferían de condiciones de los datos provistos en el entrenamiento.

En la imagen 4.2, se puede apreciar el tipo de condiciones en donde se realizó la obtención de imágenes de entrenamiento, validación y prueba en tiempo real. Con lo que se deja en claro que el experimento sí tuvo condiciones muy controladas con la intención de que se pudiera hacer un buen binarizado y fuera clasificable para un perceptrón.

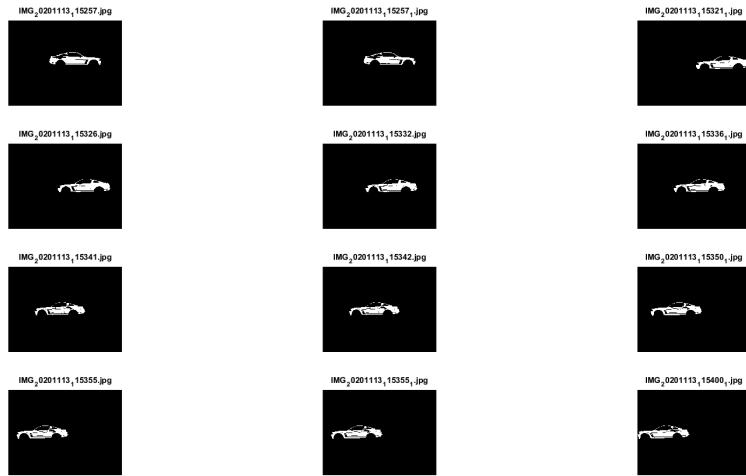


Figura 4.5: Exp2-. Ejemplos de coches binarizados.



Figura 4.6: Representación de coches y personas en escenario controlado.

Conclusiones

En este proyecto se realizó la implementación de un sistema controlador de un semáforo inteligente, capaz de detectar a una persona, o a un auto, y ceder el paso, conforme a el objeto presentado frente a la cámara de detección, para esto se utilizaron dos lenguajes de programación distintos, los cuales brindan bondades, en uno de los casos, para manejo de datos, y pre-procesamiento de imágenes, y en el segundo caso, rapidez de cálculo para la toma de una decisión mediante el procesamiento de la información. El objetivo principal de este proyecto en donde se establece la construcción de forma teórica y práctica un sistema completo con hardware y software, para la creación de un semáforo inteligente utilizando como unidad de procesamiento un perceptrón, esto permitió a su vez una comprensión mas profunda del funcionamiento y aplicabilidad de un perceptrón, lo cual indica que dicho objetivo fue cumplido de forma satisfactoria. Pudimos conocer distintos métodos de planteamiento de la unidad básica de procesamiento de una red neuronal, y sus ligeras variaciones al método de optimización para convergencia. La sincronización de funcionamiento entre lenguajes de programación y entidades de software y hardware también fue exitosa. Se encontró una aplicabilidad, que pudiese ser escalable, siguiendo las nociones básicas planteadas en este proyecto, pero potenciando los algoritmos utilizados, para la resolución de un problema real, como lo es el control de flujo de tráfico en un cruce. De la misma forma, se pudo realizar experimentación con el sistema creado fuera del ámbito virtual, y comprobando su respectivo funcionamiento, con pruebas en tiempo real, tomando siempre en cuenta que debido a las limitaciones de esta unidad de procesamiento, tuvo que realizarse experimentación bajo condiciones muy controladas. Denotando que si se quisiese resolver los problemas que representa crear un sistema a mayor escala y bajo condicione más variables, se deben contemplar aspectos como la utilización de redes complejas de

perceptrones, u otro tipo de redes o algoritmos propio de inteligencia artificial. Por ultimo se concreto el objetivo de mejorar las habilidades de programación específicamente ene lenguaje C++ por parte de los desarrolladores del proyecto.

5.1 | Posibles extensiones del proyecto

Como uno de los resultados obtenidos en este proyecto, destaca el planteamiento de un problema que puede ser solucionable mediante sistemas basados en redes neuronales, como lo es el control de un semáforo inteligente que pudiese analizar, la afluencia de tráfico peatonal y vehicular en tiempo real, optimizando así los tiempos de paso para ambos sectores. Sin embargo, este resulta ser muy complejo, para ser únicamente resuelto por el procesamiento de un perceptrón. Pero no así, para algoritmos mas capaces dentro de la misma área, debido a que los datos obtenidos son imágenes, este proyecto pudiese realizarse utilizando como método de procesamiento una red neuronal convolucional, o en su caso realizar un pre-procesamiento más robusto y donde se haga extracción de características de la forma común de personas y vehículos, como pueden ser los momentos invariantes, que representan, áreas, excentricidad, relleno de objeto binarizado, orientación, perímetro, solidez, entre otros. Una ves teniendo estos datos que son más descriptivos, que la información de los píxeles en crudo únicamente binarizados de la imagen, pudiendo utilizar un perceptrón multi-capa para clasificación. Así mismo para poder lograr llevar este tipo de sistemas a la práctica en condiciones reales, se necesitará de un microcontrolador con mayor capacidad de procesamiento y calculo, capaz de llevar a cabo todos los cálculos realizados en esta ocasión por la computadora y al mismo tiempo controlar el encendido y apagado de los semáforos.

5.2 | Comentarios Finales

Con la realización de este proyecto, quedamos satisfechos con los resultados obtenidos dadas las condiciones de hardware y software utilizadas, con la experimentación realizada, el cumplimiento de todos los objetivos y la experiencia de desarrollo adquirido. Debido a la inexperiencia de programación en C++, algunas soluciones a problemas ajenos al desarrollo del algoritmo del perceptrón, tal como la escritura y lectura de archivos, o la consulta de registros modificaciones de archivos se utilizaron, soluciones adaptadas a nuestro proyecto, extraídas de preguntas realizadas en foros de resolución de problemas en programación y documentación de lenguajes, tal como <https://>

[stack overflow .com/](https://stackoverflow.com/), <https://la.mathworks.com/help/index.html>, <http://www.cplusplus.com/forum/>, entre otros.

Código de Pre-procesamiento

■ Código de definición de funciones de mejora de contraste

```
1 classdef ContrastFunctions
2     properties
3         Value {mustBeNumeric}
4     end
5     methods
6         function p = Inversa(m)
7             p = 255 - [m.Value];
8             p = uint8(p);
9         end
10        function p = Cuadrada(m)
11            p = [uint16(m.Value)].^2/255;
12            p = uint8(p);
13        end
14        function p = Cubica(m)
15            p = [double(m.Value)].^3/255^2;
16            p = uint8(p);
17        end
18        function p = RCuadrada(m)
19            p = sqrt(255.*[double(m.Value)]);
20            p = uint8(p);
21        end
22        function p = RCubica(m)
23            p = ((255^2).*[double(m.Value)]).^(1/3);
24            p = uint8(p);
```

```
25     end
26     function p = Logaritmica(m)
27         p = 255*(log(1+double(m.Value))/log(1+255));
28         p = uint8(p);
29     end
30 end
31 end
```

■ Código de función proceso de binarización.

```
1 function C = BinarizeImg(C, imgreduced)
2
3 if imgreduced >= 1
4     C_25 = C;
5 else
6     F25 = 0.25;
7     C_25 = imresize(C,F25);
8     C_25 = C_25;
9 end
10
11
12 F = makecform('srgb2cmyk');
13 T = applycform(C_25,F);
14
15 M = T(:,:,2);
16
17 Lcont = ContrastFunctions;
18 Lcont.Value = M;
19 M = Cubica(Lcont);
20
21 Ot = graythresh(M)*256;
22
23 IOT = M;
24 i = IOT <= Ot;
25 j = IOT > Ot;
26 IOT(i) = 0;
27 IOT(j) = 256;
```

```

29
30 Ori = IOT;
31 BW = logical(IOT);
32 s = regionprops(BW, 'Centroid', 'Area', 'SubarrayIdx');
33 c = regionprops(BW, 'Circularity');
34 centroids = cat(1,s.Centroid);
35
36
37 for k = 1 : length(c)
38     Ar = s(k).Area;
39     %BB = s(k).BoundingBox;
40     s(k).isGreater = Ar < 380;
41     for i = find(s(k).isGreater).'
42         BW(s(k).SubarrayIdx{i,1},s(k).SubarrayIdx{i,2}) = false;
43     end
44     %rectangle('Position', [thisBB(1),thisBB(2),thisBB(3),thisBB(4)
45     %                           ],'FaceColor','black','EdgeColor','black','LineWidth',2 )
46 end
47
48 C = BW;

```

■ Script de lectura, aplicación binarización, re-dimensión, creación de matriz de características y vector de valores esperados.

```

1 clc; close all; clear all;
2 size_img = [960, 1280];
3 %% Binarizacion Carros Entrenamiento
4 cars= dir('Imgexp/Train/Cars/*.jpg');
5 for k=1:length(cars)
6     car= imread(string(cars(k).folder) + '\' + string(cars(k).name));
7     car = imresize(car,[size_img]);
8     car_bin = BinarizeImg(car, 1);
9     car_flatten = car_bin(:)';
10    entradas(k,:) = car_flatten;
11    target(k) = 0;
12 end
13 % squeezed = reshape(c(1,:,:,:),[z(2:end)]);

```

```

14 %% Binarizacion Humanos Entrenamiento
15 humans= dir('Imgexp/Train/Humans/*.jpg');
16 for j=1:length(humans)
17     human= imread(string(humans(j).folder) + '\' + string(humans(j).
18         name));
19     human = imresize(human,[size_img]);
20     human_bin = BinarizeImg(human, 1);
21 % human_r = imresize(human_bin,[size(car_bin)]);
22     human_flatten = human_bin(:)';%human_r
23     entradas(k+j,:) = human_flatten;
24     target(k+j) = 1;
25 end
26
27 %% Binarizacion Carros Validacion
28 valscl=dir('Imgexp/Val/Cars/*.jpg');
29 for k=1:length(valscl)
30     valimgc = imread(string(valscl(k).folder) + '\' + string(valscl(k).
31         name));
32     valimg_rc = imresize(valimgc,[size_img]);
33     valimg_binc = BinarizeImg(valimg_rc, 1);
34     valimg_flattenc = valimg_binc(:)';
35     entradasv(k,:) = valimg_flattenc;
36 end
37
38 %% Binarizacion Humanos Validacion
39 valsh=dir('Imgexp/Val/Humans/*.jpg');
40 for j=1:length(valsh)
41     valimgh = imread(string(valsh(j).folder) + '\' + string(valsh(j).
42         name));
43     valimg_rh = imresize(valimgh,[size_img]);
44     valimg_binh = BinarizeImg(valimg_rh, 1);
45     valimg_flattenh = valimg_binh(:)';
46     entradasv(k+j,:) = valimg_flattenh;
47 end
48
49 %% Guardado de Matriz de caracteristicas.

```

```
47 fileID = fopen('MatrizCaracEnt1.csv','wt');
48 formatSpec = '%d,';
49 formatSpec2 = '%d %d';
50 ent = size(entradas);
51 entradasT = entradas';
52 for i= 1:ent(1)
53     for j=1:ent(2)
54         fprintf(fileID, formatSpec, entradas(i,j));
55     end
56     fprintf(fileID, '\n');
57 end
58
59 %% Guardado de Vector de Valores Esperados.
60 fileTarget = fopen('VectorTarget.csv','wt');
61 formatSpec = '%d\n';
62 formatSpec2 = '%d %d';
63 tar = size(target);
64 targetT = target';
65 for i= 1:tar(2)
66     if i < tar(2)
67         fprintf(fileTarget, formatSpec, targetT(i));
68     else
69         fprintf(fileTarget, '%d', targetT(i));
70     end
71 end
72 fclose(fileTarget);
```

Código del Perceptrón

■ Código de entrenamiento perceptrón

```
1 ****
2
3     Codigo Entrenamiento de un perceptron para imagenes
4     La teoria de funcionamiento es adaptada al perceptron de Rosenblatt
5     Autores del Codigo - Github Users: @Natalia-SP y @Mariuki
6
7 ****
8
9 #include <iostream>
10 #include <fstream>
11 #include <chrono>
12 #include <random>
13 #include <vector>
14 #include <tuple>
15 #include <stdio.h>
16 #include <time.h>
17 #include <stdlib.h>
18 #include <unistd.h>
19 #include <fstream>
20 #include <algorithm>
21 #include <string>
22 #include <utility> // std::pair
23 #include <stdexcept> // std::runtime_error
24 #include <sstream> // std::stringstream
25 #include "csvfile.h"
26 #include <typeinfo>
27 #include <ctime>
28 #include <sys/types.h>
29 #include <sys/stat.h>
```

```

30 #include <cerrno>
31 #include <cstring>
32 #include <Windows.h>
33 #include <winbase.h>
34
35
36 using namespace std;
37
38 vector<vector<int>> GetTrainData(){
39     ifstream in(R"(D:\Directorio\MatrizCaracEnt1.csv)");
40
41     string line, field;
42
43     vector< vector<int> > array; // the 2D array
44     vector<int> v; // array of values for one line only
45
46     while ( getline(in,line) ) // get next line in file
47     {
48         v.clear();
49         stringstream ss(line);
50
51         while (getline(ss,field,',')) // break line into comma delimited fields
52         {
53             v.push_back(stoi(field)); // add each field to the 1D array
54         }
55
56         array.push_back(v); // add the 1D array to the 2D array
57     }
58 // cout << array[1].size() << "\t" << array.size() << endl;
59
60     return array;
61 }
62
63 vector<int> GetTrainTarget(){
64     ifstream in(R"(D:\Directorio\VectorTarget.csv'.csv)");
65
66     string line, field;
67
68     vector<int> array; // the 2D array
69     // array of values for one line only
70
71     while ( getline(in,line) ) // get next line in file
72     {
73         array.push_back(stoi(line)); // add the 1D array to the 2D array
74     }

```

```

75 // cout << array[1].size() << "\t" << array.size() << endl;
76 // for (auto& row : array) { /* iterate over rows *//* iterate over vals */
77 // std::cout << row << " "; /* output value */
78 // /* tidy up with '\n' */
79 //}
80     return array;
81 }
82
83
84 std::vector<float> PerceptronTrain() {
85     srand(time(NULL));
86
87     // Ejercicio Frutas //
88
89 // int input[6][6] = {{-1, 1, 1, 1, 1, 1},
90 // {1,-1, 1, 1, 1, 1},
91 // {-1,-1,-1, 1, 1, 1},
92 // {1, 1,-1,-1, 1, 1},
93 // {-1, 1,-1, 1, 1, 1},
94 // {1,-1, 1, 1, 1, 1}};
95 // std::vector<int> target = {1,1,-1,-1,-1,1};
96 // float intervalos[] = {-0.98,-
97 // 0.87,-0.65,-0.75,0.45,0.95,-0.32,-0.28,-0.11,-0.06,0.023,0.23,0.23,0.46,0.67,0.78,
98 // 0.75,0.82,0.98};
99
100    // Compuertas //
101 // int input[4][2] = {{0, 0},
102 // {0, 1},
103 // {1, 0},
104 // {1, 1}};
105 // std::vector<int> target = {1, 1, 1, 0};
106
107    // Nuestros Datos//
108
109    vector<vector<int>> input = GetTrainData();
110    vector<int> target = GetTrainTarget();
111    std::vector<float> pesos;
112
113    for (int i=0; i < input[0].size(); i++)
114        pesos.push_back(0);
115

```

```

116
117
118     float lr = .3;
119     float bias = intervalos[rand() % 18];
120
121     //// Operaciones /////
122     int NumMuestras = input[0].size();
123     int NumEntradas = input.size();
124
125     cout << NumMuestras << "\t" << NumEntradas << endl;
126
127     int corr = 0;
128     std::vector<int> vecsalida = {0, 0, 0, 0};
129     int error;
130     int salida;
131     float neurona;
132     int n = 1;
133     int k = 1;
134     int cont = 0;
135     /*cout << pesos[0] << "\t" << pesos[1] << "\t" << pesos[2] << "\t" << pesos[3]
136     <<
137     "\t" << pesos[4] << "\t" << pesos[5] << endl;*/
138     cout << "n" << "\t" << "W1" << "\t" << "W2" << "\t" << "b" <<
139     "\t" << "error" << "\t" << "salida" << "\t" << "vecs[i]" << "\t"
140     << "target[i]" << endl;
141     while (corr == 0) {
142         // n++;
143         k++;
144         for (int i = 0; i < NumMuestras;) {
145             neurona = 0.0;
146             for (int j = 0; j < NumEntradas;j++) {
147                 int input_ij = input[i][j];
148                 neurona += input[i][j] * pesos[j];
149             }
150             neurona += bias; //Original = signo -
151             salida = neurona < 0 ? 0 : 1; //Original = neurona <= 0 ? 0 : 1;
152             if (salida == target[i]) {
153                 error = 0;
154                 /* cout << n << "\t" << pesos[0] << "\t" << pesos[1] << "\t" << bias
155                 <<
156                 "\t" << error << "\t" << salida << "\t" << vecsalida[i] << "\t"
157                 << target[i] << endl;*/
158                 vecsalida[i] = salida;
159                 i++;
160             } else {

```

```

159         error = target[i] - salida;
160         for (int j = 0; j < NumEntradas;j++) {
161             pesos[j] = pesos[j] + /*lr */ error * input[i][j]; //Original =
162                                         lleva lr
163         }
164         bias = bias + /*(-lr) */ error; //Original = lleva lr
165         cout << n << "\t" << pesos[0] << "\t" << pesos[1] << "\t" << pesos
166             [2] /*<< "\t"
167             << pesos[3] << "\t" << pesos[4] << "\t" << pesos[5] */ << "\t" <<
168             bias <<
169             "\t" << error << "\t" << salida << "\t" << vecsalida[i] << "\t"
170             << target[i] << endl;
171             vecsalida.clear();
172             n++;
173             if(n > 2000) {
174                 cout << "[!] Demasiadas épocas realizadas!\n" << endl;
175                 cout << "[!] Intente nuevamente con otros pesos.\n" << endl;
176                 exit(1);
177             }
178         }
179         for (int i = 0; i < NumMuestras; i++) {
180             neurona = 0.0;
181             for (int j = 0; j < NumEntradas;j++) {
182                 neurona = neurona + input[i][j] * pesos[j];
183             }
184             neurona = neurona + bias; //Original = signo -
185             salida = neurona < 0 ? 0 : 1; //Original = neurona <= 0 ? 0 : 1;
186             if (salida == target[i]) {
187                 cont++;
188             } else { cont = 0; }
189             if (cont == NumMuestras) {
190                 corr = 1;
191             }
192         }
193     }
194     for (auto i = 0; i < NumMuestras; i++) {
195         neurona = 0.0;
196         for (int j = 0; j < NumEntradas;j++) {
197             neurona = neurona + input[i][j] * pesos[j];
198         }
199         neurona = neurona + bias; //Original = signo -
200

```

```

201     salida = neurona < 0 ? 0 : 1; //Original = neurona <= 0 ? 0 : 1;
202     cout << "Verificacion:\t" << salida << "\t" << target[i] << endl;
203 }
204 cout << "Pesos finales:\t" << pesos[0] << "\t" << pesos[1] << "\t" << bias <<
205 // for (auto& p : pesos) { /* iterate over rows *//* iterate over vals */
206 // std::cout << p << " "; /* output value */
207 // /* tidy up with '\n' */
208 //}
209
210 vector<float> pfinales = pesos;
211 pfinales.push_back(bias);
212 try
213 {
214     csvfile csv("PesosEnCero.csv"); // throws exceptions!
215     for(int i = 0; i< pfinales.size(); i++)
216         csv << pfinales[i] << endl;
217 }
218 catch (const std::exception &ex)
219 {
220     std::cout << "Exception was thrown: " << ex.what() << std::endl;
221 }
222 return pfinales;
223 }
224
225
226 int main(int argc, char **argv)
227 {
228
229     PerceptronTrain();
230
231     return 0;
232 }
```

■ Código de simulación Perceptrón

```

1 ****
2
3     Código Entrenamiento de un perceptron para imágenes
4     La teoría de funcionamiento es adaptada al perceptron de Rosenblatt
5     Autores del Código - Github Users: @Natalia-SP y @Mariuki
6
7 ****
8
9 #include <iostream>
```

```
10 #include <iostream>
11 #include <chrono>
12 #include <random>
13 #include <vector>
14 #include <tuple>
15 #include <stdio.h>
16 #include <time.h>
17 #include <stdlib.h>
18 #include <unistd.h>
19 #include <iostream>
20 #include <algorithm>
21 #include <string>
22 #include <utility>
23 #include <stdexcept>
24 #include <iostream>
25 #include "csvfile.h"
26 #include <typeinfo>
27 #include <ctime>
28 #include <sys/types.h>
29 #include <sys/stat.h>
30 #include <cerrno>
31 #include <cstring>
32 #include <Windows.h>
33 #include <winbase.h>
34
35
36 using namespace std;
37
38
39 vector<int> GetImageTest(){
40
41     // Este código fue obtenido y adaptado de un foro de stackoverflow para la
        lectura de archivos .csv
42     ifstream in(R"(D:\Directorio\ImgRead.csv)");
43
44     string line, field;
45
46     vector<int> array;
47
48     while ( getline(in,line) )
49     {
50         array.push_back(stoi(line));
51     }
52 // cout << array[1].size() << "\t" << array.size() << endl;
53 // for (auto& row : array) {
```

```
54 // std::cout << row << " ";
55 //
56 // }
57     return array;
58 }
59
60 vector<float> GetPesos(){
61     ifstream in(R"(C:\Directorio\MyTable.csv)");
62
63     string line, field;
64
65     vector<float> array;
66
67     while ( getline(in,line) )
68     {
69         array.push_back(stof(line));
70     }
71 // cout << array[1].size() << "\t" << array.size() << endl;
72 // for (auto& row : array) {
73 // std::cout << row << " ";
74 //
75 // }
76     return array;
77 }
78
79
80
81 void PerceptronTest(/*std::vector<float> pesosrec*/){
82     vector<float> pesosrec = GetPesos();
83     vector<float> pesos;
84     float neuron = 0.0;
85     cout << "Pesosrec:\t" << pesosrec.size() << endl;
86     for (int i = 0; i< pesosrec.size()-1; i++){
87         pesos.push_back(pesosrec[i]);
88     }
89
90     cout << "Pesos size:\t" << pesos.size() << endl;
91     int bias = pesosrec[pesosrec.size()];
92     vector<int> image = GetImageTest();
93     int j = 0;
94     while( j < 1228800 ) {
95         j++;
96         neuron += image[j] * pesos[j];
97
98     }
```

```

99     neuron += bias;
100    cout << "Neurona:\t" << neuron << endl;
101    int salida = neuron < 0 ? 0 : 1;
102    string sal = salida == 0 ? "Coche" : "Persona";
103    cout << "Resultado:\t" << sal << endl;
104    ofstream myfile; //Código obtenido de stackoverflow para la escritura de archivos
105    myfile.open ("D:/Directorio/Respuesta.txt");
106    myfile << salida;
107    myfile.close();
108 // cout << "Pesos test finales:\t" << pesosrec[0] << "\t" << pesosrec[1] << "\t" <<
109 // pesosrec[2] << endl;
110 }
111
112 int main(int argc, char **argv)
113 {
114
115     for(int i =0; i<3; i++) {
116 //int control = 0;
117 //while(control == 0){
118         struct stat fileInfo;
119
120         if (stat(
121             "D:/Directorio/ImgRead.csv", //Código obtenido de stackoverflow para
122             &fileInfo) != 0) { //la lectura de fechas de archivos
123             std::cerr << "Error: " << strerror(errno) << '\n';
124             return (EXIT_FAILURE);
125         }
126         char* a = ctime(&fileInfo.st_mtime);
127         cout << "Resultados para imagen #" << i+1 << "\n" << endl;
128         PerceptronTest();
129         cout << "\n " << "-----Consulta Finalizada -----" << "\n \n \n"
130             << endl;
131         char* b = ctime(&fileInfo.st_mtime);
132         std::string stra(a);
133         std::string strb(b);
134         int g = 0;
135         while(stra== strb){
136             g++;
137             Sleep(50000);
138             struct stat fileInfo;
139             if (stat(
140                 "D:/Directorio/ImgRead.csv",
141                 &fileInfo) != 0) {
142                 std::cerr << "Error: " << strerror(errno) << '\n';
143                 return (EXIT_FAILURE);
144             }

```

```
142         }
143 // std::cout << strb << '\n' << std::flush;
144
145         b = ctime(&fileInfo.st_mtime);
146         std::string strc(b);
147         strb = strc;
148     }
149
150 // cout << a << endl;
151
152 }
153
154 return 0;
155 }
```

Integración Matlab - C++ - Arduino

```
1 clear a; clc; clear all; close all;
2 a=arduino();
3 Carro='D6';
4 SCarro = 'D7';
5 Humano='D4';
6 SHumano = 'D5';
7
8 % url = 'http://XXX.XXX.X.XX:XXX/shot.jpg';
9 url = 'http://XX.X.X.XX:XXX/shot.jpg'; % IP Obtenida de la app IP Webcam
10 ss = imread(url);
11 fh = image(ss);
12 size_img = [960,1280];
13
14
15 formatSpec = '%d\n';
16 formatSpec2 = '%d %d';
17
18 for j=1:3
19     fprintf('\n \n Analizando imagen #%d \n',j);
20     ss = imread(url);
21 % set(fh,'CData',ss);
22 % drawnow;
23     fprintf('Imagen Leda \n')
24     testimg_r = imresize(ss,[size_img]);
25     testimg_bin = BinarizeImg(testimg_r, 1);
```

```
26     testimg_flatten = testimg_bin(:)';
27 % imshow(testimg_bin);
28     fprintf('Binarizada y Vectorizada \n')
29 % Guardar imagen en archivo
30 fileTarget = fopen('D:/Documentos/ImgRead.csv','wt');
31 img_size = size(testimg_flatten);
32 imgT = testimg_flatten';
33 for i= 1:img_size(2)
34     if i < img_size(2)
35         fprintf(fileTarget, formatSpec, imgT(i));
36     else
37         fprintf(fileTarget, '%d', imgT(i));
38     end
39 end
40 fprintf('CSV Escrito \n')
41 fclose(fileTarget);
42 % Leer el archivo de respuesta
43 txt = dir('D:/Documentos/Respuesta.txt');
44 date1 = txt.date;
45 date2 = date1;
46 while date1 == date2
47     txt2 = dir('D:/Documentos/Respuesta.txt');
48     date2 = txt2.date;
49 end
50 fileID = fopen('D:/Documentos/Respuesta.txt','r');
51 formatSpecResp = '%d';
52 fprintf('Respuesta Leda \n')
53 Resp = fscanf(fileID,formatSpecResp);
54 fclose(fileID);
55 % Actuar sobre arduino
56 if Resp == 0
57     fprintf('Estoy viendo un coche \n')
58     writeDigitalPin(a,SCarro,0)
59     writeDigitalPin(a,SHumano,1)
60     writeDigitalPin(a,Carro,1)
61     writeDigitalPin(a,Humano,0)
```

```
62 else
63     fprintf('Estoy viendo una persona \n')
64     writeDigitalPin(a,SCarro,1)
65     writeDigitalPin(a,SHumano,0)
66     writeDigitalPin(a,Carro,0)
67     writeDigitalPin(a,Humano,1)
68 end
69 pause(2);
70 end
71 fileTarget = fopen('D:/Documentos/ImgRead.csv','wt');
72 fprintf(fileTarget, "%s", "Fin");
73 fclose(fileTarget);
74 fprintf('Fin')
75 writeDigitalPin(a,SCarro,0)
76 writeDigitalPin(a,SHumano,0)
77 writeDigitalPin(a,Carro,0)
78 writeDigitalPin(a,Humano,0)
```

Referencias

- Heber Nehemías Chui Betancur, Jael Julia Chambi Grande, and Alberto Chui Mamani. El aprendizaje y el modelo matematico de una red neuronal denominado perceptron. *Investigación educacional*, 12(22): 39–46, 2008.
- Howard Demuth, Mark Beale, and Martin Hagan. Neural network toolbox. *For Use with MATLAB. The MathWorks Inc*, 2000, 1992.
- Jean-Paul Ebejer, Garrett M. Morris, and Charlotte M. Deane. Freely available conformer generation methods: How good are they? *J. Chem. Inf. Model.*, 52(5):1146–1158, 2012.
- Wilfrido Gómez Flores. Reconocimiento de objetos en fotografias. *CINVESTAV Tamaulipas*, 2015.
- Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.
- I Stephen. Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, 50(2):179, 1990.