Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

DISEÑO DE UN SISTEMA IDENTIFICADOR DE BILLETES PARA PERSONAS CON DEFICIENCIA VISUAL, EMPLEANDO TECNOLOGÍAS OPEN SOURCE TANTO EN SOFTWARE COMO EN HARDWARE

Milton Rolando Vásquez Hernández
Asesorador por el Ing. José Anibal Silva de Los Angeles

Guatemala, noviembre de 2020

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO DE UN SISTEMA IDENTIFICADOR DE BILLETES PARA PERSONAS CON DEFICIENCIA VISUAL, EMPLEANDO TECNOLOGÍAS OPEN SOURCE TANTO EN SOFTWARE COMO EN HARDWARE

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

MILTON ROLANDO VÁSQUEZ HERNÁNDEZ
ASESORADO POR EL ING. JOSÉ ANIBAL SILVA DE LOS ANGELES

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO ELECTRÓNICO

GUATEMALA, NOVIEMBRE DE 2020

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANA	Inga. Aurelia Anabela Cordova
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martinez
VOCAL III	Ing. José Milton De León Bran
VOCAL IV	Br. Christian Moisés De La Cruz Leal
VOCAL V	Br. Kevin Vladimir Armando Cruz Lorente
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. José Aníbal Sival de los Angeles
EXAMINADOR	Ing. Helmunt Federico Chicol Cabrera
EXAMINADOR	Ing. Carlos Alberto Navarro Fuentes
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO DE UN SISTEMA IDENTIFICADOR DE BILLETES PARA PERSONAS CON DEFICIENCIA VISUAL, EMPLEANDO TECNOLOGÍAS OPEN SOURCE TANTO EN SOFTWARE COMO EN HARDWARE

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 26 de julio de 2018.

Milton Rolando Vásquez Hernández

Dedicado a mis padres
Rolando y María Elena

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por todo el conocimiento adquirido y porque me permitió convertirme en un profesional.
Mi familia	Por su comprensión y apoyo incondicional para alcanzar esta meta en mi vida.
Mis amigos	Por compartir las alegrías y decepciones durante esta etapa de mi vida, que siempre estarán en mi memoria.
Los Ciudadanos de Guatemala	A todos aquellos trabajadores honestos y responsables, que a pesar de la desigualad y los problemas que enfrenta nuestra sociedad, ayudan con sus impuesto a dar la oportunidad de educarse a muchos guatemaltecos.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
AV	Valor de agudeza visual
a'	Minutos de arco $(a \in \mathbb{Q})$
$\mathbb{X}$	Subconjunto de los números naturales
$\rightarrow$	Mapeo entre funciones
$\mathbb{R}^n$	Conjunto de puntos definidos sobre un espacio vectorial de dimensión n.
$\mathbb{N}$	Conjunto de los números naturales
$\vec{x}$	Vector columna
nm	Nanómetro
V	Voltio
A	Amperio
GHz	Gigahertz
MB	Megabyte
$\%$	Porcentaje
fps	Fotogramas por segundo
$\times;\cdot$	Multiplicación
$I_{i,j}$	Valor dentro de la matriz de intensidad
$\mathbf{F}$	Representación matricial de una imagen
$\vec{x}_h$	Vector columna en coordenadas homogéneas
$\mathbf{R}_\theta$	Matriz de rotación
$\mathbf{Z}_s$	Matriz de escalamiento
$\mathbf{T}_{t_x,t_y}$	Matriz de traslación
$\mathbf{R}_{\theta h}$	Matriz de rotación en coordenadas homogéneas
$\mathbf{Z}_{sh}$	Matriz de escalamiento en coordenadas homogéneas

Símbolo	**Significado**
$f * g$	Operación de convolución entre las funciones f y g
$F(u)$	Función en el dominio de la frecuencia
$F(u, v)$	Función de dos variables en el dominio de la frecuencia
$\sum$	Sumatoria
$f(x)$	Función con dominio continuo o discreto
$f(x, y)$	Función de dos variables en un dominio discreto o continuo
e^n	Función exponencial
$\mathbb{Q}$	Conjunto de los número racionales.

GLOSARIO

Amplificador Circuito electrónico que tiene como función incrementar al corriente, tensión o potencia de una señal eléctrica que se aplica a su entrada.

And Compuerta lógica que realiza la función de conjunción, su salida se encuentra en estado lógico alto (1) si y solo sí todas sus entradas se encuentran en estado lógico alto (1) de lo contrario su salida muestra un estado lógico bajo (0).

Android Sistema operativo específicamente diseñando para dispositivos portátiles como teléfonos inteligentes con pantalla táctil y creado por Google basado en el núcleo de Linux junto a diversos software de código abierto.

API Siglas en inglés para *application programming interface*, es un conjunto de subrutinas, funciones y métodos que ofrece una determinada biblioteca para la comunicación entre componentes de software.

ARM Arquitectura de procesadores que cuenta con un reducido conjunto de instrucciones (RISC) ampliamente utilizada en aplicaciones de bajo consumo. Fue desarrollada por la empresa ARM Holdings con sede en el Reino Unido.

CCD	Siglas en inglés para *charge-coupled device*, es un circuito integrado que contiene un arreglo de capacitores enlazados, principalmente usado en tecnologías de imagen digital.
CMOS	Abreviatura del inglés para *complementary metal oxide semiconductor*, tecnología que emplea funciones lógicas elaboradas mediante pares de transistores de efector de campo MOSFET tipo n y tipo p para la construcción de circuitos integrados.
Convolución	Es un operador matemático entre dos funciones (f y g) que produce una tercera función, que expresa como la forma de una es modificada por la otra. Es un tipo muy general de media móvil.
Coordenadas homogéneas	Son un instrumento para designar un punto dentro del espacio proyectivo, se encuentra estrechamente relacionado con la idea de perspectiva. Son ampliamente usadas para la representación de objetos tridimensionales y transformación de imágenes.
CSI	Interfaz serie para cámara (del inglés, *camera serial interface*) es una especificación de la Mobile Industry Processor Interface Alliance que define el bus de comunicación en sistemas embebidos.

DC	Corriente continua (siglas del inglés *direct current*) flujo de carga continuo que se desplaza a través de un circuito con parámetros de tensión e intensidad constantes.
Debian	Es un sistema operativo basado en software libre desarrollado por miles de voluntarios de todo el mundo. Es una distribución de Linux.
DFT	Transformada discreta de Fourier (del inglés *discrete Fourier transform*), es una transformada utilizada en el análisis de señales en tiempo discreto y dominio finito.
Difracción	Desviación de una onda al chocar con el borde de un cuerpo opaco o una rendija.
Estadística inferencial	Parte de la estadística dedicada a obtener conclusiones útiles para realizar deducciones sobre una población estadística a partir de la información numérica de una muestra.
Fotodiodo	Dispositivo semiconductor sensible a la incidencia de luz visible o infrarroja.
Fotosensible	Sensible a la luz o a la energía radiante.

GPIO	Siglas en inglés de *general-purpose input/output*, para designar la entradas y salida en un sistema embebido.
HDMI	Siglas en inglés para *high-definition multimedia interface*, que es una norma, que permite el uso de video de alta definición y audio digital multicanal en un solo cable.
Hiperplano	Generalización del concepto de plano para cualquier espacio dimensional.
Información semántica	Información que capaz de aportar significado,interpretación y sentido.
iOS	Sistema operativo propietario desarrollado por Apple para dispositivos móviles.
IoT	Internet de las cosas (del inglés, *internet of things*), concepto implica la conexión de objetos ordinarios con internet.
Isotrópico	Propagación en todas las direcciones.

Java Uno de los lenguajes multiplataforma más usados en la actualidad, implementa el paradigma de programación orientado a objetos y se ejecuta de forma nativa en el sistema operativo Android por lo que es usado para el desarrollo de aplicaciones para teléfonos inteligentes.

Jupyter Proyecto de software libre que permite la computación interactiva mediante varios lenguajes de programación.

Kernel Matriz pequeña empleada como mascara de convolución para resaltar características de una imagen.

Latencia Demora entre la ejecución de una inferencia y otra para un modelo de aprendizaje automático.

Linux Hace referencia a una distribución GNU/Linux que es un sistema operativo completo de código abierto con características de un sistema operativo tipo Unix.

Minuto de arco Unidad de medida angular que representa $1/60$ de un grado sexagesimal o $1/21600$ del arco de un círculo, también equivalente a $\pi/10800$ radianes.

Multiplexación Técnica que permite transmitir varias señales por el mismo canal.

Optotipo	En geometría es un instrumento visual empleado para evaluar la agudeza visual, generalmente son símbolos o figuras de diferentes tamaños impresos en un tabla.
Or	Compuerta lógica que realiza la función de disyunción, su salida muestra un estado lógico alto (1), si cualquiera de sus entradas se encuentra en un estado alto (1) si todas sus entradas se encuentran en estado lógico bajo (0), su salida es un estado bajo (0).
Piezoeléctrico	Es un tipo de material que al ser sometido a fuerza capaz de causar deformación mecánica muestra una tensión eléctrica que puede producir cargas eléctricas en la parte exterior.
Píxel	Para una imagen digital un píxel es el elemento físico más pequeño que puede ser controlado individualmente en color y que se forma por los tres valores en la misma posición de la representación como arreglo matricial de un imagen RGB.
Refracción	Cambio de dirección y velocidad que experimenta una onda al pasar de un medio a otro con distinto indice refractivo.
RGB	Siglas del inglés *red, green and blue*, es un esquema de color para representar imágenes digitales.

RNA	Siglas para red neuronal artificial.
SBC	Siglas del inglés *single-board computer*, es una computadora construida sobre una única placa de circuito impreso.
Subtender	Unir con una línea recta los extremos de un arco de curva o de una línea quebrada.
Swift	Lenguaje de programación multiparadigma creado por Apple para el desarrollo de aplicaciones en sus sistemas operativos.
Tricromático	Capaz de percibir tres colores imaginarios.
Tricromatismo	Capacidad de los individuos de percibir información de color mediante tres canales independientes.
USB	Abreviatura que proviene del inglés *universal serial bus*. Es un estándar que define las especificaciones como protocolo, número de cables, los voltajes de alimentación, la forma de los conectores entre otros, para el bus de comunicación serial entre computadoras o entre computadoras y periféricos.

Widget	Es un pequeño programa definido dentro de un lenguaje de programación que tiene como función dar fácil acceso a funciones frecuentemente usadas y proveer de información visual al usuario, por ejemplo botones, lista, calendario, etc.
Xor	Compuerta lógica que realiza la función de disyunción exclusiva, su salida se encuentra en estado lógico alto (1) si sus entradas se encuentra en estados lógicos diferentes, es decir una en estado alto (1) y la otra en estado bajo (0), si ambas tiene el mismo estado, o sea ambas alto (1) o ambas bajo (0) la salida tendrá un estado bajo (0).

RESUMEN

Este trabajo presenta una introducción a la teoría del procesamiento digital de imágenes y los sistemas de aprendizaje automático, muestra los elementos que conforman un clasificador de imágenes construido mediante redes neuronales y el procedimiento para construir un modelo de clasificación con la intención de aplicarlo al problema de reconocimiento de billetes con para facilitar el desarrollo de un sistema permita a una persona con discapacidad visual identificarlos.

Con el propósito de ofrecer contexto al problema, el primer capítulo aborda la discapacidad visual, su definición, clasificación y algunas de sus causas. En el segundo capítulo se presentan los billetes, que al momento de la elaboración de este trabajo circulan en Guatemala; esto con el deseo de mostrar claramente las clases que debe reconocer el clasificador. El tercer capítulo trata sobre la formación de imágenes digitales, su representación, los elementos que las forman para finalizar exponiendo algunas técnicas de procesamiento y manipulación. Las bases teóricas sobre las redes neuronales, algunas tipos de red junto la idea general de entrenamiento se presentan se tratan en el capítulo cuatro.

Para finalizar, en el quinto capítulo, se describen de forma genérica las herramientas de software *open source* empleadas para la construcción del clasificador así como conceptos prácticos sobre el entrenamiento de redes neuronales, se expone el desempeño del modelo en la tarea de clasificación mediante las métricas de exactitud y pérdida, además muestra los elementos de hardware necesarios para el despliegue del modelo en un dispositivo diferente a

una estación de trabajo y se describen las tecnologías empleadas para el desarrollo de una aplicación prototipo para teléfonos inteligentes con la cual se pueda poner a prueba el modelo de clasificación.

OBJETIVOS

General

Desarrollar un sistema que sea capaz de identificar las denominaciones de papel moneda de Guatemala en circulación a la fecha de elaboración de este trabajo y transmitir esta información al usuario mediante audio, empleado tecnologías con licencia abierta.

Específicos

1. Presentar una introducción al procesamiento digital de imágenes y a la visión por computador.
2. Dar a conocer las herramientas de software libre disponibles para realizar tareas de aprendizaje de máquina, así como el entrenamiento y creación de modelos de clasificación de imágenes.
3. Crear un prototipo con suficiente exactitud para la identificar la denominación de los billetes de libre circulación en Guatemala, a la fecha de elaboración del trabajo.
4. Publicar el conjunto de datos y el clasificador entrenado para que puede accesible a cualquiera que desee adaptarlo a sus necesidades.

INTRODUCCIÓN

La visión por computador, una rama de la inteligencia artificial que intenta imitar la visión humana, tiene un papel indiscutible en el desarrollo de nuevas tecnologías en la actualidad debido a la variedad de aplicaciones en las que está presente, desde el reconocimiento de rostros en las imágenes que se cargan a las redes sociales hasta el control de robots autónomos en líneas de producción.

Una de las tareas clave dentro de la visión por computador es la clasificación de imágenes, que consiste en asignar una clase de un conjunto de clases posibles a una imagen, con base en su contenido. Esta tarea que es simple para un ser humano, presenta un desafío para un sistema computacional pero tiene infinidad de aplicaciones prácticas.

En este trabajo se proporciona una introducción a la visión por computador, a las técnicas de procesamiento de imágenes y se presenta además uno de los modelos para realizar clasificación de imágenes mediante redes neuronales convolucionales, por último se utiliza el modelo para el reconocimiento de papel moneda de Guatemala con el fin de asistir a una persona con discapacidad visual a identificarlo.

1. DEFICIENCIA VISUAL

En este capítulo se define que algunos términos relacionados a la deficiencia visual y como se ubica dentro de la discapacidad visual, se aportan datos que el autor considera relevantes para dar contexto al trabajo.

1.1. Discapacidad Visual

Es un término utilizado para aludir a cualquier daño o pérdida de la función visual que impida o restrinja, a una persona, realizar tareas cotidianas con normalidad. Se refiere a cualquier daño en la visión que no puede ser corregido mediante el uso de anteojos o lentes de contacto.

Existen diferentes términos para distinguir a las personas con limitaciones visuales como ceguera, baja visión y deficiencia visual, tiene en común que todos expresan un daño severo en la vista. Estos términos se encuentran englobados dentro de la discapacidad visual.

1.2. Baja Visión

Esta expresión abarca una serie de problemas visuales muy amplios con orígenes y causas diversas pero que en última instancia se aplica a todas aquellas personas que tienen un daño en la visión, pero aún conservan algún remanente útil que les permite obtener información para realizar sus tareas diarias.

1.3. Ceguera

Incluye a todos los individuos que no son capaces de percibir la luz, incapaces de distinguir la luz de la obscuridad o a aquellas personas que perciben luz, pero no les es posible obtener más información mediante la vista y se ven obligadas a emplear mayormente otros sentidos para desarrollar tareas cotidianas.

El Consejo Internacional de Oftalmología en el año 2002 recomendó el siguiente uso para los términos antes mencionados:

- Baja visión: para su uso en situaciones donde las personas pueden ser ayudadas mediante aparatos que mejoren la visión.
- Deficiencia visual: cuando la pérdida de visión se caracteriza por una pérdida de funciones visuales a nivel orgánico tales como agudeza visual y campo visual.
- Ceguera: cuando la pérdida de visión es total y las persona tiene que emplear técnicas para reemplazar la visión.

Algunos datos importantes a considerar sobre la discapacidad visual y la ceguera que provee la Organización Mundial de la Salud (OMS) se presentan a continuación:

- Para el año 2015 en el mundo se estimaba que unas 253 millones de personas padecían alguna forma de discapacidad en la vista; de las anteriores 36 millones con ceguera y 217 millones con discapacidad visual que van de moderada a discapacidad grave.
- El 28 % de las personas que padecen discapacidad visual moderada o grave están en edad de trabajar, dificultado así sus oportunidades de

encontrar un empleo y llevar una vida productiva.

- El 90 % de las personas que tienen discapacidad visual viven en países en vías de desarrollo.
- Cerca de 1.4 millones de niños padecen ceguera.
- El 81 % de las personas con ceguera o discapacidad visual moderada a grave tiene edades mayores a los 50 años.

1.4. Medición de discapacidad visual

Para determinar el porcentaje de visión que conserva una persona existen diversas técnicas que puede medir de manera cuantitativa aspectos de la función visual como la agudeza y el campo visual.

1.4.1. Agudeza visual

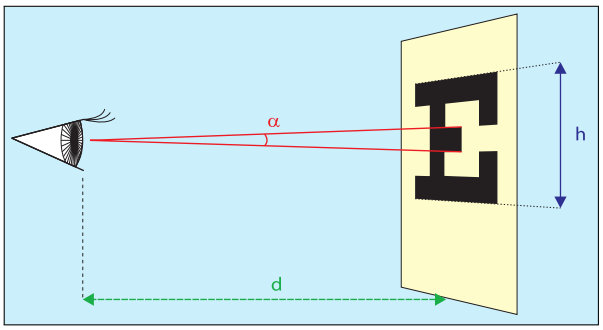
Es en esencia la claridad con la que un individuo ve un objeto y la definición de los detalles que logra percibir, de manera formal se refiere a la capacidad de la función visual para distinguir dos estímulos separados por un determinado ángulo (α). Está definida matemáticamente por el inverso del ángulo en minutos de arco que separa los estímulos.

$$AV = \frac{1}{\alpha} \tag{1}$$

Como se muestra en la figura 1 el ángulo mínimo de resolución α representa la distancia angular del detalle más pequeño que el observador es capaz de identificar en un optotipo, d representa la distancia al optotipo y h su altura.

Los valores de agudeza visual obtenidos mediante el ángulo mínimo de resolución no son empleados en la práctica, en su lugar es ampliamente

Figura 1. **Ángulo α en la medida de agudeza visual**



Fuente: MARTÍN HERRANZ, Raúl; VECILLA ANTOLÍNEZ, Gerardo. *Manual de Optometría*. https://www.medicapanamericana.com/datos/Works.4328.Sample.bin. Consulta: 3 de noviembre de 2019.

utilizada la prueba de Snellen que expresa la agudeza visual como una fracción donde el numerador es la distancia entre el individuo y el optotipo, mientras que el denominador representar el tamaño de la letra (optotipo) representado mediante la distancia a la que esta subtendería un ángulo de cinco minutos.

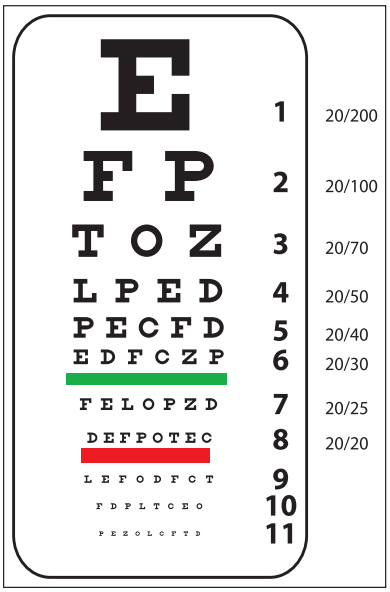
$$AV = \frac{\textit{Distancia de la prueba}}{\textit{Distancia a la que la letra subtendría un ángulo de 5'}} \tag{2}$$

En palabras más simples si una persona tiene un valor de agudeza visual de 20/70, expresada en pies, significa que la mínima letra que es capaz de reconocer correctamente a 20 pies sería vista por una persona con una agudeza visual normal a 70 pies. La figura 2 muestra la escala de Snellen con los optotipos de uso más común.

1.4.2. Campo visual

Es el espacio de realidad en la cual se puede reconocer estímulos con la visión, manteniendo la vista fija en un punto estático sin mover la cabeza. Una

Figura 2. **Escala de Snellen**



Fuente: MARTÍN HERRANZ, Raúl; VECILLA ANTOLÍNEZ, Gerardo. *Manual de Optometría*. https://www.medicapanamericana.com/datos/Works.4328.Sample.bin. Consulta: 3 de noviembre de 2019.

persona normal tiene un campo visual de aproximadamente 190 grados. El campo visual se cuantifica mediante la campimetría que es un examen médico que cosiste en fijar la vista de la persona en un punto estático e introducir puntos luminosos de forma aleatoria dentro de su campo visual, la persona evaluada debe indicar el momento en que percibe el estímulo.

1.5. Categorías

De acuerdo a la Organización Mundial de la Salud (OMS) y basándose en la Clasificación Internacional de Enfermedades 10 (CIE-10), la función visual se divide en cuatro categorías principales:

- Visión normal

- Discapacidad visual moderada
- Discapacidad visual grave
- Ceguera

Tal como se mostró en la sección anterior, para caracterizar la discapacidad visual se emplea principalmente la agudeza visual. La CIE ofrece entonces una clasificación de la severidad de la discapacidad visual en relación con la agudeza visual de una persona, que se presenta en la tabla I. Los valores en cada una de las celdas de la tabla corresponde una fracción de Snellen expresada en metros para el primer valor, a la escala decimal para el segundo y la fracción Snellen expresada en pies para el tercero.

Tabla I. **Categorías de discapacidad visual**

Categoría	Agudez visual lejana	
	Menor a:	Igual o mejor que:
0 Discapacidad visual leve o sin discapacidad		6/18 3/10 (0.3) 20/70
1 Discapacidad visual moderada	6/18 3/10 (0.3) 20/70	6/60 1/10 (0.1) 20/200
2 Discapacidad visual grave	6/60 1/10 (0.1) 20/200	3/60 1/20 (0.05) 20/400
3 Ceguera	3/60 1/20 (0.05) 20/400	1/60 1/20 (0.05) 20/400
4 Ceguera	1/60 1/50 (0.02) 5/300 (20/1200)	Percepción de luz
5 Ceguera	No percepción de luz	
9	Indeterminado o no especificado	

Fuente: Organización Mundial de la Salud. *CIE-10*. `http://apps.who.int/classifications/icd10/browse/2016/en#/H54`. Consulta: 4 de noviembre de 2019.

1.6. Causas

Las estimaciones que presenta la OMS indican que las causas principales de discapacidad visual moderada o grave son: errores de refracción no corregidos que representa un 53 % de los casos, cataratas 25 %, degeneración macular debido a la edad 4 % y retinopatía diabética 1 %; mientras que, la ceguera se debe principalmente a cataratas 35 %, errores de refracción no corregidos 21 % y glaucoma 8 %.

1.6.1. Errores de refracción

Son afecciones que no permiten a una persona enfocar adecuadamente, se deben a cambios en la estructura del ojo, como en el globo ocular, ya sea más corto o más largo de lo normal, la forma de la córnea o envejecimiento del cristalino. La visión borrosa, nublada o doble son algunos de los síntomas que presentan. Los errores más habituales son:

- Miopía: este problema se manifiesta como visión borrosa al enfocar objetos lejanos y clara al enforcar objetos cercanos.

- Hipermetropía: los objetos cercanos se ven borrosos mientras que los lejanos claros.

- Astigmatismo: deformación de la imagen al enfocar objetos debido a una curvatura irregular de la cornea.

- Presbicia: pérdida de la capacidad de enfocar objetos cercanos, debido a la rigidez del cristalino causada por el envejecimiento.

1.6.2. Catarata

El cristalino, que es la lente natural del ojo, está conformado por proteínas ordenadas adecuadamente que lo hace transparente, con la edad la acumulación de proteínas nubla pequeñas áreas del cristalino que se extienden lentamente dificultando la visión, a esta afección se le denomina catara.

Aunque el envejecimiento es considerada la causa más común de las cataratas, no es la única, las cataratas pueden ser causadas por traumas, exposición a radiación, enfermedades de la piel o factores genéticos.

1.6.3. Retinopatía diabética

Es una complicación ocular de la diabetes que surge debido al daño en los vasos sanguíneos de la retina por los altos niveles de azúcar en sangre, estos puede inflamarse, dejar escapar fluidos u obstruirse completamente. La retina es el tejido sensible a la luz que se encuentra en la parte posterior del ojo, convierte las imágenes en señales eléctricas que luego son interpretadas por el cerebro, debido a esto cualquier daño en la retina afecta directamente la visión y puede causar ceguera.

1.6.4. Degeneración macular

La mácula es la una parte de la retina que es capaz de percibir detalles finos, permite reconocer rostros, leer y además asociada a ella está la visión central que es el sector donde está presente la mayor agudeza visual. La degeneración macular se manifiesta como pérdida de la agudeza visual debido al daño en la mácula, por lo que se pierde resolución en la visión y se dificulta captar todos los pequeños detalles de una imagen. Existe dos tipos de

degeneración macular húmeda y seca.

La degeneración húmeda se produce debido al crecimiento anormal de vasos sanguíneos en la retina que pueden dejar escarpar sangre u otros fluidos, provocando cicatrices en la mácula. Para el caso de la degeneración seca, que es el tipo más común, esta se produce por el adelgazamiento debido a la edad de algunas partes de la mácula causada por la acumulación de proteínas llamadas drusas.

1.6.5. Glaucoma

Es un trastorno capaz de dañar el nervio óptico y provocar pérdida de la visión o ceguera, en la parte frontal de un ojo sano está presente un líquido transparente llamado humor acuoso, el ojo produce y drena este líquido continuamente para mantener una circulación constante, la red trabecular es el tejido encargado de drenarlo, si el humor acuoso no fluye adecuadamente se incrementa la presión en el ojo, provocado una fuerza adicional sobre el nervio óptico, lo que provoca daño en las fibras nerviosas.

1.7. Discapacidad visual en Guatemala

En Guatemala al momento de realizar este trabajo se cuentan con datos que proceden de la Encuesta Nacional de Discapacidad (ENDIS) realizada en el año 2016, los resultados que esta presenta, indican que en el país el 10.2 % de la población sufren algún tipo de discapacidad, de los adultos encuestados de 18 años o mayores el 4.2 % tiene discapacidad visual, mientras que solo el 0.5 % de niños entre 2 y 17 años presentan este tipo de discapacidad.

2. MONEDA DE GUATEMALA

Este capítulo tiene como objetivo presentar y describir los billetes de actual circulación en Guatemala, se describe las características de cada uno y se presenta todas las denominaciones junto con ilustraciones para que puedan ser fácilmente reconocidos por el lector.

2.1. Reseña histórica

La civilización maya en Guatemala, utilizó como moneda objetos como plumas de quetzal, conchas, jade y cacao. Con la llegada de los españoles durante la conquista fue necesario instaurar un sistema comercial que no estuviera basado el trueque, por lo que se adoptó la moneda española como base para las operaciones comerciales. Las monedas españolas circularon hasta la tercera década del siglo XIX.

Al formarse la República de Guatemala después de la disolución de la Federación Centroamericana, no se estableció una moneda nacional sino que durante varios años circuló la moneda de la federación junto a varias monedas extranjeras, fue durante el gobierno de Rafael Carrera que se creó el peso, moneda aún acuñada en oro. En el año 1873 se crea el Banco Nacional, durante el gobierno de Justo Rufino Barrios se ponen en circulación los primeros billetes en Guatemala, bajo la denominación de peso.

Con la reforma monetaria de 1925 en el gobierno del general José María Orellana se crea el Quetzal, con la intención de corregir el desorden económico y monetario causado por los gobiernos anteriores. Inicialmente se emitieron

solamente monedas y fue hasta 1927 que se emitió papel moneda bajo el signo del Quetzal. Después de la Revolución de Octubre de 1944 se impulsó una reforma monetaria que concluyo en 1946 con la creación del Banco de Guatemala constituido como el único emisor de moneda. Fue el 15 de septiembre de 1948 que el Banco de Guatemala emite los primeros billetes con diseños propios y con denominaciones de 50 centavos de Quetzal, 1, 5, 10, 20, 50, y 100 Quetzales.

2.2. Características

En 1971 con la intención de añadir nuevas características de seguridad y de renovar las herramientas de impresión de los billetes, el Banco de Guatemala pone en circulación billetes con nuevos diseños, todo ellos en su anverso portaban a imágenes personajes de renombre en la historia guatemalteca, se añadió también numeración maya para indicar la denominación de los billetes. En los reversos se emplearon diversas imágenes de base, como algunas obras de artista guatemaltecos o lugares reconocidos de Guatemala.

Los diseños de los billetes han sufrido algunos cambios a lo largo de los años pero la mayoría de los elementos que se introdujeron en 1971 aún se conservan. La figura 3 muestra los billetes que actualmente circula en Guatemala.

2.3. Seguridad

El Banco de Guatemala ha implementado medidas para evitar la falsificación de los billetes, la mayoría de las medidas son similares y están presentes en todos los billetes, pero algunas están presentes solamente en ciertas denominaciones. Se describen a continuación las características presentes en la mayoría de los

billetes:

- Microtextos: denominación del billete impresa con caracteres minúsculos, que puede ser leída con una lente de aumento.
- Ventana transparente: figura transparente que se combina con una imagen sombreada y solamente está presente en los billetes de 1 y 5 Quetzales, con diseños particulares de cada denominación.
- Alto relieve: resaltado de la impresión sensible al tacto, en la imagen del personaje, en la denominación al centro del billete y en las letras que indican Banco de Guatemala en la parte superior. Presente en las denominaciones de 10, 20, 50, 100 y 200 Quetzales.

Figura 3. **Billetes de Guatemala**

(a) Billete de un quetzal (b) Billete de cinco quetzales

(c) Billete de diez quetzales

Continuación de la figura 3.

(d) Billete de veinte quetzales

(e) Billete de cincuenta quetzales

(f) Billete de cien quetzales

(g) Billete de doscientos quetzales

Fuente: Banco de Guatemala. https://www.banguat.gob.gt/es/page/monedas-y-billetes-de-actual-circulacion. Consulta: 3 agosto de 2019.

- Imagen latente: al inclinar el billete y observarlo de forma horizontal se pueden observar las iniciales BG y la denominación, en una zona específica.

- Imagen de alto registro: imagen de un quetzal, ave símbolo, impresa en tonos intensos con tres colores observables presente solamente en los

billetes de 1 y 5 Quetzales.

- Imágenes coincidentes: imagen de la bandera que se corresponde perfectamente en ambos lados del billete al observarlo a trasluz.
- Imagen sombreada: imagen con diseño idéntico al busto principal en el anverso, ubicada en la parte derecha de la ventana transparente. Solamente para los billetes de 1 y 5 Quetzales.
- Resaltado seco: relieve sensible al taco ubicado en la parte izquierda de la ventana transparente. Solamente para los billetes de 1 y 5 Quetzales.
- Figuras con cambio de color: elementos del diseño que cambian de color al girar el billete y cambiar el ángulo de observación, para los billetes de 20, 50, 100 y 200 Quetzales.
- Marca de agua: imágenes observables a tras luz en los billetes de 10, 20, 50, 100 y 200 Quetzales
- Cinta entretejida: cinta que al observar el billete a trasluz, es posible leer un texto continuo que depende del tipo de la denominación, también cambia de color al girar el billete. Está presente en los billetes 20, 50, 100 y 200 Quetzales.

En la figura 4, a manera de ejemplo, se muestra la localización de los elementos de seguridad para los billetes de uno y cincuenta quetzales. Para la figura 4.a el número 1 corresponde a la imagen de alto registro; los números 2, 3 y 4 corresponde a la ventana transparente, al resaltado en seco a la imagen sombreada; el 5 a las imágenes coincidentes; el 6 al microtexto y por último el 7 a la imagen latente. En la figura 4.b el número 1 corresponde al alto relieve; 2 a la marca de agua, 3 a la cinta entretejida, 4 para una figura con cambio de color, 5 para una imagen latente, 6 y 7 para el microtexto, por último 8 para las imágenes coincidentes.

Figura 4. **Características de seguridad**

(a) Billete de un quetzal (b) Billete de cincuenta quetzales

Fuente: Banco de Guatemala. https://www.banguat.gob.gt/page/seguridad-en-los-billetes-0. Consulta: 4 noviembre de 2019.

3. INTRODUCCIÓN A LA VISIÓN POR COMPUTADOR

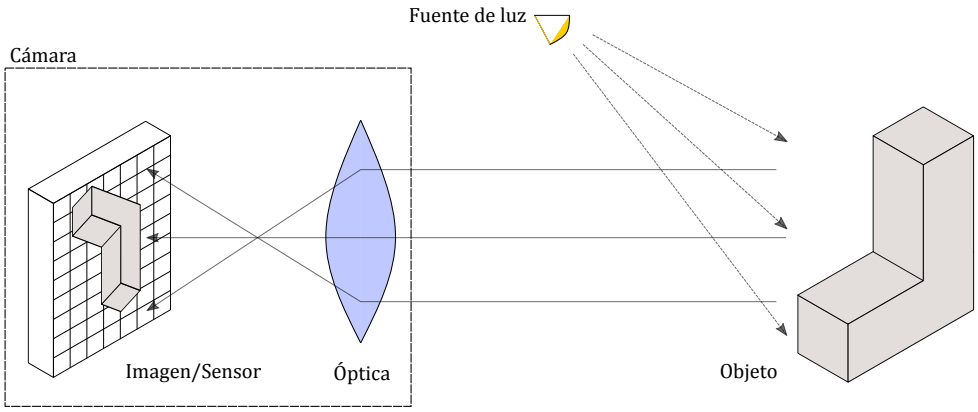
Diversos autores afirman que la visión por computador, también llamada visión artificial, es una rama de la inteligencia artificial especializada en imitar la visión humana, otros sin embargo consideran que es un campo aparte dentro de las ciencias de la computación, lo que es indudable es la íntima relación que existe entre la visión por computador y la inteligencia artificial, más aun cuando se tiene en consideración que el objetivo de la visión por computador es que un sistema computacional sea capaz de extraer información de las imágenes que se le presentan, e interpretarla para tomar decisiones. La visión por computador además aplica conocimiento de muchas otras áreas del saber como la óptica, la estadística inferencial, la neurología y el procesamiento de señales, por mencionar algunos.

El propósito de este capítulo es presentar brevemente algunos elementos que son esenciales para realizar tareas de visión por computador, como la forma en que los sistemas computacionales son capaces de capturar imágenes y representarlas adecuadamente, así como también se describen técnicas básicas del procesamiento de imágenes.

3.1. Formación de imágenes digitales

Una imagen es una proyección en dos dimensiones de una escena del mundo real, en el proceso para la formación de una imagen interviene diversos elemento, como una o más fuentes de luz, la escena o el objeto que se captura, un sistema óptico que colecta la luz, y un sistema de captura capaz de representar la imagen en un formar apropiada.

Figura 5. **Modelos simplificado de la formación de una imagen**



Fuente: Elaboración propia usando Inskape, 5 de agosto de 2019.

Para producir una imagen es necesario que una escena u objeto sea irradiado por una o varias fuentes de luz, esta interactúa con el objeto ya sea, por reflexión, refracción, absorción, difracción, etc. El sistema óptico consiste en una serie de lentes y mecanismo encargados de colectar la luz que viaja desde el objeto, para proyectarla adecuadamente en el sistema de captura.

El sistema de captura es un sensor sensible a la radiación lumínica que transforma la señal recibida en una señal eléctrica apta para su procesamiento, en la actualidad se emplean sensores CCD (*charge-couple device*) o CMOS (*complementary metal oxide semiconductor*), ambas tecnologías usan el efecto fotoeléctrico como principio de funcionamiento, pero cada una posee características que las hace más apropiadas para determinadas aplicaciones, por ejemplo el nivel de ruido al momento de capturar una imagen con un sensor CCD es menor que con un CMOS, también los sensores CCD tiene un rango dinámico más amplio, mientras que los sensores CMOS tienen un menor consumo de energía y una mayor velocidad.

Los sensores, tanto CCD como CMOS, están formados por celdas fotosensibles dispuestas en forma de rejilla rectangular donde incide la luz de la imagen, para el caso de los sensores CCD en cada celda se genera una carga eléctrica, proporcional a la intensidad de la luz que recibe, que carga un condensador. Una vez terminado el proceso de cargar un circuito de control desplaza la cargar de un condensador a otro dentro de la rejilla actuando como un registro de desplazamiento hasta que la carga se deposita en un amplificador que transfiera la señal a un convertidor analógico digital.

En los sensores CMOS la luz que impacta cada celda fotosensible afectan su conductividad provocando una corriente eléctrica proporcional a su intensidad, la señal de cada celda es amplificada individualmente y los valores son recuperados por el circuito de control mediante un esquema de multiplexación.

Es importante resaltar que el modelo presentado anteriormente (representado en la figura 5) es sumamente simplificado, dado que no toma en cuenta todos los factores que interviene en la formación de una imagen, como la iluminación de la escena, las longitudes de onda que componen la fuente de luz, la posición de la cámara o la distorsión ocasionada por dispositivos ópticos. Además, existen diversos métodos de captura que no siguen este modelo como las imágenes obtenidas mediante escáneres, donde el sensor no está dispuesto en forma de rejilla sino que es una sola fila de fotodiodos, o la formación de imágenes por ultrasonido que emplea transductores piezoeléctricos para la emisión y recepción de las ondas acústicas.

3.1.1. Captura del color

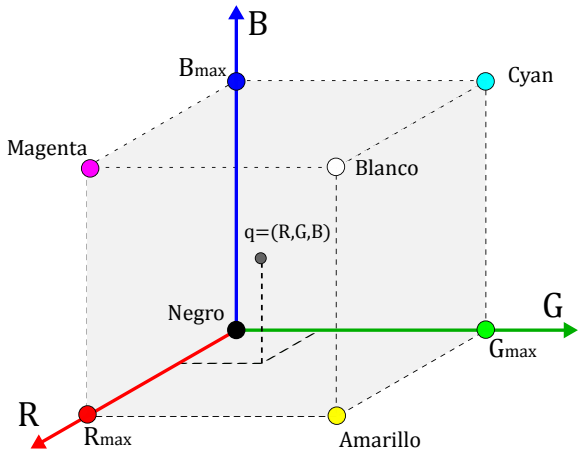
La percepción del color es la interpretación que el cerebro da a las señales nerviosas causas por diferentes longitudes de onda del espectro visible, ubicadas entre los 400 y $700\ nm$. El sistema visual humano pose tres tipos de células fotorreceptoras llamadas conos, cada tipo es aproximadamente sensible a una longitud de onda que corresponde al color rojo, al verde o al azul. Se denomina a este tipo de visión tricromática y es debido a la proporción con que la luz estimula cada uno de los tipos de conos, que se formar la gama de colores visibles.

La visión humana es solamente capaz de reconocer unas pocas docenas de tonos de gris mientras que es capaz de reconocer cientos de miles de diferentes colores, debido a esto mucha de la información que un ser humano puede obtener mediante la visión se debe a su capacidad para discriminar los colores. La información contenida en el color es útil para resolver muchos problemas en el ámbito de la visión por computador, a pesar de que los sistemas computacionales no son capaces de representar satisfactoriamente todas las características de la visión, como lo es la invariancia al color debido a cambios en la iluminación.

Debido al tricromatismo de la visión es posible la existencia del modelo de color RGB, que representa cada color por la adición de cada una de las componentes primarias rojo, verde y azul. El espacio de color que conforma este modelo se puede visualizar en coordenadas cartesianas al representar cada uno de los ejes coordenados como una de las componentes del modelo RGB, entonces el espacio de color está conformado por el conjunto de puntos dentro y sobre un cubo, definido por una diagonal, que se traza desde el origen (color negro) hasta la intersección de los valores máximos de cada componente (color

blanco), esta diagonal representa la escala de grises. Ver figura 6.

Figura 6. **Espacio de color RGB**



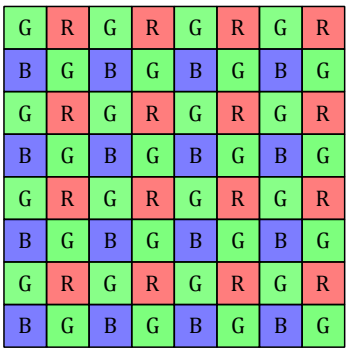
Fuente: Elaboración propia usando Inskape, 5 de agosto de 2019

La mayoría de las cámaras digitales y las pantallas de los dispositivos de uso diario, emplean este modelo para representar el color. Un sensor CMOS o CCD no es por sí solo capaz de captura el color de una imagen estos registra únicamente la intensidad que reciben produciendo así imágenes monocromáticas (escala de grises), para que una cámara se capaz de capturar el color básicamente existen dos métodos, el primero implica el uso de un prisma para separar la luz junto a tres sensores, uno para cada canal RGB; para el segundo se emplea solamente un sensor al que se le coloca un filtro de Bayer, siendo este último el de más amplio uso por ser más barato de implementar, mientras que el primero se reserva situaciones donde sea necesario tomar imágenes de alta calidad y mayor resolución.

Un filtro de Bayer es un filtro de luz que tiene un patrón específico, como el que se muestra en la figura 7, permite captar la intensidad de los tres canales RGB al permitir el paso de solo una longitud de onda por celda fotosensible. La

información para formar cada pixel de una imagen capturada con este método procede entonces de cuatro celdas fotosensibles (verde, rojo, azul, verde). La razón para emplear una mayor número de puntos de color verde es que el ojo humano es más sensible a ese color.

Figura 7. **Patrón de Bayer**



Fuente: Elaboración propia usando Inskape, 15 de agosto 2019.

3.2. Representación de una imagen

3.2.1. Imágenes como funciones

Una imagen puede representarse como una función discreta con un dominio que es un conjunto de coordenadas discretas y finitas en forma de una rejilla rectangular. La función mapea el conjunto de coordenadas hacia un conjunto de valores vectoriales en $\mathbb{R}^n$, donde n equivale a la dimensión de la señal, para imágenes monocromáticas $n = 1$ y para imágenes a color $n = 3$. Entonces una la función se define como:

$$f : \mathbb{X} \to \mathbb{R}^n \tag{3}$$

Donde, $\mathbb{X}$ representa al conjunto de coordenadas que son posiciones validas, que a su vez es un subconjunto de los números naturales. Formalmente

$\mathbb{X} = \{0, 1, 2, ..., D-1\}^d \in \mathbb{N}^d$, la d denota el número de dimensiones espaciales del conjunto de coordenadas. Se puede decir entonces que $f(\vec{x})$ es una función que devuelve un vector que representa lo composición espectral de la imagen en la posición $\vec{x}$.

Si el rango $\mathbb{R}^n$ de la función $f(\vec{x})$ es limitado a un conjunto valores que pertenecen a $\mathbb{N}^n$ se tiene una imagen digital.

De la definición anterior se puede entender que para el caso de imágenes monocromáticas $f(\vec{x})$, reescrita como $f(x, y)$ para cuando $d = 2$, retorna únicamente un valor de intensidad I fácil de representar junto a (x, y) en un sistema cartesiano tridimensional, mientras que para el caso de imágenes a color f se puede interpretar como una regla de correspondencia entre el conjunto de puntos (x, y) en un plano hacia puntos dentro y sobre el cubo RGB.

3.2.2. Imágenes como matrices

Una imagen digital es discreta tanto en el espacio en que está definida (coordenadas discretas), así como en los valores de intensidad, de cada componente espectral, asociados a cada posición. Se suma a este hecho que en la mayoría de los casos se trabajara con imágenes con dominio definido en dos dimensiones $(d = 2)$, por lo que se puede representar una imagen como una matriz en la que la fila y la columna representan un punto en la imagen mientras que el valor del elemento en la matriz corresponde a la intensidad en ese punto.

La matriz $\mathbf{F}$ representa a la imagen definida por la función f si cumple que

$$I_{ij} = f(i, j) \tag{4}$$

y además

$$\mathbf{F} = \begin{pmatrix} I_{0,0} & I_{0,1} & \dots & I_{0,n-1} \\ I_{1,0} & I_{1,1} & \dots & I_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ I_{m-1,0} & I_{m-1,1} & \dots & I_{m-1,n-1} \end{pmatrix} \tag{5}$$

Para el caso de una imagen a color, las componentes son valores que depende de la posición pero independientes entre sí, es posible representar una imagen como un arreglo de tres matrices, una por cada canal.

3.3. Procesamiento de imágenes

El propósito procesar una imagen es resaltar alguna característica de interés o eliminar información innecesaria para la tarea que se quiere ejecutar, básicamente lo que hacen las técnicas y algoritmos contenidos dentro de este campo es, tomar una imagen para producir otra con mejor calidad o en la cual es más fácil buscar información. En esta sección como en las secciones posteriores se discute de manera general algunas de las operaciones y algoritmos que se emplean para alcanzar este objetivo.

3.3.1. Tipos de imágenes

La representación matricial de una imagen es muy útil en sistemas computacionales, debido a que facilita la manipulación de una imagen mediante álgebra lineal. Como se mencionó anteriormente la posición de los punto que conforman una imagen y la intensidad de cada uno son variables discretas, para cuantificar la intensidad generalmente se usan 8 bits por lo que los valores de intensidad van de 0 a 255.

A continuación se presentan algunos de los tipos de imágenes que es posible distinguir en base a los valores de intensidad presentes en las matrices que las forman.

3.3.1.1. Imagen RGB

Se representa la imagen en forma de un arreglo de tres matices, uno para cada canal, con valores que especifican intensidad. Cada matriz tiene un tamaño igual a la imagen original, también cada una corresponde a uno de los colores primarios, rojo, verde y azul, estableciendo cuanto de cada uno de estos colores debe ser usado en un determinado píxel. La figura 8 muestra una imagen de color separada en los tres canales que la compone, en la parte superior derecha se muestra el canal rojo, en la inferior izquierda el canal verde y el rojo en la inferior derecha.

3.3.1.2. Imagen en escala grises

Referidas en algunas ocasiones como imágenes monocromáticas, cada uno de los elementos de la matriz que las conforman tienen un valor entre 0 y 255, cuando el valor sea 0, el píxel en concreto, tendrá el color negro más intenso. Por lo contrario, si el valor de un elemento es 255, el píxel pasará a ser completamente blanco, y los valores en el medio se interpolan linealmente para obtener diferentes tonalidades de gris.

Un imagen en escala de grises puede obtenerse a partir de una imagen de color, existen diferentes algoritmos para llevar acabo la tarea y cada uno resalta diferentes características de la imagen, los más sencillos implican multiplicar por un factor escalar (a, b o c, según el canal), cada una de las matrices que

Figura 8. **Descomposición de canales de color**

Fuente: Elaboración propia usando OpenCV, basado en https://pixabay.com/es/orange-kiwi-madera-mesa-azul-1599264. Consulta: 5 septiembre de 2019

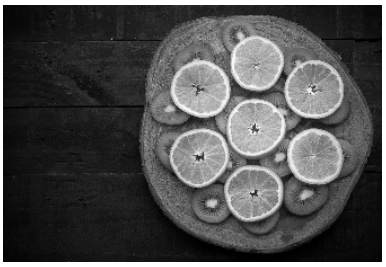
representan la imagen ($\mathbf{R}$, $\mathbf{G}$ y $\mathbf{B}$) y luego sumarlas elemento por elemento para obtener la imagen. La siguiente ecuación ilustra esta situación

$$\mathbf{F} = a \cdot \mathbf{R} + b \cdot \mathbf{G} + c \cdot \mathbf{B} \tag{6}$$

3.3.1.3. Imagen binaria

Una imagen binaria, es una imagen en la que cada píxel toma uno de dos valores discretos 1 o 0. Normalmente el 1 representar el color blanco y el 0 el color negro. La imagen se obtiene partiendo de una imagen en escala de grises a la cual se le aplica un umbral t sobre el valor de intensidad para cada punto, de modo que para cualquier elemento f_{ij} de la matriz que representa la imagen, si es mayor que el umbral se establecen en 1 y para cualquier otro caso se establece

Figura 9. **Imagen en escale de grises**



Fuente: Elaboración propia usando OpenCV, basado en https://pixabay.com/es/orange-kiwi-madera-mesa-azul-1599264. Consulta: 5 septiembre de 2019

en 0.

$$f_{ij} = \begin{cases} 1 & si \;\; f_{ij} \geq t \\ 0 & otro\ caso \end{cases} \tag{7}$$

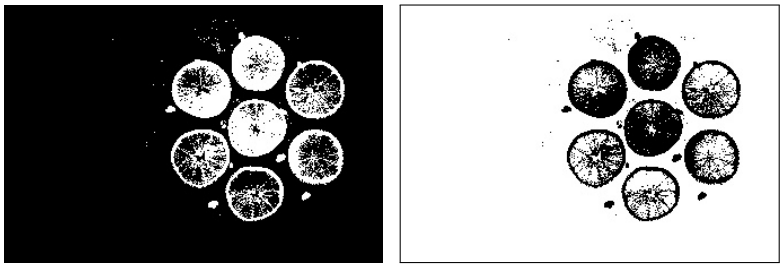
Es posible aplicar un umbral invertido de modo que para cualquier intensidad mayor o igual que t se asigna el 0 y 1 para otro caso. La figura 10 muestra dos imágenes binarias, del lado derecho se muestra una imagen con un umbral invertido.

$$f_{ij} = \begin{cases} 0 & si \;\; f_{ij} \geq t \\ 1 & otro\ caso \end{cases} \tag{8}$$

3.4. Transformaciones geométricas

Estas transformaciones son una manipulación de la imagen que provoca un cambio de la posición de los píxeles. Es importante decir que cuando se trabaja con imágenes se utiliza un sistema coordenado donde el origen se sitúa en la esquina superior izquierda de la imagen, el eje x se utiliza para desplazarse horizontalmente sobre los píxeles de izquierda a derecha y el eje y

Figura 10. **Imagen binaria**



Fuente: Elaboración propia usando OpenCV, basado en https://pixabay.com/es/orange-kiwi-madera-mesa-azul-1599264. Consulta: 5 septiembre de 2019

para hacerlo verticalmente de arriba hacia abajo.

La posición de un pixel en una imagen pude ser representado como un par de valores en forma de vector columna o alternativamente también como vector columna en coordenadas homogéneas

$$\vec{x} = \begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow \vec{x}_h = \begin{pmatrix} \alpha x \\ \alpha y \\ \alpha \end{pmatrix} \tag{9}$$

dado que una transformación lineal puede expresarse como una matriz, la representación de vector columna permite obtener la nueva posición del pixel mediante una multiplicación matricial. Para la representación en coordenadas homogéneas se emplean un factor de escala α el valor de este factor es arbitrario pero no igual a cero, cuando se habla de transformaciones geométricas elementales (rotación, traslación y escalado), generalmente se escoge $\alpha = 1$.

Como principal causa para representar un vector mediante coordenadas

homogéneas se tiene que al realizar varias transformaciones a una imagen, su aplicación se reduzca a una multiplicación sucesiva de matrices o a una concatenación de las transformaciones, donde una única matriz representa al mismo tiempo las operaciones de traslación, rotación y escalado.

3.4.1. Rotación

Consiste en la rotación de un ángulo θ en el plano de la imagen, sobre el origen y en contra de las agujas del reloj si el ángulo es positivo

$$\vec{u} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \cos\theta & \operatorname{sen}\theta \\ -\operatorname{sen}\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \mathbf{R}_\theta \vec{x} \tag{10}$$

para coordenadas homogéneas la transformación puede expresarse como

$$\vec{u}_h = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & \operatorname{sen}\theta & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \mathbf{R}_{\theta h} \vec{x}_h \tag{11}$$

3.4.2. Escalamiento

Esta transformación implica variar el tamaño de la imagen original. La variación implica multiplicar cada coordenada por un factor s cuando se trata de un escalamiento isotrópico, si $s < 1$ la imagen reduce su tamaño y si $s > 1$ el tamaño aumenta, está definida como

$$\vec{u} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} s & 0 \\ 0 & s \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \mathbf{Z}_s \vec{x} \tag{12}$$

en coordenadas homogéneas puede expresarse la operación de escalamiento como se indica a continuación

$$\vec{u}_h = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \mathbf{Z}_{\mathbf{s}h}\vec{x}_h \tag{13}$$

3.4.3. Traslación

La traslación consiste en mover un pixel desde la posición desde su posición original (x, y) hasta que alcance la posición $(x + t_x, y + t_y)$, esta operación corresponde a una suma vectorial entre $[x \;\; y]^T$ y $[t_x \;\; t_y]^T$, pero puede ser expresada en coordenadas homogéneas donde corresponde también a una multiplicación

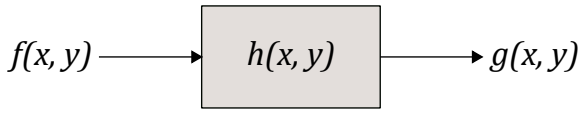
$$\vec{u}_h = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \mathbf{T}_{(t_x,t_y)}\vec{x}_h \tag{14}$$

3.5. Filtrado de imágenes

En el procesamiento de imágenes un filtro tiene como propósito resaltar información, como por ejemplo el aumento de contraste, y/o la eliminación de ruido para facilitar el procesamiento en etapas posteriores. Consiste en aplicar una transformación T, a la función f de la imagen para obtener una nueva función g que representa la imagen con las características requeridas.

$$g(x, y) = T\left[f(x, y)\right] \tag{15}$$

Figura 11. **Proceso de filtrado**



Fuente: Elaboración propia usando Inskape, 8 de septiembre de 2019.

Dado que un filtro es un sistema lineal puede entonces ser caracterizado por su respuesta al impulso $h(x, y)$ por lo que al pasar una señal f por el sistema, su salida g es la convolución entre la respuesta al impulso y la señal de entrada.

$$g(x, y) = h(x, y) * f(x, y) \tag{16}$$

Por el teorema de convolución, la expresión anterior es equivalente a la multiplicación de ambas funciones en el dominio de la frecuencia.

$$G(u, v) = H(u, v)F(u, v) \tag{17}$$

Existen entonces dos caminos para filtrar una imagen, mediante la operación de convolución en el dominio espacial o mediante la aplicación de la transformada de Fourier para pasar las señales h y f al dominio de la frecuencia, multiplicarlas y para luego retornarlas al dominio espacial con la transformada inversa.

3.5.1. Filtros de frecuencia

Como fue mencionado antes el filtrado de una imagen también puede realizarse en el dominio de la frecuencia, para ello es necesario el cambio del dominio sobre el que está definida la función. La transformada de Fourier es la operación que permite este cambio, al aplicar la transformación es posible

determinar el contenido espectral o contenido por frecuencias que conforman una función, está definida como

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux} dx \tag{18}$$

para obtener la función $f(x)$ a partir de $F(u)$ la transformada inversa de Fourier está definida como

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux} du. \tag{19}$$

Cuando se trabaja con imágenes sus funciones están definidas en un plano por lo que es necesario extender el análisis de Fourier a funciones de dos variables, entonces el par de transformadas está definido como

$$F(u,v) = \int_{-\infty}^{\infty} f(x,y) e^{-j2\pi(ux+vy)} dxdy \tag{20}$$

$$f(x,y) = \int_{-\infty}^{\infty} F(u,v) e^{j2\pi(ux+vy)} dudv \tag{21}$$

en los pares de transformadas anteriores se considera a las funciones $f(x)$, $F(u)$, $f(x,y)$ y a $F(u,v)$ como funciones continuas.

Un desarrollo teórico más amplio sobre las propiedades de la transformada de Fourier y el análisis armónico, es largo como extenso, además se encuentra fuera de los objetivos de este trabajo, por lo que se saltara directamente a la transformada discreta de Fourier (DFT, por sus siglas en ingles), que es la base a partir de la cual se construyen los algoritmos para el filtrado en frecuencia.

Considerando a $f(x)$ como una función discreta que contiene N muestras separadas una de otra por una distancia Δx, la transformada discreta de Fourier

está definida por

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)e^{\frac{-j2\pi ux}{N}}; \qquad u = 0, 1, 2, ..., N-1 \tag{22}$$

$$f(x) = \sum_{u=0}^{N-1} F(u)e^{\frac{j2\pi ux}{N}}; \qquad x = 0, 1, 2, ..., N-1 \tag{23}$$

para cuando se tiene una función discreta de dos variables $f(x, y)$ con M muestras para x y N muestras para y se tiene las siguientes expresiones

$$F(u, v) = \frac{1}{MN}\sum_{y=0}^{N-1}\sum_{x=0}^{M-1} f(x, y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{24}$$

$$f(x, y) = \sum_{v=0}^{N-1}\sum_{u=0}^{M-1} F(u, v)e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}. \tag{25}$$

El siguiente paso luego de obtener la transformada discreta de Fourier de una imagen, es multiplicar por alguna función de transferencia $H(u, v)$ que esta definida según el tipo de filtro que se quiera implementar.

3.5.1.1. Filtro pasa bajos

Este filtro como indica su nombre permite el paso a las bajas frecuencia y atenúa las componentes espectrales de alta frecuencia, esta operación se traslada al dominio espacial como el suavizado de los detalles de la imagen.

- Filtro Ideal: La función de transferencia $H(u, v)$ más sencilla para un filtro pasa bajo es aquella que simplemente anula las componentes de alta frecuencia, está es la función de un filtro pasa bajo ideal en dos

dimensiones y está definida como

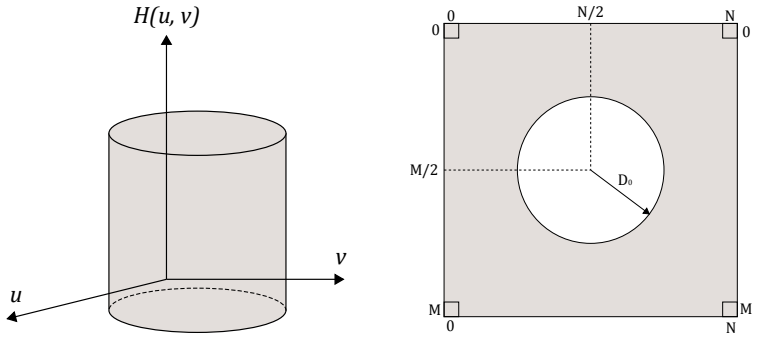
$$H(u,v) = \begin{cases} 1 & si \quad D(u,v) \geq D_0 \\ 0 & si \quad D(u,v) < D_0 \end{cases} \tag{26}$$

donde D_0 es un valor positivo que representa la frecuencia de corte y $D(u,v)$ es la distancia desde el centro del espectro de frecuencias de la imagen hacia un punto (u,v) dentro del espectro y está dada por

$$D(u,v) = \sqrt{(u - \frac{M}{2})^2 + (v - \frac{N}{2})^2}. \tag{27}$$

En la figura 12 se muestra una representación gráfica del filtro ideal como se pude ver en esencia es un cilindro de altura 1, para cualquier valor bajo de frecuencia que se encuentre cerca del centro del espectro de frecuencias el filtro no actuá, pero cualquier valor de frecuencia alto mayor a D_0 queda anulado.

Figura 12. **Filtro Ideal**



Fuente: Elaboración propia usando Inskape, 10 de septiembre 2019.

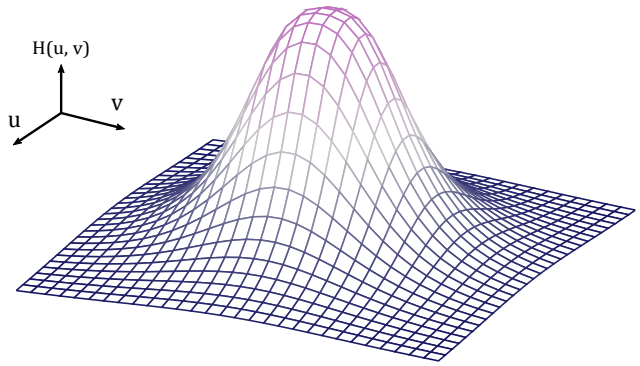
- Filtro de Butterword: Este tipo de filtro tiene una curva más suave, como la que se puede apreciar en la figura 13, que ofrece mejores resultados

y evita que aparezcan patrones indeseados en el espectro de frecuencia como si sucede con el filtro ideal. La función de transferencia de un filtro de Butterword pasa bajo de orden n con una frecuencia de corte D_0, es

$$H(u,v) = \frac{1}{1 + \left[\frac{D(u,v)}{D_0}\right]^{2n}} \tag{28}$$

dado el tipo de curva suave que dibuja la función de transferencia la frecuencia de corte está definida, como el punto en el que $H(u,v) = 1/2$.

Figura 13. **Filtro Butterword**



Fuente: Elaboración propia usando pgfplot, 25 septiembre 2019.

3.5.1.2. Filtro pasa altos

Los filtros pasa altos se encargar en atenuar las componentes de baja frecuencia mientas no perturba la componentes de alta frecuencia, debido a que los cambios brusco en la intensidad, como bordes en la imagen, están asociados a las componentes de alta frecuencia, un filtro pasa altos actúa como un filtro de acentuación en el dominio espacial.

Debido a que un filtro pasa alto actúa precisamente en forma contraria a un

filtro pasa bajo, las funciones transferencia de los filtros pasa alto pueden obtener a partir de las funciones de los filtros pasa bajo, de la siguiente manera

$$H_{hp}(u,v) = 1 - H_{lp}(u,v) \tag{29}$$

donde $H_{lp}(u,v)$ representa la función de transferencia del filtro pasa bajo y $H_{hp}(u,v)$ la función de transferencia del filtro pasa alto correspondiente.

3.5.2. Filtros de convolución

Cuando se realiza el filtrado en el dominio espacial, las operaciones se hacen directamente sobre los píxeles de la imagen. Para obtener cada uno de los píxeles que conforman la imagen filtrada se utiliza un pixel de referencia y los píxeles a su alrededor ubicados dentro de una ventana de tamaño fijo. Esta ventana denomina máscara o *kernel*, es una pequeña matriz con valores predeterminados escogidos para resaltar o atenuar alguna propiedad de la imagen.

La operación de convolución se traduce entonces en desplazar la máscara a lo largo de todos lo píxeles de la imagen, como se muestra en la figura 14, y con cada iteración se multiplica cada pixel de la imagen, dentro del área común, con el correspondiente elemento de la máscara, luego se suman todo los valores obtenidos.

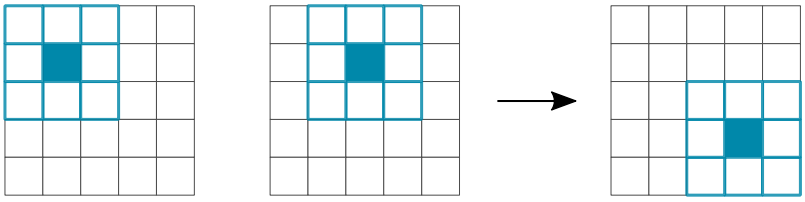
La siguiente ecuación expresa la aplicación del filtro sobre una imagen

$$g(x,y) = \sum_{(k,l)} f(x+k, y+l)h(k,l) \tag{30}$$

donde h es la máscara y (k,l) representa el recorrido sobre el tamaño de la

máscara por ejemplo en si tenemos una máscara de con 3 filas y 3 columnas entonces $(k, l) = \{(0, 0), (0, 1), ..., (2, 2)\}$.

Figura 14. **Desplazamiento de la máscara**



Fuente: Elaboración propia usando Inskape, 10 de octubre de 2019.

3.5.2.1. Filtros de suavizado

Son un grupo de filtros empleados para eliminar ruido en la imagen debido a un mal muestreo o detalles pequeños que no son de interés. Son análogos a los filtros pasa bajos en el dominio de la frecuencia. Existen varios tipos de filtro, según los valores que toman los elementos de la máscara:

- Media móvil: Es un filtro que promedia el valor de todos los píxeles dentro del área de la máscara, esta es cuadrada y todos sus valores son iguales a $1/S$, donde S representa el número de píxeles en la máscara.
- Mediana: En este filtro todos los elementos de la máscara son 1, no realiza una sumatoria sino que retorna el valor de la mediana, del conjunto de pixeles que caen dentro de la máscara.
- Gaussiano: Utiliza una máscara que es una aproximación a una distribución gaussiana para dos variables. Asumiendo una varianza y un media iguales para ambos eje los valores de la máscara están dados por

$$h(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{31}$$

El valor máximo debe centrarse en el pixel de referencia y a partir la función disminuye en función de la varianza σ^2 hacia los píxeles que lo rodean.

3.5.2.2. Filtros de acentuación

En este tipo de filtro el objetivo es realzar los detalles y los cambios repentinos de intensidad, el resultado de esto es que los bordes de la imagen, donde la intensidad cambia bruscamente, son acentuados; mientras que las zonas uniformes son atenuadas. Son análogos a los filtros pasa alto, en el dominio de la frecuencia.

Una de las formas como se implementan es mediante una máscara de valores con una sumatoria de cero, como la que se muestra a continuación

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \tag{32}$$

dado que la suma de los elementos de la máscara es cero, si todos los píxeles que caen dentro de la máscara tiene el mismo valor (región con intensidad constante) la operación retorna un valor de cero, en cambio sí en la región hay un pixel con diferente intensidad la operación retorna un valor distinto de cero y mientras más grande la diferencia mayor el valor de retorno.

Otro método de para realizar este tipo de filtrado consiste en aplicar un filtro de suavizado a la imagen y luego restar la imagen resultante elemento por elemento de la imagen original

$$\mathbf{G} = \mathbf{F} - \mathbf{F}' \tag{33}$$

donde F es la matriz de la imagen original F' el resultado de aplicarle un filtro de suavizado a la imagen F y G es el resultado del filtro de acentuación.

4. REDES NEURONALES EN LA VISIÓN POR COMPUTADOR

Como se mencionó al inicio del capítulo anterior el principal objetivo de la visión por computador es enseñar a un sistema computacional a interpretar la información que forman una imagen (píxeles) de la manera en la que lo logran los seres humanos, esta tarea tan genérica se encuentra segmentada en los principales problemas que concierne a este campo, como reconocimiento de contenido, análisis de video y visión 3D. Al inicio de este capítulo se explora el problema del reconocimiento de contenido, directamente relacionado con el sistema identificador de moneda.

En años recientes dentro del campo de la visión por computador surgió una nueva ola de innovación alimentada por la gran cantidad de datos disponibles en la red y el crecimiento en el poder de procesamiento de las computadoras, que permitieron el empleo de algoritmos de aprendizaje profundo para construir soluciones más eficientes y precisas para los problemas tradicionales de área.

Las redes neuronales forman el núcleo del aprendizaje profundo y son herramientas muy poderosas para las tareas de visión por computador. En este capítulo se dará una breve descripción a su funcionamiento, la teoría que se encuentra detrás, luego se discutirán las redes neuronales convolucionales y su aplicación en tareas reales de visión por computador.

4.1. Reconocimiento de contenido

Una de las principales tareas que aborda la visión por computador es el reconocimiento del contenido, otorgarle sentido a las imágenes, en esencia

extraer significado o información semántica a partir los píxeles, como por ejemplo identificar los tipos objetos presentes en las imágenes, su posición, orientación o cantidad. Este complejo problema es usualmente dividido por los investigadores del área en algunos subdominios que se enumeran brevemente a continuación.

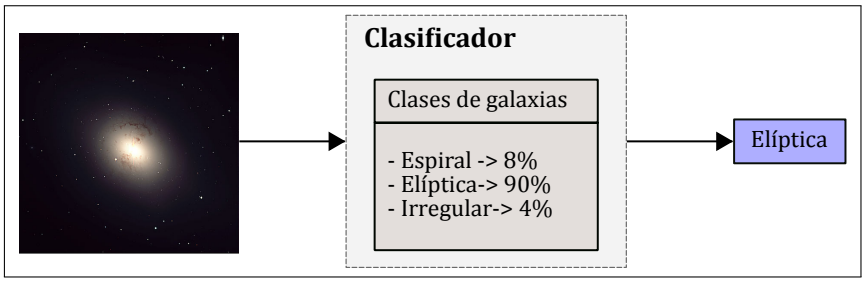
4.1.1. Clasificación de objetos

La clasificación de una imagen consiste en que un sistema computacional tome una imagen como entrada y de forma autónoma devuelva una o varias etiquetas, de un conjunto definido, en función del contenido de la imagen.

Si suponemos por ejemplo, que dado un conjunto de imágenes de galaxias se desea separarlas en tres categorías básicas (espirales, elípticas e irregulares) que representan la forma de la galaxia. La función del sistema es asignar una etiqueta a partir del gran conjunto de números para establecer a que categoría pertenece la imagen. En la figura 15 se muestra un diagrama simplificado de la función del clasificador.

Este problema, trivial para un ser humano, es difícil de abordar para una computadora debido a que el contenido dentro de la imagen puede presentar muchas variaciones como, cambios de tamaño, puntos de vista, rotaciones de los objetos dentro de la imagen, cambios en la condición de iluminación, cambios en la escala, por mencionar algunos. Sin embargo la solución a este problema fue el primer gran éxito de los modelos basados en redes neuronales, tal fue el progreso en esta tarea que incluso para algunos casos de uso se ha logrado alcanzar un desempeño sobre humano. Una aplicación real para las herramientas de clasificación de objetos es el reconocimiento de caracteres escritos a mano para la digitalización de documentos.

Figura 15. **Ejemplo de un clasificador de imágenes de galaxias**



Fuente: Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Giant _Elliptical_Galaxy_NGC_1316_in_Fornax_Cluster.jpg. Consulta: 2 enero de 2020.

4.1.2. Identificación de objetos

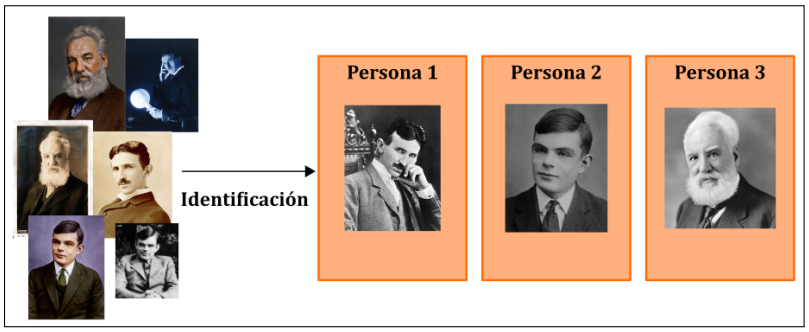
Debido a las variaciones que pueden presentar los objetos dentro de una imagen, incluso si pertenecen a una misma clase, como tamaño, color o forma, se puede hablar una ocurrencia concreta, una instancia de clase. A diferencia de la clasificación de objetos que asigna una etiqueta a una imagen de un conjunto previamente definido, la tarea de identificación consiste en reconocer la instancia específica de una clase.

Por ejemplo, un programa clasificador de objetos podría aprender a separar imágenes que contengan rostros humanos de las que no, mientras que un identificador estaría centrado en aprender las características de los rostros para identificar a las personas y reconocerlas en otras imágenes, como se muestra en la figura 16.

4.1.3. Detección de objetos

Implica localizar todas las instancias del objeto de interés dentro de una imagen, con las arquitecturas y métodos de aprendizaje profundo desarrollados

Figura 16. **Ejemplo de identificación de objetos**



Fuente: Elaboración propia usando Inskape, 5 de enero de 2020.

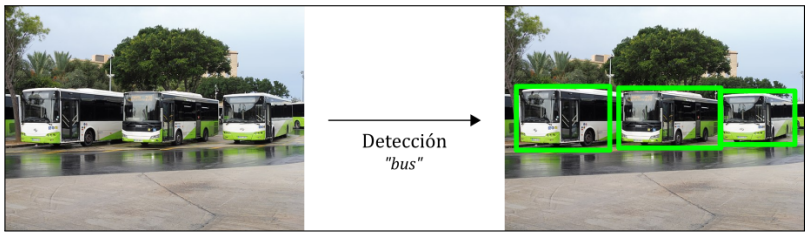
específicamente para esta tarea, se abre la capacidad para rastrear el movimiento de un objeto, contar el número de ocurrencias o recorte automático de imágenes, entre otros.

La detección de objetos es a menudo un paso previo para realizar tareas más complejas, como el reconocimiento facial, control de calidad en líneas de producción o navegación de vehículo autónomos. Por lo general una herramienta para la detección de objetos toma una imagen como entrada y retorna un recuadro que delimita el área que ocupa cada objeto, indica su posición relativa a otros elementos en la imagen y la clase a la que pertenece, como se aprecia en la figura 17.

4.1.4. Segmentación

A diferencia de la detección de objetos donde simplemente se retorna un recuadro para cada elemento que estima el área rectangular y su posición; un método de segmentación retornar una máscara que cubre cada píxel de todos los objetos que pertenecen a una clase o retorna una máscara por cada instancia de clase presente en la imagen, esta es una tarea mucho más compleja, porque

Figura 17. **Detector de buses que delimita candidatos**



Fuente: Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Malta_Buses.jpg. Consulta: 2 de enero de 2020.

implica dibujar un contorno sobre la silueta de cada objeto, en la figura 18 se aprecia, que en el método segmentación de objetos se retorna una sola máscara para todos los buses, en la segmentación de instancia se retorna una máscara que etiqueta cada instancia de bus que se reconoce.

Figura 18. **Ejemplo de segmentación de imagen**

Fuente: Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Malta_Buses.jpg. Consulta: 2 de enero de 2020.

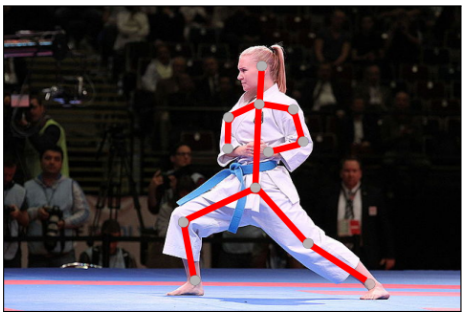
4.1.5. Estimación de pose

Esta tarea puede referirse a técnicas diferentes, dependiendo del problema que se quiera solucionar, para cuerpos rígidos se refiere a la estimación de la posición y orientación de los objetos relativos a una cámara en el espacio

tridimensional, como los algoritmos empleados para que un robot interactué con su entorno (tomar de objetos, evitar colisiones, navegación autónoma) o a la correcta ubicación de elementos cuando se trabaja con realidad aumentada.

También puede significar la estimación de la posición de las partes que conforman un objeto, relativas entre sí, como por ejemplo cuando se intenta reconocer poses humanas para extraer información del lenguaje corporal.

Figura 19. **Estimación de pose humana**



Fuente: Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Karate_WM_2014_(2)_017.JPG. Consulta: 2 de enero de 2020.

4.2. Red neuronal artificial

Las redes neuronales artificiales son modelos ligeramente inspirados en el funcionamiento del cerebros biológicos. Por ejemplo el cerebro humano es una red extremadamente compleja de neuronas que intercambian información así como transforman la información sensorial de entrada en pensamientos y acciones. Cada neurona biológica es una célula dedicada a procesar información, son los componentes principales de sistemas complejos como el sistema nervioso o el cerebro humano. La idea básica de su funcionamiento consiste en que una neurona recibe tanto señales eléctricas como químicas por medio de la conexión con otras neuronas (sinapsis) a través de sus dendritas y

transmite las señales generadas por el cuerpo principal de la célula (soma) mediante el axón.

Concretamente las dendritas junto al soma recibe señales de entrada; el cuerpo celular las incorpora e integra y emite señales de salida (ver figura 20). Las señales de entrada pueden ser atenuadas o amplificadas dependiendo la sinapsis de origen, la combinación de las señales inhibe o activa la neurona, si la estimulación acumulada excede un umbral específico, la neurona se activa y un impulso eléctrico es propagado a la siguiente neurona a través de los terminales del axón, es una situación de todo a nada. Se puede pesar en las neuronas como unidades simples de procesamiento de señal que al ser agrupadas en redes desarrollan propiedades emergentes, como el pensamiento o la conciencia.

Las redes neuronales artificiales (RNA) por otra parte no son más que aproximaciones no lineales de la manera en que funciona el cerebro humano porque solo emulan una parte muy simple y no están basadas únicamente en redes biológicas; por lo tanto las RNA son incapaces de alcanzar el nivel tan alto de complejidad de las redes biológicas. Para describir las redes neuronales artificiales es necesario definir primero la unidad fundamental que las conforma.

4.2.1. Neurona artificial

En el modelo de la neurona artificial propuesto a partir del comportamiento simplificado que muestra la contraparte biológica, se identifican tres elementos básicos:

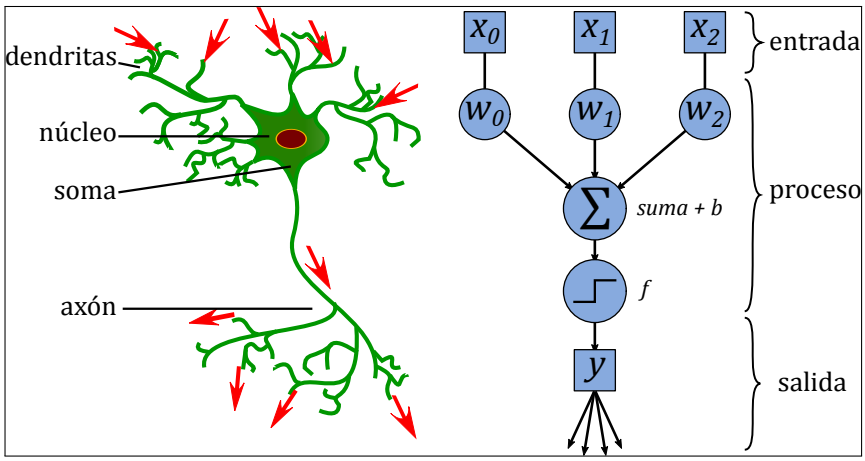
- Entradas: que son un conjunto de enlaces de conexión caracterizado por los pesos w_i por los que ingresan las señales x_i, donde los pesos pueden

tomar un rango de valores tanto positivos como negativos.

- Sumador: para realizar la suma ponderada de las señales, esta operación representa una combinación lineal de la entrada.
- Función de activación: encargada de limitar la amplitud a la salida de la neurona, delimita el rango de la señal de salida y a valores finitos.

La neurona artificial toma los valores de entrada, realiza una suma ponderada y finalmente aplica una función de activación para obtener una señal de salida, esto se puede ver en el gráfico 20.

Figura 20. **Neurona biológica vs artificial**



Fuente: Wikimedia Commons. https://commons.wikimedia.org/wiki/File:ANN_neuron.svg. Consulta: 3 de enero 2020.

Debido a la presencia de un peso en cada entrada los valores son escalados en función de un peso específico. Estos pesos son los parámetros que son ajustados durante la fase de entrenamiento con la intención de que la neurona reaccione a las características correctas. Otro parámetro que se añade al proceso de entrenamiento es el sesgo (*bias* por su nombre en inglés) un ajuste que se encarga de disminuir o incrementar el valor total de la señal de

salida, considerado un parámetro opcional.

Formalizando el proceso matemáticamente, si se tiene una neurona con dos valores de entrada x_0 y x_1. Cada uno de esos valores será ponderado por un peso w_0 y w_1 antes de ser sumados y luego combinados junto al *bias*, b. Se puede expresar los valores de entrada como un vector fila y los valores de los pesos con un vector columna:

$$x = (x_0 \;\; x_1), \quad w = \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} \tag{34}$$

La operación previa a la función de activación se puede expresar de manera directa como:

$$z = x \cdot w + b \tag{35}$$

donde el producto punto de los vectores conforma la suma ponderada.

$$z = x \cdot w + b = \sum_i x_i \cdot w_i + b = x_0 w_0 + x_1 w_1 + b \tag{36}$$

Es necesario aplicar la función de activación al resultado z para obtener la salida de la neurona. Como se menciona anterior mente una neurona biológica se activa o inhibe dependiendo de la estimulación que recibe, una función de activación binaria tal que si el valor de z esta por arriaba de una umbral t retorno una valor de 1 o de lo contrario retorna 0 puede emular de forma elemental el comportamiento de salida de una neurona biológica. Formalmente la función de activación se expresa como:

$$y = f(z) = \begin{cases} 0 & z < t \\ 1 & z \geq t \end{cases} \tag{37}$$

4.2.2. Perceptrón

El modelo para neurona artificial representado por las ecuaciones 35 y 37, fue el primero o uno de los primeros modelos computacionales propuesto para el aprendizaje supervisado basado en redes neuronales. Inicialmente diseñado para realizar tareas de clasificación de patrones, rápidamente se descubrió que tenía severas limitaciones, debido a que solamente podía abordar problemas con clases linealmente separables y por la simplicidad del sistema de una sola neurona solo era capaz de realizar tareas de clasificación binaria.

Por ejemplo es posible ajustar los pesos de un perceptrón con una sola neurona para que imite el comportamiento de compuertas lógicas, como las compuertas *or*, *and* o *not*; sin embargo es imposible ajustar los pesos para tener la función de una compuerta *xor*, con una sola neurona. Intentar agrupas neuronas una seguida de otra (en serie) no incrementa las capacidades de aprendizaje del sistema debido a que al ser un modelo lineal, esto es equivalente a tener una sola neurona.

4.2.3. Funciones de activación

Debido a las desventajas de la función escalón unitario las redes neuronales modernas emplean funciones de activación con propiedades mucho más ventajosa como ser continuamente diferenciables o no lineales, que permite obtener modelos mucho más complejos y facilitan el desarrollo de algoritmos para el aprendizaje de las redes neuronales.

Algunas de las funciones de activación más comunes se presentan a continuación:

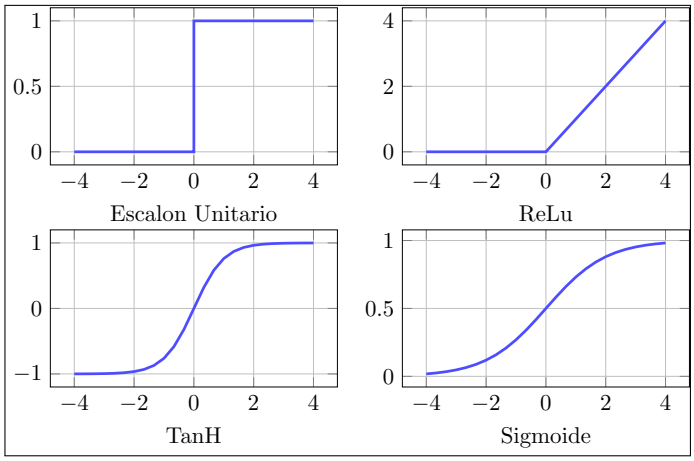
$$\text{Función sigmoide} \quad \sigma(z) = \frac{1}{1+e^{-z}} \tag{38}$$

$$\text{Tangente hiperbólica} \quad \tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \tag{39}$$

$$\text{Unidad lineal rectificada} \quad ReLu(z) = \begin{cases} 0 & z < t \\ z & z \geq t \end{cases} \tag{40}$$

Estas funciones se emplean dependiendo del tipo de problema que se intente resolver, debido a las particularidades que cada una presenta, por ejemplo la introducción de la no linealidad permite desarrollar aplicaciones de clasificación donde el conjunto de datos no es separable linealmente siendo esta la más habitual de las situaciones.

Figura 21. **Gráfica de funciones de activación comunes**



Fuente: Elaboración propia usando pgfplot, 15 de febrero de 2020.

4.2.4. Redes multicapa

Como fue expuesto anteriormente una red neuronal con escalón unitario como función de activación de neurona o nodo único, solo puede clasificar patrones linealmente separables; es por ello que no pude reproducir correctamente la función lógica *xor*. En el caso esta operación entre dos valores binarios, es necesario considera que el conjunto de datos de entrada es $(0, 0)$, $(0, 1)$, $(1, 0)$ y $(1, 1)$; la neurona se debe activar, retornar 1, con los valores $(0, 1)$, $(1, 0)$ o retornar 0 para $(0, 0)$ y $(1, 1)$.

Si se coloca los puntos en un plano cartesiano, se forma un cuadro en el que es imposible separar los puntos $(0, 1)$, $(1, 0)$ del resto mediante una única línea recta, esta línea recta representa la operación que realiza la neurona sobre los datos. Para poder solucionar este problema es necesario agrupar neuronas en capas, una arquitectura como la que se muestra del lado izquierdo de la figura 22 que consiste de tres neuronas, es capaz de aprender a realizar la operación *xor*.

Las neuronas 1 y 2 construyen una frontera de decisión (línea recta) sobre los datos, cada una solo es capaz de separar una parte del conjunto de entrada en forma correcta, en otros términos aprenden solo una característica del problema a resolver.

Por ejemplo si con el modelo para una neurona definido por las ecuaciones 35 y 37 se ajustan los pesos de la neurona uno con los valores:

$$w_0 = 1, w_1 = 1 \text{ y } b = -\frac{3}{2} \tag{41}$$

y para la neurona dos se escogen los siguientes:

$$w_0 = 1, w_1 = 1 \text{ y } b = -\frac{1}{2} \tag{42}$$

Se pueden evaluar el conjunto de datos de entrada en ambas neuronas para obtener los valores de salida que cada una genera, esto se muestra en la tabla II. Se puede observar que cada neurona es capaz de separar uno de los valores de entrada del resto si se ignora el valor de salida que le asigna; $(1, 1)$ para el caso de la neurona uno y $(0, 0)$ para la número dos. La neurona de salida se puede caracterizar por los siguientes valores:

$$w_0 = -2, w_1 = 1 \text{ y } b = -\frac{1}{2} \tag{43}$$

Empleando los valores de salida de las neuronas uno y dos como entradas para la última neurona, es posible ver como la arquitectura de tres neuronas resuelve el problema de la operación *xor*, la tabla II muestra el resultado. La función de la neurona de salida es construir una combinación lineal de las fronteras de decisión de las dos neuronas previas.

Tabla II. **Valuación del modelo con tres neuronas**

Neurona 1

x_0	x_1	z	$f(z)$
0	0	$-3/2$	0
0	1	$1-3/2$	0
1	0	$1-3/2$	0
1	1	$2-3/2$	1

Neurona 2

x_0	x_1	z	$f(z)$
0	0	$-1/2$	0
0	1	$1-1/2$	1
1	0	$1-1/2$	1
1	1	$2-1/2$	1

Neurona 3

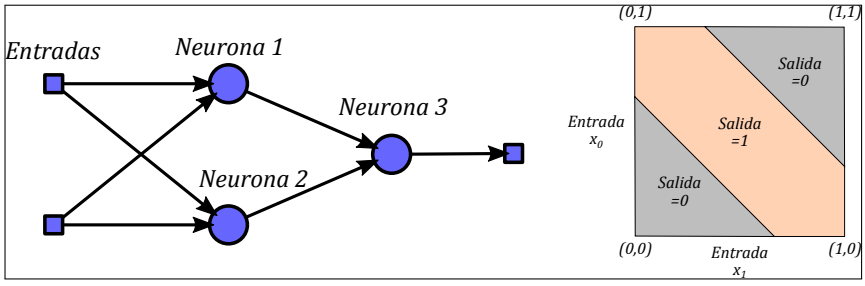
x_0	x_1	z	$f(z)$
0	0	$-1/2$	0
0	1	$1-1/2$	1
1	0	$1-1/2$	1
1	1	$-1-1/2$	0

Fuente: Elaboración propia, 16 de febrero de 2020.

En la parte derecha de la figura 22 se puede observar una gráfica que muestra la separación de los datos correctamente, se pueden ver las dos fronteras de decisión necesarias para resolver el problema.

Una forma de entender mejor la frontera de decisión es ver la ecuación 35 e interpretar una neurona como una función de varias variables, dos variables(x_0, x_1) en este caso, que describe a un plano en tres dimensiones. La frontera de decisión corresponde a la línea de intersección entre el plano descrito por la ecuación de la neurona y el plano $z = 0$, en el caso de neuronas con un número arbitrario de entradas n, el conjunto de posibles entradas corresponde a un conjunto de puntos en un espacio $n-$dimensional sobre el que se define una frontera de decisión como un hiperplano, una extensión del concepto de plano que es capaz de separar el conjunto de n puntos en dos partes.

Figura 22. **Arquitectura de red para resolver el problema *xor***



Fuente: Elaboración propia usando Inskape, 20 de febreo de 2020.

Con el ejemplo anterior queda claro que para lograr resolver problemas complejos, es necesario incrementar el número de neuronas, básicamente construir redes neuronal. En una red las neuronas están organizadas en capas, que son conjuntos de neuronas que reciben las mismas entras y aplican la

misma operación a los datos, lo que diferencia a cada una son los pesos que las caracterizan.

En una red neuronal, como la que se muestra en la figura 23, las capas que no están conectadas directamente a una entrada o a una salida son llamadas capas ocultas. La información se mueve a través de una red neuronal de la capa de entrada a la capa de salida, en contra posición, cuando se entrena una red neuronal (ajuste automático de sus pesos) para que pueda realizar una determinada tarea, es necesario tomar en cuenta el error, la información de error viaje en sentido apuesto desde la salida a la entrada; para entrenar la red, esto se explicara en detalle más adelante.

Para poder trabajar con redes neuronales con varias capas, nuevamente es necesario compactar los cálculos al representar sus elementos como vectores y matrices. Si se identifican las neuronas de la capa de entrada como A, B y C en la figura 23 su operación se puede ver como:

$$\begin{aligned} z_A &= x \cdot w_A + b_A \\ z_B &= x \cdot w_B + b_B \\ z_C &= x \cdot w_C + b_C \end{aligned} \tag{44}$$

que puede ser expresada

$$z = x \cdot W + b \tag{45}$$

donde sus elementos están definidos de la siguiente manera:

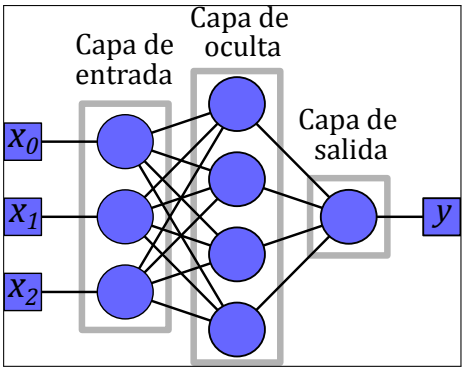
$$\begin{gathered} x = \begin{pmatrix} x_0 & x_1 & \cdots \end{pmatrix} \qquad W = \begin{pmatrix} w_{a0} & w_{b0} & w_{c0} \\ w_{a1} & w_{b1} & w_{c1} \\ \vdots & \vdots & \vdots \end{pmatrix} \\ b = \begin{pmatrix} b_A & b_B & b_C & \cdots \end{pmatrix} \quad z = \begin{pmatrix} z_A & z_B & z_C & \cdots \end{pmatrix} \end{gathered} \tag{46}$$

La activación de la primera capa puede ser escrita como el vector

$$y = f(z) = \begin{pmatrix} f(z_A) & f(z_B) & f(z_C) \end{pmatrix} \tag{47}$$

Como fue mencionado anterior mente la red neuronal más simple según su topología, es la que corresponde a un solo perceptrón denomina red neuronal monocapa. La arquitectura básica de la figura 23 es una forma generalizada de la red monocapa, en la que existen capas ocultas denominadas redes multicapa, estas capas en las que cada una de las neuronas está conectada a todas las entradas o a todas las neuronas de la capa previa, se denominan por su nombre en inglés *fully connected layers* o *dense*.

Figura 23. **Red neuronal de 3 capas**



Fuente: Elaboración propia usando Inskape, 4 de marzo de 2020.

4.2.5. Aprendizaje y entrenamiento

Las redes neuronales artificiales son un conjunto de algoritmos que forman parte de una rama del campo de la inteligencia artificial llamando aprendizaje automático o comúnmente referido por su nombre en inglés *machine learning*,

que se encarga de desarrollar técnicas para que un sistema computacional sea capaz de aprender y mejorar su desempeño con la experiencia. Esta idea de aprender y mejorar con la experiencia conlleva al hecho de que las RNA sean algoritmos que necesiten ser entrenados (desarrollar experiencia).

Las RNA pueden resolver un amplio y variado rango de tareas, que van desde problemas simples de clasificación hasta el incremento de fotogramas, resolución o calidad en videos antiguos, las tareas específicas que pueden resolver depende en parte de su arquitectura junto con sus conexiones internas; la arquitectura de redes neuronales presentada en este capítulo se conoce como rede neuronal prealimentada (*feedforward neural network* por su nombre en inglés) fue la primera y la más simple porque no posee ningún ciclo o bucle. Este tipo de RNA es capaz de abordar tareas de reconocimiento de patrones en conjuntos de datos numéricos como clasificación de datos con ruido, regresión, inferencia o incluso clasificación de imágenes sencillas como caracteres escritos a mano.

Para resolver un problema mediante una RNA primero es necesario escoger la arquitectura apropiada y a continuación que la red pase por un proceso de aprendizaje, para el caso de una red prealimentada el proceso de aprendizaje se traduce en el ajuste automático de los pesos de cada neurona mediante un algoritmo de entrenamiento.

Existen diferentes paradigmas para entrenar una red neuronal dependiendo del problema que se desee resolver y de los datos disponibles, para el objetivo de este trabajo se presentará solamente el paradigma de aprendizaje supervisado siendo el más común además del único relevante para nuestros propósitos.

4.2.6. Aprendizaje supervisado

Es la forma de aprendizaje más fácil de aplicar, en forma general se emplea cuando se quiere establecer una mapeo entre dos conjuntos de datos, como por ejemplo en la clasificación de imágenes. Requiere tener disponible un conjunto de datos (*dataset* por su nombre en inglés) identificados por sus respectivas clases, en el caso de la clasificación de imágenes estas deben ser etiquetadas y separadas según su contenido.

El fin del aprendizaje supervisado es entrenar un modelo a partir del conjunto de pares de datos, que sea capaz de predecir el valor corrector que corresponde a cualquier entrada válida después de haber visto una serie de ejemplos, por lo que el modelo debe ser capaz generalizar el conocimiento, partiendo de los datos presentados a situaciones no vistas previamente.

El procedimiento general para entrenar una red neuronal mediante aprendizaje supervisado es el siguiente:

- Pasar el conjunto de datos a la red y recolectar los resultados.
- Evaluar la función pérdida de la red, que no es más que el método establecido para comparar las predicciones de la red con las etiquetas correctas.
- Ajustar los parámetros para reducir la pérdida.
- Repetir hasta que la red converja, dicho de otra manera, hasta que no puede mejorar más con los datos de entrenamiento.

4.2.7. Gradiente descendiente

El proceso de entrenamiento es análogo a un problema de optimización en el que se intenta encontrar el valor mínimo de una función, por ello es necesario acudir a los principios del gradiente, una generalización multivarible de la derivada. Para que la red converja es necesario minimizar una función, llamada función de pérdida.

La función de pérdida evalúa la calidad de las predicciones realizadas por la red en función de sus parámetros actuales (pesos y sesgo) mientras más pequeño su valor mejores parámetros, el método del gradiente descendente consiste en aproximar un mínimo de la función mediante pasos proporcionales al negativo del gradiente de la función en el punto actual, para ello es necesario establecer una función de pérdida y calcular su la derivada para cada parámetro de la red en cada paso.

Existen diferentes funciones de pérdida y se emplean según el problema que se aborda, la función más simple y ampliamente usada es el error cuadrático, que se expresa:

$$L(t, y) = \sum_i (t_i - y_i)^2 \tag{48}$$

donde el error t representa el valor real y representa el valor de predicho por la red y el subíndice i denota el número de salidas en una red neuronal prealimentada.

Para calcular la derivada de la función de pérdida se emplea la regla de la cadena, en una red con un número arbitrario de capas, es posible establecer que la derivada depende de los parámetros de la capa k, las entradas x_k, salidas y_k de la capa y las derivadas de la capa siguientes $k + 1$, formalmente se plantea la

deriva de la función de perdida L respecto de W_k, para este caso W_k representa la matriz que contiene los pesos de cada una de las neuronas de la capa, como se muestra a continuación:

$$\begin{aligned}\frac{dL}{dW_k} &= \frac{dL}{dy_k}\frac{dy_k}{dW_k} = \frac{dL}{dy_k}\frac{dy_k}{dz_k}\frac{dz_k}{dW_k} = \frac{dL}{x_{k+1}}\frac{dy_k}{dz_k}\frac{d(W_k \cdot x_k + b_k)}{dW_k} \\ &= l'_{k+1} \odot f'_k \frac{dy_k}{dz_k}\frac{d(W_k \cdot x_k + b_k)}{dW_k} = x_k^T \cdot (l'_{k+1} \odot f'_k)\end{aligned} \tag{49}$$

En la ecuación 49 el símbolo l'_{k+1} representa la derivada para la capa $k+1$ respecto de su entrada, implica que $x_{k+1} = y_k$, con f'_k como la derivada de la función de activación y x^T representa la transpuesta del vector de entrada x o un vector columna, z_k representa el resultado de la operación de la neurona justo antes de aplicar la función de activación y $\odot$ indica multiplicación elemento a elemento entres dos matrices.

La derivada para el sesgo se puede calcular como se muestra en la ecuación 50

$$\frac{dL}{db_k} = \frac{dL}{dy_k}\frac{dy_k}{db_k} = \frac{dL}{dy_k}\frac{dy_k}{dz_k}\frac{dz_k}{db_k} = l'_{k+1} \odot f'_k \frac{d(W_k \cdot x_k + b_k)}{db_k} = l'_{k+1} \odot f'_k \tag{50}$$

Para resumir lo que es fundamental entender es que las ecuaciones nos indican que es posible calcular como cada parámetro afecta la función de pérdida recursivamente, comenzando al calcular la derivada en la capa de salida y retrocediendo para calcular capa tras capa, cada una de sus derivadas, además nos indican que la función de activación debe ser una función diferenciable. El procedimiento descrito anteriormente se conoce como retropropagación (del inglés *backpropagation*). Los algoritmos para el

entrenamiento de las redes neuronales prealimentadas tiene como base las ecuaciones 49 y 50.

Para iniciar el proceso de retropropagación es necesario conocer primero la derivada de la pérdida en la capa de salida, para el caso del error cuadrático es simplemente:

$$\frac{dL(t,y)}{dy} = -2(t-y) \tag{51}$$

Una vez se conocen las derivadas para la función de pérdida respecto de cada parámetro, solamente es necesario actualizar acorde a:

$$W_k \leftarrow W_k - \varepsilon \frac{dL}{dW_k}, \quad b_k \leftarrow b_k - \varepsilon \frac{dL}{db_k} \tag{52}$$

Se puede ver un valor ε añadido, proporcional a las derivadas, este valor se conoce como tasa de aprendizaje (*learning rate* por su nombre en inglés), ayuda a controlar la intensidad con la que cada parámetro debe ser modificado en cada iteración y deber ser cuidadosamente seleccionado.

4.3. Redes neuronales convolucionales

La arquitectura de la red neuronal prealimentada a pesar de tener un variado rango de aplicaciones, es ampliamente superada por la red convolucional en cuanto a tareas de visión por computador se refiere. Cuando se trata de trabajar con imágenes las redes prealimentadas presentan principalmente los siguientes problemas.

4.3.1. Falta de razonamiento espacial

Debido a la forma en la que se representa la información, en la entrada de una capa totalmente conectada (vector x) las neuronas reciben todos los valores de la capa previa sin distinción, es por ello que la red neuronal carece de cualquier noción de distancia o espacio entre los valores de entrada. Para ejemplificar, en el capítulo 2 se mencionó que una imagen se puede representar como un arreglo de tres matrices (una por canal) donde un píxel está formado por los tres elemento que comparten posición de cada matriz, esta información dimensional se pierde al convertir la matriz en un vector unidimensional para que ingrese en una red prealimentada, también sucede con la información útil como la posición de los píxeles que podría indicar si pertenecen a una misma región de la imagen.

4.3.2. Incremento en el número de parámetros

La representación matricial de una imagen nos indica que el número total de valores numéricos que la forman estado dado por $h \times w \times d$ donde w y h representan el ancho y la altura de la imagen mientras que D representa la cantidad de canales ($d = 3$ para RGB), ahora para entender mejor el problema supongamos emplear una red prealimentada para clasificar pequeñas imágenes monocromáticas ($d = 1$) de tamaño 32×32, el tamaño del vector entrada para la red sería de 1024, si asumimos que la primera capa oculta posee 50 neuronas además de que es totalmente conectada a la capa de entrada, el número de pesos que es necesario optimizar es de 51200, solamente para la primera capa oculta.

Este simple cálculo nos muestra como para imágenes más grandes el número de parámetros que es necesario entrenar crece desproporcionadamente

y lo hace aún más cuando tenemos en cuenta que para que una red neuronal puede aprender a realizar tareas como la clasificación de imágenes es necesario un alto grado de complejidad en la red, esto se traduce como un incremento en el número de capas ocultas y en el número de neuronas. Este rápido crecimiento hace inviable el entrenamiento para imágenes grandes.

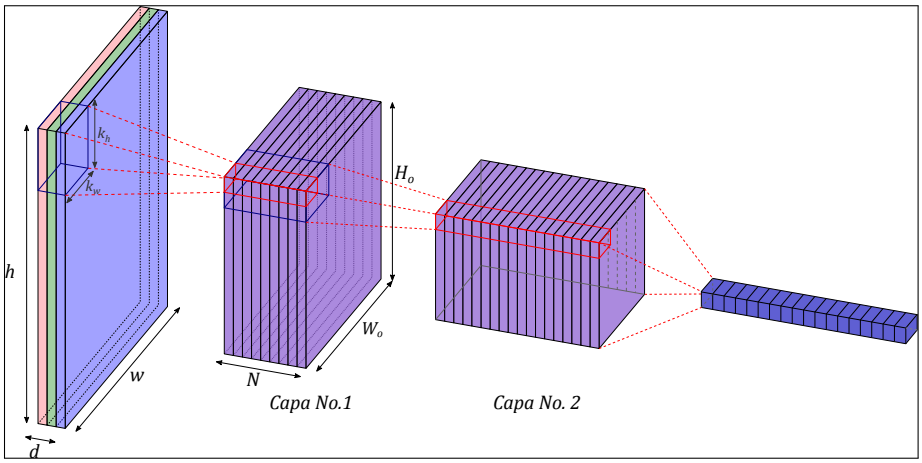
4.3.3. Fundamentos

Las redes neuronales convolucionales (CNN, por su nombre en inglés *convolutional neural network*) ofrecen soluciones simples a los problemas que tienen las redes neuronales prealimentadas (*feedforward*). Muchos de los conceptos vistos anterior mente son aplicables para las CNN y su funcionamiento general es similar, la información fluye de la capa de entrada a la de salida y sucede la retropropagación del error durante el entrenamiento, pero presentan algunos cambios en su arquitectura.

La primera característica de las CNN es que son capaces de manejar datos multidimensionales, cuando se trata de imágenes la entrada puede ser incluso el arreglo tridimensional ($h \times w \times d$) y las neuronas están dispuestas en un volumen similar. Otra diferencia importante con respecto a las redes prealimentadas, que tienen neuronas totalmente conectadas a las neuronas de la capa previa, es que cada neurona solo tiene acceso a una región de la imagen delimitada de la capa previa. Esta región remarcada en la figura 24 abarca todos los canales y recibe el nombre de campo receptivo o tamaño del filtro.

Al enlazar las neuronas solamente a las vecinas de la capa anterior se reduce drásticamente el número de parámetros y se preserva la información espacial de la imagen.

Figura 24. **Representación de CNN**



Fuente: Elaboración propia usando Inskape, 21 de marzo de 2020.

Al igual que las redes neuronales prealimentadas las CNN tienen una inspiración biológica en las funciones que realiza la corteza visual como la selección y detección de característica, orientación y reconocimiento de bordes en las imágenes. La arquitectura para una CNN cosiste de varias o incluso cientos de capas ocultas que realiza diferentes operaciones sobre los datos de entrada, las principales son las capas de convolución, agrupamiento (*pooling*) y capas totalmente conectadas (*dense*).

4.3.4. Capa de convolución

Este tipo de capa es la esencia de la red convolucional, es la que le otorga su nombre, gracias a ella el número de parámetros se reduce, debido a que las neuronas comparten los pesos y sesgos a lo largo de todas las neuronas conectadas al mismo canal. En el capítulo 2 se prestaron los filtros de convolución, definidos en el dominio espacial, que se aplican desplazando un *kernel* a lo largo de todos lo píxeles de una imagen e ir multiplicando y sumando

los valores de la imagen con dicho *kernel*, a medida que sucede el desplazamiento, con la intención de resaltar o atenuar características de la imagen en función de los valores de este.

En forma similar la operación que realizan las neuronas en una capa de convolución puede ser analizada como una sola neurona que se desliza a lo largo de la matriz de los datos de entrada con conectividad espacial a la capa previa limitada al tamaño del filtro (k_h, k_w). Si la imagen de entrada tiene más de un canal, el volumen de entrada corresponde a ($h \times w \times d$) y la conectividad de la neurona se limita al volumen $(k_h \times k_w \times d)$ por que el filtro iguala en número de dimensiones al volumen de entrada. La neurona se comporta igual a las mostradas anteriormente, realiza una combinación lineal de los valores de entrada con los valores del filtro para cada canal antes de aplicar una función de activación.

La respuesta $z_{i,j}$ de la neurona sobre un área del tamaño del filtro (k_h, k_w) en los datos de entrada para la posición (i, j) puede ser expresada como:

$$z_{i,j} = \sigma \left(b + \sum_{l=0}^{k_h-1} \sum_{m=0}^{k_w-1} \sum_{n=0}^{d-1} w_{l,m,n} \cdot x_{i+l,j+m,n} \right) \tag{53}$$

donde w representa los pesos de la neurona, que son los valores que conforman el filtro con forma $(k_h \times k_w \times d)$, b es el sesgo y σ es la función de activación sigmoide. Repitiendo esta operación para todas las posiciones que la neurona puede tomar sobre los datos de entrada se obtiene la matriz de repuesta z con dimensiones $H_o \times W_o$, que representa el número de veces que el filtro se puede deslizar a lo largo y ancho sobre los datos de entradas. Para poder entender mejor la operación previamente descrita se puede interpretar a la figura 14, del capítulo 2, como la operación de una capa de convolución sobre una matriz con un solo canal.

Una capa de convolución puede tener N conjuntos de neuronas, en otras palabras, puede aplicar N filtros sobre los datos de entrada, cada conjunto generan una matriz de repuesta z, estas matrices se apilan una tras otra para formar la repuesta completa de una capa de convolución que tiene forma $(H_o \times W_o \times N)$ como se ilustra en la figura 24, la respuesta de la capa de convolución se denomina tensor de salida. Un tensor es una generalización matemática de los conceptos de escalar, vector y matriz; por el momento puede ser visto como un arreglo multidimensional de matrices.

4.3.4.1. Propiedades

Los parámetros a entrenar para una capa convolucional son los valores que contiene el conjunto de N *kernels* más el sesgo, o sea que, si los datos de entrada tienen más de un canal el número total de parámetros para una capa es $N(d \cdot k_h \cdot k_w + 1)$ comparando una capa totalmente conectada equivalente tendría un número de parámetro de $(h \times w \times d) \times (H_o \times W_o \times N)$.

Con lo anterior queda claro que para una capa totalmente conectada el número de parámetros depende del tamaño de los datos de entrada, pero esto no afecta a la capa de convolución lo que le otorga propiedades muy útiles en la visión por computador, primero que es posible entrenar redes para clasificar imágenes grandes sin afectar el número de parámetros y también implica que una capa de convolución puede ser aplicada a cualquier imagen independientemente de su tamaño, no es necesario repetir el entrenamiento para diferentes tamaños de entrada.

Mediante el entrenamiento las capas de convolución son capaces de reaccionar a las características locales específicas de una imagen, una capa con

N filtros (conjunto de neuronas que comparten parámetros) tiene la posibilidad de reaccionar a N diferentes características. En las primeras capas de una CNN el entrenamiento provoca que cada *kernel* aprenda a reaccionar a características de bajo nivel, como una orientación especifica de los bordes o un gradiente de color. Las capas posteriores usan estos resultados para localizar características más avanzadas o abstractas, como el contorno de un objeto particular o la forma de un rostro.

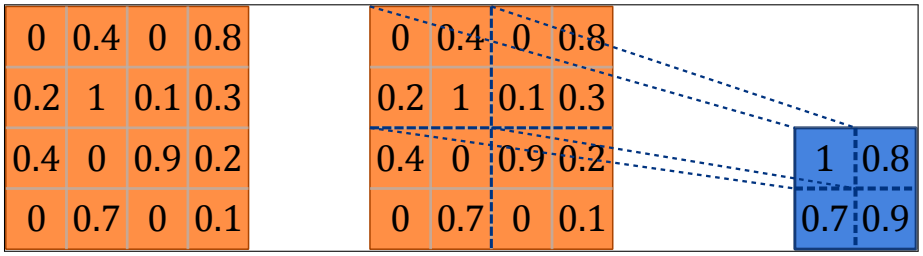
Las matrices de repuesta generadas por cada filtro aplicado a los datos de entrada pueden ser descritas como mapas de ubicación de la característica objetivo, por esta razón los resultados intermedios de una CNN son comúnmente llamados mapas de característica. Una capa con N filtros retornará N mapas de característica, cada uno corresponde a la detección de una característica particular en los datos de entrada. El arreglo de N mapas de característica que retornar una capa de convolución es llamado volumen de características y sus dimensiones son $(H_o \times W_o \times N)$.

4.3.5. Capa agrupamiento

Las capas de agrupamiento también son conocidas como capas tipo *pooling*, este tipo de capa no posee parámetros que deban ser ajustados durante el entrenamiento, su funcionamiento consiste en reducir el tamaño del mapa de característica para ser procesado por capas posteriores. Es necesario reducir el tamaño de cada uno de los mapas de características para controlar el número de parámetros, debido al incremento en el número de mapas según la cantidad de filtros en la capa previa; reducir la dimensión espacial de los datos reduce significativamente la cantidad de cálculo necesario durante el entrenamiento.

Los métodos de agrupamiento más comunes son *max-pooling* y *average-pooling*. Las neuronas en una capa de agrupamiento simplemente toman los valores dentro de su ventana (campo receptivo) y retornan un único valor, *Max-pooling* devuelve el máximo valor dentro del área que agrupa para cada capa (ver figura 25) y *average-pooling* calcula el valor promedio de los valores que agrupa.

Figura 25. **Ilustración de operación *Max-pooling***



Fuente: Elaboración propia usando Inskape, 2 de abril de 2020.

Para ejemplificar, una capa de *max-pooling* con un tamaño de ventana de 2×2 agruparía cuatro valores en cada mapa y retornarían un solo número, al recorrer un mapa de características de arriba abajo y de izquierda a derecha dividiría tanto su ancho como su alto por un factor de dos. Al realizar la operación de agrupamiento se preserva la posición relativa de las características más no su posición exacta con lo que se añade invariancia espacial al clasificador.

4.3.6. Capas totalmente conectadas

Este tipo de capas son usadas en las CNN de la misma forma que en las redes *feedforward*, para poder ser utilizadas primero es necesario cambiar la

forma del tensor de entrada a un vector columna de una sola dimensión, a este procedimiento se le conoce como aplanar (*flatten* en inglés) el volumen de características, este tipo de capa también es llamado densamente conectada o simplemente densa (*dense* en inglés).

Las capas densas son las últimas capas en una arquitectura de CNN y su propósito es realizar la fase final de clasificación a partir de los mapas de características que le suministran las capas previas, la etapa final con capas densas genera un vector de longitud K que contiene la probabilidad de pertenencia a cada clase de la imagen suministrada a la red, donde K representa el número total de clases que la red es capaz de identificar.

De forma general una arquitectura de CNN para clasificación de imágenes consta de dos partes principales, la primera que son las capas dedicas a la extracción de características formada por las capas de convolución y las capas de agrupamiento, la segunda parte consta de una o varias capas densas, donde la última capa tiene un número de neuronas igual al número K de clases que la red es capaz de identificar.

5. IMPLEMENTACIÓN DE PROTOTIPO

Para concluir el trabajo se presentarán con detalle las tecnologías empleadas en el desarrollo del prototipo para la detección e identificación de billetes, primero se abordará el software y la construcción del clasificador mediante redes neuronales convolucionales. En la segunda parte se describirá los componentes de hardware sobre los que se ejecutara el software, tales como una computadora SBC, el módulo de cámara e iluminación.

El componente principal de software en el proyecto es en una red neuronal convolucional para la clasificación. En el capítulo previo se presentó una breve introducción a las redes neuronales convolucionales desde el punto de vista teórico, en este apartado se tratarán a las redes neuronales desde una perspectiva técnica así como práctica, se mostrará la arquitectura de red neuronal que emplea el clasificador, se hará referencia a conceptos como *overfitting*, técnicas de aprendizaje automático como *transfer learning*; además se presentarán las bibliotecas necesarias para construir el clasificador y varias herramientas que facilitan el desarrollo.

El lenguaje de programación empleado para desarrollar el proyecto es *Python* en su versión $3.6.9$, debido a la facilidad en sus sintaxis, a la cantidad de librerías disponibles para aprendizaje de máquina, ciencia de datos o visión por computadores entre muchas otras; este lenguaje se ha convertido en una herramienta muy poderosa dentro de las ciencias de la computación, en investigación, desarrollo comercial y elaboración de prototipos.

5.1. *Tensorflow*

Es una biblioteca *open source* desarrollada por Google que integra los paradigmas de programación diferencial junto a la programación de flujo de datos *(dataflow)*, es una librería de procesamiento numérico principalmente usada para desarrollar y entrenar modelos de aprendizaje automático o aprendizaje profundo. Debido a sus características *Tensorflow* ofrece una forma simple para desarrollar modelos de aprendizaje automático al igual que desplegarlos en distintas plataformas como computadoras con CPU, computadoras con GPU, dispositivos móviles así como recientemente a través de un navegador web.

Debido a la cantidad de recursos computacionales necesarios para entrenar un modelo de aprendizaje profundo *Tensorflow* ofrece la posibilidad de entrenamiento a través del uso de una unidad de procesamiento gráfico (GPU por sus siglas en inglés) reduciendo considerablemente el tiempo de entrenamiento. Para el desarrollo de este proyecto se utilizó *Tensorflow* en la versión $2.3.0$

5.2. Keras

Es una API (*application progamming interface*) de alto nivel que interactúa con otras bibliotecas para desarrollo de redes neuronales como *Tensoflow* o *Theano*, su función principal es facilitar la construcción de aplicaciones de aprendizaje profundo al proveer funciones más amigables al usuario, una mayor modularidad y suficiente flexibilidad. Con la versión 2 de *Tensoflow*, Keras fue incorporado por defecto, ya no es necesaria su instalación por separado para poder utilizarlo.

5.3. *Google Colaboratory*

Es una herramienta sin costo, disponible en línea para desarrolladores e investigares de inteligencia artificial que provee un entorno de desarrollo interactivo basado en la web, que permite crear documentos similares a los del proyecto *Jupyter*. El entorno de desarrollo soporta la ejecución del lenguaje *Python* en el navegador, tiene por defecto una instalación funcional de *Tensoflow*, además permite conectarse con datos externos y almacenar los documentos de trabajo mediante el servicio de *Google Drive*. Una de sus mayores ventajas es que permite el acceso temporal a una GPU para acelerar el entrenamiento de modelos de aprendizaje automático.

5.4. *MobileNet V2*

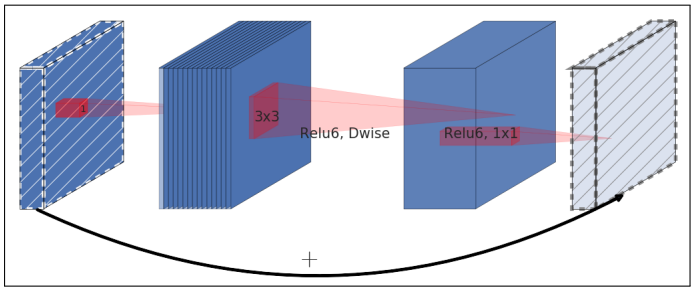
Es una arquitectura abierta desarrollada por Google, disponible dentro de *Tensorflow*. Está diseñada para utilizar poca memoria y ser utilizada en dispositivos móviles con recursos limitados. Las arquitecturas que son eficientes en cuanto al uso de memoria y recursos del computador, utilizan un tipo especial de convolución llamada *depthwise separable* para reducir el número de parámetros así como los cálculos necesarios para generar predicciones, este tipo de convolución implica alternar dos tipos de convolución:

- Convolución *pointwise*: Son convoluciones regulares que tienen un *kernel* de dimensión 1×1. Su propósito es combinar los canales en la imagen, en esencia realiza la suma ponderada de todos los canales.
- Convolución *depthwise*: La diferencia con una convolución regular es que este tipo no combina los canales de los datos de entrada. Al igual que la operación de convolución del capítulo anterior su rol principal es detectar patrones, en el caso de imágenes construye un mapa de características

para cada canal de entrada.

Al combinar estos dos tipos de convolución se obtiene una operación similar a la convolución presentada en el capítulo previo, sin embargo debido al pequeño tamaño del *kernel* la arquitectura requiere de menos parámetros y menos recursos de computación. Para el caso de *MobileNet V2* los múltiples bloques que la conforman son llamados *linear bottlenecks*, estas son capas compuestas por una convolución 1×1 (*pointwise*), seguido por una capa de convolución 3×3 (*depthwise*) y finaliza por una convolución 1×1.

Figura 26. **Bloque *linear bottleneck***



Fuente: MobileNetV2: Inverted Residuals and Linear Bottlenecks. https://arxiv.org/abs/1801.04381. Consulta: 10 de junio 2020

5.5. *Transfer learning*

Dentro del campo del aprendizaje automático, existe el subdominio del aprendizaje profundo (en inglés, *deep learning*), con extrema simplificación se puede afirmar que a medida que se incrementa el número de capas ocultas, de unas pocas a cientos de ellas, una red neuronal *feedforward* pasa a ser un modelo de aprendizaje profundo; que requiere grandes recursos computacionales y grandes cantidades de datos para poder extraer abstracciones de alto nivel y hacer predicciones con más exactitud.

Debido al coste computacional y a la inmensa cantidad de datos necesaria para entrenar un modelo de aprendizaje profundo, se desarrolló la técnica llamada *transfer learning*, que consiste en aprovechar un modelo disponible y previamente entrenado en una tarea similar a la tarea que se intenta solucionar. Para ejemplificar si comparamos el proceso de entrenamiento de una red neuronal con el problema de encontrar las raíces de una función, se necesitan menos iteraciones para encontrar una raíz mediante un método numérico, si se conoce de antemano una solución aproximada o cercana a la raíz y esta se emplea como valor inicial, de manera similar reentrenar una red requiere menos recursos, potencia de cómputo y datos.

Como se mencionó en el capítulo anterior un clasificador de imágenes mediante redes neuronales tiene dos partes principales, la primera es una parte convolucional que se emplea para extraer características de las imágenes y la segunda consiste en capas de redes neuronales *feedforward* que se emplean para clasificar la información que provee la parte convolucional. Para construir un modelo mediante *transfer learning* primero es necesario seleccionar uno de los modelos libres, ya entrenados.

El siguiente paso es eliminar del modelo las capas de clasificación, las últimas capas, para emplear solamente las capas de convolución que extraen características de las imágenes. Para construir el nuevo modelo se añade una nueva etapa de clasificación en sustitución de la que fue eliminada, luego se procede a entrenar nuevamente el modelo con el nuevo conjunto de datos que contiene ejemplos de las imágenes que se desea clasificar.

Durante esta nueva fase de entrenamiento no se actualizan los parámetros (pesos y sesgos) de las capas de convolución, las capas base, que provienen del modelo ya entrenado, se congelan para que sus parámetros no puedan ser

modificados. La razón para esto se debe a que estas capas ya fueron entrenadas mediante un gran conjunto muy variado de imágenes, algunas arquitecturas libres son capaces de reconocer más de mil clases distintas; las capas de convolución ya son capaces de reconocer patrones y figuras de manera eficiente. La nueva etapa de clasificación que se añade al final se inicializa con parámetros aleatorios que se actualizan y mejoran durante el proceso de entrenamiento; la capa de salida contiene un número de neuromas igual al número de clases que deseamos identificar.

Con el procedimiento previamente descrito se reduce drásticamente el número de parámetros a ser entrenados, por ello también se reduce considerablemente el tiempo de entrenamiento y tamaño del conjunto de datos, de cientos de miles de imágenes a unas decenas de miles, sin embargo todo esto tiene un coste y es que por lo general un modelo no podrá alcanzar una exactitud tan alta, comparado a un modelo entrenado desde cero con un gran conjunto de datos y gran potencia de cómputo, a pesar de esta desventaja la exactitud que ofrece un modelo desarrollado con esta técnica es considerablemente alta y además existen técnicas, como *fine-tuning* para incrementar el desempeño del modelo.

5.6. Construcción del *Dataset*

Tener un adecuado conjunto de datos, es esencial para poder entrenar un modelo de aprendizaje automático, además es necesario un procesado previo para poder ingresarlos al modelo y que las imágenes sean relevantes para el entrenamiento, en esencia que contenga ejemplos representativos para cada clase con los que el modelo pueda generalizar. El *dataset* fue elaborado tomando fotografías de cada una de las denominaciones, fue necesario ordenar los datos en quince clases, dos por cada una de las denominaciones, una clase

para el anverso y una para el reverso, siendo un total de catorce clases para las denominaciones, además fue necesario agregar una clase adicional que no contiene billetes, para que el clasificador sea capaz identificar la presencia de un billete y diferenciarlo del fondo.

Para obtener un modelo con rendimiento y exactitud decente que pueda ser aplicado en una situación real se hicieron las siguientes consideraciones al momento de tomar las fotografías para elaborar el *dataset*:

- El billete debe ocupar como mínimo una porción de $1/5$ de tamaño de la imagen.
- Los billetes deben estar en diferentes posiciones, rotaciones y ángulos de visión para poder tener la mayor diversidad posible en los ejemplos.
- Debe haber ejemplos con oclusión (objetos que cubran una porción de billete) para tener ejemplos fieles a una situación real.
- El fondo de la imagen debe ser lo más variado posible.
- Las fotografías deben ser tomadas bajos diferentes condiciones de luz.

El *dataset* consiste de aproximadamente 320 ejemplos para cada una de las catorce clases que representan billetes mientras la clase que representa el fondo contiene mil ejemplos, esto con la intención de reforzar el entrenamiento para que el modelo aprenda con exactitud a separar imágenes que contiene billetes de las que no; en la figura 27 se muestran algunos ejemplos del conjunto de datos.

El conjunto de datos empleado en este trabajo está disponible en línea y las fotografías se encuentran bajo licencia abierta para que puedan ser empleadas en trabajos similares, el repositorio que las contiene este indicado en la sección de apéndice.

Figura 27. **Ejemplos del conjunto de datos**

(a) Billete de cinco (b) Billete de diez

(c) Billete de cincuenta (d) Billete de cien

Fuente: Elaboración propia, 12 de junio 2020.

5.7. Entrenamiento

En esta sección se exponen algunos de los conceptos prácticos sobre el entrenamiento del modelo propuesto para clasificación de billetes, como la necesidad de preprocesamiento de las imágenes, el motivo por el cual se separan en diferentes conjuntos los datos y los problemas de sobre ajuste que pueden surgir por el tamaño relativamente pequeño del *dataset*.

Debido a que la resolución de la cámara en un teléfono inteligente moderno supera fácilmente mil pixeles, tanto en ancho como en alto, una

imagen puede tener un tamaño de unos cuantos *megabytes*, por ello es necesario reducir el tamaño de las imágenes para que puedan ser manipuladas fácilmente y también hacer más ligero el peso total del *dataset*.

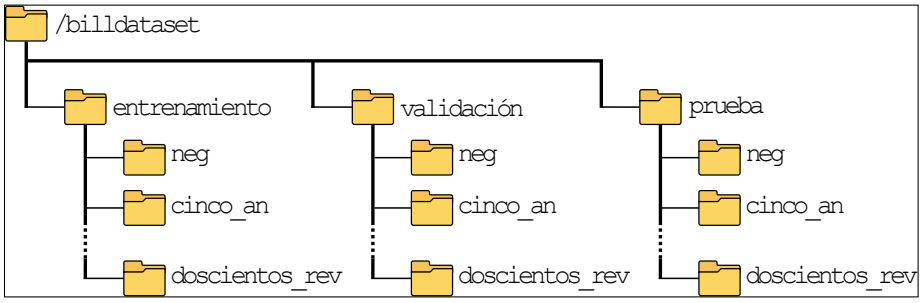
Otra razón relevante para reducir el tamaño de las imágenes es que un modelo construido a partir de *MobileNet V2* debe recibir como entrada una imagen con dimensiones específicas, en este proyecto se emplearon las dimensiones por defecto de la arquitectura $(224, 224, 3)$ originalmente utilizadas para entrenar las capas extractoras de características.

Mientras más grande sea la imagen que ingrese a la red neuronal, una mayor cantidad de cálculos son necesarios para el entrenamiento de la red y para realizar una predicción, durante las pruebas empíricas las dimensiones de $(224, 224, 3)$ ofrecen resultados satisfactorios, si se reducen las dimensiones con la intención de reducir la potencia del cómputo necesaria, la exactitud de la red se reduce considerablemente debido a la perdida de detalles y características para identificar correctamente los billetes.

En el capítulo anterior se describió el procedimiento genérico para entrenar una red neuronal dicho procedimiento brevemente consiste en ajustar los parámetros con cada nueva iteración, para incrementar la exactitud y reducir la pérdida. En la práctica una mejor estrategia de entrenamiento requiere reservar una parte de todas las imágenes de tal modo que el algoritmo emplea dos conjuntos uno de entrenamiento y uno de validación. Los datos de entrenamiento se usan para ajustar los parámetros e incrementar la exactitud de la red, mientras que los datos de validación evalúan la exactitud de la red con datos que no son usados para ajustar parámetros durante el proceso de entrenamiento. Utilizar un 80 % del total de los datos para entrenamiento, un 10 % para el conjunto de validación y el 10 % restante como datos de prueba, es

una recomendación por parte de los desarrolladores e investigadores en este campo, para ejemplificar en la figura 28 se muestra la estructura del árbol de archivos que contiene el *dataset*.

Figura 28. **Árbol de archivos del conjunto de datos**



Fuente: Elaboración propia usando Inskape, 15 de junio 2020.

El conjunto de validación sirve para hacer predicciones con datos que el modelo desconoce; permite evaluar durante el entrenamiento, el desempeño del modelo al momento de generalizar, realizando predicciones con datos nuevos, de este modo se verifica el conocimiento aprendido durante la fase de entrenamiento. El algoritmo de entrenamiento devuelve dos valores de exactitud en cada iteración, uno corresponde a la exactitud que se obtiene al predecir la clase para el conjunto de entrenamiento y otro al predecir la clase para el conjunto de validación. Estos dos valores sirven como referencia para conocer si el entrenamiento es efectivo, si ambos valores son similares y se incrementan a medida que aumenta el número de iteraciones se puede asumir, con cautela, que el modelo está aprendiendo características relevantes para resolver el problema de clasificación.

Si los dos valores de exactitud difieren drásticamente puede atribuirse a varios motivos pero hay uno especialmente relevante, más aún si la exactitud del conjunto de entrenamiento es alta y la exactitud del conjunto de validación se

mantiene baja y no incrementa durante el entrenamiento, esto puede ser por el sobreajuste (*overfitting*) de los parámetros.

5.7.1. Overfitting

Es un problema presente en un modelo que no es capaz de aplicar el conocimiento adquirido a nuevos datos, no es capaz de generalizar, por lo que no puede clasificar correctamente nuevos datos (baja exactitud) mientras que es capaz de clasificar correctamente casi todos los datos con los que fue entrenado (muy alta exactitud) mediante una analogía se puede entender que el modelo no está aprendiendo sino que está memorizando los datos.

En modelos de aprendizaje profundo y aprendizaje automático este problema surge a menudo como consecuencia de un *dataset* pequeño, cuenta con pocos ejemplos por clase, debido a ello los parámetros del modelo sufren sobreajuste y se hacen sensibles a características muy específicas del conjunto de entrenamiento por lo que no es capaz de reconocer esas características en datos nuevos.

5.7.2. *Data augmentation*

Una solución al problema de *overfitting* es incrementar el tamaño del *dataset* lo que incrementaría el tiempo de entrenamiento y los recursos necesarios para entrenar un modelo, sin embargo esto no siempre el posible ya sea porque es demasiado costoso o difícil recolectar nuevos datos, una solución alternativa es sintetizar nuevos datos a partir de los datos ya existentes.

Es habitual la situación en la que no se cuentan con suficientes datos para entrenar apropiadamente un modelo, por lo que incrementar los datos

artificialmente es una técnica ampliamente utilizada para mitigar el *overfitting*, en el caso de la clasificación de imágenes, consiste en aplicar pequeñas transformación a los datos existentes, como las transformaciones geométricas mencionadas en el capítulo 3, otras transformaciones podría ser cambios en el color, iluminación o inserción de ruido en las imágenes.

5.7.3. Regularización mediante *dropout*

Es una técnica en la que un porcentaje aleatorio de neuronas son ignoras durante cada iteración del entrenamiento, esto quiere decir su contribución al flujo de activación es temporalmente removida y sus parámetros no son actualizados durante la retropropagación, esto causa que las neuronas aprendan pesos que no están basados en la cooperación con las neuronas circundantes, se vuelven independientemente suficientes. Este proceso minimiza el *overfitting* y permite obtener una red capaz de generalizar mejor.

En este trabajo con la finalidad de mantener el conjunto de datos pequeño y reducir el *overfitting* se sintetizan nuevos datos durante la fase de entrenamiento, esto es posible gracias a las funciones implementadas tanto en *Tensorflow* como *Keras* que permiten realizar transformaciones como corrimientos, rotación y amplificación de forma sencilla.

5.8. Optimización para producción

Un modelo de *deep learning* elaborado mediante *Tensorflow* puede ser optimizado para que consuma menos recursos en detrimento de la exactitud si este proceso se hace adecuadamente la reducción en la exactitud es mínima y no afecta el desempeño de la aplicación, para realizar este proceso se emplea *Tensorflow Lite*.

5.8.1. *Tensorflow Lite*

Es un conjunto de herramientas dentro de *Tensorflow* que permite la ejecución de modelos de *Tensorflow* en dispositivos móviles, dispositivos con Linux incorporado o de IoT. Permite realizar predicciones con modelos de aprendizaje automático con baja latencia y ayuda a reducir el tamaño que ocupan en memoria. Tiene dos componentes principales un intérprete que permite la ejecución de los modelos optimizados para el tipo de hardware (teléfonos inteligentes, computadores SBC o microcontroladores) y un conversor que permite obtener modelos con formato eficiente para el intérprete, mejorando su tamaño y rendimiento.

Tensorflow Lite, se encuentra disponible individualmente, separado de la biblioteca principal, con el fin de ser empleado con un conjunto de funciones limitado y especialmente optimizado para la inferencia en dispositivos portátiles, que empleen Android, iOS o computadoras de una sola placa que cuenten con un procesador ARM. El fin último de todo el proceso de entrenamiento es obtener un modelo con suficiente precisión, de tamaño reducido y con una latencia adecuada para la aplicación que se está desarrollando, todo esto almacenado en un solo archivo con extensión *.tflite* que puede ser empleado las distintas plataformas (Linux, Windows, iOS o Android) que son capaces de correr el intérprete en *Tensorflow Lite*.

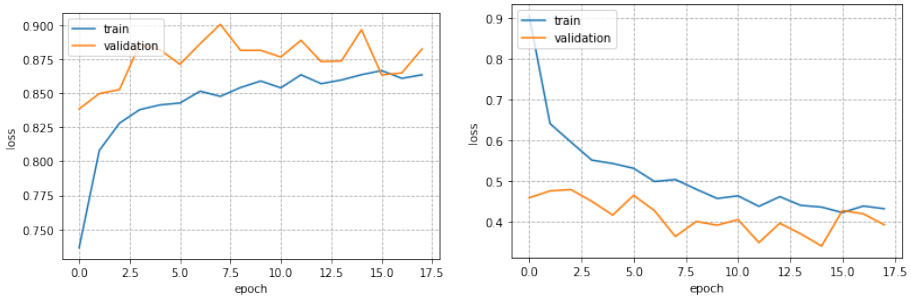
5.9. Resultados del entrenamiento

Para realizar el entrenamiento del modelo fue necesario utilizar la herramienta en línea de *Google Colaboratory* que permite acelerar el entrenamiento mediante una GPU, también ahorra el tiempo de instalación de

Tensorflow en un entorno local y los problemas que puedan surgir durante el proceso. El modelo utilizado para clasificación emplea como base (extractor de característica) la red neuronal *Mobilenet V2* a la que se lee ha retirado la última capa de clasificación y se sustituye por una capa de regularización *Dropout* y por una capa *Desen* (totalmente conectada) al congelar el entrenamiento para todo el modelo base, toda la red MobileNet V2, para aplicar *transfer learning* se obtiene un modelo con un total de $2,282,319$ de parámetros de los cuales son entrenables $21,775$.

El entrenamiento se realizó en dos sesiones empleando el mismo conjunto de datos, en la primera se puede observar en el figura 29 que el modelo es ofrece una exactitud mayor al 80 % sobre el conjunto de validación tan solo después de la primera iteración, tras 8 iteraciones se alcanza el 90 % y oscila cerca de ese valor el resto de iteraciones mientras que la exactitud para el conjunto de entrenamiento se incrementa paulatinamente hasta que alcanza un máximo de 86 % a partir de la cual ya no mejora.

Figura 29. **Evolución del entrenamiento**



(a) Exactitud (b) Pérdida

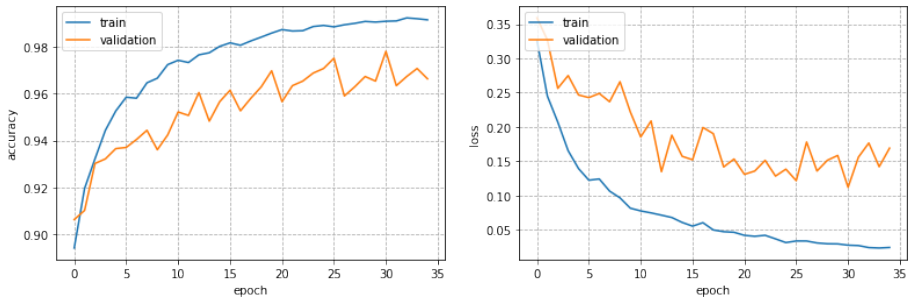
Fuente: Elaboración propia usando Matplotlib, 19 junio 2020.

En la segunda parte del entrenamiento se empleó la técnica llamada

Fine-tuning este es un paso adicional en el proceso de entrenamiento que pretende mejorar la exactitud del modelo, consiste entrenar nuevamente el modelo pero en esta ocasión, se permite que algunas capas en la salida del modelo base puedan ser entrenadas aumentado el número de parámetros disponibles para ajuste. También se emplea un *learning rate* muy bajo debido a que se espera que los pesos sean bastante buenos por lo que no se pretende distorsionarlos, debe ser usada con precaución por que puede ocasionar *overfitting* dañando el modelo en el proceso.

Para aplicar *fine-tuning* al modelo se estableció que las últimas 15 capas del modelo base pudieran ser entrenadas, así como un *learning rate* 100 veces menor al que se empleó en la primera parte. En la figura 30 se muestra como se incrementa gradualmente el valor de exactitud y se minimiza la pérdida tanto en el conjunto de entrenamiento como en el conjunto de validación, siendo después de 31 iteraciones que se obtienen los mejores pesos para la red y que corresponde a una exactitud de 97.8 % para el conjunto de validación.

Figura 30. **Evolución del entrenamiento (*fine-tuning*)**



(a) Exactitud (b) Pérdida

Fuente: Elaboración propia usando Matplotlib, 21 de junio de 2020.

Para poder evaluar correctamente el modelo obtenido es necesario ponerlo

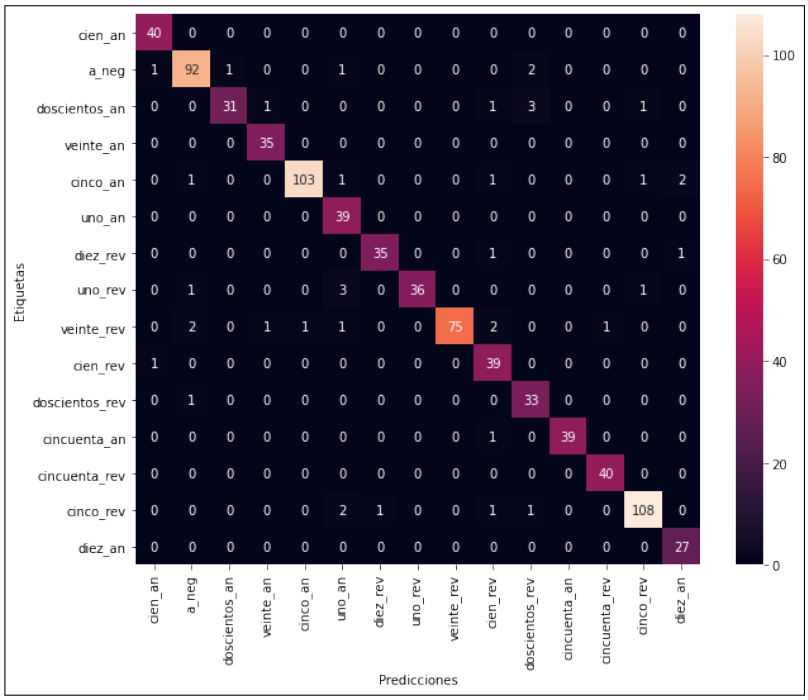
a prueba con datos que no fueran usados en el conjunto de entrenamiento o en el conjunto de validación, para ello es necesario el conjunto de prueba, al realizar predicciones sobre estos datos es posible observar si existe algún sesgo en la clasificación o si el sistema confunde clases, la herramienta que permita visualizar esta información es la matriz de confusión, en ella cada fila representa el número de imágenes que pertenecen a la clase, mientras que las columnas representan el número de predicciones para cada clase, con ella es fácil observa si el sistema confunde clases entre sí. Si no existe dispersión de los datos a lo largo de la comuna y si el valor vas grande se ubica en la diagonal de la matriz se puede afirmar que clasifica correctamente.

En la figura 31 se puede observar la matriz de confusión elaborada a partir de las predicciones en el conjunto de prueba. Otra medida de fiable de rendimiento es la exactitud que el modelo ofrece al predecir sobre los datos de prueba la cual corresponde a 96.4 %.

5.10. Raspberry Pi

Para poder realizar pruebas sobre el modelo es necesario ejecutarlo sobre algún dispositivo que sea capaz correr la biblioteca completa de *Tensorflow* o su versión limitada *Tensorflow lite*, a parte de una computadora de escritorio, un opción es recurrir a computadoras como las Raspberry PI; computadores de placa reducida con procesadores de arquitectura ARM, especialmente orientados al desarrollo de prototipos y para el cual Google provee un paquete oficial altamente optimizado de *Tensorflow lite*. Otra ventaja al recurrir a una Raspberry Pi es poder usar un sistema operativo Linux capaz de ejecutar el lenguaje Python, esto hace sumamente sencillo su despliegue, debido a que no es necesario realizar demasiados cambios en el código de base, en cambio si se

Figura 31. **Matriz de confusión**



Fuente: Elaboración propia usando Seaborn, 22 de junio de 2020.

desea emplear dispositivos móviles es necesario emplear el lenguaje de desarrollo específico de la plataforma, Java para el caso de Android o Swift para el caso de iOS.

5.10.1. Reseña histórica

Con la intención de crear una computadora de bajo coste que fuera abierta y permitiera ser modificada para estimular la enseñanza de la informática en las escuelas fue creada la Fundación Raspberry Pi en el Reino Unido en año 2009, por el físico e ingeniero Eben Upton que había estado trabajado en la idea de un computador SBC desde el año 2006; que junto a un grupo de profesores, académicos y entusiasta de la informática desarrollan la idea hasta logra su

primer lanzamiento al mercado en febrero del 2012. La computadora cosechó un gran demanda desde el principio gracias a su bajo precio, su versatilidad y su tamaño reducido; acabo siendo un más popular de lo que esperaban sus desarrolladores, debido a ventas fuera del mercado objetivo, que era educación, empleándose incluso en aplicaciones de robótica.

El software de todos los modelos, es software libre, siendo su sistema operativo oficial una versión derivada de Debian llamada Raspberry Pi OS, aunque permite la instalación de otras distribuciones Linux para arquitecturas ARM. El lenguaje utilizado para desarrollo es principalmente Python. Según el modelo la computadora cuenta en general con procesador Broadcom, memoria RAM, GPU, puertos USB, HDMI, Ethernet, 40 pines GPIO y un conector para cámara. La computadora en la actualidad aparte del entorno de la educación figura en diversas aplicaciones para la industria y el campo de IoT.

5.10.2. Modelo Raspberry Pi 3 A+

De las diferentes opciones de computadoras Raspberry Pi, para implementar un dispositivo pequeño y portátil el modelo Raspberry Pi Zero W sería ideal debido a su factor de forma, pero a pesar de que es posible la instalación de *Tensorflow lite* este modelo tiene un procesador con solamente un núcleo no cuenta con suficiente potencia de cómputo para ejecutar apropiadamente el clasificador de moneda. Es necesario contar con el hardware suficientemente potente para ejecutar tareas de inteligencia artificial. El modelo Raspberry Pi 3 A+ cuenta con un factor de forma más pequeño que el modelo estándar de la versión 3 y que el modelo más reciente Raspberry Pi 4 por lo que a pesar de contar con recursos limitados debido a su bajo coste es una muy buena opción para implementar un prototipo del clasificador.

En la tabla III se muestra las especificaciones del modelo Raspberry Pi 3 A+ entre las que destacan el procesador Broadcom BCM2837B0 de 64 bits a 1.4 GHz, la salida de audio con un conector de 4 polos que permite la conexión de directamente de auriculares para la salida de audio al usuario y el puerto CSI disponible para la conexión de cámara de un módulo de cámara.

Tabla III. **Especificaciones de Raspberry PI**

Procesador	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC a 1.4GHz
Memoria RAM	512MB LPDDR2 SDRAM
Conectividad	2.4GHz y 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2/BLE
GPIO	40 pines de propósito general
Salida de video	Full-size HDMI
Puertos de Conexión	USB 2.0
	CSI para la conexión con el módulo de cámara de Raspberry Pi
	DSI para la conexión con la pantalla táctil de Raspberry Pi
Audio	salida estéreo y video compuesto de 4 polos
Almacenamiento	Puerto micro SD para la carga del sistema operativo y almacenamiento de datos
Alimetación	5V/2.5A DC

Fuente: Raspberry Pi Fundation. https://www.raspberrypi.org/products/raspberry-pi-3-model-a-plus/. Consulta: 10 de julio de 2020

5.10.3. Módulo de cámara V2

Es la segunda edición del módulo de cámara para Raspberry Pi, que cuenta con un lente fijo, un sensor de imagen Sony IMX219 de 8 megapíxeles y tiene conexión a través de puerto CSI. Es capaz de captura de video en con resoluciones de 1080 píxeles a 30 fps (fotogramas por segundo), de 720 píxeles a 60 fps y VGA90, así como la captura de imágenes fijas con una resolución máxima de 1080 píxeles. El sensor que incluye esta cámara es una sustancial mejora respecto al módulo de cámara original de Raspberry Pi en aspectos

como la fidelidad del color, calidad de imagen y el desempeño con baja iluminación. Es compatible con todos los modelos de Raspberry Pi que dispongan de un puerto CSI. Para acceder a la cámara se dispone de las APIs MMAL y V4L, sin embargo la forma más sencilla y rápida es a través de la librería de Python llamada Picamera.

Este módulo es necesario para la captura de imágenes y clasificación de billetes. Es más que suficiente para una aplicación como la que pretende este trabajo, dado que el clasificador funciona con imágenes con dimensiones de 224×224 será necesario escalar las imágenes.

Figura 32. **Módulo de cámara V2**

Fuente: Raspberry Pi Fundation. https://www.raspberrypi.org/products /camera-module-v2/?resellerType=home. Consulta: 10 de julio de 2020

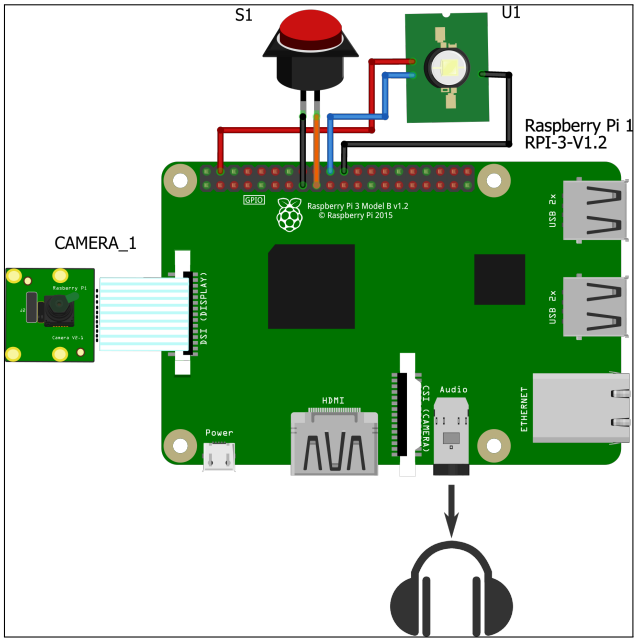
5.10.4. Conexiones de GPIO

Para poder elaborar un prototipo y poner a prueba el modelo de clasificación es necesario contar con algunos componentes mínimos para la ejecución como un botón par que el usuario indique la captura una imagen y se prediga la clase a la que pertenece, en el caso de una aplicación es posible

emplear un *widget* que ejecute la operación, para el caso de la Raspberry Pi es necesario recurrir a un sensor como botón pulsador para indicar el inicio de la operación.

Otro elemento que las pruebas mostraron necesario es una fuente de iluminación para mejorar la exactitud del modelo, debido a que es sensible a las condiciones de luz, su desempeño decae si la iluminación es muy pobre, es necesaria una fuente de luz que siempre ilumine el billete a identificar, para el caso de un teléfono móvil esto consiste simplemente en encender la luz led incorporada, para el caso de una Raspberry Pi es necesario añadir un módulo con led de alta luminosidad que apunte en la misma dirección que la cámara.

Figura 33. **Diagrama de componentes**

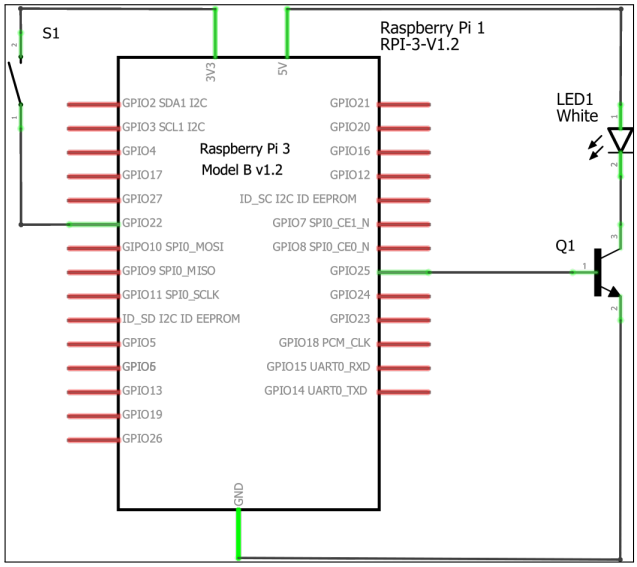


Fuente: Elaboración propia empleando Fritzing, 11 de julio de 2020.

En la imagen 33 se muestran un esquema general de conexión de los

componentes sobre la Raspberry Pi, para probar el modelo. En la figura 34 se muestra con más claridad las conexiones en los pines GPIO necesario para el funcionamiento del prototipo, se muestra el botón pulsador para iniciar la inferencia y el módulo de luz alta intensidad que consiste simplemente en un led de alta luminosidad controlado por un transistor para manejar una corriente más alta.

Figura 34. **Diagrama de conexiones en GPIO**



Fuente: Elaboración propia empleando Fritzing, 11 de julio de 2020.

5.11. Ejecución de la aplicación

Debido al funcionamiento del intérprete de *Tensorflow lite* el algoritmo para realizar una predicción es bastante directo, al proceso de ejecutar el modelo para realizar predicciones en función de los datos de entrada se le llama inferencia. Durante la inferencia la mayor parte del trabajo lo realiza el intérprete junto al modelo almacenado, dentro del programa estos actúan como una caja negra, una función ya definida que requiere como argumento una imagen con el

formato y dimensiones adecuadas. El procedimiento general para clasificar una imagen puede resumirse en los siguientes pasos:

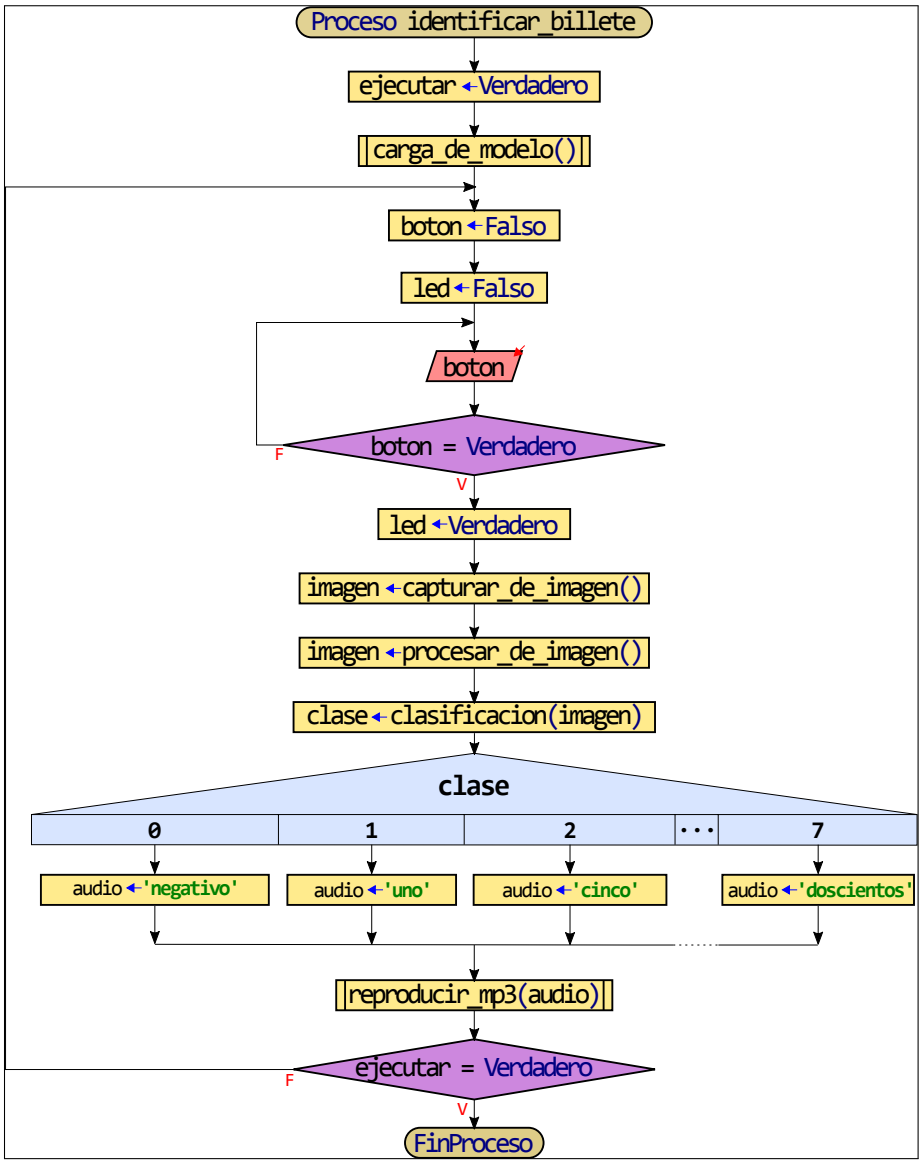
1. Carga del archivo `.tflite` en memoria que contiene el modelo.
2. Procesamiento de los datos de entrada, el modelo espera un formato específico para los datos, para nuestro caso consiste en redimensionar y normalizar las imágenes.
3. La inferencia, implica el uso de *Tensorflow lite* para ejecutar el modelo.
4. Interpretación de la salida, en el momento que se recibe el resultado de la inferencia se debe interpretar el resultado de una manera que sea útil a la aplicación.

Los pasos descritos previamente son el punto de partida para desarrollar cualquier aplicación que ejecute un modelo de inteligencia artificial a partir de *Tensorflow lite* independiente mente del dispositivo que se planea utilizar. La aplicación que ejecuta la Raspberry Pi para clasificar billetes se basa en el procedimiento anterior y puede resumirse en el diagrama de flujo que se muestra en la figura 35, hay que tener en cuenta las consideraciones de hardware que se mencionara en la sección anterior.

5.12. Aplicación Móvil

Una opción adicional para poner aprueba el modelo de clasificación es mediante un teléfono inteligente que ya posee todo el hardware necesario, la diferencia respecto a ejecutar el modelo en una Raspberry Pi con el hardware mencionado previamente, es que en un dispositivo móvil se hace necesario desarrollar una aplicación que sea capaz de ejecutar el modelo, por ello es necesario recurrir a nuevas herramientas como lenguajes de programación y

Figura 35. **Diagrama de flujo simplificado**



Fuente: Elaboración propia empleando Inskape, 13 de junio de 2020.

entornos de desarrollo mientras que en una Raspberry es posible ejecutar el lenguaje de programación Python e instalar el intérprete de Tensorflow lite directamente.

La ventaja principal de un teléfono inteligente, es que al ser un sistema totalmente integrado, con un flujo de desarrollo establecido permite poner a prueba el modelo sin la necesidad de integrar hardware. Por ello para poner el modelo de clasificación se decidió optar por del desarrollo de una aplicación para un teléfono inteligente con sistema operativo Android, esto mediante el kit de desarrollo llamado Flutter y el lenguaje de programación Dart.

5.12.1. Flutter

Es un SDK (siglas en inglés de *software developtment kit*) de código abierto creado por Google para el desarrollo de aplicaciones multiplataforma. Puede ser utilizado para crear aplicaciones para web, iOS, Android, Windows y Linux entre otros, empleando una única base de código. Es relativamente reciente debido a que sus primearas versiones fueron publicadas en el año 2015.

Las aplicaciones de Flutter emplean el lenguaje de programación Dart, un lenguaje de programación optimizado para clientes también desarrollado por Google, tiene muchas de las características de un lenguaje de programación moderno, como programación orientada a objetos, basado en clases y recolector de basura; sus sintaxis es parecida al lenguaje de programación C. Es uno de los lenguajes de más rápido crecimiento en la actualidad.

Sin entrar en detalles con respecto al desarrollo de aplicaciones en Flutter la aplicación para ejecutar el modelo sigue los pasos del procedimiento general establecido en la sección 11 de este capítulo y los pasos principales que ejecuta para realizar una inferencia son similares a los pasos establecidos en el diagrama de la figura 35, todo el código para ejecutar la aplicación se encuentra disponibles en línea el enlace para consultarlo se encuentra en la sección de apéndice.

CONCLUSIONES

1. Se desarrolló un modelo que es capaz de clasificar cada una de las denominaciones de los billetes de Guatemala a la fecha de la elaboración de este trabajo, empleando exclusivamente software con licencia abierta, como se presenta en el capítulo 5.

2. Mediante el capítulo 3 se desarrolla una introducción a la formación, representación y procesamiento de imágenes digitales, que fue complementada por la exposición teórica de la relación entre las redes neuronales con la visión por computador en el capítulo 4.

3. En las secciones 1 a 4 del capítulo 5 se presentaron las herramientas de software libre modernas como los son *Tensorflow*, *Tensorflow lite*, *Keras*, que junto al lenguaje programación Python y con el nuevo auge de la inteligencia artificial se han convertido en la norma para el desarrollo de aplicaciones que emplean aprendizaje de máquina.

4. El modelo desarrollado ofrece una exactitud del $96.4\,\%$ con un conjunto de datos externo que contiene imágenes tomadas bajo condiciones reales, de los billetes en circulación a la fecha de la elaboración de este trabajo, como se muestra en las figuras 29 y 30 en el sección 9 del capítulo 5.

5. Se publicó el conjunto de datos empleado para la elaboración de este trabajo bajo la licencia *Creative Commons* y el código fuente bajo una licencia GNU GPL en su versión 3, ambos se encuentra disponibles en un repositorio en línea, indicado con un enlace en la sección de apéndices.

RECOMENDACIONES

1. Una manera sumamente sencilla para crear modelos de aprendizaje automático para tareas de clasificación es la herramienta disponible en línea y provista por Google llamada `Teachable Machine` que para tareas simples con condiciones de luz controlada puede ser muy útil.

2. El último paso en el proceso de entrenamiento de un clasificador el *fine-tunnig* es considerado opcional debido a que no siempre se vale la pena porque la ganancia en la exactitud puede ser insignificante.

3. Es posible optimizar el modelo para que pueda realizar clasificación en tiempo real con un proceso de cuantización a enteros de 8 bits mediante *Tensoflow lite*, sin embargo este proceso reduce la exactitud aunque se obtiene reducción en el tiempo de latencia y en el tamaño del modelo pero debe utilizarse en dispositivos con arquitectura ARM.

4. Una aplicación para un dispositivo móvil, que ejecuta un modelo de *Tensorflow Lite* puede emplear aceleración por hardware mediante su GPU obteniendo cálculos optimizados y una mayor eficiencia energética.

5. Para entrenar un modelo desde cero especializado para la tarea de clasificación de billetes, que pudiera otorgar un mejor desempeño que el mostrado en este trabajo sería necesario contar con un conjunto de datos mucho más grande (cientos de miles de imágenes) y mayor potencia de cómputo para el entrenamiento.

BIBLIOGRAFÍA

1. ABADI, Martín y col. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467 [cs]* [en línea]. 2016 [consulta: 10 de jul. de 2020]. Disponible en arXiv: 1603.04467.

2. ALEGRE, Enrique; PAJARES, Miguel y DE LA ESCALERA, Arturo. *Conceptos y métodos en visión por computador*. España: s.l., 2016. ISBN 978-84-608-8933-5. OCLC: 957745488.

3. ALVARADO, José. *Procesamiento y Análisis de Imágenes Digitales*. Costa Rica: Tecnológico de Costa Rica, 2012. Disponible en: http://www.tec.ac.cr/sites/default/files/media/doc/paid.pdf.

4. ANILKUMAR, B y SRIKANTH, KRJ. Design and Development of Real Time Paper Currency Recognition System of Demonetization New Indian Notes by Using Raspberry Pi for Visually Challenged. *International Journal of Mechanical Engineering and Technology* [en línea]. 2018, vol. 9, n.º 3, págs. 884-891 [consulta: 9 de jun. de 2019]. ISSN 0976-6359. Disponible en: https://www.iaeme.com/MasterAdmin/uploadfolder/IJMET_09_03_090/IJMET_09_03_090.pdf.

5. BANCO DE GUATEMALA (BANGUAT). *Reseña histórica de la moneda de Guatemala* [en línea] [consulta: 4 de nov. de 2019]. Disponible en: http://www.banguat.gob.gt/page/resena-historica-de-la-moneda-de-guatemala-0.

6. BANCO DE GUATEMALA (BANGUAT). *Seguridad en los Billetes* [en línea] [consulta: 4 de nov. de 2019]. Disponible en: http://www.banguat.gob.gt/page/seguridad-en-los-billetes-0.

7. BIBLIOTECA NACIONAL DE MEDICINA (EE. UU.) *Degeneración macular : MedlinePlus en español* [en línea] [consulta: 7 de nov. de 2020]. Disponible en: https://medlineplus.gov/spanish/maculardegeneration.html#cat_51.

8. BOURNE, Rupert R A y col. Magnitude, temporal trends, and projections of the global prevalence of blindness and distance and near vision impairment: a systematic review and meta-analysis. *The Lancet Global Health* [en línea]. 2017, vol. 5, n.º 9, págs. 888-897 [consulta: 9 de nov. de 2019]. ISSN 2214109X. Disponible en DOI: 10.1016/S2214-109X(17)30293-0.

9. BOYD, Kierstan. *¿Qué es el glaucoma?* [American Academy of Ophthalmology] [en línea] [consulta: 6 de ago. de 2019]. Disponible en: https://www.aao.org/salud-ocular/enfermedades/que-es-la-glaucoma. Library Catalog: www.aao.org.

10. BOYD, Kierstan. *¿Qué es la degeneración macular relacionada con la edad?* [American Academy of Ophthalmology] [en línea] [consulta: 6 de ago. de 2019]. Disponible en: https://www.aao.org/salud-ocular/enfermedades/dmre-degeneracion-macular-relacionada-edad. Library Catalog: www.aao.org.

11. BOYD, Kierstan. *¿Qué es la retinopatía diabética?* [American Academy of Ophthalmology] [en línea] [consulta: 6 de ago. de 2019]. Disponible en: https://www.aao.org/salud-

ocular/enfermedades/retinopatia-diabetica. Library Catalog: www.aao.org.

12. CHOLLET, François. *Keras*. 2015. Disponible en: https://keras.io.

13. CONADI; CBM y UNICEF. *Informe de la II Encuestas Nacional de Discapacidad en Guatemala*. Guatemala: UNICEF, 2016. Disponible en: https://www.unicef.org/guatemala/media/461/file/ENDIS%202016.pdf.

14. CROSSLAND, Michael; GUSTAFSSON, Jörgen; RUMNEY, Nicholas y VEREZEN, Anton. *Documento de Posición Oficial BAJA VISIÓN* [en línea]. Consejo Europeo de Optometría y de Óptica, 2011 [consulta: 9 de nov. de 2019]. Disponible en: https://www.ecoo.info/wp-content/uploads/2011/03/BAJA-VISION.pdf.

15. FROMAGET, Patrick. *The awesome story of Raspberry Pi* [Raspberry tips] [en línea] [consulta: 10 de abr. de 2020]. Disponible en: https://raspberrytips.com/raspberry-pi-history/.

16. GONZALEZ, Rafael C. y WOODS, Richard E. *Digital image processing*. 2nd ed. Upper Saddle River, N.J: Prentice Hall, 2002. ISBN 978-0-201-18075-6.

17. HAYKIN, Simon S. *Neural networks and learning machines*. 3rd ed. New York: Prentice Hall, 2009. ISBN 978-0-13-147139-9. OCLC: ocn237325326.

18. KLETTE, Reinhard. *Concise computer vision: an introduction into theory and algorithms*. London: Springer, 2014. Undergraduate topics in computer science. ISBN 978-1-4471-6319-0 978-1-4471-6320-6. Disponible en DOI: 10.1007/978-1-4471-6320-6. OCLC: ocn865495798.

19. MARTÍN, Raúl y VECILLA, Gerardo. *Manual de optometría*. Madrid: Editorial Médica Panamericana, 2012. ISBN 978-84-9835-272-6. OCLC: 795921651.

20. Museo en la billetera. *Prensa Libre* [en línea]. 2015 [consulta: 8 de oct. de 2019]. Disponible en: https://www.prensalibre.com/hemeroteca/museo-en-la-billetera/.

21. ORGANIZACIÓN MUNDIAL DE LA SALUD (OMS). *10 datos acerca de la ceguera y la discapacidad visual* [en línea] [consulta: 26 de oct. de 2019]. Disponible en: https://www.who.int/features/factfiles/blindness/blindness_facts/es/.

22. ORGANIZACIÓN MUNDIAL DE LA SALUD (OMS). *Ceguera y discapacidad visual* [en línea] [consulta: 5 de nov. de 2019]. Disponible en: https://www.who.int/es/news-room/fact-sheets/detail/blindness-and-visual-impairment.

23. PLANCHE, Benjamin y ANDRES, Eliot. *Hands-On Computer Vision with TensorFlow 2: Leverage deep learning to create powerful image processing apps with TensorFlow 2.0 and Keras*. S.l.: Packt Publishing Limited, 2019. ISBN 978-1-78883-064-5. OCLC: 1103673594.

24. PONCE, Pedro y HERRERA, Alejandro. *Inteligencia artificial con aplicaciones a la ingeniera*. México: Alfaomega Grupo Editor, 2010. ISBN 978-607-7854-83-8. OCLC: 1010618488.

25. RASPBERRY PI FOUNDATION. *About Us* [Raspberry Pi] [en línea] [consulta: 6 de jun. de 2020]. Disponible en: https://www.raspberrypi.org/about/.

26. RASPBERRY PI FOUNDATION. *Raspberry Pi 3 Model A+* [Raspberry Pi] [en línea] [consulta: 10 de jun. de 2020]. Disponible en: https://www.raspberrypi.org/products/raspberry-pi-3-model-a-plus/?resellerType=home.

27. SANDLER, Mark; HOWARD, Andrew; ZHU, Menglong; ZHMOGINOV, Andrey y CHEN, Liang-Chieh. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv:1801.04381 [cs]* [en línea]. 2019 [consulta: 22 de jul. de 2020]. Disponible en arXiv: 1801.04381.

28. SUCAR, L. Enrique y GÓMEZ, Giovani. *Visión computacional*. México, 2011. Disponible en: https://ccc.inaoep.mx/~esucar/Libros/vision-sucar-gomez.pdf.

29. SZELISKI, Richard. *Computer vision: algorithms and applications*. London: Springer, 2011. Texts in computer science. ISBN 978-1-84882-935-0 978-1-84882-934-3. OCLC: 845585642.

APÉNDICE

Repositorios

Conjunto de datos

El conjunto de datos recolectado para el entrenamiento del modelo se encuentra bajo licencia *Creative Commons* y puede ser consultado mediante el siguiente enlace:

```
https://github.com/MiltonVH/gua-currencybills
```

Código de entrenamiento

El código fuente empleado para la elaboración de este trabajo, tanto el *notebook* de entrenamiento, los archivos con extensión .tflite para ejecutar el modelo mediante Tensorflow Lite y el programa de inferencia escrito en Python para probar el modelo en una aplicación de escritorio se encuentran disponibles en el siguiente enlace:

```
https://github.com/MiltonVH/ml-classifier
```

Notebook de entrenamiento

El acceso al código de entrenamiento ejecutado en la herramienta de Google Colaboratory se encuentra disponible en:

```
https://colab.research.google.com/drive/14aYn4Z08XcsHyo49_
-Mlm9QAkb6Xty4n?usp=sharing
```

Prototipo de aplicación

Empleado el kit de desarrollo Flutter, para el desarrollo de aplicaciones multiplataforma se elaboró una aplicación de Android para poder apreciar el funcionamiento del modelo de clasificación, el código fuente esta disponible en:

```
https://github.com/MiltonVH/classifier_app
```

Video demostrativo

Demostración del modelo de clasificación en un dispositivo móvil:

```
https://youtu.be/ZaXZwRSwYso
```