Salas, Federico ; Martino, Julián

Reconocimiento de billetes utilizando redes neuronales, java y android

Tesis de Licenciatura en Sistemas y Computación
Facultad de Química e Ingeniería "Fray Rogelio Bacon"

Facultad de Química e Ingeniería del Rosario

Pontificia Universidad Católica Argentina

RECONOCIMIENTO DE BILLETES UTILIZANDO REDES NEURONALES, JAVA Y ANDROID

Cátedra de Seminario de Sistemas

SALAS, Federico
MARTINO, Julián

PRESENTACIÓN DE LA PROPUESTA

Esta propuesta de trabajo de Tesis está pensada en la investigación y desarrollo científico y a su vez brindar un aporte a la sociedad.

El enfoque de la problemática se centra en la dificultad para las personas con discapacidad visual poder reconocer al momento de realizar un pago utilizando billetes, cual es el valor de los mismos, debiendo confiar en la mayoría de los casos en la persona con la cual está tratando.

En este trabajo se busca crear un prototipo de aplicación para dispositivos móviles basado en el reconocimiento de imágenes para la detección de billetes para personas con discapacidad visual.

Al momento no existe ninguna solución para resolver esta problemática aplicable a nuestro sistema de billetes argentinos.

En el comienzo del trabajo abordaremos la temática de discapacidad y la discapacidad visual para comprender las necesidades y problemáticas diarias que afrontan las personas que son afectadas.

Luego se resumen algunos aspectos teóricos correspondientes a las tecnologías utilizadas para el desarrollo de la investigación.

En el apartado 3 se muestran algunas investigaciones y desarrollos que se han llevado adelante sobre el tema.

En el apartado 4 se evalúan casos de estudios en la búsqueda de alternativas para lograr la solución. Luego se elabora la propuesta teniendo como base los casos de estudio del apartado anterior.

Finalmente se presentan las conclusiones y aspectos no tratados en este trabajo.

UNIVERSIDAD CATÓLICA ARGENTINA

Índice de contenido

1 - Introducción **4**

2 - Marco Teórico **5**

2.1 - Billetes Argentinos (Casos de estudio) 5

2.2 – Discapacidad 9

2.3 – Discapacidad visual 10

2.4 – Redes neuronales artificiales 14

2.5 – Android 22

2.6 – Java 25

2.7 – Algoritmo de Canny 27

3 – Revisión del Estado del Arte **28**

3.1 – Proyectos comercializados 28

3.1.1 – Looktel Money Reader 28

3.1.2 – MCT Money Reader 29

3.2 – Proyectos experimentales 30

3.2.1 – INTI 30

4 – Desarrollo de la propuesta **33**

4.1 – Metodología de trabajo 33

4.2 – Implementaciones prácticas de redes neuronales de Back-propagation 34

4.3 – Caso de Estudio Nº 1 : Leaves Recognition 34

4.4 – Caso de Estudio Nº 2 : BackPropagation Network for Image Recognition (BP Simplified Demo) 49

4.5 – Caso de Estudio Nº 3 : Aplicación de redes neuronales 64

4.6 – Análisis de los casos de estudios 74

4.7 – Desarrollo Aplicación Java 78

4.8 – Desarrollo Aplicación Móvil 102

5 – Conclusiones **172**

6 – Bibliografía **173**

1 - INTRODUCCIÓN

Actualmente las personas que sufren pérdida de visión no cuentan con herramientas que faciliten el uso de dinero en efectivo, sumado a que en nuestro país no se ha implementado un sistema de billetes que contemplen de manera efectiva el caso como sí lo han hecho en otros lugares del mundo.

Aunque existe una perspectiva alentadora con el uso de medios de pagos electrónicos aunque en nuestro país aún es muy baja la implementación.

Esta problemática constituye el puntapié inicial de nuestro trabajo que abordará la temática de discapacidad y las cuestiones técnicas para desarrollar la solución

La problemática del reconocimiento de billetes para personas con discapacidad visual se constituye debido a que si bien nuestra moneda contempla algunas medidas, no son efectivos.

La creación de un software para dispositivos móviles puede resultar útil para mejorarles la calidad de vida a estas personas.

Si bien en este trabajo se desarrollará un prototipo, el mismo será abierto para que sea utilizable por cualquier persona que desea involucrarse en el tema.

Tomamos como condición para resolver la problemática trabajar sobre dispositivos con sistema operativo ANDROID. A lo largo del trabajo explicaremos los motivos.

El lenguaje de programación a utilizar será JAVA y utilizaremos algoritmos de procesamiento de imágenes para obtener patrones que luego evaluaremos a través de una RED NEURONAL de Back Propagation.

El mismo debe ser capaz de lograr capturar la imagen de un billete argentino, reconocer su denominación y brindar el resultado.

Existe una gran complejidad en el desarrollo de software utilizado por personas con discapacidad visual debido a diversos obstáculos que se presentarán en el camino.

2 - MARCO TEÓRICO

2.1 – Billetes argentinos y comparaciones

La muestra de billetes que utilizamos en el desarrollo del trabajo son los existentes al momento del inicio de nuestra investigación y con los cuales entrenamos la red neuronal. El trabajo posibilita la apertura para tomar lo desarrollado, trabajarlo y aplicarlo a todas las denominaciones recientes y futuras.

El trabajo toma como muestra las siguientes denominaciones que se presentan en las figuras 1, 2, 3, 4, 5 y 6:

Figura 1: Billete de 2 pesos - BCRA

Figura 2: Billete de 5 pesos - BCRA

Figura 3: Billete de 10 pesos - BCRA

Figura 4: Billete de 20 pesos - BCRA

Figura 5: Billete de 50 pesos - BCRA

Figura 6: Billete de 100 pesos – BCRA

El único modo de identificar billetes sin verlos es a través del relieve de las marcas en tinta con las que se imprimen, las que pierden su efectividad con el uso, teniendo en cuenta la dificultad que eso representa para quienes tienen discapacidad visual –cerca de 1 millón de personas en Argentina-

Actualmente en la página del Banco Central Argentino informan que los billetes poseen en relieve una identificación, pero los hechos demuestran que esto imposibilita su tacto cuando los mismos adquieren un desgaste de poco tiempo de uso (ver figura 7).

En el momento no existe ninguna iniciativa para cambiar el sistema de impresión ya que esto demanda un costo importante.

Figura 7: Medidas de seguridad en billete de 100 pesos – Identificación para ciegos - BCRA

A continuación se detallaran (ver Tabla 1) algunos casos sobre el sistema de billetes utilizado por algunos países elegidos para representar la problemática.

Cabe destacar que se estudian los billetes porque la moneda gracias a su superficie metálica con relieves denotados claramente al tacto no es un problema para su reconocimiento.

El caso de estudio y su aplicación será realizado sólo y únicamente para la moneda Argentina (Peso), los demás citados nos servirán para ejemplificar y aportar información para la investigación.

Tabla 1- Sistemas de billetes utilizados en algunos países

Moneda	**Sistema de reconocimiento para ciegos.**	**Problemas del sistema de reconocimiento**	**Soporte tecnológico**
Peso argentino	Sistema de relieve para identificación táctil.	Con el uso el relieve pierde su altura.	No existe versión disponible para descarga..
Peso Chileno	Diferentes tamaños. Material resistente al desgaste. Líneas con relieve.	Todavía no se conocen.	No necesita.
Córdoba Nicaragüense	Ventana que a través del tacto se puede reconocer el valor numérico del billete	Todavía no se conocen.	No necesita.
Euro	Sistema de diferenciación por tamaños. A mayor valor, mayor tamaño.	Todavía no se conocen.	No necesita.
Dólar estadounidense	No posee.		Comercializado.

2.2- Discapacidad

Discapacidad ([OMS, Wikipedia]) es un término general que abarca las deficiencias, las limitaciones de la actividad y las restricciones de la participación. Las deficiencias son problemas que afectan a una estructura o función corporal; las limitaciones de la actividad son dificultades para ejecutar acciones o tareas, y las restricciones de la participación son problemas para participar en situaciones vitales.

Por consiguiente, la discapacidad es un fenómeno complejo que refleja una interacción entre las características del organismo humano y las características de la sociedad en la que vive.

La Clasificación Internacional del Funcionamiento, de la Discapacidad y de la Salud (CIF) define la discapacidad como un término genérico que abarca deficiencias, limitaciones de la actividad y restricciones a la participación. Se entiende por discapacidad la interacción entre las personas que padecen alguna enfermedad (por ejemplo, parálisis cerebral, síndrome de Down y depresión) y factores personales y ambientales (por ejemplo, actitudes negativas, transporte y edificios públicos inaccesibles y un apoyo social limitado).

Se calcula que más de mil millones de personas —es decir, un 15% de la población mundial— están aquejadas por la discapacidad en alguna forma. Tienen dificultades importantes para funcionar entre 110 millones (2,2%) y 190 millones (3,8%) personas mayores de 15 años. Eso no es todo, pues las tasas de discapacidad están aumentando debido en parte al envejecimiento de la población y al aumento de la prevalencia de enfermedades crónicas.

La discapacidad es muy diversa. Si bien algunos problemas de salud vinculados con la discapacidad acarrean mala salud y grandes necesidades de asistencia sanitaria, eso no sucede con otros. Sea como fuere, todas las personas con discapacidad tienen las mismas necesidades de salud que la población en general y, en consecuencia, necesitan tener acceso a los servicios corrientes de asistencia sanitaria. En el artículo 25 de la Convención sobre los derechos de las personas con discapacidad se reconoce que las personas con discapacidad tienen derecho a gozar del más alto nivel posible de salud sin discriminación.

2.3 - DISCAPACIDAD VISUAL[1]:

Con arreglo a la Clasificación Internacional de Enfermedades (CIE-10, actualización y revisión de 2006), la función visual se subdivide en cuatro niveles:

- visión normal;
- discapacidad visual moderada;
- discapacidad visual grave;
- ceguera.

La discapacidad visual moderada y la discapacidad visual grave se reagrupan comúnmente bajo el término «baja visión »; la baja visión y la ceguera representan conjuntamente el total de casos de discapacidad visual.

Principales causas de discapacidad visual

La distribución mundial de las principales causas de discapacidad visual es como sigue:

- errores de refracción (miopía, hipermetropía o astigmatismo) no corregidos: 43%;
- cataratas no operadas: 33%;
- glaucoma: 2%.

¿Quién está en riesgo?

Aproximadamente un 90% de la carga mundial de discapacidad visual se concentra en los países de ingresos bajos.

Personas de 50 años o más

Alrededor de un 65% de las personas con discapacidad visual son mayores de 50 años, si bien este grupo de edad apenas representa un 20% de la población mundial. Con una población anciana en aumento en muchos países, más personas estarán en riesgo de sufrir discapacidad visual por enfermedades oculares crónicas y envejecimiento.

[1] CEGUERA Y DISCAPACIDAD VISUAL - OMS : http://www.who.int/es/news-room/fact-sheets/detail/blindness-and-visual-impairment

Menores de 15 años

Se estima que el número de niños con discapacidad visual asciende a 19 millones, de los cuales 12 millones la padecen debido a errores de refracción, fácilmente diagnosticables y corregibles. Unos 1,4 millones de menores de 15 años sufren ceguera irreversible y necesitan intervenciones de rehabilitación visual para su pleno desarrollo psicológico y personal.

Evolución en los últimos 20 años

En términos generales, las tasas mundiales de discapacidad visual han disminuido desde comienzos de los años noventa. Ello pese al envejecimiento de la población en el mundo entero. Esa disminución se debe principalmente a la reducción del número de casos de discapacidad visual por enfermedades infecciosas, mediante:

- el desarrollo socioeconómico en general;
- una actuación concertada de salud pública;
- un aumento de los servicios de atención oftalmológica disponibles;
- el conocimiento por parte de la población general de las soluciones a los problemas relacionados con la discapacidad visual (por ejemplo, cirugía o dispositivos correctores).

Respuesta mundial para prevenir la ceguera

En todo el mundo, el 80% de todas las discapacidades visuales se pueden prevenir o curar. En los últimos 20 años se han realizado progresos en las esferas siguientes:

- implantación, por los gobiernos, de programas y normas para la prevención y el control de la discapacidad visual;
- incorporación paulatina de los servicios de oftalmología en los sistemas de atención primaria y secundaria, con énfasis en la prestación de servicios accesibles, asequibles y de alta calidad;
- campañas de educación y sensibilización sobre la importancia de la función visual, incluida la educación en las escuelas; y
- liderazgo gubernamental reforzado en las alianzas internacionales, con una creciente participación del sector privado.

- Los datos de los últimos 20 años revelan que en muchos países se han logrado progresos importantes en lo que respecta a la prevención y cura de las discapacidades visuales. Además, la reducción masiva de la ceguera asociada a la oncocercosis y el tracoma es parte de una disminución significativa de la distribución de la enfermedad, y ha permitido limitar sustancialmente la carga de morbilidad derivada de esas enfermedades infecciosas. Ello ha sido posible gracias a la fructífera labor de algunas alianzas internacionales público-privadas.

Datos y cifras

- En el mundo hay aproximadamente 285 millones de personas con discapacidad visual, de las cuales 39 millones son ciegas y 246 millones presentan baja visión.
- Aproximadamente un 90% de la carga mundial de discapacidad visual se concentra en los países de ingresos bajos.
- El 82% de las personas que padecen ceguera tienen 50 años o más.
- En términos mundiales, los errores de refracción no corregidos constituyen la causa más importante de discapacidad visual, pero en los países de ingresos medios y bajos las cataratas siguen siendo la principal causa de ceguera.
- El número de personas con discapacidades visuales atribuibles a enfermedades infecciosas ha disminuido considerablemente en los últimos 20 años.
- El 80% del total mundial de casos de discapacidad visual se pueden evitar o curar.

ACCESIBILIDAD DE PERSONAS CON DISCAPACIDAD VISUAL A LOS MEDIOS ELECTRÓNICOS[2]

La accesibilidad de los medios electrónicos es la facilidad de uso de las tecnologías de la información y la comunicación (TIC), tales como Internet, por personas con discapacidad. La presentación de los sitios web debe permitir que los usuarios discapacitados tengan acceso a la información. Por ejemplo:

- para los usuarios ciegos, los sitios web deben ser interpretados por programas que lean los textos en voz alta y describan las imágenes;
- para los usuarios con grave discapacidad visual, el tamaño de los textos debe ser modificable, y los colores deben contrastar claramente;
- para los usuarios sordos o con deficiencias auditivas, los documentos en audio deben ir acompañados de las correspondientes transcripciones o de un video con lenguaje de signos.

A nivel internacional, el World Wide Web Consortium (W3C) ha preparado unas directrices sobre la accesibilidad de los contenidos en la web. La Convención sobre los derechos de las personas con discapacidad, que entró en vigor el 3 de mayo de 2008, también subraya que hay que garantizar a las personas con discapacidad la igualdad de acceso a las TIC, y contribuirá a eliminar los obstáculos que impiden el acceso a la información, en particular por Internet. Las leyes y reglamentos nacionales pueden fomentar el cumplimiento de las normas de accesibilidad.

Para muchas personas, las TIC, entre ellas Internet, son hoy en día indispensables para la economía, la educación y la vida social. Para que las personas con discapacidad tengan iguales posibilidades de acceso a la información que los demás, los sitios web deben poder ser consultados por todos.

[2] ¿Qué es la accesibilidad de los medios electrónicos? – OMS - http://www.who.int/features/qa/50/es/

2.4 - REDES NEURONALES ARTIFICIALES ([Wikipedia, UTN])

ACERCA DE LAS REDES NEURONALES:

El hombre se ha caracterizado siempre por su búsqueda constante de nuevas vías para mejorar sus condiciones de vida. Estos esfuerzos le han servido para reducir el trabajo en aquellas operaciones en las que la fuerza juega un papel primordial. Los progresos obtenidos han permitido dirigir estos esfuerzos a otros campos, como por ejemplo, a la construcción de máquinas calculadoras que ayuden a resolver de forma automática y rápida determinadas operaciones que resultan tediosas cuando se realizan a mano.

Uno de los primeros en acometer esta empresa fue Charles Babbage, quien trató infructuosamente de construir una máquina capaz de resolver problemas matemáticos.

Posteriormente otros tantos intentaron construir máquinas similares, pero no fue hasta la Segunda Guerra Mundial, cuando ya se disponía de instrumentos electrónicos, que se empezaron a recoger los primeros frutos. En 1946 se construyó la primera computadora electrónica, ENIAC. Desde entonces los desarrollos en este campo han tenido un auge espectacular.

Estas máquinas permiten implementar fácilmente algoritmos para resolver multitud de problemas que antes resultaban engorrosos de resolver. Sin embargo, se observa una limitación importante: ¿qué ocurre cuando el problema que se quiere resolver no admite un tratamiento algorítmico, como es el caso, por ejemplo, de la clasificación de objetos por rasgos comunes? Este ejemplo demuestra que la construcción de nuevas máquinas más versátiles requiere un enfoque del problema desde otro punto de vista. Los desarrollos actuales de los científicos se dirigen al estudio de las capacidades humanas como una fuente de nuevas ideas para el diseño de las nuevas máquinas. Así, la inteligencia artificial es un intento por descubrir y describir aspectos de la inteligencia humana que pueden ser simulados mediante máquinas. Esta disciplina se ha desarrollado fuertemente en los últimos años teniendo aplicación en algunos campos como visión artificial, demostración de teoremas, procesamiento de información expresada mediante lenguajes humanos.

Las redes neuronales son más que otra forma de emular ciertas características propias de los humanos, como la capacidad de memorizar y de asociar hechos. Si se examinan con atención aquellos problemas que no pueden expresarse a través de un

algoritmo, se observará que todos ellos tienen una característica en común: la experiencia.

El hombre es capaz de resolver estas situaciones acudiendo a la experiencia acumulada. Así, parece claro que una forma de aproximarse al problema consista en la construcción de sistemas que sean capaces de reproducir esta característica humana. En definitiva, las redes neuronales no son más que un modelo artificial y simplificado del cerebro humano, que es el ejemplo más perfecto del que disponemos para un sistema que es capaz de adquirir conocimiento a través de la experiencia. Una red neuronal es "un nuevo sistema para el tratamiento de la información, cuya unidad básica de procesamiento está inspirada en la célula fundamental el sistema nervioso humano: *la neurona*".

Todos los procesos del cuerpo humano se relacionan en alguna u otra forma con la (in)actividad de estas neuronas. Las mismas son un componente relativamente simple del ser humano, pero cuando millares de ellas se conectan en forma conjunta se hacen muy poderosas.

Lo que básicamente ocurre en una neurona biológica es lo siguiente: la neurona es estimulada o excitada a través de sus *entradas* (inputs) y cuando se alcanza un cierto umbral, la neurona se dispara o activa, pasando una señal hacia el *axon*. Posteriores investigaciones condujeron al descubrimiento de que estos procesos son el resultado de eventos electroquímicos.

Como ya se sabe, el pensamiento tiene lugar en el cerebro, que consta de billones de neuronas interconectadas. Así, el secreto de la "inteligencia" -sin importar cómo se defina- se sitúa dentro de estas neuronas interconectadas y de su interacción. También, es bien conocido que los humanos son capaces de aprender. Aprendizaje significa que aquellos problemas que inicialmente no pueden resolverse, pueden ser resueltos después de obtener más información acerca del problema. Las redes neuronales

- ***Consisten de unidades de procesamiento que intercambian datos o información.***
- ***Se utilizan para reconocer patrones, incluyendo imágenes, manuscritos y secuencias de tiempo (por ejemplo: tendencias financieras).***
- ***Tienen capacidad de aprender y mejorar su funcionamiento.***

Una primera clasificación de los modelos de redes neuronales podría ser, atendiendo a su similitud con la realidad biológica:

1) ***El modelo de tipo biológico.*** Este comprende las redes que tratan de simular los sistemas neuronales biológicos, así como las funciones auditivas o algunas funciones básicas de la visión.

2) ***El modelo dirigido a aplicación.*** Este modelo no tiene por qué guardar similitud con los sistemas biológicos. Su arquitectura está fuertemente ligada a las necesidades de las aplicaciones para la que es diseñada.

DEFINICIÓN DE RED NEURONAL ARTIFICIAL:

Existen numerosas formas de definir a las redes neuronales; desde las definiciones cortas y genéricas hasta las que intentan explicar más detalladamente qué son las redes neuronales. Por ejemplo:

1) Una nueva forma de computación, inspirada en modelos biológicos.

2) Un modelo matemático compuesto por un gran número de elementos procesales organizados en niveles.

3) Un sistema de computación compuesto por un gran número de elementos simples, elementos de procesos muy interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas.

4) Redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico.

Una red neuronal se compone de unidades llamadas neuronas. Cada neurona recibe una serie de entradas a través de interconexiones y emite una salida. Esta salida viene dada por tres funciones:

1. ***Una función de propagación*** (también conocida como función de excitación), que por lo general consiste en el sumatorio de cada entrada multiplicada por el peso de su interconexión (valor neto). Si el peso es positivo, la conexión se denomina *excitatoria*; si es negativo, se denomina *inhibitoria*.

2. ***Una función de activación,*** que modifica a la anterior. Puede no existir, siendo en este caso la salida la misma función de propagación.

3. ***Una función de transferencia***, que se aplica al valor devuelto por la función de activación. Se utiliza para acotar la salida de la neurona y generalmente viene dada por la interpretación que queramos darle a dichas salidas. Algunas de las más utilizadas son la función sigmoidea (para obtener valores en el intervalo (0,1)) y la tangente hiperbólica (para obtener valores en el intervalo (-1,1)).

VENTAJAS:

Las redes neuronales artificiales (RNA) tienen muchas ventajas debido a que están basadas en la estructura del sistema nervioso, principalmente el cerebro.

- Aprendizaje: Las RNA tienen la habilidad de aprender mediante una etapa que se llama *etapa de aprendizaje*. Esta consiste en proporcionar a la RNA datos como entrada a su vez que se le indica cuál es la salida (respuesta) esperada.

- Auto organización: Una RNA crea su propia representación de la información en su interior, quitándole esta tarea al usuario.

- Tolerancia a fallos: Debido a que una RNA almacena la información de forma redundante, ésta puede seguir respondiendo de manera aceptable aun si se daña parcialmente.

- Flexibilidad: Una RNA puede manejar cambios no importantes en la información de entrada, como señales con ruido u otros cambios en la entrada (por ejemplo si la información de entrada es la imagen de un objeto, la respuesta correspondiente no sufre cambios si la imagen cambia un poco su brillo o el objeto cambia ligeramente).

- Tiempo real: La estructura de una RNA es paralela, por lo cual si esto es implementado con computadoras o en dispositivos electrónicos especiales, se pueden obtener respuestas en tiempo real.

Topología

Una primera clasificación de las redes de neuronas artificiales que se suele hacer es en función del patrón de conexiones que presenta. Así se definen tres tipos básicos de redes:

- Dos tipos de redes de propagación hacia delante o acíclicas en las que todas las señales van desde la capa de entrada hacia la salida sin existir ciclos, ni conexiones entre neuronas de la misma capa de red neuronal y su clasificación.

- ***Monocapa.*** Ejemplos: perceptrón, Adaline.

- ***Multicapa.*** Ejemplos: perceptrón multicapa.

- Las redes recurrentes que presentan al menos un ciclo cerrado de activación neuronal. Ejemplos: Elman, Hopfield, máquina de Boltzmann.

Aprendizaje

Una segunda clasificación que se suele hacer es en función del tipo de aprendizaje de que es capaz (si necesita o no un conjunto de entrenamiento supervisado). Para cada tipo de aprendizaje encontramos varios modelos propuestos por diferentes autores:

- ***Aprendizaje supervisado***: necesitan un conjunto de datos de entrada previamente clasificado o cuya respuesta objetivo se conoce. Ejemplos de este tipo de redes son: el perceptrón simple, la red Adaline, elperceptrón multicapa, red backpropagation, y la memoria asociativa bidireccional.

- ***Aprendizaje no supervisado o autoorganizado***: no necesitan de tal conjunto previo. Ejemplos de este tipo de redes son: las memorias asociativas, las redes de Hopfield, la máquina de Boltzmann y lamáquina de Cauchy, las redes de aprendizaje competitivo, las redes de Kohonen o mapas autoorganizados y las redes de resonancia adaptativa (ART).

- ***Redes híbridas***: son un enfoque mixto en el que se utiliza una función de mejora para facilitar la convergencia. Un ejemplo de este último tipo son las redes de base radial.

- ***Aprendizaje reforzado***: se sitúa a medio camino entre el supervisado y el autoorganizado.

Tipo de entrada

Finalmente también se pueden clasificar las RNAs según sean capaces de procesar información de distinto tipo en:

- ***Redes analógicas***: procesan datos de entrada con valores continuos y, habitualmente, acotados. Ejemplos de este tipo de redes son: Hopfield, Kohonen y las redes de aprendizaje competitivo.

- ***Redes discretas***: procesan datos de entrada de naturaleza discreta; habitualmente valores lógicos booleanos. Ejemplos de este segundo tipo de redes son: las máquinas de Boltzmann y Cauchy, y la red discreta de Hopfield.

MODELO DE NEURONA ARTIFICIAL

El modelo de Rumelhart y McClelland (1986) define un elemento de proceso (EP), o neurona artificial, como un dispositivo que a partir de un conjunto de entradas, x_i (i=1...n) o vector x, genera una única salida y.

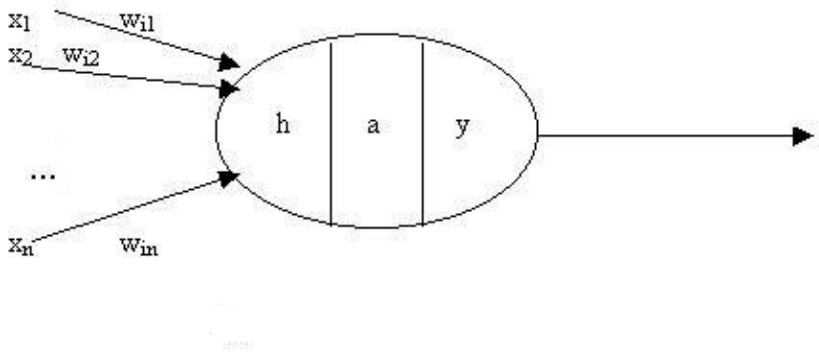


Figura 8 – Modelo de Neurona Artificial - Wikipedia

Esta neurona artificial consta de los siguientes elementos:

- Conjunto de entradas o vector de entradas x, de n componentes
- Conjunto de *pesos sinápticos* w_{ij}. Representan la interacción entre la neurona presináptica j y la postsináptica i.
- *Regla de propagación* $d(w_{ij},x_j(t))$: proporciona el potencial postsináptico, $h_i(t)$.
- *Función de activación* $a_i(t)=f(a_i(t-1), h_i(t))$: proporciona el estado de activación de la neurona en función del estado anterior y del valor postsináptico.
- F*unción de salida* $F_i(t)$: proporciona la salida $y_i(t)$, en función del estado de activación.

Las señales de entrada y salida pueden ser señales binarias (0,1 – neuronas de McCulloch y Pitts), bipolares (-1,1), números enteros o continuos, variables borrosas, etc.

Entonces, una red neuronal artificial (RNA) se puede definir (Hecht – Nielssen 93) como un grafo dirigido con las siguientes restricciones:

1. Los nodos se llaman *elementos de proceso* (EP).
2. Los enlaces se llaman *conexiones* y funcionan como caminos unidireccionales instantáneos
3. Cada EP puede tener cualquier número de conexiones.
4. Todas las conexiones que salgan de un EP deben tener la misma señal.
5. Los EP pueden tener *memoria local*.
6. Cada EP posee una *función de transferencia* que, en función de las entradas y la memoria local produce una señal de salida y / o altera la memoria local.
7. Las entradas a la RNA llegan del mundo exterior, mientras que sus salidas son conexiones que abandonan la RNA.

ARQUITECTURA

La arquitectura de una RNA es la estructura o patrón de conexiones de la red. Es conveniente recordar que las conexiones sinápticas son direccionales, es decir, la información sólo se transmite en un sentido.

En general, las neuronas suelen agruparse en unidades estructurales llamadas *capas*. Dentro de una capa, las neuronas suelen ser del mismo tipo. Se pueden distinguir tres tipos de capas:

- ***De entrada:*** reciben datos o señales procedentes del entorno.
- ***De salida***: proporcionan la respuesta de la red a los estímulos de la entrada.
- ***Ocultas***: no reciben ni suministran información al entorno (procesamiento interno de la red).

Generalmente las conexiones se realizan entre neuronas de distintas capas, pero puede haber conexiones intracapa o *laterales* y conexiones de *realimentación* que siguen un sentido contrario al de entrada-salida.

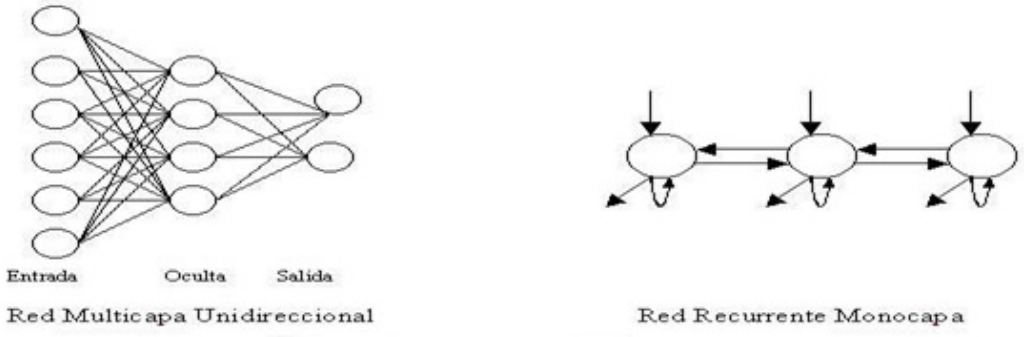


Figura 9 – Arquitectura de RNA - Wikipedia

APRENDIZAJE

Es el proceso por el que una RNA actualiza los pesos (y, en algunos casos, la arquitectura) con el propósito de que la red pueda llevar a cabo de forma efectiva una tarea determinada.

Hay tres conceptos fundamentales en el aprendizaje:

Paradigma de aprendizaje: información de la que dispone la red.

Regla de aprendizaje: principios que gobiernan el aprendizaje.

Algoritmo de aprendizaje: procedimiento numérico de ajuste de los pesos.

Existen dos paradigmas fundamentales de aprendizaje:

Supervisado: la red trata de minimizar un error entre la salida que calcula y la salida deseada (conocida), de modo que la salida calculada termine siendo la deseada.

No supervisado o autoorganizado: la red conoce un conjunto de patrones sin conocer la respuesta deseada. Debe extraer rasgos o agrupar patrones similares.

En cuanto a los algoritmos de aprendizaje, tenemos cuatro tipos:

- ***Minimización del error:*** reducción del gradiente, retropropagación, etc. La modificación de pesos está orientada a que el error cometido sea mínimo.
- ***Boltzmann:*** para redes estocásticas, donde se contemplan parámetros aleatorios.
- ***Hebb:*** cuando el disparo de una célula activa otra, el peso de la conexión entre ambas tiende a reforzarse (Ley de Hebb).
- ***Competitivo:*** sólo aprenden las neuronas que se acercan más a la salida deseada.

Los algoritmos, y en general el proceso de aprendizaje, son complejos y suelen llevar bastante tiempo computacionalmente hablando. Su ventaja es que una vez ha aprendido, la red puede congelar sus pesos y funcionar en modo *recuerdo* o *ejecución.*

2.5 - ANDROID

Android es un sistema operativo pensado para dispositivos móviles. Está basado en Linux y ofrece un núcleo libre, gratuito y multiplataforma. Esto representa una gran ventaja respecto a su gran competidor iOS ya que los dispositivos que incorporan Android son más baratos y, por lo tanto, más accesibles.

Uno de los aspectos que da más confianza a los usuarios más curiosos e informados es que es de código abierto y puede descargarse desde https://source.android.com/setup/build/downloading

Al ser de código abierto, puede ser visto por cualquiera, analizado y encontrar potenciales mejoras o, inclusive, fallos de seguridad.

Historia

En la página oficial de Android se puede ver la historia de este sistema operativo en detalle, accediendo a https://www.android.com/intl/es-419_mx/history/#/marshmallow

Los principios de este sistema operativo no fueron tan explosivos como sí lo fue más adelante. Al principio, cuando Google compró, en 2005, Android era casi desconocido.

Luego, a fines de 2010, el crecimiento de Android se volvió de carácter exponencial. Empezó a ser el SO para móviles más utilizado del mundo.

Con el avance de la tecnología y la aparición de nuevos dispositivos móviles como las Tablets, Android fue expandiéndose cada vez más. Hoy en día, hay televisores inteligentes que cuentan con Android para utilizar diferentes tipos de aplicaciones como Netflix, Youtube, etcétera.

Entornos para programar en Android

Enfocar el perfil como desarrollador Android hoy es una de las mejores demandas en el mercado ya que hay muchos puestos a llenar.

Google ha puesto a disposición el entorno Android Studio para poder desarrollar aplicaciones móviles en Java para Android. También se brinda a la siguiente página https://developer.android.com/ como soporte para los desarrolladores.

Actualmente, se cuenta con numerosos entornos de desarrollos además del provisto por Google. Algunos de los entornos disponibles son:

- Android Studio
- Basic 4 Android
- Mono para Android
- App Inventor
- LiveCode
- InDesign CS6

Ventajas

- Portabilidad: Hoy en día, gracias a la expansión IoT (Internet of Things), este sistema operativo puede encontrarse en TVs, Autos, Microondas, Lavarropas y casi cualquier dispositivo con pantalla táctil.
- Código abierto: Como se dijo antes, al ser de código libre, Android puede ser manipulado, analizado y personalizado por los usuarios. Esto, a su vez, genera muchas oportunidades de mejoras ya que hay personas que se dedican a encontrar fallas y reportarlas.
- Multitarea: Es capaz de gestionar varias tareas en paralelo. Administra la memoria y el procesador para poder ejecutar diferentes procesos en paralelo y sacar de memoria aquellos que se encuentren inactivos.
- Comunidad: Hoy en día cuenta con una amplia comunidad activa que sube material a Internet para que todos podamos acceder a estos. Ya han surgido una gran cantidad de problemas los cuales se han resuelto y documentado. Esto viene de la mano con la vasta documentación que hay disponible para los que la necesiten.

Desventajas

- Flexibilidad para aplicaciones: Un gran inconveniente para los usuarios es que Google no es tan rígido en cuanto a los criterios de aceptación para que los diferentes desarrolladores puedan subir al Store sus aplicaciones. Esto, por un lado, genera que haya una gran variedad de aplicaciones de todo tipo en el mercado de Android (ya sean libres o gratuitas) pero, por el otro lado, la calidad de estas aplicaciones no siempre es de lo más alto y se halla, con frecuencia, usuarios insatisfechos.
- Optimización de la batería: Generalmente, una de las mayores quejas de los usuarios es que se agota muy rápido la batería de sus dispositivos que usan Android. Esto se debe a una mala gestión de la memoria. A pesar que hay, en el mercado, aplicaciones encargadas de mejorar esta gestión, no es lo más óptimo que se deba instalar una aplicación extra para manejar dicho proceso.

2.6 -JAVA

Java es un lenguaje de programación que sigue el paradigma orientado a objetos. Hoy en día, su uso es extendido en cualquier dispositivo debido a su versatilidad. Tiene la ventaja que sigue la idea de WORA (Write One, Run Anywhere). Es decir, se escribe el código una única vez, se compila y se implementa. El código que es compilado para una plataforma, no debe ser recompilado para ser ejecutado en otra.

Historia

Java fue lanzado oficialmente en 1995. Este lenguaje fue desarrollado, primeramente, por Sun Microsystems aunque luego fue adquirido por Oracle que es quien hoy en día, sigue manteniendo la continuidad del lenguaje.

Java ha revolucionado el mercado. A pesar que había otros lenguajes con características similares en cuanto a posibilidades de desarrollo, fue el uso de JVM lo que permitió a este lenguaje expandirse tanto. En la sección “Compilación de código Java” se detalla este tema.

Actualmente, Java es utilizado ampliamente y ha tenido un gran empujón debido a que Google decidió utilizar este lenguaje para el desarrollo de su sistema operativo Android para dispositivos móviles.

Compilación de código Java

El proceso de compilación es, en esencia, traducir código inentendible por la computadora a código de bajo nivel que sí es entendido por ésta. Generalmente, lo que se hace en lenguajes de alto nivel como C, C++ o Python es traducir el código de alto nivel a código de máquina. En Java no es así.

Primeramente, se hace una “compilación intermedia”. Con “intermedia”, me refiero a que se traduce el código a un lenguaje intermedio llamado Byte Code. Este código, en realidad, no es entendido por la máquina real donde se va a correr la aplicación sino que se implementa una máquina virtual que entiende este código. Esta máquina virtual es denominada Java Virtual Machine, toma el Byte Code y lo traduce a código que ella entiende para, luego, ejecutar la aplicación. Gracias a esta metodología, se puede cumplir

con WORA ya que el código siempre es traducido a Byte Code y entendido por JVM. De esta manera, el código puede ser extendido a cualquier plataforma.

Ventajas

- **Lenguaje orientado a objetos:** Debido a esto, hereda todas las ventajas de usar el paradigma de orientación a objetos. Representación de objetos, código reutilizable, encapsulamiento de información, herencia, polimorfismo, entre otros.
- **Comunidad:** Hoy en día cuenta con una amplia comunidad activa que sube material a Internet para que todos podamos acceder a estos. Ya han surgido una gran cantidad de problemas los cuales se han resuelto y documentado. Esto viene de la mano con la vasta documentación que hay disponible para los que la necesiten.
- **Multiplataforma:** Como ya se dijo antes, gracias a la JVM, puede utilizarse Java para desarrollar en distintas aplicaciones sin necesidad de recompilar las soluciones.
- **Manejo automático de la memoria:** El manejo de la memoria se hace automáticamente y utilizando el garbage collector.
- **No posee licencia:** Una de las grandes ventajas que tiene usar Java es que no requiere de ningún tipo de licencia para desarrollar aplicaciones de carácter comercial.

Frameworks

Se han desarrollado una serie de Frameworks para implementar Java. Obviamente, cada uno tiene su orientación y sus ventajas y desventajas respecto a los demás. Aquí se muestra una lista de los 10 frameworks más utilizados según un estudio realizado por Rebellabs basado 4 diferentes fuentes como son Linkedin, Github, búsquedas en Google y StackOverflow

2.7 - ALGORITMO DE CANNY [3]

Algoritmo de Canny ([Wikipedia]) es un operador desarrollado por John F. Canny en 1986 que utiliza un algoritmo de múltiples etapas para detectar una amplia gama de bordes en imágenes. Lo más importante es que Canny también desarrolló una teoría computacional acerca de la detección de bordes que explica por qué la técnica funciona.

El propósito de Canny era descubrir el algoritmo óptimo de detección de bordes. Para que un detector de bordes pueda ser considerado óptimo debe cumplir los siguientes puntos:

- Buena detección- el algoritmo debe marcar el mayor número real en los bordes de la imagen como sea posible.
- Buena localización- los bordes de marca deben estar lo más cerca posible del borde de la imagen real.
- Respuesta mínima - El borde de una imagen sólo debe ser marcado una vez, y siempre que sea posible, el ruido de la imagen no debe crear falsos bordes.

Para satisfacer estos requisitos Canny utiliza el cálculo de variaciones - una técnica que encuentra la función que optimiza un funcional indicado. La función óptima en el algoritmo de Canny es descrito por la suma de cuatro términos exponenciales, pero se puede aproximar por la primera derivada de una gaussiana.

Etapas Algoritmo de Canny

Reducción de ruido

El algoritmo de detección de bordes de Canny utiliza un filtro basado en la primera derivada de una gaussiana. Ya que es susceptible al ruido presente en datos de imagen sin procesar la imagen original es transformada con un filtro gaussiano. El resultado es una imagen un poco borrosa respecto a la versión original. Esta nueva imagen no se ve afectada por un píxel único de ruido en un grado significativo.

[3] ALGORITMO DE CANNY: Wikipedia – Algoritmo de Canny : https://es.wikipedia.org/wiki/Algoritmo_de_Canny

Encontrar la intensidad del gradiente de la imagen

El borde de una imagen puede apuntar en diferentes direcciones, por lo que el algoritmo de Canny utiliza cuatro filtros para detectar horizontal, vertical y diagonal en los bordes de la imagen borrosa. El operador de detección de bordes devuelve un valor para la primera derivada en la dirección horizontal (Gy) y la dirección vertical (Gx). A partir de éste, se pueden determinar el gradiente de borde y la dirección

3 - REVISION DEL ESTADO DEL ARTE

En la actualidad existen diversos software para el reconocimiento de imágenes de todo tipo. Google ha desarrollado “Googles” la cual es capaz de reconocer una gran cantidad de objetos, pero la misma no está pensada para solucionar la problemática del reconocimiento de billetes en personas con discapacidad visual.

Para billetes argentinos existen algunos proyectos en estudio utilizando librerías de procesamiento de imágenes, pero ninguna solución utilizando JAVA, Redes Neuronales bajo sistema operativo Android.

3.1 -SOLUCIONES COMERCIALES

Para billetes como el dólar estadounidense existen aplicaciónes solo para dispositivos Apple y en algunos casos no gratuitas o con publicidad tales como:

3.1.1 - LookTel Money Reader [4]

LookTel Money Reader reconoce al instante la moneda y dice la denominación, lo que permite a las personas que experimentan impedimentos visuales o ceguera identificar y contar facturas de manera rápida y fácil. Apunte la cámara de su dispositivo iOS a una factura y la aplicación le indicará la denominación en tiempo real. Veintiuna monedas son compatibles: el dólar estadounidense, el dólar australiano, el dinar de Bahrein, el real brasileño, el rublo bielorruso, la libra esterlina, el dólar canadiense, el euro, el forint húngaro, el shekel israelí, la rupia india, el yen japonés, el dinar kuwaití y el peso mexicano. Dólar de Zelandia, Zloty polaco, Rublo ruso, Riyal de Arabia Saudita, Dólar de

[4] LookTel Money Reader : http://www.looktel.com/moneyreader

Singapur y Dirham de los Emiratos Árabes Unidos. Con la tecnología de reconocimiento de objetos patentada y patentada de LookTel, el lector de dinero de LookTel hace que sea tan fácil reconocer las facturas como sea posible. No es necesario mantener el dispositivo iOS fijo o capturar una foto y esperar un resultado.

El reconocimiento ocurre instantáneamente, en tiempo real. La aplicación no requiere una conexión a Internet, lo que significa que leerá dinero en cualquier ubicación y en cualquier momento. LookTel Money Reader es un asistente móvil útil que es simple y fácil de usar. Mientras compra, use la aplicación para verificar el dinero mientras realiza el pago o para asegurarse de recibir la cantidad correcta de cambio.

La denominación también se muestra en el dispositivo iOS en números grandes de alto contraste, para aquellos que tienen suficiente visión para hacer uso de la pantalla. LookTel Money Reader ofrece compatibilidad con Voice Over para varios idiomas, incluidos inglés, español, francés, italiano, alemán, polaco, portugués, ruso, coreano, finlandés, danés, sueco, noruego, japonés, griego, húngaro y mandarín.

LookTel Money Reader para Mac es el primer identificador de moneda disponible para Apple.

3.1.2 - MCT MONEY READER[5]

MCT Money Reader identifica al instante algunas monedas y transfiere el resultado al usuario a través de texto a voz. Esta aplicación cumple con los estándares de accesibilidad para personas ciegas o con discapacidad visual que pueden identificar y contar de manera fácil y rápida los billetes. No se requiere conexión a Internet para este uso, la aplicación funciona sin conexión a Internet. El reconocimiento tiene lugar en menos de 1 segundo en condiciones normales. El flash del teléfono funciona para permitir su uso en entornos oscuros.

Los billetes reconocidos son; Dólar estadounidense, euro, lira turca, rublo ruso, yen japonés, yuan chino, rupia indonesia, rupia pakistaní, pesos mexicanos, peso colombiano, pesos de Filipinas, won surcoreano, libra egipcia, manat de Azerbaiyán, zloty polaco,

[5] MCT Money Reader en Google Play :
https://play.google.com/store/apps/details?id=com.mctdata.ParaTanima

corona sueca, rial iraní, dong vietnamita, hryvnia ucraniano, riyali de Arabia Saudita, baht tailandés.

La aplicación te redirigirá con un tono audible cuando busques el billete. También hay un mostrador para recolectar la cantidad de dinero que introduce en la práctica. Debido a que las superficies de algunos billetes de banco son muy similares entre sí, es útil mostrar la otra superficie del billete de banco siempre que se extienda el período de reconocimiento de billetes de la aplicación. En general, la introducción de ambas superficies del billete de banco será la forma más saludable.

MCT Money Reader es compatible con varios idiomas, como inglés, español, polaco, ruso, coreano, sueco, japonés, chino, indonesio, urdu, árabe, persa, filipino, tailandés, ucraniano, vietnamita, azerbaiyano y turco.

3.2 – PROYECTOS EXPERIMENTALES

3.2.1 – INTI [6]

El INTI diseñó un prototipo de sistema de reconocimiento de billetes argentinos que utiliza software libre y funciona en dispositivos móviles y celulares.

Hoy en día, el único modo de identificar billetes sin verlos es a través del relieve de las marcas en tinta con las que se imprimen, las que pierden su efectividad con el uso. Teniendo en cuenta la dificultad que eso representa para quienes tienen discapacidad visual –cerca de 1 millón de personas en Argentina–, el Laboratorio de Desarrollo en Electrónica e Informática de INTI-Córdoba junto al Centro de Tecnologías para la Salud y Discapacidad del INTI desarrollaron el primer prototipo, en etapa de testeo, de una aplicación que utiliza *software* libre para teléfonos celulares capaz de identificar billetes argentinos y comunicar su denominación (valor) por medios auditivos.

“Junto con integrantes de la Biblioteca Argentina para Ciegos detectamos una dificultad que limita la independencia que tienen las personas con muy baja visión o no videntes. Además del desgaste del relieve que se produce en los billetes con su uso,

[6] PROYECTO EXPERIMENTAL INTI: https://www.inti.gob.ar/noticiero/2015/noticiero465.htm

advertimos el hecho de que no todos tienen el sentido del tacto tan desarrollado como para poder identificarlos", comenta Leonardo Cruder, director adjunto del Centro de Tecnologías para la Salud y la Discapacidad del INT

Si bien existen aplicaciones para dispositivos móviles que identifican la denominación de dólares y euros, hasta el momento no existe en el mundo un sistema de reconocimiento de billetes que utilice software libre.

"Ya tenemos el prototipo desarrollado en distintas plataformas, tanto en PC como en celulares. Ahora resta hacer pruebas de laboratorio para testearlo en diferentes dispositivos y poner a punto el *software* para lograr una versión final de la aplicación. Eso nos va a permitir tener un análisis estadístico de su fiabilidad y tolerancia a fallas", explica Ignacio Moretti del Laboratorio de Desarrollo en Electrónica e Informática de INTI-Córdoba, quien no obstante señala que el prototipo ha demostrado por el momento un nivel de exactitud del 95% sobre una base de datos de prueba de imágenes.

La base de datos con la que funciona el programa está conformada por imágenes de frente y dorso de los distintos tipos de billetes que son comparados con la imagen que captura la cámara del dispositivo, ya sea un celular o computadora. A partir de esa imagen, el software realiza una búsqueda de puntos característicos de referencia y, si encuentra alguna coincidencia, emite la denominación por medios sonoros en pocos segundos.

Desarrollado en lenguaje C++ y Java (Android), el *software* se basa en la utilización de la librería para procesamiento de imágenes denominada *OpenCV.* "El método utilizado para el procesamiento consta de tres etapas bien definidas. La primera es la creación de un banco de datos con las imágenes de referencia que denominamos *templates* o modelos de billetes a los que se les asocia una máscara que permite enfocarse en los detalles de cada referencia. Luego se realiza un pre procesamiento de la imagen capturada: se reduce el ruido con diferentes filtros, se compensa la falta de luz mediante el incremento del contraste y se ajusta de modo automático el brillo. La tercera etapa, la más crítica, consiste en el reconocimiento, comenzando con el análisis individual de la imagen de referencia y de captura. Por último, el sistema reconoce alguno de los billetes de referencia en la imagen de captura, emite un sonido correspondiente a la

denominación detectada, y el proceso vuelve a comenzar con una nueva imagen de captura", precisa Moretti.

El desarrollo distingue billetes argentinos en distintas posiciones y escalas, total y parcialmente, y es ejecutable en PC, tablets y smartphones. Tiene un alto grado de complejidad por la gran variedad de situaciones ante las que se lo va a utilizar, como el desgaste, posiciones e iluminación del billete. Y va a ser fácilmente configurable para que funcione en otros países con otra denominación monetaria.

"El diseño de la interfaz de la aplicación fue pensado en base a las necesidades específicas de los usuarios, que en este caso serán en su mayoría no videntes. Por eso utilizamos un método de abordaje para el diseño más riguroso. Una vez desarrollado el prototipo final, cualquier persona va a poder utilizar la aplicación intuitivamente sin ninguna ayuda externa", agrega Nicolás Candiano de INTI- Tecnologías para la Salud y la Discapacidad.

Una vez lograda la versión final de la aplicación, va a ser liberada para que los usuarios puedan descargarla de manera gratuita, al igual que su código. "La idea es que la gente se apropie de esta tecnología y pueda modificar este mismo *software* para que sea utilizado en otras aplicaciones. Porque si se reemplazara la base de datos de billetes por otra, se podrían reconocer otro tipo de objetos" explica Moretti.

El técnico del Laboratorio de Desarrollo en Electrónica e Informática de INTI-Córdoba advierte que a futuro también se podrían hacer modificaciones al programa para que pueda detectar la falsedad de un billete agregándole una luz ultravioleta. "Se podría llegar a modificar el led del flash de la cámara del celular para que funcione como tal o agregar un dispositivo anexo para conectarlo", detalla.

"Trabajar la temática de la discapacidad nos permite repensar muchas prácticas individuales y sociales y a la vez la tecnología. En ese proceso, se van enriqueciendo diferentes instancias", expresa Cruder en referencia a que esa misma tecnología va a ser utilizada para otras aplicaciones.

4 – DESARROLLO DE LA PROPUESTA

4.1 – Metodología de trabajo

Metodología inductiva

Para desarrollar la aplicación de nuestro propósito, que reconozca billetes a través del uso de redes neuronales, partiremos de casos pequeños, aislados y controlados, hasta llegar a alcanzar nuestro caso de éxito con la conjunción y avance de nuestras pruebas.

Destacamos entonces, que nuestra aplicación se enfoca en dos grandes temas:

- Utilización de una red neuronal de Back-propagation
- Algoritmos de conversión de imágenes

Para ir abarcando ambos temas, comenzaremos trabajando en la formulación de estos eventos particulares donde se puedan evaluar los comportamientos de los distintos procedimientos a realizar, y diferir a partir de los resultados los caminos de acción a seguir.

4.2 - Implementaciones prácticas de redes neuronales de Back-propagation

Como es sabido, nuestro trabajo además de tener un procedimiento teórico formal, busca tener un alto impacto pragmático.

Es por este motivo que nos es de suma importancia, analizar primeramente qué herramientas existentes poder utilizar. Cuando hablamos de herramientas, nos referimos precisamente a algoritmos de redes neuronales ya desarrollados que pueden ser comprobados y evaluados.

Cabe destacar, que el desarrollo de un algoritmo de red neuronal de back-propagation es en sí una tarea compleja que bien podría ser motivo de un trabajo de investigación formal completo, pero esto no coincide con nuestro propósito particular, y es por eso que reutilizaremos algoritmos de código abierto en procura de nuestros objetivos.

Dicha esta mención, comenzaremos indicando los distintos sistemas evaluados que nos fueron nutriendo de conocimiento y experiencias para poder definir el

desenvolvimiento del desarrollo de nuestro proyecto. A partir de este trabajo, seleccionaremos el algoritmo del software que más se adapte a nuestras necesidades, estimando que nos otorgará el mayor beneficio posible. De estos algoritmos, esencialmente nos interesa obtener una red neuronal entrenada y un correcto posterior proceso de reconocimiento. Demás procesamientos se verán afectados por la parte de procesamiento de imágenes y demás cuestiones a desarrollar en el presente trabajo.

Por lo tanto, haremos un análisis de los sistemas encontrados focalizándonos en los siguientes puntos:

- Indicar cuál es el propósito general del sistema
- Definir el lenguaje utilizado y la plataforma de desarrollo
- Comentar su funcionamiento, señalando sus prestaciones así como sus limitaciones

4.3 - Caso de Estudio Nº 1: Leaves Recognition[7]

Propósito

LeavesRecognition (Leaves Recognition Project) es una aplicación basada en Java que utiliza una red neuronal para reconocer imágenes de hojas a través de una red de back-propagation previamente entrenada.

La intención de esta aplicación es darle al usuario la posibilidad de administrar una lista jerárquica de imágenes de hojas, donde puede realizar una clase de detección de bordes para identificar los puntos de entrada para cada imagen. Estos puntos son la base para que la red neuronal pueda reconocer una imagen desconocida e indicar a qué grupo de especie de hoja pertenece.

Desarrollo

Para realizar el mencionado propósito y de lograr que sea posible usar esta aplicación en múltiples sistemas operativos, el desarrollador eligió como lenguaje de programación a Java.

[7] Leaves Recognition Project : https://sourceforge.net/projects/lrecog/

Además, la facilidad de realizarlo en Java es que también sirve de base para realizar una aplicación Java Applet para directamente correr desde un navegador.

La única complicación que se encuentra en cuanto al desarrollo, es que el proyecto está realizado a través de Java Builder X, que no es una herramienta de programación convencional ni popular (como serían Eclipse, NetBeans, etc.).

Fundamentos teóricos

Detección de los bordes de las imágenes

Una de las tareas principales de esta aplicación es la detección de los puntos, conocidos como tokens, en las imágenes de las hojas. Se asume que la imagen a utilizar consta de una única hoja completa, y para lo cual se utilizará el algoritmo de detección conocido como "Prewitt Edge".

El algoritmo "Prewitt Edge"[8] produce una imagen donde los altos niveles de grises indican la presencia de un borde entre dos objetos. El filtro de este algoritmo calcula la raíz cuadrada de una matriz de 3x3. Por lo tanto, utiliza esta matriz para calcular el valor del gradiante:

```
-1  0  1      1  1  1

-1  0  1      0  0  0

-1  0  1     -1 -1 -1

   X           Y
```

Luego si consideramos la siguiente matriz 3x3 de imagen:

```
+------------+

| a1  a2  a3 |

| a4  a5  a6 |

| a7  a8  a9 |
```

[8] Detección de bordes en una imagen : http://www4.ujaen.es/~satorres/practicas/practica3_vc.pdf

+------------+

Donde:

- a1 .. a9 - son los valores de cada nivel de gris de cada pixel
- X = -1*a1 + 1*a3 - 1*a4 + 1*a6 - 1*a7 + 1*a9
- Y = 1*a1 + 1*a2 + 1*a3 - 1*a7 - 1*a8 - 1*a9
- Gradiante de Prewitt = SQRT(X*X + Y*Y)

Tratamiento

Como se indicó anteriormente, la idea para identificar la especie de una imagen específica es que el marco exterior de la hoja sea suficiente para indicar la especie a la que pertenece. Para realizar eso, es necesario dibujar correctamente este marco. El algoritmo Prewitt Edge solamente encuentra los bordes con un umbral pre configurado y luego de esta detección, se tiene que realizar un algoritmo de reducción para minimizar este umbral basado en bordes a una línea de marco donde se pueda aplicar el algoritmo de reconocimiento posterior.

El algoritmo que utiliza procesa la imagen recursivamente y minimiza las líneas encontradas a un pixel de ancho.

Token de las hojas

La parte central de esta aplicación la componen los tokens que son encontrados luego del procesamiento de cada imagen.

La idea detrás de la transformación del marco de la hoja a una red neuronal aplicable, se concibe a través de los ángulos cosenos y senos de este marco o contorno, representando los criterios para un patrón de reconocimiento.

La siguiente imagen (ver figura 10) muestra una parte de una hoja que ya fue procesada a través de los mecanismos de detección de bordes y de refinamiento.

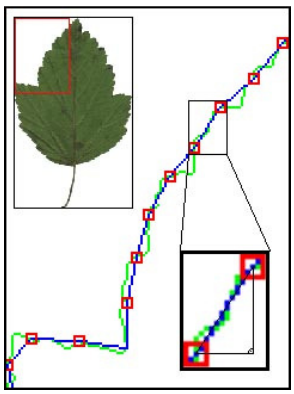


Figura 10 – Imagen tomada del software Leaves Recognition

Las distintas indicaciones en la imagen reflejan lo siguiente:

- Línea verde: la forma de la hoja luego de una correcta detección de bordes y refinamiento
- Cuadrados rojos: Representan un punto del contorno de la hoja desde donde se dibujará una línea al próximo cuadrado.
- Línea azul: la combinación del centro de los dos cuadrados desde donde se harán los cálculos de los ángulos coseno y seno. Esta línea es la representación de un token.

Si se mira con detenimiento en la imagen agrandada del triángulo que se forma entre dos cuadrados rojos, se notará que se forma un triángulo rectángulo. Este y el conjunto de todos los triángulos de la imagen son la representación de los token desde los cuales se realizarán los cálculos en la red neuronal.

La siguiente figura (ver figura 11) ilustra un simple token de una imagen. Aquí se ve claramente cómo los ángulos A y B son las dos partes necesarias que ingresarán a las capas de las redes neuronales.

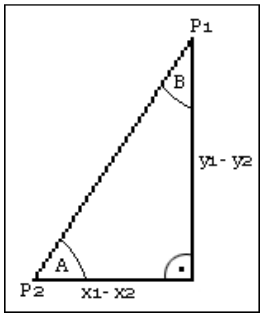


Figura 11 – Token de Imagen - Wikipedia

Con estos dos ángulos se puede representar exactamente la dirección de la hipotenusa entre el punto P1 y P2 lo cual es absolutamente necesario para la representación de la hoja.

Red Neuronal

La otra parte importante de este trabajo es la integración de la red neuronal de back-propagation. Como se describió anteriormente, las entradas a esta red neuronal son los token individuales para una hoja, y como un token normalmente consiste de sus ángulos coseno y seno, la cantidad de neuronas de entrada para esta red será la cantidad de tokens multiplicados por dos (2).

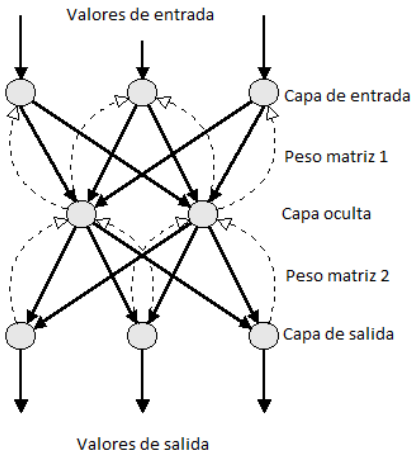


Figura 12 – Red neuronal de 2 capas -Wikipedia

La figura 12 resume el procesamiento de la red neuronal en esta aplicación.

Para llenar las neuronas de entrada de la red, se usan los tokens calculados anteriormente. La cantidad de neuronas de salida es normalmente especificada por la cantidad de diferentes especies ya que codificamos sus formas para especificar las salidas.

Utilización

Utilizaremos esta aplicación primeramente para los fines para los cuales fue desarrollada para así inicialmente evaluar verdaderamente su eficacia y prestaciones, para luego intentar aplicarlas a nuestro propósito (reconocimiento de billetes) y ver su factibilidad al respecto.

Procesamiento de la imagen

El primer paso de esta aplicación es realizar el procesamiento de las imágenes, como ellos lo indican encontrar los tokens. Para eso una vez iniciada la aplicación, se procede a ir agregando distintas especies de hojas (conjuntos) y su particular hoja perteneciente a esa especie.

Entonces iniciamos el programa y agregamos cuatro especies distintas de hojas, cada una con un solo ejemplar que le pertenezca:

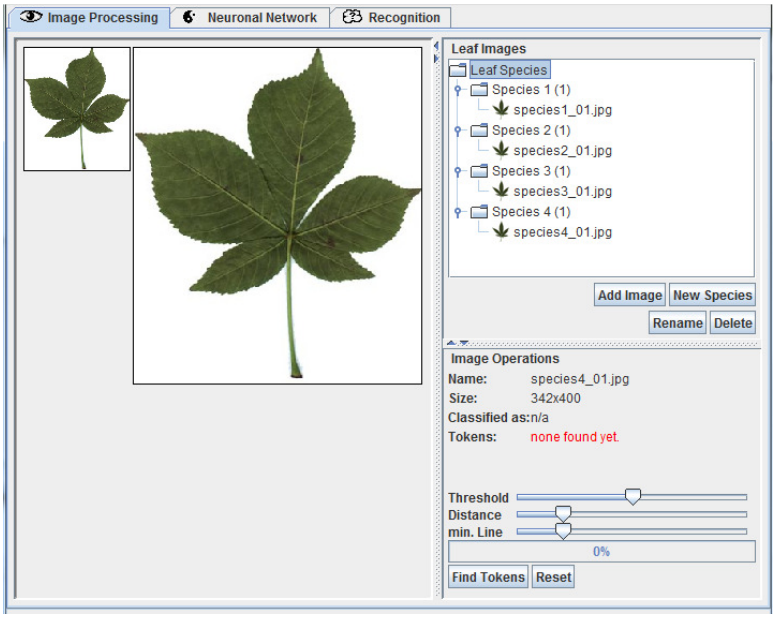


Figura 13 – Procesador de imagen, Software Leaves Recognition

Figura 14 – 4 especies de hojas, Software Leaves Recognition

Una vez agregadas las imágenes, debe producirse el algoritmo de reconocimiento de los tokens. Entonces por cada imagen se aprieta el botón "Find tokens" el cual arrojará un valor para cada hoja donde ahora dice "Tokens: none found yet".

Tokens encontrados

Especie 1 - 62 tokens

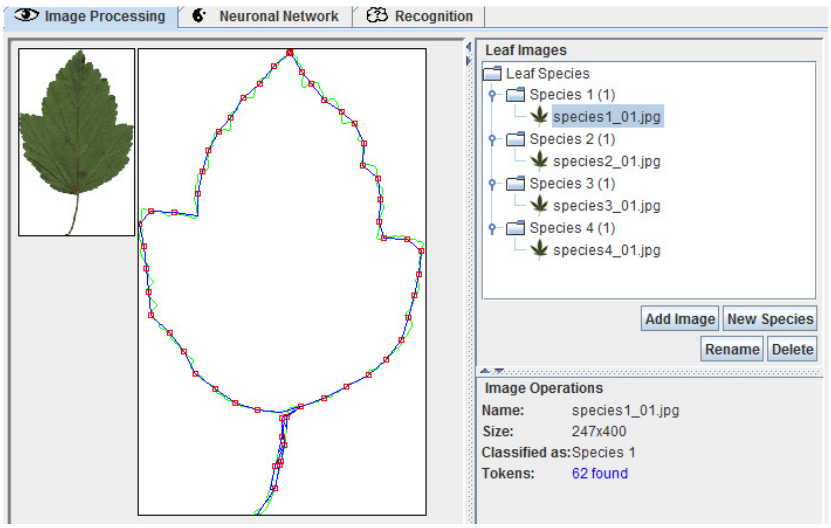


Figura 15 – Tokens encontrados en "Especie 1" - Software Leaves Recognition

Especie 2 - 96 tokens

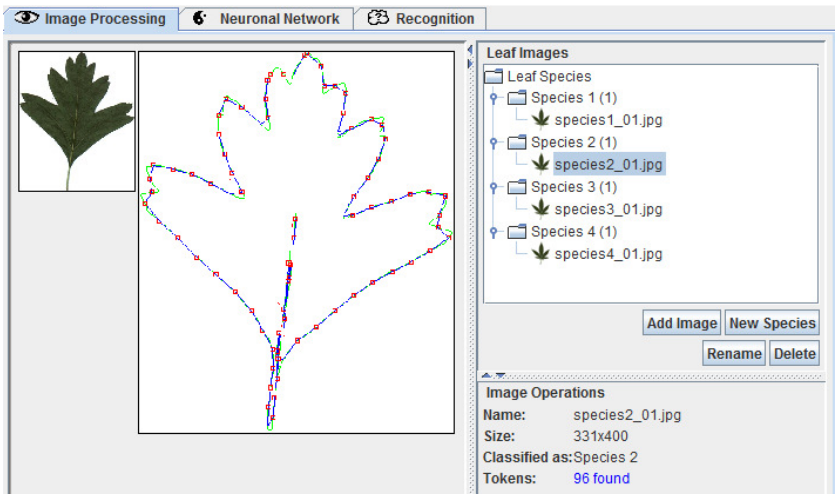


Figura 16 – Tokens encontrados en "Especie 2" - Software Leaves Recognition

Especie 3 - 48 tokens

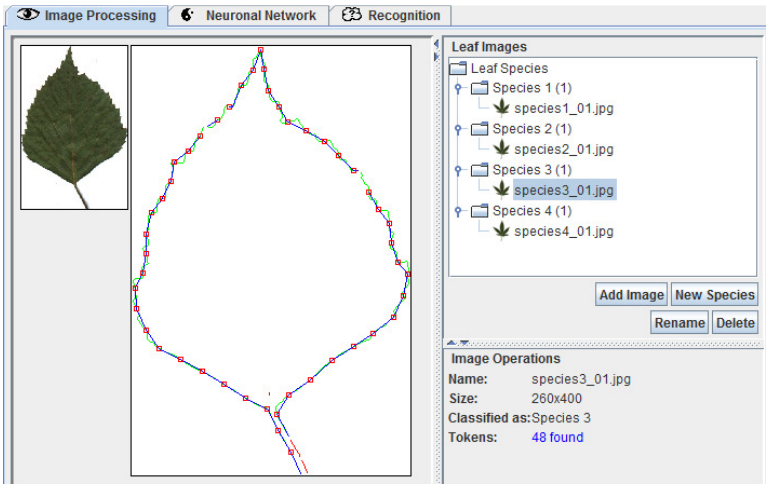


Figura 17 – Tokens encontrados en "Especie 3" - Software Leaves Recognition

Especie 4 - 121 tokens

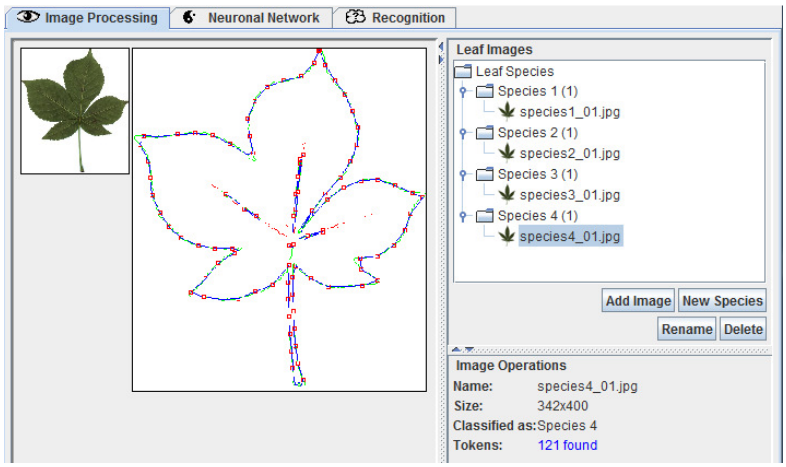


Figura 18 – Tokens encontrados en "Especie 4" - Software Leaves Recognition

Obtenidos los token de cada imagen, el paso siguiente es realizar el entrenamiento de la red neuronal.

Entrenamiento de la red neuronal

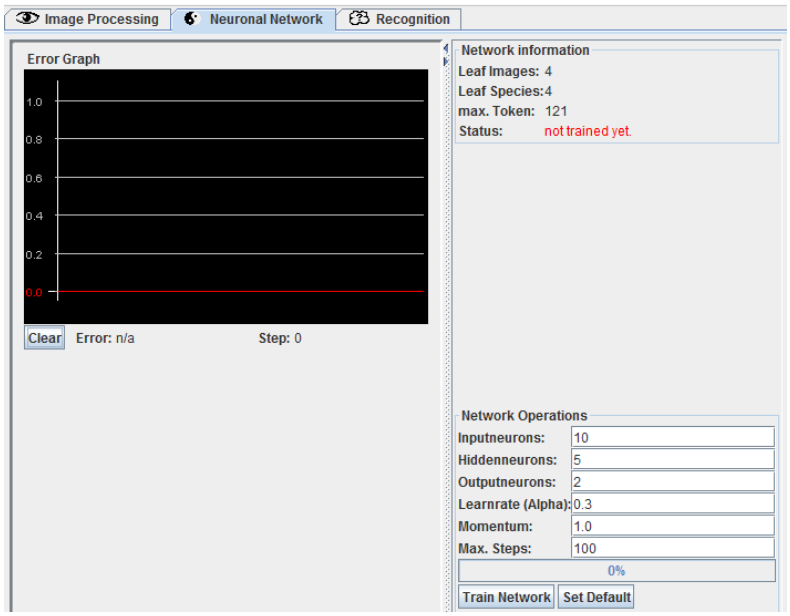


Figura 19 – Entrenamiento de la red neuronal en Software Leave Recognition.

En la figura 19 podemos ver la pestaña nos permite realizar el entrenamiento de la red, y sus particulares configuraciones. Es decir, aquí indicaremos cuántas neuronas de entrada utilizaremos, cuántas neuronas ocultas utilizaremos, cuántas salidas esperamos, el factor de aprendizaje y la cantidad máxima de pasos para ir reduciendo el error. No tenemos un parámetro que sea justamente indicar hasta qué valor umbral de error queremos que se ejecute el entrenamiento de la red.

Ahora bien, resta configurar este entrenamiento acorde a las imágenes ingresadas y sus tokens obtenidos. Para ello el programa nos facilita el botón "Set default", que toma los valores que se muestran arriba, como ser "Max. token 121" para indicarnos los valores por defecto con que puede funcionar esta aplicación. Seleccionamos esos, y realizamos el entrenamiento de la red.

El proceso es muy rápido, en cuestión de una decena de segundos ya está realizado el entrenamiento y nos arroja el siguiente umbral de error.

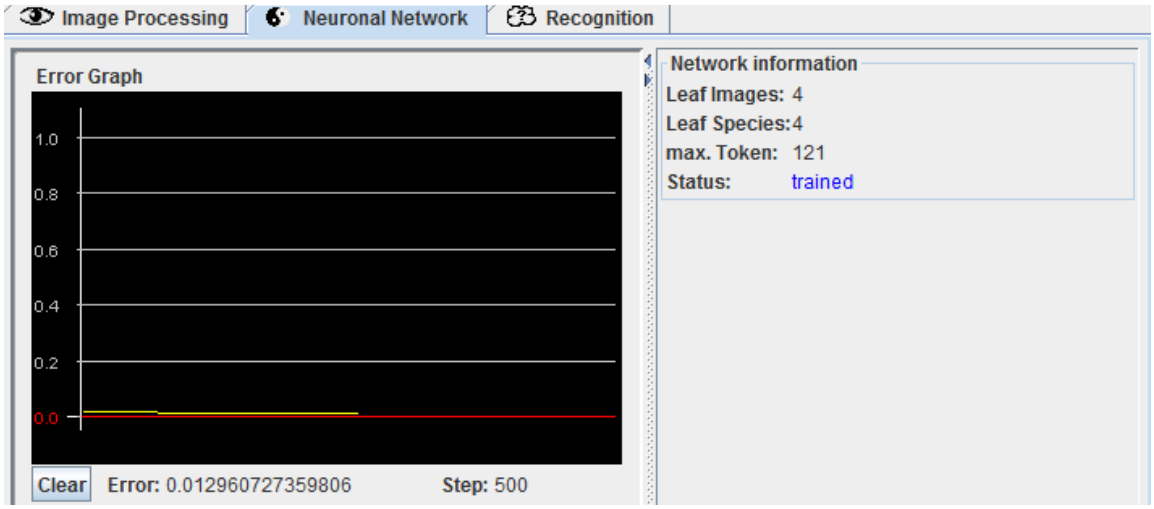


Figura 20 – Entrenamiento de la red neuronal en Software Leave Recognition.

Para 500 pasos, nos indica que el error asciende a relación de 0.013 lo cual es un número mensurable y que puede producir reconocimientos erróneos.

Reconocimiento

Una vez entrenada la red, es cuestión de realizar el reconocimiento con otra imagen para probar el funcionamiento de esta red neuronal entrenada. Para ello

primeramente realizamos pruebas con las mismas imágenes con las cuales fueron entrenadas y otras imágenes distintas pero pertenecientes a la especie. Los resultados se muestra en la Tabla 2:

Tabla 2 – Reconocimiento utilizando las imágenes con las que fue entrenada la red neuronal.

1º Prueba - Reconocimiento con la misma imagen entrenada		
Especie	**Resultado**	**Observaciones**
Especie 1 	**Recognition Info** **Image:** species1_01.jpg **Token:** 62 **Status:** recognized **Recognition Results** Species 1 — 97.524 % Species 2 — 75.095 % Species 3 — 51.545 % Species 4 — 27.322 %	Correcto reconocimiento, falta mayor precisión en la definición y mayor diferenciación contra otras especies
Especie 2 	**Recognition Info** **Image:** species2_01.jpg **Token:** 96 **Status:** recognized **Recognition Results** Species 1 — 74.482 % Species 2 — 97.642 % Species 3 — 74.306 % Species 4 — 50.367 %	Correcto reconocimiento, falta mayor precisión en la definición y mayor diferenciación contra otras especies
Especie 3 	**Recognition Info** **Image:** species3_01.jpg **Token:** 48 **Status:** recognized **Recognition Results** Species 1 — 50.817 % Species 2 — 74.018 % Species 3 — 96.860 % Species 4 — 73.206 %	Correcto reconocimiento, falta mayor precisión en la definición y mayor diferenciación contra otras especies

Especie 4 	**Recognition Info** **Image:** species4_01.jpg **Token:** 121 **Status:** recognized **Recognition Results** Species 1 — 26.306 % Species 2 — 50.924 % Species 3 — 74.587 % Species 4 — 98.001 %	Correcto reconocimiento, falta mayor precisión en la definición y mayor diferenciación contra otras especies

Como resumen podemos notar que ante la misma imagen, obtenemos un certero reconocimiento de la especie a la que pertenece la hoja, aunque podría mejorarse el porcentaje de error para así también bajar el porcentaje de las que no son de la especie.

Luego se realizaron pruebas con imágenes distintas a las entrenadas:

Esta prueba, que es la que estamos interesados en hacer ya que nuestro propósito no es reconocer imágenes iguales a las que ya fue entrenada la red sino justamente distintas imágenes, totalmente nuevas pero que puedan ser reconocibles a través de los patrones establecidos, no nos arroja resultados favorables.

Es necesario realizar ajustes para mejorar la precisión del algoritmo de reconocimiento de los tokens como así también mejorar el entrenamiento de la red para reducir el umbral de error. Por lo tanto, hay varias acciones por tomar:

- Mejorar el entrenamiento de la red con distintos parámetros
- Modificar el número de tokens reconocidos por el algoritmo
- Agregar más individuos a la especie

Para el continuado de las pruebas, realizaremos un mejor entrenamiento de la red neuronal, para así reducir el error absoluto y ver qué resultados da al reconocimiento final. Procedemos entonces a realizar el mismo entrenamiento, pero 100 veces lo que realizamos inicialmente, es decir unos 50000 pasos hasta terminar de entrenar la red.

Este entrenamiento, sí tuvo aparejado tiempos de cálculo que rondaron los 8 minutos de reloj, algo descartable ya que el sentido de la red neuronal es ser entrenada una sola vez y luego verse aplicada por lo que no nos conlleva a un problema.

Como vemos, el margen de error es inmensamente mejor a nuestra red anterior a razón de 1.025 E-4.

A simple vista vemos que ante una red mayormente entrenada algunos resultados fueron muy positivos y cercanos a lo buscado pero otros tomaron otro comportamiento variante. Se puede inferir que ante mayor sea el entrenamiento, más se reducirá el error general para cada especie. Y en pos de reducir este error, es que continuamos realizando pruebas como planeamos, por lo tanto indicaremos distintas cantidades de tokens para cada imagen para así ver cómo se comporta y qué resultados obtenemos.

Dicho esto, realizaremos dos pruebas:

- Con mayor cantidad de token: donde se encontrarán más líneas internas a la hoja, que no forman parte del relieve
- Con menor cantidad de token: donde se encontrará exclusivamente el contorno de la hoja

Mayor cantidad de token:

Para realizar esto, disminuimos el umbral para el reconocimiento de las hojas y realizamos nuevamente la búsqueda de los token para nuestras 4 especies utilizadas. Para un umbral reducido estos son los valores obtenidos:

Especie 1 - 507 tokens

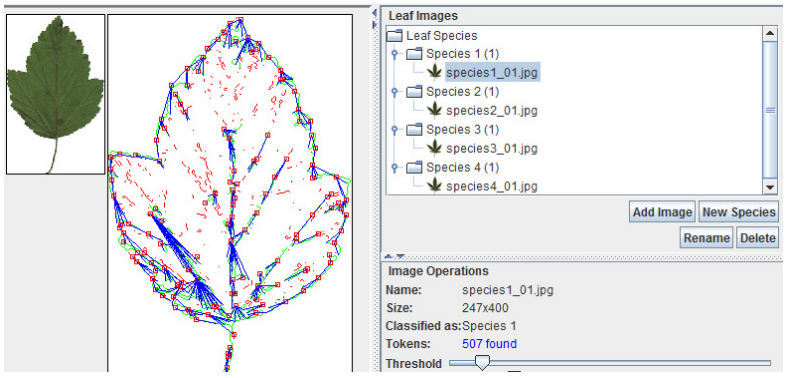


Figura 21 – Tokens encontrados en "Especie 1" - Software Leaves Recognition

Especie 2 - 731 tokens

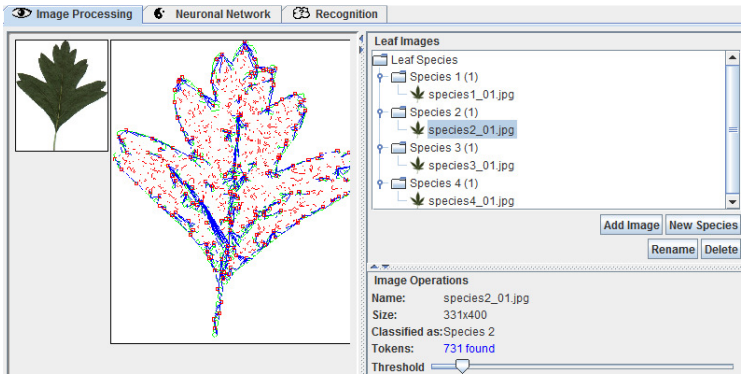


Figura 22 – Tokens encontrados en "Especie 2" - Software Leaves Recognition

Especie 3 - 467 tokens

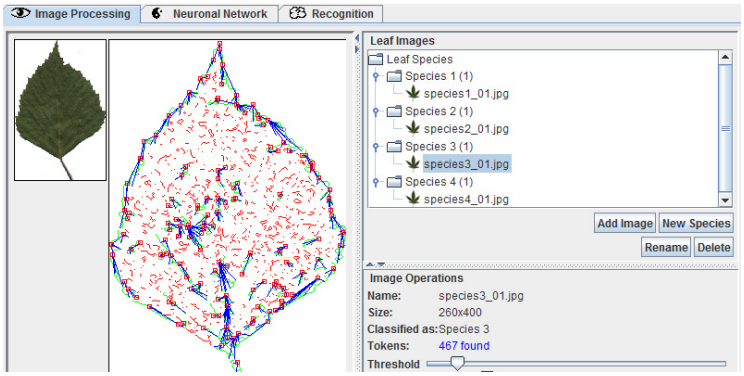


Figura 23 – Tokens encontrados en "Especie 3" - Software Leaves Recognition

Especie 4 - 1020 tokens

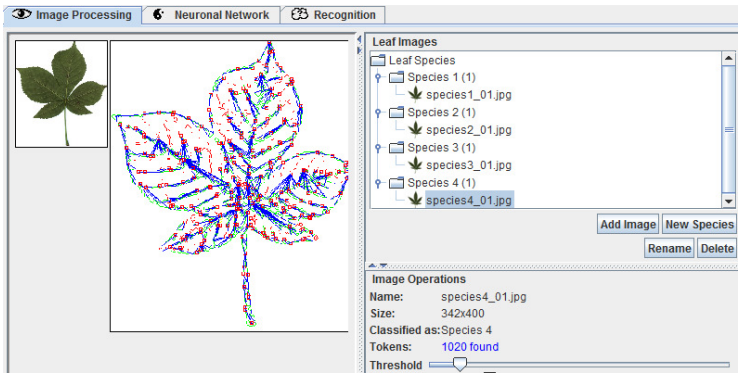


Figura 24 – Tokens encontrados en "Especie 4" - Software Leaves Recognition

Como vemos la cantidad de tokens ha aumentado a razón de 10 veces por lo que las entradas a la red neuronal que nuevamente debemos entrenar también aumentarán de tamaño. Procedemos a realizar el entrenamiento para posterior evaluación de resultados. Este entrenamiento nos arrojó un valor de error de 2.08 E-4, mayor al anterior debido a que la cantidad de neuronas aumentó y por lo tanto es factible que requiera de mayor número de iteraciones para mejorar el umbral de reconocimiento.

4º Prueba - Reconocimiento con distinta imagen a la entrenada (50000 pasos y más tokens)

Como resultado de esta nueva prueba, podemos ver y empezar a demostrar el propósito de esta aplicación. Al reducir el valor del umbral en el procesamiento de la imagen, hemos traído demasiados tokens que pertenecían a lo que sería el interior de la hoja, así como sus comisuras y tallo. Esto nos trajo problemas en el reconocimiento ya que esta aplicación está orientada a descubrir las especies de las hojas a través de su contorno. Cuando le agregamos detalles de su interior, estos dependen de la calidad de la imagen, del deterioro de la hoja y dejando de lado su verdadera forma, por lo tanto hemos obtenido esos resultados completamente errados en algunos casos ya que la hoja en su interior era bastante diferente a la entrenada, pero manteniendo la misma forma exterior.

Queda ahora entonces aumentar este umbral para que tome aún más los contornos y evaluar si produce mejores resultados a las pruebas con valores por defecto.

Mayor cantidad de token - Mayor umbral

Realizamos la obtención de los token para cada imagen, con el mayor umbral posible, por el cual obtendremos exclusivamente el contorno de cada una de las hojas. Vemos que la cantidad de tokens es similar a la obtenida con los valores por defecto, y que además nos muestra solamente el contorno de cada hoja en su procesamiento.

Primer red neuronal

Dada la necesidad de programar una red neuronal, es que definiremos, entrenaremos y luego finalmente utilizaremos.

Para el primer desarrollo de pruebas con redes neurales, seremos acotados y específicos: buscaremos comprobar el funcionamiento del algoritmo y sus bondades

4.4 Caso de estudio nº 2: BackPropagation Network for Image Recognition (BP Simplified Demo)[9]

Propósito

El siguiente software [CODEPROJECT] a evaluar está realizado bajo Visual C++, con la intencionalidad de a partir de un entrenamiento de una red neuronal bajo ciertas parametrizaciones, proponer o dibujar una figura y equipararla con las que fue entrenado.

Desarrollo

En procura de realizar esto, el software funciona de la siguiente manera: en primer medida, se debe colocar en una carpeta aquellas imágenes que servirán de patrones para entrenar nuestra red neuronal. Ahí mismo el software base propone imágenes con las letras del abecedario más los números del 0 al 9, pero también queda abierto a la utilización de las imágenes que uno quisiera, punto que nos interesa y motiva.

Además de la selección de imágenes, se permite realizar una configuración de a qué nivel de capas deseamos trabajar (1 a 3), proponiendo un umbral máximo de error y sus salidas. Una parametrización interesante de este aplicativo, es que a mayor nivel de capas, nos permite seleccionar la cantidad de puntos de entrada de trabajo para experimentación.

Utilización

Comenzaremos comprobando el funcionamiento del aplicativo tal cual es provisto por sus desarrolladores para ir conociendo sus bondades y deficiencias. Para ello ejecutamos el programa y visualizamos la pantalla que se ilustra en la figura 25:

[9] BackPropagation Network for Image Recognition (BP Simplified Demo): https://www.codeproject.com/Articles/19323/Image-Recognition-with-Neural-Networks

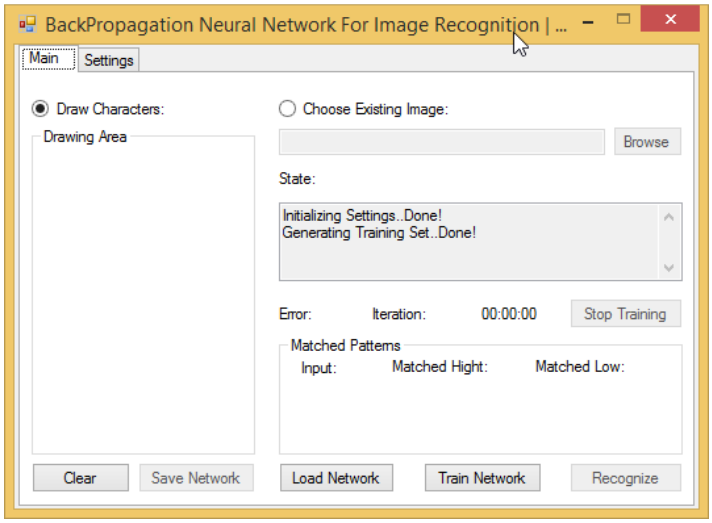


Figura 25 – Pantalla Principal – Software BackPropagation Neural Network for Image Recognition

En esta primera pantalla (figura 25) podemos ver que tenemos dos solapas, una que dice ser la principal que es la que estamos viendo, y otra que tiene opciones donde pueden realizarse las parametrizaciones que fuimos adelantando.

Luego podemos ver que está seleccionado el botón tipo radio que dice “Draw Characters:” y debajo aparece una “Drawing Area” que es el área de dibujo que podemos utilizar una vez que tengamos entrenada nuestra red para realizar el proceso de reconocimiento. La otra opción que aparece sin seleccionar es la de utilizar una imagen existente, por lo tanto el sistema nos permite proceder de las dos formas, ya sea dibujando manualmente o realizar comparativas contra imágenes.

En el centro a la derecha, aparece un cuadro de estado (“State”) que indicará los distintos procesamientos que se están realizando y su finalización. Debajo del mismo algunas consideraciones importantes que serán vistas cuando realicemos el entrenamiento que son: el margen de error alcanzado, la cantidad de iteraciones realizadas, y el tiempo neto insumido.

Siguiendo hacia abajo dentro del cuadro de “Matched Patterns” visualizaremos los resultados que nos arroja el programa al realizar su proceso de reconocimiento. Nos indicará la imagen más y menos probable, junto a su porcentaje de éxito.

Por último, los botones inferiores realizan las siguientes acciones: “Clear” sirve para borrar algo que hayamos dibujado, “Save network” nos permite guardar una red entrenada para futuras utilizaciones, “Load network” es el botón para cargar redes guardads con anterioridad, “Train network” comenzará el proceso de entrenamiento con las opciones actuales, y “Recognize” hará el proceso de reconocimiento.

Explicada la pantalla principal del aplicativo, podemos continuar viendo la configuración que nos permite modificar y con la cual realizaremos entrenamientos experimentales (ver figura 26).

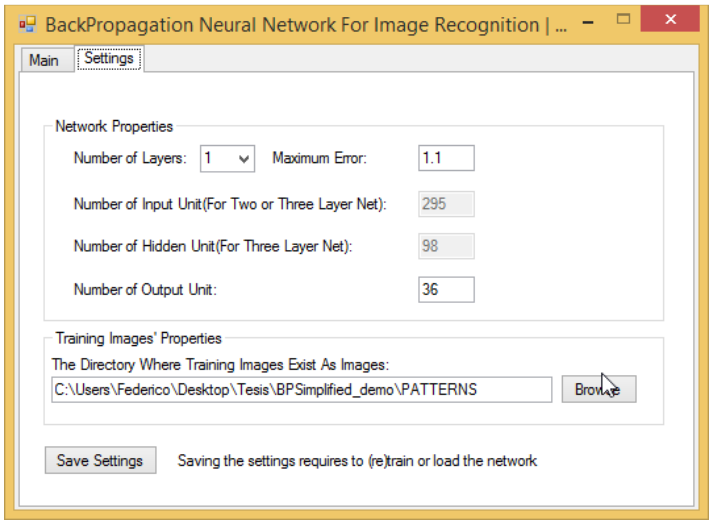


Figura 26 – Pantalla Ajustes – Software BackPropagation Neural Network for Image Recognition

Como se puede vislumbrar en la figura 26, en el centro de la pantalla comienzan las propiedades de la red neuronal. En primer medida podremos configurar a qué número de capas trabajar (de 1 a 3) en el campo “Number of Layers” y seguido a éste el error máximo permitido. Debajo, si elegimos mayor nivel de capas podremos elegir la cantidad de puntos de entrada y de unidades ocultas inclusive en las que queremos que nuestra red neuronal se propague. Finalmente el campo de “Output Unit” nos permite seleccionar cuántos resultados posibles tendremos, que para el caso son 36 compuestos por 26 letras del abecedario y 10 números.

Finalmente tenemos una caja de texto para buscar nuestra carpeta donde estarán contenidas nuestras imágenes para entrenar nuestra red, y un botón para guardar la configuración de esta pantalla, independientemente de si está o no entrenada la red.

A continuación, en la figura 27, un ejemplo de las imágenes con las que entrenaremos la red neuronal:

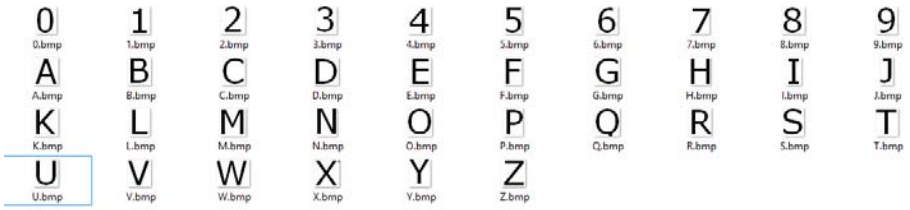


Figura 27 – Imágenes para entrenamiento de red neuronal - PAINT

Con la información indicada estamos en condiciones de proceder a realizar el primer entrenamiento de la red y empezar a conocer sus funcionalidades. Por lo tanto, como primer corrida utilizaremos los valores propuestos por defecto para luego ir probando nuevos escenarios.

Entonces, realizamos el primero entrenamiento de la red a valores por defecto y obtenemos el siguiente resultado en cuanto a error e iteraciones (ver figura 28):

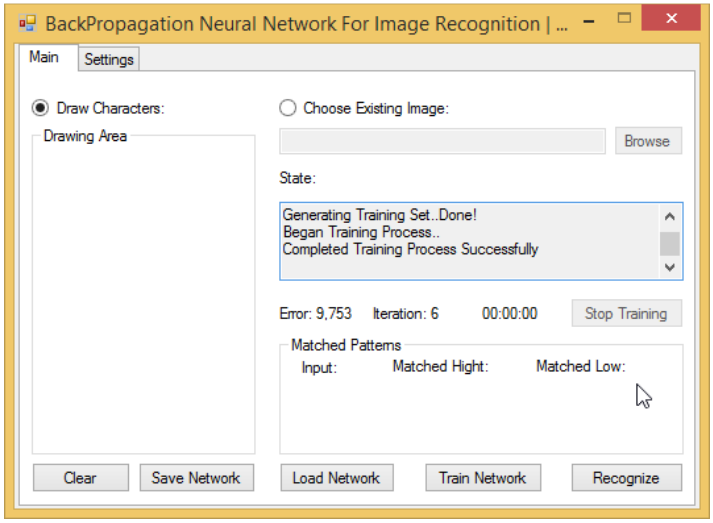


Figura 28 – Entrenando Red Neuronal con imágenes de "Figura 27" – Software BackPropagation Neural Network for Image Recognition.

Al parecer el entrenamiento fue demasiado rápido con muy pocas iteraciones ya que tenemos un valor de error muy grande (1.1). Ahora debemos ver cómo se comporta al reconocer imágenes y qué grado de acierto tiene. Empezaremos utilizando la opción de cargar imágenes existentes, y luego dibujaremos a mano para mayor complejidad.

Cargamos la letra B de las imágenes existentes y ejecutamos el botón “Recognize” para que tenga lugar el procesamiento de reconocimiento (ver figura 29).

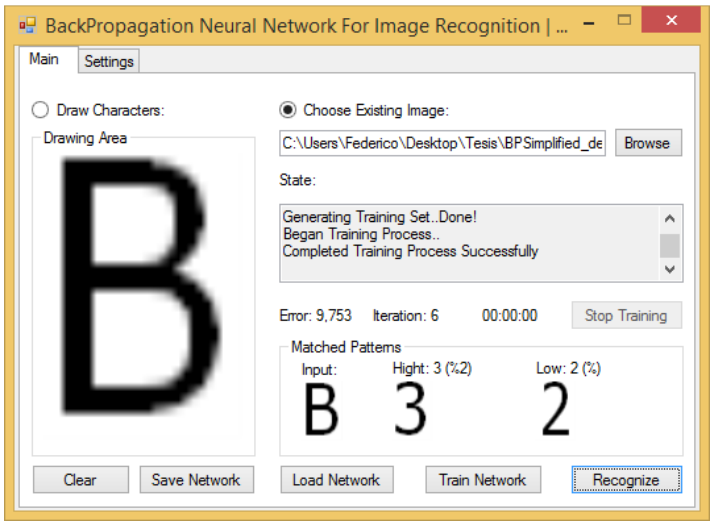


Figura 29 – Carga de imagen y obtención de resultados – Software BackPropagation Neural Network for Image Recognition.

El resultado es no satisfactorio. A misma imagen de entrada, los valores propuestos como resultado son incorrectos. El número 3 aparece con mayor probabilidad aunque casi inexistente, por lo cual este resultado debería descartarse. A continuación pasaremos a figuras más simples para comprobar si el proceso mejora. Los resultados se muestran en la Tabla 3.

Tabla 3 – Resultados de entrenamiento

LETRA	PATRONACIERTO HIGH - PORCENTAJE (%)	PATRON ACIERTO LOW - PORCENTAJE (%)	COMENTARIOS
1	1 – 86%	0 – (2%)	La herramienta trabajó mejor al ser una figura más simple y menos parecida a otras por las cual fue entrenada
X	X – (86%)	K – (6%)	La letra X también tuvo gran porcentaje de éxito, apareciendo la letra K visiblemente similar como segunda candidata en un porcentaje muy bajo

Con estos podemos empezar a delinear una línea de pensamiento que nos apunta a que la red neuronal necesita de mayor entrenamiento para discernir entre figuras más completas y semejantes a otras, pero que con este entrenamiento rápido se pudo identificar figuras que son más disímiles.

Realizaremos el mismo procedimiento ahora, pero dibujando manualmente las 3 figuras para comprobar si tenemos resultados parecidos. Los mismos se encuentran referenciados en la tabla 4.

Tabla 4 – Resultados de entrenamiento

LETRA	PATRONACIERTO HIGH - PORCENTAJE (%)	PATRON ACIERTO LOW - PORCENTAJE (%)	COMENTARIOS
B	P – 12%	F – (6%)	
1	1 – (50%)	K – (0%)	
X	X – (28%)	2 – (20%)	

Los resultados alcanzados con la escritura a mano, nos dan cuenta de que hemos tenido los mismos desenlaces, con diferencia en el grado de éxito. Inclusive, para el caso de la letra B hemos tenido un mejor comportamiento ya que la letra resultante fue la P que comparte claramente su composición con la objetivo. Para los otros dos casos, el porcentaje de acierto no fue tan contundente pero sí efectivo. Esto puede deberse claramente a las deficiencias de la escritura a mano en el trazo irregular y dibujo de las formas.

Ahora pasaremos a una prueba con un entrenamiento mayor, a un nivel de error de al menos 0,1 para evidenciar si los resultados mejoran (ver figura 30).

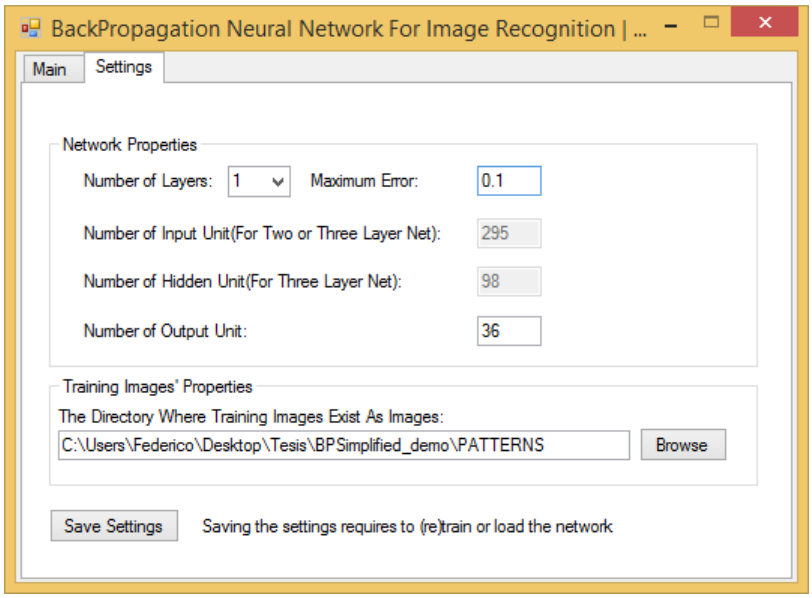


Figura 30 – Setteo de la red neuronal– Software BackPropagation Neural Network for Image Recognition.

Este nuevo entrenamiento arroja una cantidad numerosa de iteraciones (284 iteraciones) y un procesamiento que llevó al menos un corto tiempo (5 segundos) de realizar. Utilizaremos el lote de pruebas presentado anteriormente para ver los nuevos resultados, los que se resumen en la siguiente Tabla 5.

Tabla 5 – Resultados de entrenamiento

LETRA	TIPO	PATRONACIERTO HIGH - PORCENTAJE (%)	PATRON ACIERTO LOW - PORCENTAJE (%)	COMENTARIOS
B	IMAGEN	3 – 1%	0 – (0%)	
1	IMAGEN	1 – (97%)	0 – (0%)	
X	IMAGEN	X – (96%)	k– (2%)	

Los resultados para el procesamiento por imágenes de archivo, se presentan similares a los anteriormente presentados con un margen de error de 1.1. Se tienen sí mayor grado de acierto en los casos del número 1 y la letra X, pero se siguen cayendo en fallos para la letra B.

El procesamiento manual que se visualiza en las figuras 31, 32 y 33 también derivó en resultados similares a la primer prueba, también mejorando su porcentaje de acierto y agrandando la brecha entre sus alternativas.

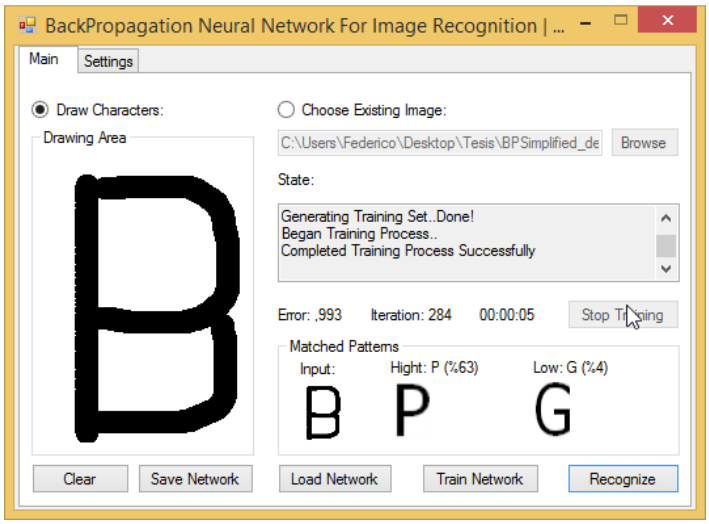


Figura 31 – Entrenamiento con carga manual, letra "B" – Software BackPropagation Neural Network for Image Recognition.

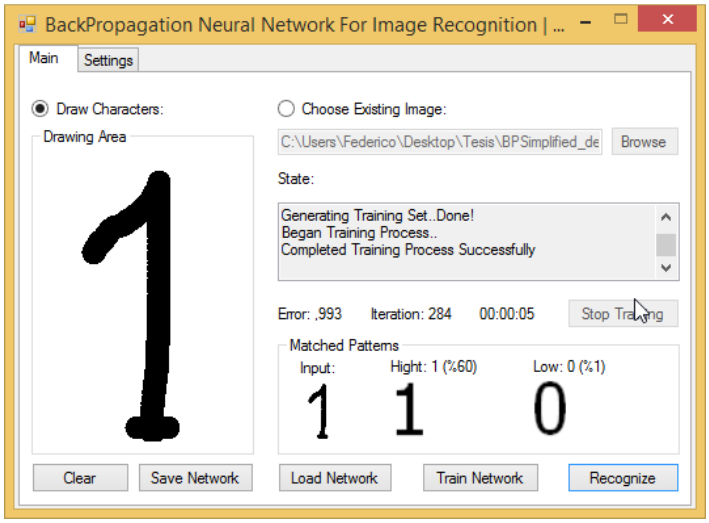


Figura 32 – Entrenamiento con carga manual, número "1" – Software BackPropagation Neural Network for Image Recognition.

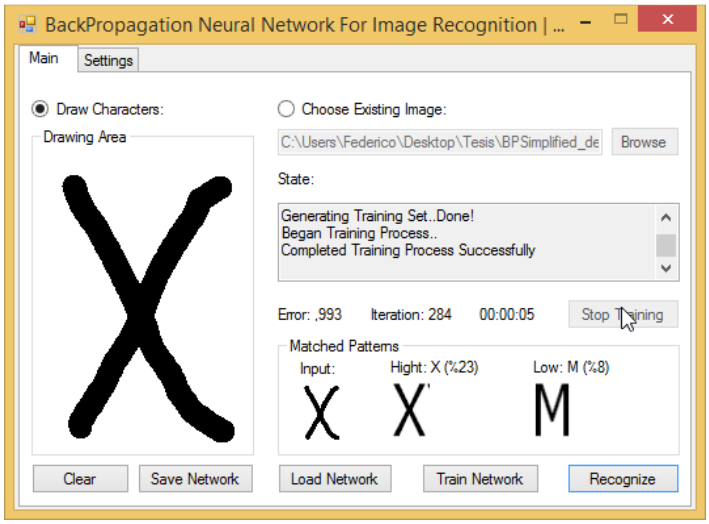


Figura 33 – Entrenamiento con carga manual, letra "X" – Software BackPropagation Neural Network for Image Recognition.

Sin embargo, podemos conferir que no se ha alcanzado el resultado esperado todavía lo que nos lleva a exigir aún más el valor del error, antes de modificar el nivel de capas de la red neuronal. En esta tercer prueba descenderemos un cero a un valor de error de 0,01.

Se alcanzó ese nivel de error en aproximadamente 5000 iteraciones continuando trabajando sin capas intermedias. A continuación los resultados de nuestra batería de pruebas se indican en la Tabla 6.

Tabla 6 – Resultados de entrenamiento utilizando imágenes y escritura manual

LETRA	TIPO	PATRONACIERTO HIGH - PORCENTAJE (%)	PATRON ACIERTO LOW - PORCENTAJE (%)	COMENTARIOS
B	IMAGEN	B – 75%	3 – (3%)	
1	IMAGEN	1 – (99%)	0 – (0%)	
X	IMAGEN	X – (99%)	K– (0%)	
B	ESCRITURA MANUAL	R – 50%	M – (12%)	
1	ESCRITURA MANUAL	1 – (79%)	0 – (1%)	
X	ESCRITURA MANUAL	X – (97%)	V– (1%)	

El procesamiento por imágenes de archivo finalmente dio satisfactorio en todos los casos de prueba. Por primera vez pudimos obtener correctamente la letra B y en un alto porcentaje de 75%. Los otros dos casos superaron su porcentaje alcanzando un 99% el cual se nos presenta inmejorable. Realmente la mayor cantidad de iteraciones realizó un mejor entrenamiento de la red a valores que estábamos buscando.

La carga manual sigue presentándose con los problemas antes vistos, lo que nos augura complicaciones cuando las imágenes no aparezcan tal cual fueron entrenadas (cuestión que nos ocurrirá en nuestro trabajo pensando en las imágenes de los billetes). Se obtuvieron buenos porcentajes para los casos de éxito ya conocidos.

Dada esta mejoría haremos una prueba con este nivel de error a dos capas para conocer si la utilización de una capa más influye o no en el proceso de reconocimiento.

También se analizarán los tiempos e iteraciones necesarias para alcanzar el valor de error buscado, si estos se vieron afectados ya sea en aumento o decrimento.

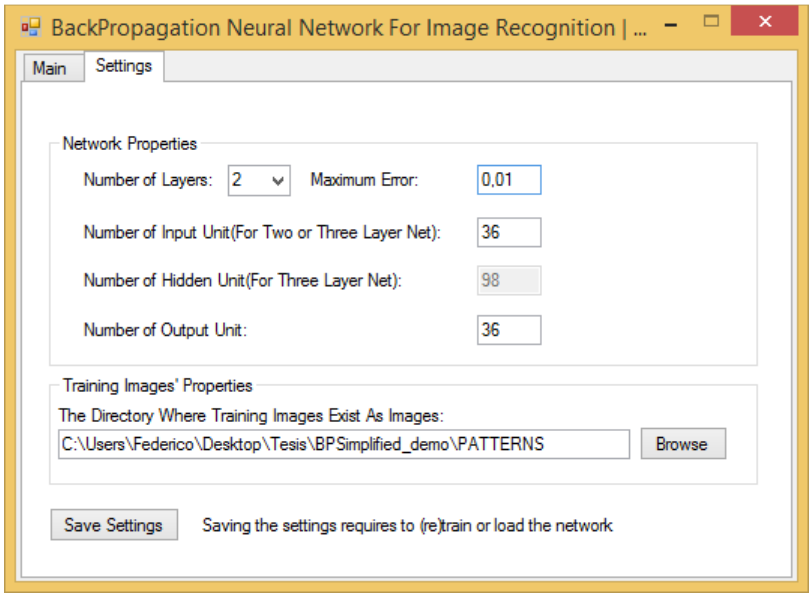


Figura 34 – Setteo de entrenamiento con 2 capas – Software BackPropagation Neural Network for Image Recognition.

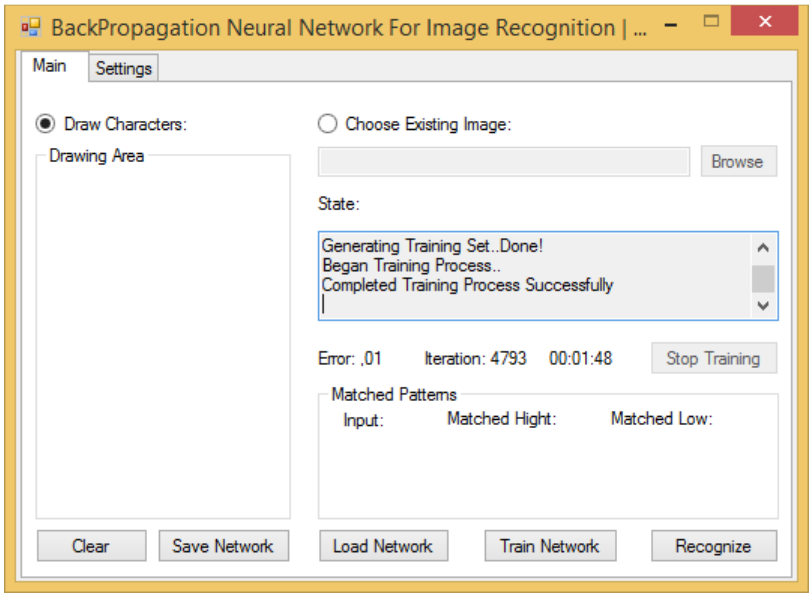


Figura 35 – Setteo de entrenamiento con 2 capas – Software BackPropagation Neural Network for Image Recognition.

Estas imágenes (figuras 34 y 35) nos arrojan como conclusión que se logró alcanzar el mismo grado de error en algunas iteraciones menos (5173 [1 capa] contra 4793 [2 capas]) y por lo tanto menor tiempo de entrenamiento aunque la diferencia no es demasiado significante, ni irreversible ya que puede deberse a varios factores. Cabe destacar que durante el proceso de entrenamiento, la valoración del error fue mucho más grande que cuando se trabaja a una capa, es decir que el error arrancó desde un gran valor (aproximadamente 17) siendo que a una capa rápidamente en las primeras iteraciones el valor se situa alrededor del 1.

Procederemos a evaluar y comparar los resultados de este entrenamiento a dos capas, contra su par a una capa y determinar si se mejoró o no el grado de suceso. Los resultados para las dos capas se muestran en la Tabla 7.

Tabla 7 – Resultados de entrenamiento utilizando imágenes y escritura manual

LETRA	TIPO	PATRONACIERTO HIGH - PORCENTAJE (%)	PATRON ACIERTO LOW - PORCENTAJE (%)	COMENTARIOS
B	IMAGEN	B – 75%	3 – (3%)	
1	IMAGEN	1 – (99%)	0 – (0%)	
X	IMAGEN	X – (99%)	K– (0%)	
B	ESCRITURA MANUAL	R – 16%	D – (3%)	
1	ESCRITURA MANUAL	1 – (89%)	0 – (1%)	
X	ESCRITURA MANUAL	X – (60%)	V– (2%)	

Para el reconocimiento por imágenes, vemos idénticos resultados que el entrenamiento a una capa. Mismos porcentajes y mismos aciertos, por lo cual podemos inferir que no afectó.

Dibujando a mano alzada seguimos teniendo los mismos problemas por lo tanto se nos dificulta encontrar una forma de obtener el correcto reconocmiento de la letra “B”. Además los porcentajes de acierto han ido en decrimento por lo cual podemos inclinarnos a decir que el entrenamiento a dos capas nos perjudicó el términos de correcto reconocimiento a mano alzada.

Como hemos venido mencionando, la presencia de tanta incertidumbre al trabajar con dibujos a mano alzada nos ha de preocupar porque es el escenario que imaginamos vamos a encontrarnos a lo largo de nuestro trabajo: los billetes se presentan irregulares y la captura de una foto de ellos se presentará aún más irregular, por lo tanto las probabilidades de que tengamos una imagen igual a una entrenada son remotamente posibles, por lo tanto necesitamos que nuestra red esté entrenada al punto de poder trabajar a partir de estas diferencias.

Nos queda realizar una última prueba con imágenes de billetes para ver el comportamiento de este software y compararlo contra nuestro propósito. Como es lógico, al trabajar con imágenes de billetes solamente podremos evaluar el procesamiento por imágenes y no a mano alzada de los dibujos que realicemos, ya que el software no nos permite dibujar tanta extensión y además termina trabajando monocromático (blanco y negro).

Intentaremos entrenar entonces nuestra red con el siguiente conjunto de imágenes que se ilustran en la figura 36.

Figura 36 – Imágenes de billetes para entrenamiento - BCRA

Luego de 3 horas y 41 minutos de procesamiento, se obtuvo la red entrenada. En ese tiempo solo se produjeron 869 iteraciones, lo que da la pauta de que el entrenamiento de las imágenes fue un proceso exigente y lento, por lo tanto mejorar este nivel de error sería muy demandante en términos de recursos y tiempos de procesamiento.

Una vez entrenada la red procedemos a realizar las pruebas con las mismas imágenes que fueron entrenadas. Los resultados se enuentran en la siguiente Tabla 8

Tabla 8 – Resultados de entrenamiento utilizando imágenes de billetes

BILLETE	TIPO	PATRON ACIERTO HIGH - PORCENTAJE (%)	PATRON ACIERTO LOW - PORCENTAJE (%)	COMENTARIOS
2 PESOS	IMAGEN	2 PESOS – 82%	20 PESOS – 3%	
5 PESOS	IMAGEN	5 PESOS – 95%	10 PESOS – 3%	
10	IMAGEN	10 PESOS –95%	100 PESOS – 0%	

PSOS				
20 PESOS	IMAGEN	20 PESOS – 94%	100 PESOS – 2%	
50 PESOS	IMAGEN	50 PESOS – 97%	100 PESOS – 2%	
100 PESOS	IMAGEN	50 PESOS – 63%	100 PESOS – 41%	

Los resultados se presentaron bastante exitosos con aciertos del 97% al 82% siendo que fueron utilizadas las mismas imágenes con las que fueron entrenadas. Se obtiene un caso de error a tener en cuenta para el billete de 100 pesos donde se resuelve como más probable el billete de 50 pesos, con porcentajes considerables.

Dada esta última evaluación, se presenta al aplicativo como una buena opción ante el procesamiento de imágenes que necesitamos realizar, aunque debemos tener en cuenta estos casos de fallo que ocurren inclusive cuando se utiliza la misma imagen con la que fueron entrenadas. Realizado este análisis, podemos escribir las siguientes ventas y desventajas.

Ventajas

- Permite parametrizar la red neuronal en capas y luego las neuronas de cada una
- El reconocimiento de las imágenes puede hacerse por archivo o dibujo a mano
- Posibilidad de exportar el entrenamiento de la red a un formato codificado (se requiere modificar el código fuente)
- Numerosos resultados de éxito en el reconocimento de imágenes

Desventajas

- Lenguaje C++ no adaptable nativamente a Android
- El valor del error entrenado no se ve reflejado en los posteriores reconocimientos
- Reconocimientos erróneos para las mismas imágenes con la cual la red fue entrenada

4.5 - Caso de estudio nº 3: Aplicación de Redes Neuronales[10]

Propósito

Este tercer software para entrenamiento y procesamiento de redes neuronales, es una herramienta en Java que permite ingresar una cantidad de patrones (10 en principio) dibujados a mano, para luego proceder a un entrenamiento de la red con ciertos parámetros y finalmente realizar un procesamiento nuevamente dibujando la figura que nos interesa reconocer sobre los patrones entrenados.

Desarrollo

El funcionar de esta aplicación es similar a los anteriores casos, salvo que ésta en particular no provee un entrenamiento en base a imágenes de archivo sino que permite crear sus patrones al comenzar su ejecución. Para ello tiene un modo inicial de "aprendizaje", en donde debemos completar los 10 patrones que queramos utilizar, dibujando sobre una zona de dibujo de 5x7 píxeles. A cada patrón se le puede indicar un nombre representativo de lo que hayamos dibujado.

Luego existe un botón de entrenamiento donde se pueden definir un factor de aprendizaje, un valor de error buscado y una cantidad de iteraciones máximas por si este error no puede ser alcanzado.

Finalmente permite hacer un reconocimiento nuevamente dibujando sobre el cuadro de dibujo y realizando el proceso de cálculo. También el programa provee una consola que indica el estado del procesamiento como así también la matriz que se está dibujando.

Utilización

A continuación en la figura 37 se mostrarán la pantalla principal del programa junto con el comentario de cada una de sus funciones, para luego comenzar a analizar y probar el funcionamiento en conjunto del mismo.

[10] Aplicación de redes neuronales en java – Pablo Borbón :

http://pabloborbon.com/2011/06/07/aplicacion-de-redes-neuronales-en-java/

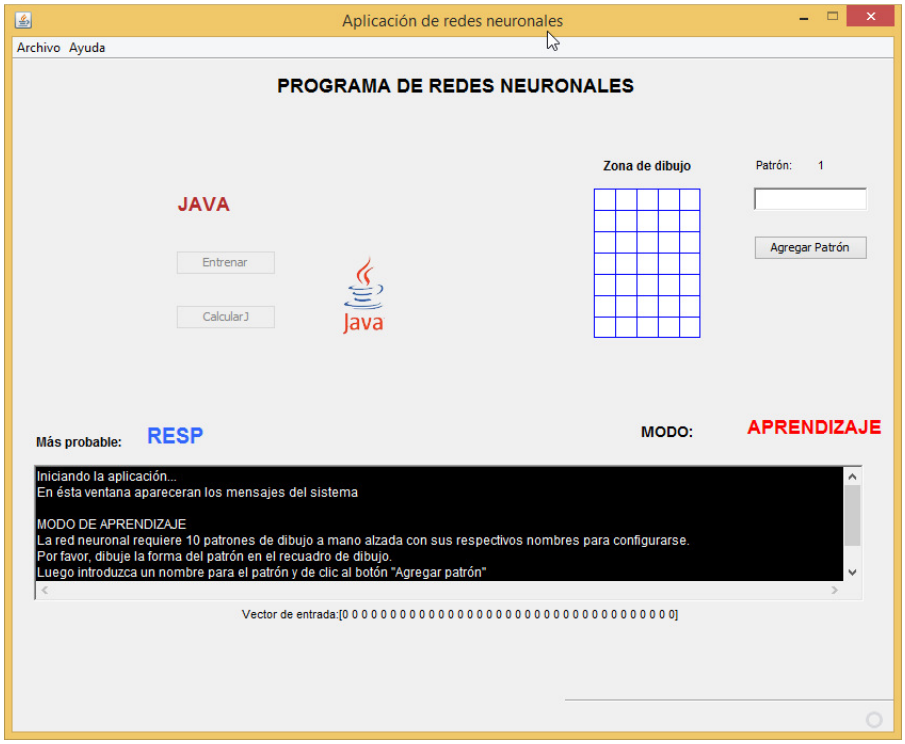


Figura 37 – Programa de redes neuronales JAVA - CODEPROJECT

El aplicativo posee esta pantalla principal en donde se realizarán todas las acciones pertinentes. Debajo a la derecha, y en esta figura en color rojo, se indica el estado en que se encuentra el aplicativo o bien el modo como ahí se expresa. Como todavía no se ha entrenado ninguna red, el modo es “aprendizaje”.

En el recuadro de fondo negro, es la consola donde irán apareciendo los mensajes tanto del entrenamiento como de los reconocimientos. Ahí se indica que deben dibujarse 10 patrones a mano en la zona de dibujo, para que se habilite el botón de entrenar.

Para crear los patrones basta con dibujarlas con el mouse en la zona de dibujo de 5x7, y debajo de patrón 1 en la caja de texto indicar el nombre que se desee para esa figura. Ese procedimiento se repite hasta alcanzar los 10 patrones, y luego se realiza el entrenamiento y posterior reconocimiento.

Comenzaremos a dibujar nuestros patrones que en este caso serán del número 0 al 9, y entrenar la red neuronal con distintas configuraciones (ver figura 38).

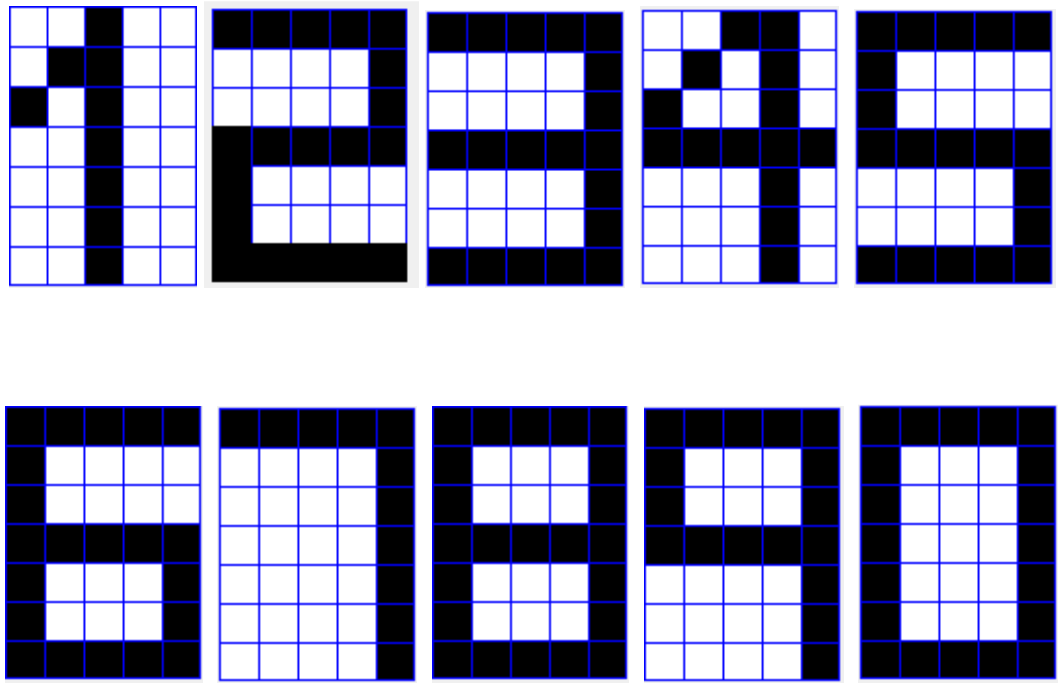


Figura 38 – Imágenes de patrones entrenamiento de red neuronal.

Una vez completados los patrones, la aplicación cambia a modo “Ejecución”.

Realizados nuestros patrones, se procede a coomenzar el entrenamiento de la red apretando en el botón de “Entrenar”. Ahí mismo se podrán colocar los valores para el factor de aprendizaje, el error buscado y una cantidad máxima de iteraciones.

Procederemos a realizar la primer prueba con un factor de aprendizaje igual a 1, un error del 0,01 y 10000 iteraciones máximo.

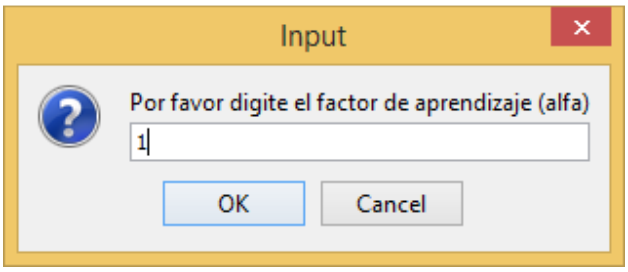


Figura 39 – Configuración de entrenamiento de Programa de redes neuronales JAVA

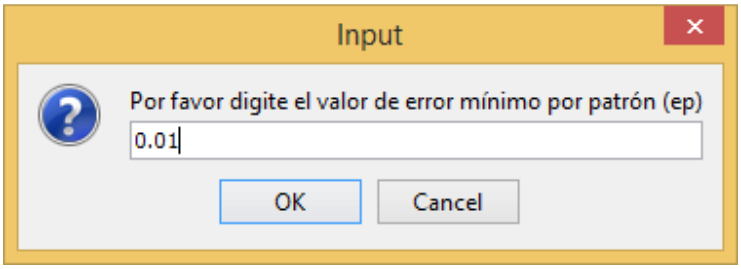


Figura 40 – Configuración de entrenamiento de Programa de redes neuronales JAVA

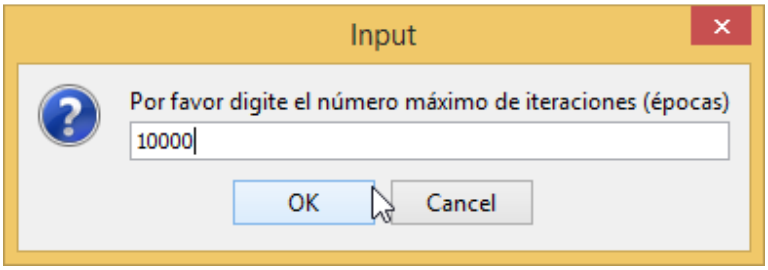


Figura 41 – Configuración de entrenamiento de Programa de redes neuronales JAVA

Completados estos tres datos (figuras 39, 40 y 41), comienza realmente el entrenamiento de la red neuronal. Pocos segundos después ya tenemos el resultado del proceso que se ilustra en la figura 42.

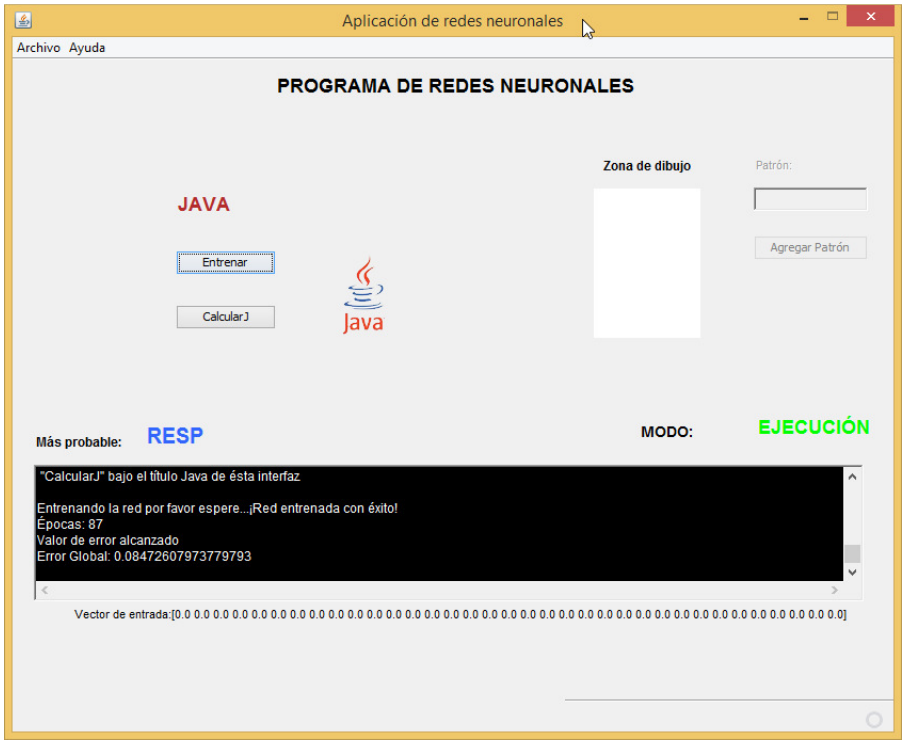


Figura 42 – Resultado de entrenamiento - Programa de redes neuronales JAVA

Podemos visualizar en la figura 42 que solamente tomó 87 iteraciones conseguir el error global menor a 0,01 lo cual en principio se nos presenta como un cálculo demasiado rápido. Procederemos a dibujar una figura para probar la bondad de la función de reconocimiento. La figura a dibujar es la del número 1 (Figura 43):

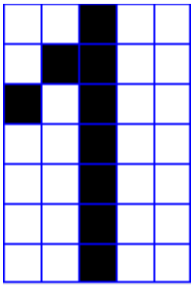


Figura 43 – Patrón "Número 1"

A continuación se aprieta el botón CalcularJ y se espera a que tenga lugar el reconocimiento.

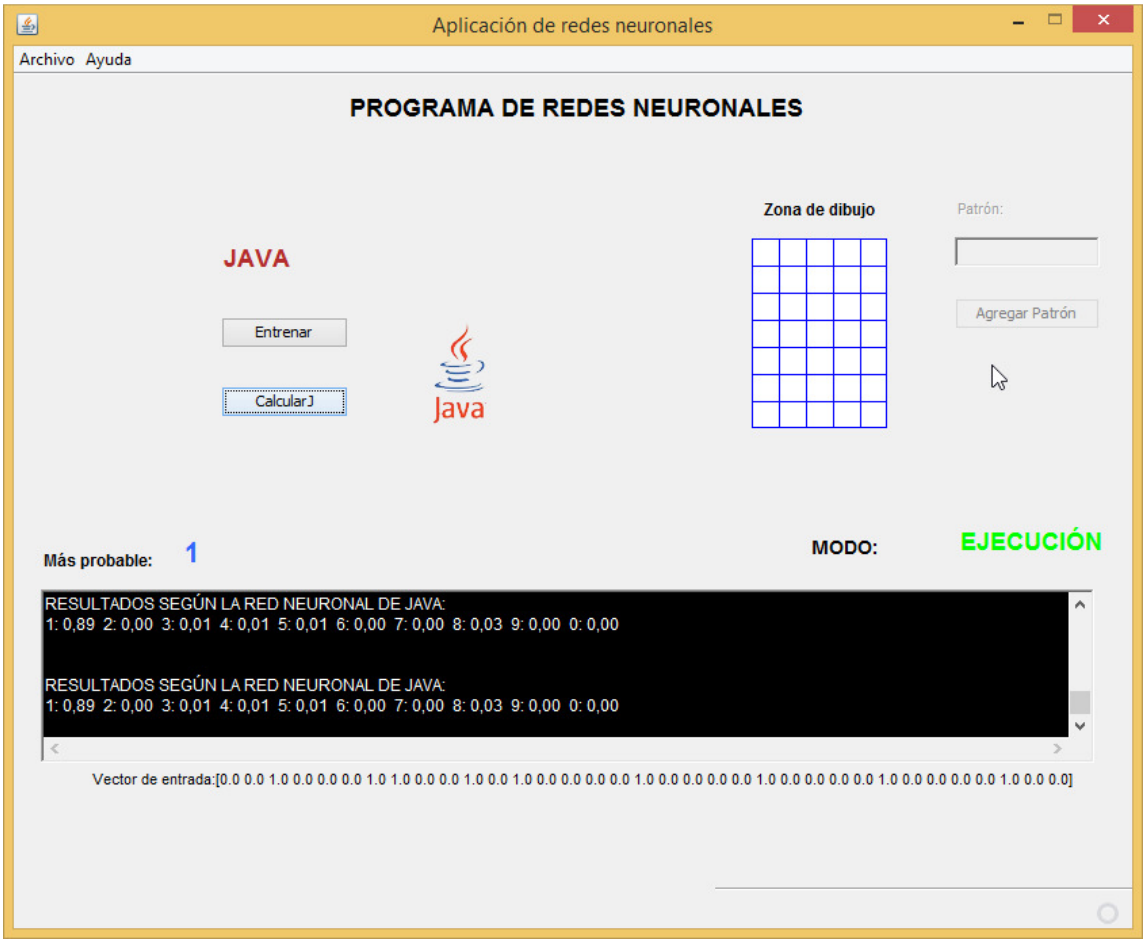


Figura 44 – Resultado de ejecución - Programa de redes neuronales JAVA

Como se visualiza en la figura 44 nuestro primer intento fue exitoso. Nos indicó como resultado el número 1 con un 89% de acierto. Algo interesante de este aplicativo es que nos indica cuáles fueron los porcentajes de cada valor. Podemos ver que el número 1 tuvo el mayor porcentaje, pero que también hubieron otros números como el 4 y el 5 que aportaron algún grado de reconocimiento aunque sea mínimo. A continuación analizaremos el comportamiento para todos los números y sus valores resultado (ver Tabla 9).

Tabla 9 – Resultados para los números 2 a 0

Número 2	91%	Número 3	89%	Número 4	87%
Número 5	89%	Número 6	90%	Número 7	89%
Número 8	88%	Número 9	90%	Número 0	91%

Todos los casos fueron de éxito a un valor aceptable, utilizando la misma figura dibujada por la que fueron entrenados. Para continuar con nuestro análsis, se procederá a entrenar la red a un nivel menor del error para comprobar si realmente los resultados mejoran en su precisión, permitiendo también que cuando utilicemos imágenes disímiles, la respuesta continúe siendo satisfactoria.

Por lo tanto entrenaremos la red a un factor de aprendizaje 1; un nivel de error de 0,0001; y 10000000 iteraciones.

A este nivel de error los resultados del reconocimiento de cada número indican lo siguiente (ver Tabla 10):

Tabla 10: Resultados del 0 al 9

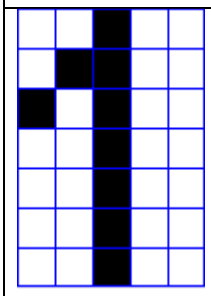
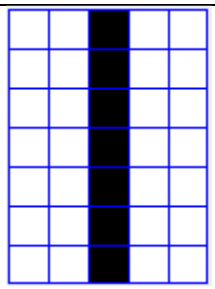
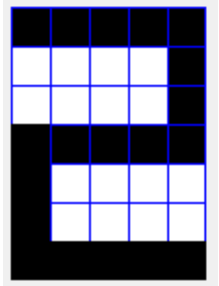
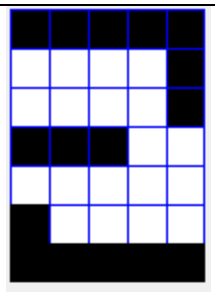
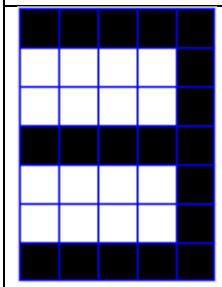
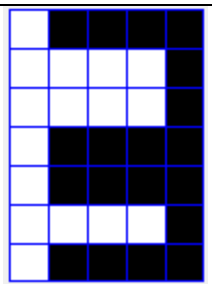
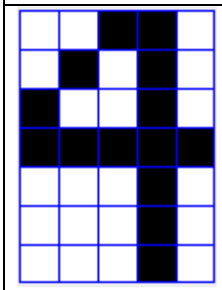
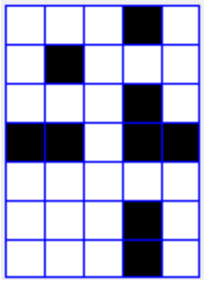
Número 1	99%	Número 2	99%	Número 3	99%
Número 4	99%	Número 5	99%	Número 6	99%
Número 7	99%	Número 8	99%	Número 9	99%
Número 0	99%				

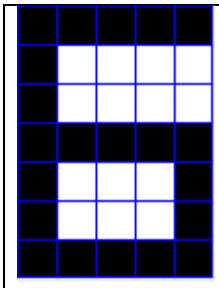
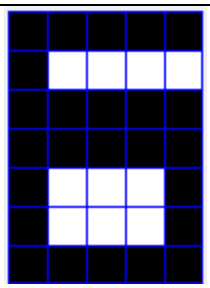
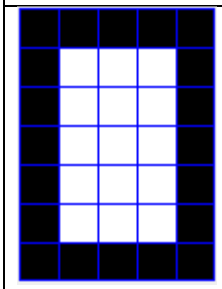
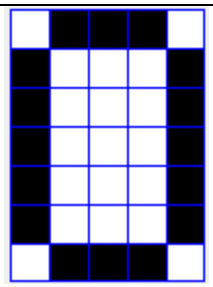
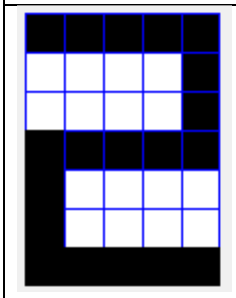
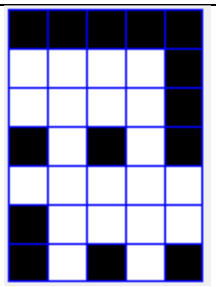
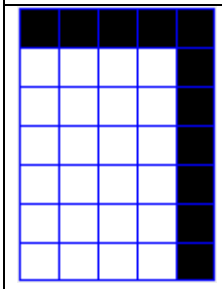
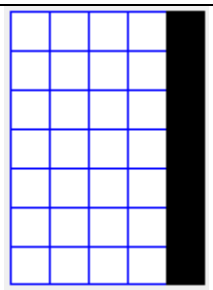
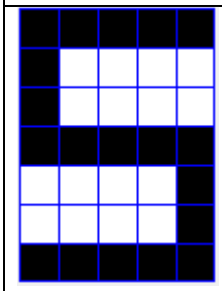
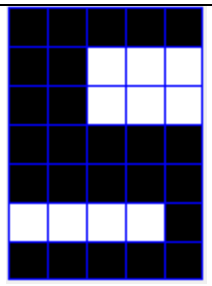
Todos los reconocimientos con figuras iguales nos devuelven el grado de error por el cual fueron entrenados, de 0,0001. Si bien la red se presenta como potencialmente acrecentable en su grado de error, para nuestro análisis ya podemos inferir que a mayor grado de error los resultados van a seguir presentándose en evidencia con su entrenamiento, por lo cual de necesitar más iteraciones será solamente en caos de utilizarla finalmente como método elegido y así obtener una red más robusta.

En pos del estudio del software, un punto interesante a examinar es el comportamiento de la red a la hora de reconocer figuras con píxeles distorcionados o diferentes a los cuales fueron entrenados. Generaremos un lote de pruebas para ver

hasta qué punto la red puede encontrarnos un reconocimiento correcto. A continuación la documentación de estas pruebas (ver Tabla 11).

Tabla 11 : Resultados ingresando patrones numéricos dibujados manualmente

Figura entrenada buscada	Figura a reconocer	Resultado
		Patrón 1 (99%)
		Patrón 2 (83%)
		Patrón 3 (89 %)
		Patrón 4 (95%)

		Patrón 6 (53%) Patrón 8 (43%)
		Patrón 0 (91%) Patrón 8 (21%)
		Patrón 2 (18 %)
		Patrón 3 (30%) Patrón 7 (9%)
		Patrón 6 (45%) Patrón (23%)

Las pruebas con figuras similares se presentan en principio muy satisfactorias también ya que ha habido aciertos reales cuando a las figuras les faltan algunos píxeles o tiene alguno extra sin perder la forma real del número.

En otros casos, hemos optado por no dibujar muchos de los píxeles que componen la figura como por ejemplo para el número 7 que no se pudo determinar correctamente la figura. En este caso la red neuronal nos demuestra que tiene razón ya que con esa sola línea no se puede realmente confirmar lo que se quiso realizar porque es insuficiente, amén de que el software nos provea como válida la respuesta más probable. Otro caso similar es el número 5, que al haberlo extendido en sus píxeles, también ha tomado forma muy similar a la del número 6 entrenado por lo cual la red nos indica ese número como más probable, ya que en términos morfológicos, de haber dibujado un píxel más que cierra la panza del número 5, nuestro entrenamiento humano podría aseverar que lo que se realizó era un 6 y no un 5.

Por lo tanto es muy interesante lo que ofrece este sistema en cuanto a niveles de entrenamiento y posibilidades del reconocimiento de las imágenes. En primer lugar nos permite entrenar la red a un nivel de detalle superior a lo que veníamos trabajando, que realmente después se ve reflejado a ese nivel cuando se procesan los reconocimientos. Por otra parte, esta conjunción de entrenamiento y reconocimiento, facilitan la obtención de resultados en escenarios reales donde las figuras son distintas a las que produjeron el entrenamiento.

A partir de las tareas realizadas, concluímos en los siguientes puntos fuertes y débiles de este sistema:

Ventajas

- Lenguaje Java 100% adaptable a Android
- Entrenamiento de la red neuronal a un error comprobable
- Facilidad de exportación de los pesos de cada neurona de la red
- Reconocimiento exitoso para casos homogéneos y heterogéneos
- Algoritmos reutilizables y programabables para nuestros requerimientos de reconocimiento de billetes (requerirá convertir imágenes a matriz de 1 y 0)

Desventajas

- Matriz de dibujo manual
- Matriz de trabajo standard pequeña

4.6 Análisis de los casos de estudio

En este punto repasaremos brevemente los casos de estudio mencionados para poder elegir el camino de acción a seguir basándonos en algoritmos y procedimientos antes vistos.

Primeramente publicamos un aplicativo que su funcionalidad era más orientada al reconocer formas de los objetos más que su contenido, por lo tanto su utilidad a nuestro proyecto no era muy viable. Este aplicativo que si bien es usado para clasificar las formas de las hojas y que realmente cumple su cometido, también ha promovido ideas a utilizar en nuestro desarrollo. Ideas importantes que rescatamos aquí es el tratamiendo de las imágenes y la utilización de la clasificación de imágenes. En cuanto al tratamiento de imágenes que realiza, nos ilumina a utilizar algún mecanismo similar en donde las imágees que obtengamos también sean dibujadas de esta forma lineal, evitando problemas con el color y la luz al momento de obtener la foto. El otro punto acerca de la clasificación, como el sistema permite agrupar distintas imágenes a una determinada clasificación, nosotros podremos aplicar esto a nuestro sistema a implementar, agregando distintas imágenes de entrenamiento a nuestros grupos de clasificación que serán finalmente los billetes. Si bien el entrenamiento de las redes neuronales, el proceso de reconocimiento y su forma de implementación no están relacionadas a nuestro propósito, estos temas nos sirven de guía ante problemas futuros que se presentarán.

Luego en el segundo caso utilizamos una aplicación más orientada a nuestros objetivos, en donde pudimos experimentar luego de varias pruebas el reconocimiento basadao en el entrenamiento de letras y números. Aquí pudimos realizar muchas pruebas diversas debido a la posiblidad del sistema de cargar nuestros patrones a evaluar de forma manual o por archivo, generando gran variedad de información resultante. Si bien algunos casos la precisión no era la correcta, parece ser adaptable a nuestra idea de aplicación, aunque requiere de un trabajo de transformación entre el lenguaje actual a nuestro lenguaje objetivo para la movilidad en Android.

Y por último, la tercera aplicación también relacionada a nuestro propósito, permitiendo entrenar la red con dibujos hechos a mano y luego encontrar su semejanza a través del proceso de reconocimiento. Aquí vimos muy buenos resultados de exactitud en cuanto a valores de error que nos auguran un buen proceder con esta herramienta, aunque con algunas limitantes expuestas anteriormente.

Llega un punto en que debemos elegir qué algoritmo realizar, y aunque podríamos tener un favorito, preferiremos que los números nos indiquen también su parecer. Nos referimos a que, ya que tanto la opción dos y tres se presentan viables para nuestro proyecto, compararlos en el mismo terreno con mismos casos de prueba y así tener una razón adicional y empírica de que el algoritmo elegido funcionará mejor en nuestro aplicativo.

Es por ello que propondremos a continuación, un estudio comparativo entre los aplicativos del caso 2 y caso 3, en donde la imagen que dibujaremos a mano éste último será utilizada como imagen de entrenamiento y reconocimiento en el aplicativo 2. A mismas imágenes, mismo grado de error, veremos cuál de los dos aplicativos consigue mayor grado de confianza.

Procedemos a presentar las imágenes que serán utilizadas para estos casos. Las mismas son en total 10 y están compuestas por 5 píxeles de ancho por 7 píxeles de alto, con sus puntos pintados en color negro que conforman la figura del número que representan. A continuación en la figura 45 dejamos una imagen ampliada de cómo se visualizan las mismas:

Figura 45 – Imágenes del 0 al 9 utilizadas en entrenamiento

Como pueden verse, son los mismos números que utilizamos en el entrenamiento de la red en el caso 3, pero llevados a archivos así son comparables para el aplicativo 2. Una vez generadas las imágenes, procedemos a realizar el entrenamiento de cada programa. Comenzaremos entrenando la red del aplicativo “BP simplificado” a una sola capa ya que no obtuvimos mejores resultados al hacerlo con otra cantidad de neuronas, y para un error de 0,001.

Una vez entrenadas ambas redes, es momento de realizar la comparativa final en cuanto al reconocimiento de imágenes. Debajo dejaremos para cada aplicativo, cuál fue la opción obtenida como correcta y sus porcentajes de acierto (ver Tabla 12).

Tabla 12 – Comparativa de resultados utilizando los casos de estudio 2 y 3

Imagen	BP Simplificado (caso 2)	Redes Neuronales (caso 3)
	**0 (97%)**	0 (96%)
	**1 (99%)**	1 (96%)
	**2 (98%)**	2 (97%)
	3 (96%)	**3 (97%)**
	**4 (99%)**	4 (96%)
	5 (96%)	**5 (97%)**
	6 (95%)	**6 (96%)**
	**7 (98%)**	7 (97%)
	8 (92%)	**8 (96%)**
	9 (96%)	**9 (97%)**
	0 (94%)	**0 (96%)**

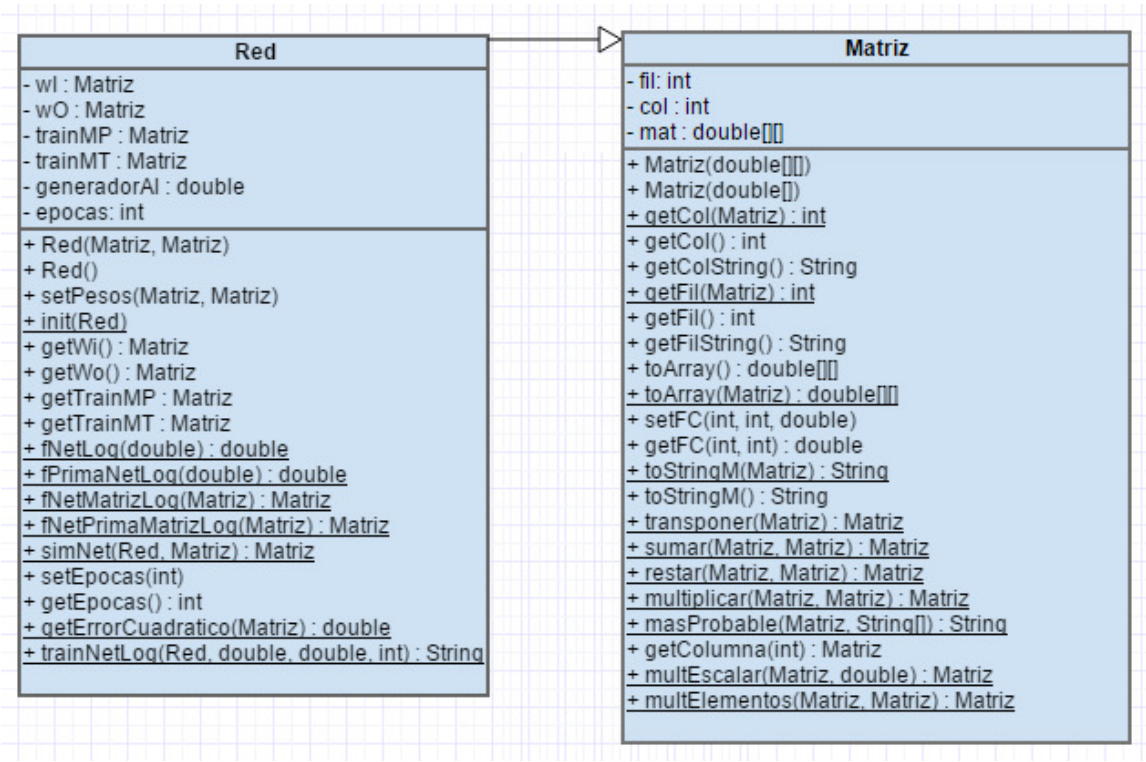
	**1 (96%)**	**1 (96%)**
	2 (94%)	**2 (95%)**
	3 (89%)	**3 (94%)**
	**4 (96%)**	**4 (96%)**
	**5 (93%)**	5 (92%)
	6 (94%)	**6 (96%)**
	**7 (97%)**	7 (96%)
	8 (92%)	**8 (95%)**
	**9 (91%)**	9 (90%)

La comparación de ambos aplicativos fue extremadamente similar, a tal punto que generamos una tanda de pruebas con figuras alteradas para conocer su comportamiento y definir entre los dos sistemas. Para el BP Simplificado, hubieron 10 casos en donde obtuvo el mayor % de acierto. Y para el de Redes Neuronales, se obtuvieron 12 casos en esa condición (visualizar que ante empate, ambos sumaron como correctos). Por lo tanto y por escaso margen, el aplicativo de Redes Neuronales basado en Java se presentó ligeramente mejor a su par evaluado.

Como conclusión de esta prueba, esperábamos obtener algún resultado mayormente dispar pero solamente confirmó lo que habíamos planteado en un principio: ambos programas están aptos o nos proveerán del mismo beneficio en cuanto a su utilización del algoritmo de entrenamiento y reconocimiento en nuestra aplicación para dispositivos móviles. Con este ligero punto a favor, y con las ventajas presentadas anteriormente, nuestro algoritmo elegido a trabajar será el utilizado por el mencionado en el caso de estudio nº 3, **"Aplicación de Redes Neuronales".**

4.7 - DESARROLLO DE APLICACIÓN JAVA

Diagrama de clases



El diagrama de clases del algoritmo que entrena y realiza el procesamiento de reconocimiento, es bastante sencillo. Consta de una clase principal que es nuestra red neuronal, donde están definidos atributos como los pesos de entrada y salida, que son de un tipo Matriz, la otra clase utilizada por este algoritmo, y con métodos para simular la red neuronal a través de funciones de propagación sigmoidales. Define las matrices de pesos, los métodos de entrenamiento, de propagación y en general todos los aspectos de campos y funcionalidad de la red neuronal. La clase Matriz provee de objetos de este tipo a la clase Red junto con métodos para realización de cálculos vectoriales rápidos y sencillos.

Descripción de los métodos

A continuación se expresa una tabla con una breve y resumida documentación de cada método indicado en el diagrama de clases (ver Tabla 13).

Tabla 13 – Descripción de métodos

Clase Red		
Método	**Código**	**Observaciones**
Constructor con matrices de patrones (Red)	public Red(Matriz mP1, Matriz mT1)	Crear clase Red * @param mP1 Matriz de patrones de entrada * @param mT1 Matriz de objetivos
Constructor vacío (Red)	public Red()	Crear clase Red
Establecer pesos	public void setPesos(Matriz pesosI, Matriz pesosO)	Coloca las matrices de la capa oculta y la capa de salida en el objeto Red. * @param pesosI pesos de la capa oculta * @param pesosO pesos de la capa de salida
Inicializar red	public static void init(Red redNeu)	Inicializa los pesos de la red neuronal pasada como parámetro con un número aleatorio dentro de una distribución de probabilidad Gaussiana con valores entre [-0.5,0.5] * @param redNeu Red Neuronal Objetivo
Obtener pesos ocultos	public Matriz getWi()	Devuelve la matriz de pesos de la capa oculta * @return Matriz Wi
Obtener pesos salida	public Matriz getWo()	Devuelve la matriz de pesos de la capa de salida * @return Matriz Wo
Obtener patrones de entrada	public Matriz getTrainMP()	Devuelve la matriz con los patrones de entrada de entrenamiento * @return Matriz patronesEntrada
Obtener salidas objetivo	public Matriz getTrainMT()	Devuelve la matriz con los objetivos de entrenamiento * @return Matriz con los vectores de salida
Función sigmoidal	public static double fNetLog(double value)	Evalúa la función sigmoidal 1/(1+e^-n) de un valor dado * @param value valor a evaluar * @return1/(1+e^-value)
Derivada sigmoidal	public static double fPrimaNetLog(double value)	Evalúa la primera derivada de la función sigmoidal 1/(1+e^-n) * @param value valor a evaluar * @return (1/(1+e^-n))*(1-1/(1+e^-n))
Función sigmoidal de la matriz	public static Matriz fNetMatrizLog(Matriz mat)	Evalúa la función sigmoidal a toda una matriz * @param mat matriz a evaluar * @return Matriz con sus elementos evaluados
Derivada sigmoidal de la matriz	public static Matriz fNetPrimaMatrizLog(Matriz mat)	Evalúa la derivada de la función sigmoidal a toda una matriz * @param mat matriz a evaluar * @return Matriz con sus elementos evaluados

Simular Red Neuronal	public static Matriz simNet(Red redNeu, Matriz Ventrada)	Simula el comportamiento de la red neuronal. Multiplica matrices, evalúa las funciones de propagación y en general propaga la red hacia adelante. * @param redNeu Red Neuronal modelo * @param Ventrada Vector de entrada a evaluar * @return Matriz con un vector de respuestas de la red
Establece iteraciones del entrenamiento	public void setEpocas(int num)	Establece el número de épocas de entrenamiento han transcurrido en un proceso de entrenamiento tras la condición de salida * @param num contador de épocas
Obtiene iteraciones del entrenamiento	public int getEpocas()	Devuelve el número de épocas que tuvo una red para alcanzar la condición de salida. * @return int numEpocas
Error cuadrático	public static double getErrorCuadratico(Matriz errores)	Obtener Error cuadrático * @param errores Matriz con los errores (yd-o) * @return Double con la sumatoria de los errores al cuadrado multiplicado por ½

Entrenar red neuronal	public static String trainNetLog(Red redNeu,double alpha,double error, int iteraciones)	Entrena la red neuronal con el siguiente algoritmo:
		1.Se inicializa a red (valores aleatorios de wi y wo)
		2.huboError=true;
		Ciclo1.Mientras (epocas < iteracionesMáximas)&& (huboError==true))
		Ciclo2 for j=0;j<10;j++
		3.Se presenta el patrón j y se propaga la red hacia adelante
		4 Se calcula el error
		Cond1 .If error>error{
		-huboError=true
		-Se propaga la red hacia atrás
		-Se actualizan los pesos}
		fin del ciclo2
		epocas++;
		fin del cliclo1
		El algoritmo se termina cuando todos los patrones tengan un valor de error medio cuadrático menor que el establecido como parámetro o cuando se supera el número máximo de iteraciones.
		* @param redNeu Red neuronal modelo
		* @param alpha factor de aprendizaje
		* @param error error objetivo por patrón
		* @param iteraciones número máximo de iteraciones del algoritmo
		* @return String cadena con los resultados del entrenamiento

Clase Matriz		
Método	**Código**	**Observaciones**

Constructor Matriz bidimensional	public Matriz(double m[][])	Crear clase Matriz * @param m Arreglo bidimensional de double
Constructor Matriz unidimensional	public Matriz(double m[])	Crear clase Matriz * @param m Arreglo unidimensional de double
Columnas de la la Matriz	public static int getCol(Matriz m)	Obtener número de columnas * @param m Matriz * @return int NumColumnas
Obtener número de columnas	public int getCol()	Obtener número de columnas * @return int NumColumnas
Obtener número de filas	public static int getFil(Matriz m)	Obtener número de filas * @param m Matriz * @return int NumFilas
Convertir matriz en array	public static double[][] toArray(Matriz m)	Convertir matriz en array * @param m Matriz * @return Double array[][]

Colocar valor en Matriz	public void setFC(int f,int c, double value)	Setear valor en índices de la Matriz * @param f índice de fila * @param c índice de columna * @param value Valor que se coloca en el arreglo de la matriz
Obtener valor en Matriz	public double getFC(int f,int c)	Obtener valor en Matriz por índices * @param f índice fila * @param c índice columna * @return double valor
Matriz a String	public static String toStringM(Matriz m)	Convertir Matriz a String para visualización * @param m Matriz a convertir * @return String con las filas y las columnas de la matriz debidamente ordenadas.
Transponer Matriz	public static Matriz transponer(Matriz m)	Transposición de la Matriz * @param m matriz a transponer * @return Matriz m'

Sumar matrices	public static Matriz sumar(Matriz mA, Matriz mB)	Cálculo de suma entre Matrices * @param mA sumandoA * @param mB sumandoB * @return Matriz (ma+mB)
Restar matrices	public static Matriz restar(Matriz mA, Matriz mB)	Cálculo de resta entre Matrices * @param mA Minuendo * @param mB Sustraendo * @return Matriz diferencia= mA-mB
Multiplicar matrices	public static Matriz multiplicar(Matriz mA, Matriz mB)	Cálculo de multiplicación entre Matrices * @param mA factor A * @param mB factor B * @return mAxmB * La matriz de retorno tiene dimensions [fA]x[cB]

Procesar reconocimiento	public static String masProbable(Matriz resp,String nom[])	Calcular la respueta más probable * @param resp Matriz con las respuestas de la simulación * @param nom Arreglo de Strings con los nombres de los patrones * @return String con el patrón más probable del arreglo
Obtener columna	public Matriz getColumna(int index)	Obtiene columna según índice * @param index índice de la columna a obtener * @return Matriz Columna con todos los valores de la columna indicada
Multiplicar Matriz por escalar	public static Matriz multEscalar(Matriz m, double esc)	Función para multiplicar Matriz por un número escalar * @param m Matriz a multiplicar * @param esc factor escalar * @return Matriz escalada

Multiplicar elementos entre matrices	public static Matriz multElementos(Matriz m1, Matriz m2)	Multiplica elemento a elemento entre dos Matrices * @param m1 Matriz 1 * @param m2 Matriz 2 * @return Devuelve una matriz con las mismas dimensiones con los productos de los elementos de cada matriz

Funcionamiento del algoritmo

Como ya hemos adelantado el algoritmo consta de dos partes principales, el proceso de entrenamiento y el proceso de reconocimiento. Para entender cómo es que se logran ambos procesos, se resumirán los métodos que son ejecutados en cada clase para cada comportamiento.

Para el proceso de entrenamiento, el esquema es el siguiente:

```
trainNetLog(red, alfa, error, iteraciones)
```

El primer paso es llamar a la función trainNetLog que se encaga de realizar el entrenamiento de nuestra red neuronal. El parámetro que figura como red, no es más que el conjunto de las matrices de entrada y de salida. Una vez concluída la carga de los patrones a entrenar, la clase Matriz guarda estos valores y los transpose, y repite el procedimeinto para los patrones de salida guardándolos y transponiéndolos. Ahí se tiene la primer imagen de la red neuronal sin entrenar, y es pasada como parámetro. Los otros parámetros son los que el programa solicita para control de errores e iteraciones.

Aquí dentro de esta función se ejecutan varios métodos encargados de realizar la comprobación de errores y determinar si es necesario continuar iterando en búsqueda de alcanzar el error configurado. Comienza evaluando si se ha alcanzado o no algunas de las condiciones de cierre:

```
(contEpocas<iteraciones)&&huboError==true
```

```
huboError=true
j=0
contEpocas=0;
While ((contEpocas<iteraciones) AND huboError==true)
//PROPAGAR LA CAPA OCULTA
netI=Matriz.multiplicar(redNeu.getWi(), redNeu.getTrainMP().getColumna(j))
fNetI=Red.fNetMatrizLog(netI)
//PROPAGAR LA SALIDA
netO=Matriz.multiplicar(redNeu.getWo(),fNetI)
fNetO= Red.fNetMatrizLog(netO)
//CALCULAR LOS ERRORES
eO=Matriz.restar(redNeu.getTrainMT().getColumna(j), fNetO)
```

Implementación de red neuronal en Android:

Conocido nuestro objetivo de crear un prototipo de aplicación para que pueda reconocer el valor de cada billete a través de la cámara de un dispositivo móvil, utilizando redes neuronales, nuestro primer punto a comprobar es justamente esto último: implementar una red neuronal en un celular.

Para ello y haciéndonos uso del algoritmo anteriormente descripto, comenzaremos comprobando y evaluando el comportamiento del trabajo de esta red neuronal en un dispositivo móvil. Entonces necesitamos comenzar esta prueba a través de una aplicación más sencilla que trabaje con una red neuronal. Es por ello que bajo las indicaciones de nuestro tutor, elegimos simular lo que sería la operatoria que tienen los operadores lógicos. Es decir, intentaremos recrear y ejemplificar el uso de nuestra red neuronal, desarrollando una primera aplicación que sirva para hacer algunas operaciones lógicas simples que demuestren el funcionamiento correcto o incorrecto de nuestra red.

De esta forma, para alcanzar este primer objetivo, diagramamos nuestro trabajo en dos grandes tareas de la siguiente forma:

- Entrenar una red neuronal, a través de la inserción de valores de entradas (0 y 1) y sus correspondientes salidas que simulen la operatoria de algunos operadores lógicos. Esta tarea la realizaremos dentro del aplicativo Java.
- Exportar los pesos de esta red neuronal a un aplicativo móvil que pueda a través de distintas entradas, devolver tanto un resultado acorde a su entrenamiento como a su operación lógica estipulada. Esta tarea la llevaremos a cabo directamente sobre un equipo móvil.

Entrenamiento de la red neuronal para operadores lógicos

Como primer paso de este procedimiento, se debe adaptar el aplicativo Java de Redes Neuronales, para que su funcionamiento y entramiento sea basado en matrices más pequeñas: actualmente los patrones de entrada están compuestos por matriz de 5x7, por lo que no sirven para nuestro caso. Por lo tanto es necesario modificar el lienzo del dibujo hasta reducirlo a 2 elementos que nos permitan trabajar con los valores 0 y 1. Con estos valores de entrada generaremos un patrón, que dependiendo del operador lógico y del vector de entrada, resultaremos en si es un patrón tipo 0 o tipo 1. Así, se simula el

comportamiento de una tabla de verdad, con 4 patrones resultantes que son producto de la combinatoria para las columnas X e Y de dicha tabla. Cada uno de estos grupos serán nuestros patrones de entrenamiento, como en anteriores ocasiones a lo largo de este trabajo han sido números dibujados a mano o imágenes, ahora el resultado final será tan solamente un valor 0 o un valor 1, encendido o apagado, indicando la salida de cada puerta lógica.

Ahora bien, es momento de seleccionar cuál o qué operadores lógicos serán utilizados. Se plantea esta incógnita debido a la duda de si con un solo operador es necesario para inferir que las pruebas realizadas sean suficientes, logrando alcanzar resultados aplicables y generales, o se requiere de al menos una prueba con más individuos. Es por esto que se decide utilizar no un solo operador logístico sino tres de ellos para tener una justificación con mayor peso acerca del uso de la red neuronal en un dispositivo móvil. Por lo tanto seleccionamos los siguientes tres operadores lógicos:

- OR
- AND
- XOR

A tener en cuenta, es que cada operador lógico sería una propia red neuronal, con sus determinadas entradas y distintos patrones, por lo tanto también distintos pesos de entrada y salida. Además cada operador lógico tiene su forma de funcionar distinta a los demás, por eso la necesidad de crear una red neuronal por cada uno.

Cabe destacar que la adaptación del programa no será detallada en este documento ya que no es su fin demostrar ni enumerar los pasos seguidos para modificar el aplicativo, ya que no es más que una implementación técnica y aplicada de un determinado lenguaje de programación, pero sí veremos su aplicación en los siguientes análisis.

Para finalizar la preparación de la red neuronal de compuertas lógicas, debemos definir nuestros patrones de entrada. Recordemos que en nuestro algoritmo aplicado, los patrones de entrada son representados a través de matrices. De esta forma debemos generar todas las combinaciones posibles para cada operador de manera bidimensional

con su correspondiente nombre de patrón. Por lo tanto, para cada operación tendremos los siguientes arreglos:

- Para todos los operadores OR/AND/XOR:

  patronEntradaOperador[0][0]=0;

  patronEntradaOperador [0][1]=0;

  patronEntradaOperador [1][0]=1;

  patronEntradaOperador [1][1]=0;

  patronEntradaOperador [2][0]=0;

  patronEntradaOperador [2][1]=1;

  patronEntradaOperador [3][0]=1;

  patronEntradaOperador [3][1]=1;

Tabla de verdad de entrada (resumen):

X	Y
0	0
0	1
1	0
1	1

De esta manera logramos tener los 3 operadores/patrones de entrada, cada uno con su matriz/arreglo de 4x2 asociado con todas las combinatorias posibles. A partir de allí, tendremos que definir qué resultado se espera a la salida da cada combinatoria, ahora sí dependiente de cada operador y cada entrada. Ejemplo literal: para el caso OR donde las entradas son ambas 0, retornar 0, cuando existe un 1 aplicar suma lógica y devolver 1.

A continuación se indica cómo se realizaron estas actividades en el aplicativo, previo al entrenamiento de cada red de operador lógico:

- Operador lógico OR

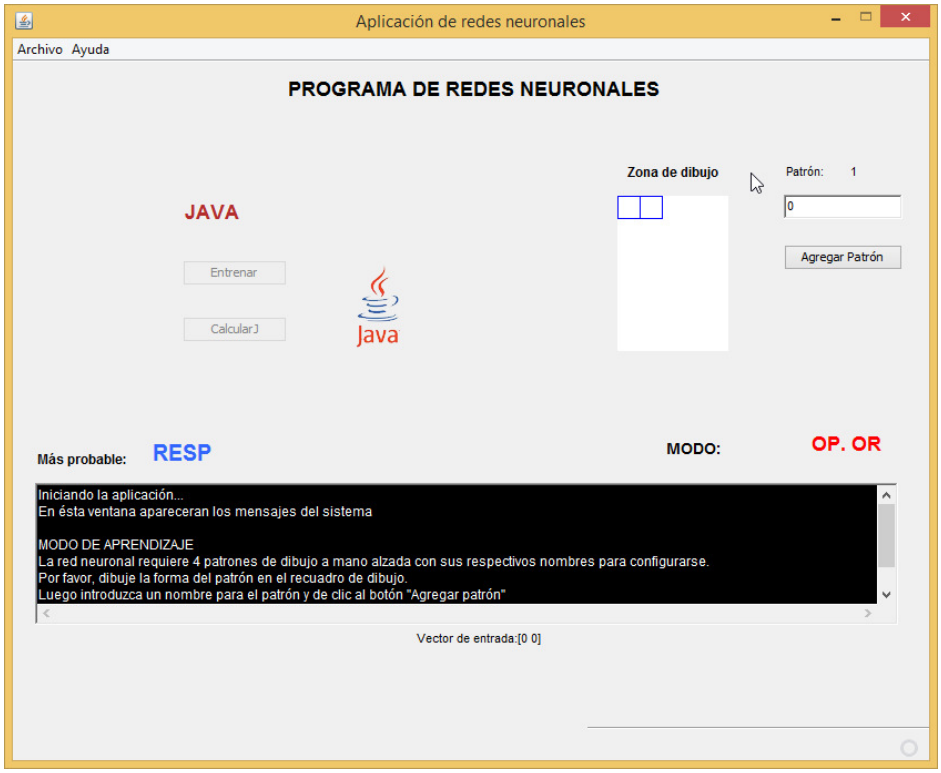


Figura 46 – Aplicación Redes Neuronales Java para resolver operadores lógicos.

Notar la zona de dibujo tiene sus dos recuadros en blancos, resultando en un vector [0 , 0].

patronEntradaOperador[0][0]=0;

patronEntradaOperador [0][1]=0;

patronSalida [0]=0;

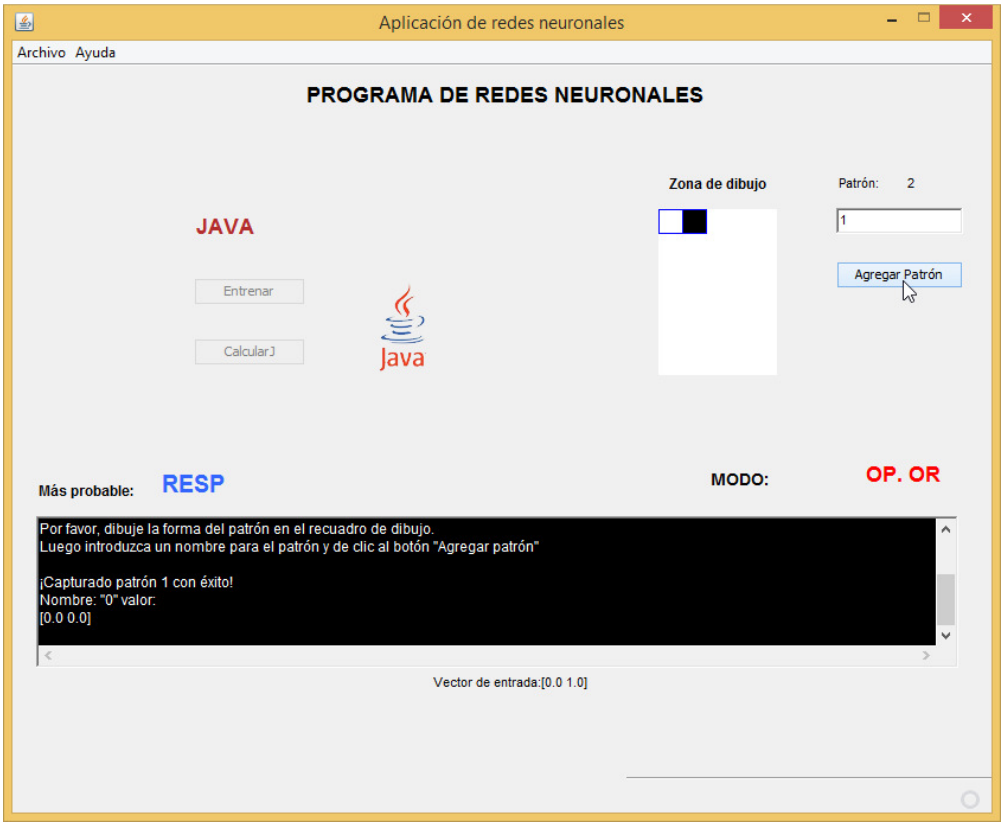


Figura 47 – Aplicación Redes Neuronales Java para resolver operadores lógicos.

patronEntradaOperador [1][0]=0;

patronEntradaOperador [1][1]=1;

patronSalida [1]=1;

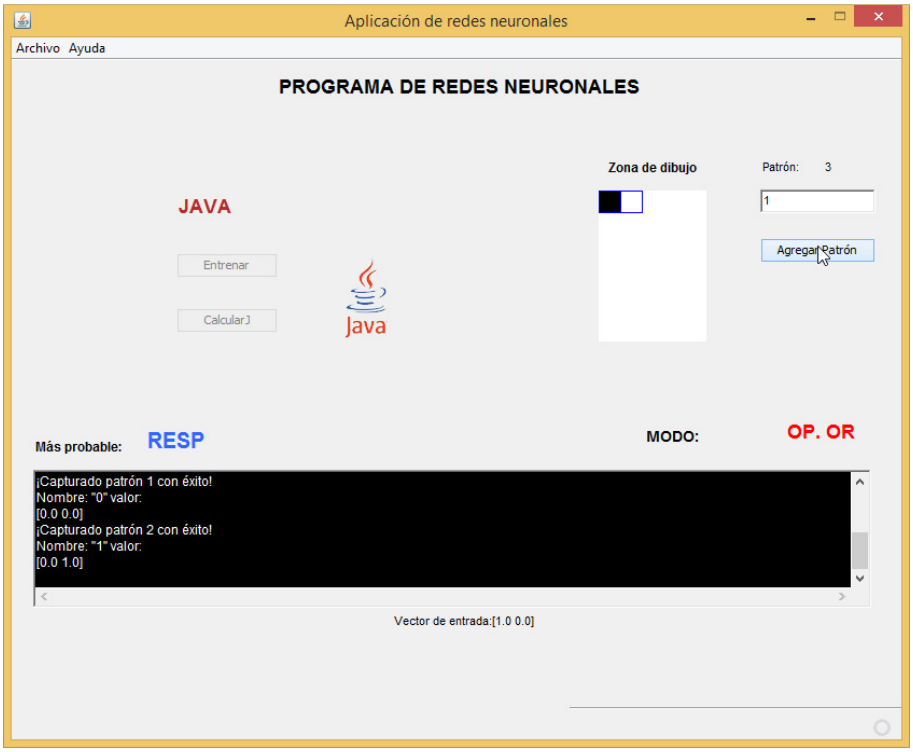


Figura 48 – Aplicación Redes Neuronales Java para resolver operadores lógicos.

patronEntradaOperador [2][0]=1;

patronEntradaOperador [2][1]=0;

patronSalida [2]=1;

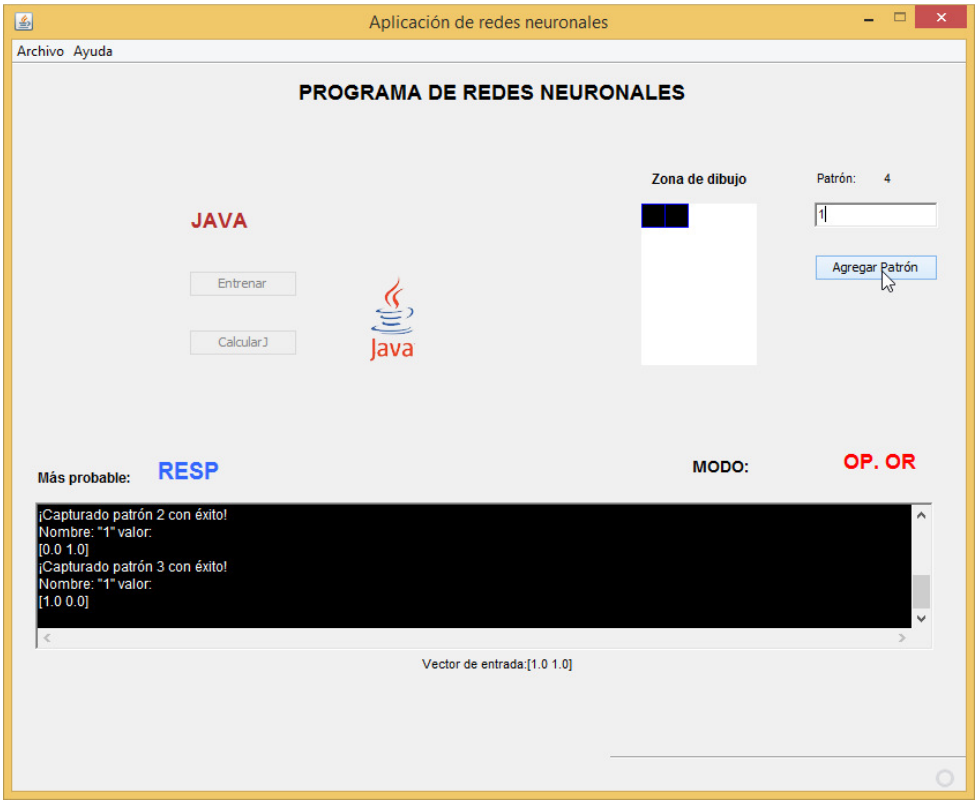


Figura 49 – Aplicación Redes Neuronales Java para resolver operadores lógicos.

patronEntradaOperador [3][0]=1;

patronEntradaOperador [3][1]=1;

patronSalida [3]=1;

Los resultados se muestran en la siguiente tabla de verdad (ver Tabla 511):

Tabla 14 – Tabla de verdad para operador OR

X	Y	Salida
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

- Operador lógico AND

  patronEntradaOperador[0][0]=0;

  patronEntradaOperador [0][1]=0;

  patronSalida [0]=0;

  patronEntradaOperador [1][0]=0;

  patronEntradaOperador [1][1]=1;

  patronSalida [1]=0;

  patronEntradaOperador [2][0]=1;

  patronEntradaOperador [2][1]=0;

  patronSalida [2]=0;

  patronEntradaOperador [3][0]=1;

  patronEntradaOperador [3][1]=1;

  patronSalida [3]=1;

Los resultados se muestran en la siguiente tabla de verdad (ver Tabla 15):

Tabla 15 – Tabla de verdad para operador AND

X	Y	Salida
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

- Operador lógico XOR

```
patronEntradaOperador[0][0]=0;
patronEntradaOperador [0][1]=0;
patronSalida [0]=0;
patronEntradaOperador [1][0]=0;
patronEntradaOperador [1][1]=1;
patronSalida [1]=1;
patronEntradaOperador [2][0]=1;
patronEntradaOperador [2][1]=0;
patronSalida [2]=1;
patronEntradaOperador [3][0]=1;
patronEntradaOperador [3][1]=1;
patronSalida [3]=0;
```

Los resultados se muestran en la siguiente tabla de verdad (ver Tabla 16):

Tabla 16 – Tabla de verdad para operador XOR

X	Y	Salida
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Entrenamiento de la red neuronal para operadores lógicos

Una vez adaptado el aplicativo y teniendo tanto los patrones de entrada como sus salidas esperadas, debemos proceder a entrenar nuestra redes a través de estos patrones. Para eso primeramente debemos definir si se usarán neuronas de la capa

oculta, y en caso afirmativo cuántas de ellas. Esto también debe estar relacionado al error que quiere alcanzarse.

Como la prueba a realizar es una mera comprobación del uso de redes neuronales en un aplicativo móvil, el grado de exactitud de la misma se nos presenta como un punto a no tener en mayor consideración en este momento: ahora el compromiso es poder volcar el entrenamiento de la red a un dispositivo móvil y validar su funcionamiento en tiempo y forma. Dicho esto, y para facilitar la primera codificación en el lenguaje Android, utilizaremos la manera más rápida de entrenamiento, haciendo uso de la misma cantidad de neuronas en capa oculta que en la capa de entrada (4 neuronas de entrada con 4 neuronas ocultas). De esta forma se ahorra tiempo de entrenamiento e igualmente se alcanza a un nivel de error suficiente para la prueba que se requiere hacer.

Para llevar a cabo este entrenamiento debemos utilizar el aplicativo, ejecutando la función trainNetLog enunciada anteriormente. Recordemos que este método necesita de un objeto “Red” que preparamos en el punto anterior a través de las entradas ingresadas, y luego configuraciones respecto a valor de error buscado, cantidad máxima de iteraciones y factor de aprendizaje. Configuraremos estos parámetros con los siguientes valores:

- Error mínimo: 0,001

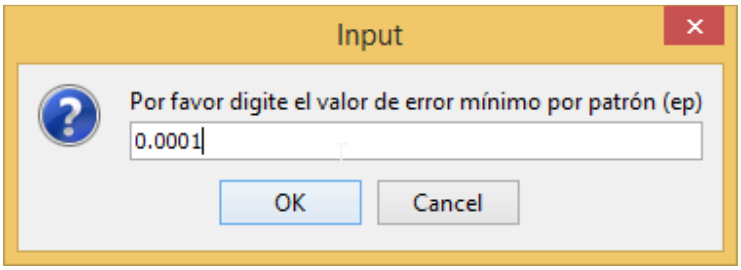


Figura 50 –Setteo de configuración en Aplicación Redes Neuronales Java para resolver operadores lógicos.

- Cantidad máxima de iteraciones: 100000

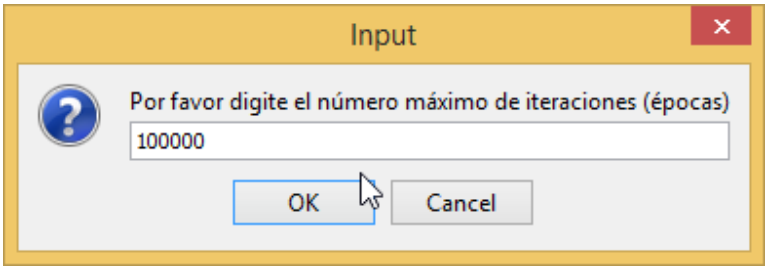


Figura 51 –Setteo de configuración en Aplicación Redes Neuronales Java para resolver operadores lógicos.

- Factor de aprendizaje: 1.0

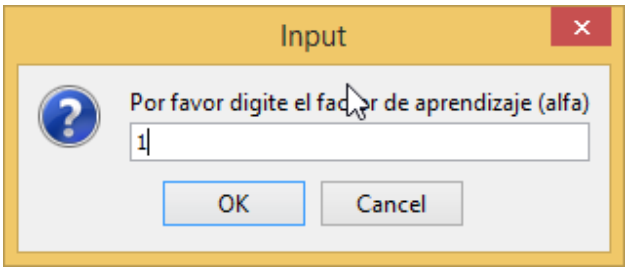


Figura 52 –Setteo de configuración en Aplicación Redes Neuronales Java para resolver operadores lógicos.

Una vez configurados estos valores, el entrenamiento se produjo correctamente para los tres grupos de redes neuronales. A continuación del detalle de cuántas iteraciones realizaron hasta alcanzar el error indicado.

- Entrenamiento OR

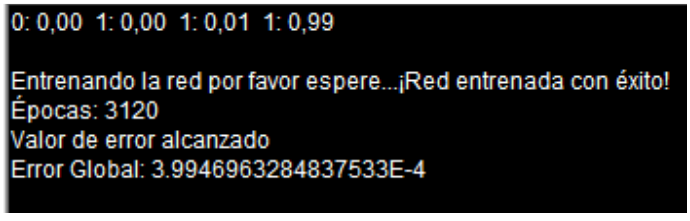


Figura 53 –Resultado entrenamiento OR en Aplicación Redes Neuronales Java para resolver operadores lógicos.

- Entrenamiento AND

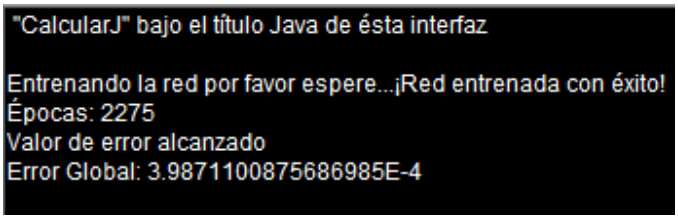


Figura 54 –Resultado entrenamiento AND en Aplicación Redes Neuronales Java para resolver operadores lógicos.

- Entrenamiento XOR

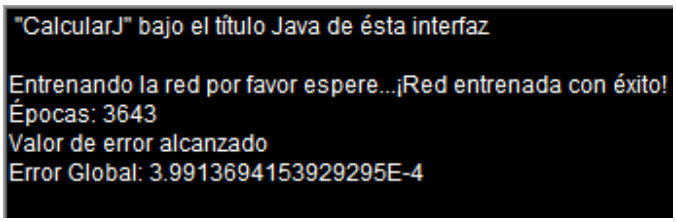


Figura 55 –Resultado entrenamiento XOR en Aplicación Redes Neuronales Java para resolver operadores lógicos.

Como resumen de estos entrenamientos, podemos ver en las figuras 53, 54 y 55 que no existen mayores diferencias en cuanto a la cantidad de iteraciones que realizaron para alcanzar ese nivel de error establecido. Ahora bien, a partir de este entrenamiento, lo interesante es obtener los pesos de cada nodo de la red neuronal, que es lo necesario a llevar al aplicativo móvil que no realizará ningún tipo de entrenamiento, sino que tomará esta red entrenada para solamente su utilización.

Se enuncian a continuación, los valores de los pesos mencionados:

- Pesos red neuronal OR
  - Entradas

```
templ[0][0]=6.946;
templ[0][1]=-2.689;
templ[1][0]=-4.719;
templ[1][1]=-4.628;
templ[2][0]=-4.544;
```

```
templ[2][1]=4.231;
templ[3][0]=3.511;
templ[3][1]=-7.765;
```

- Salidas

```
tempO[0][0]=-5.572;
tempO[0][1]=17.435;
tempO[0][2]=-4.303;
tempO[0][3]=1.033;
tempO[1][0]=-0.196;
tempO[1][1]=-4.096;
tempO[1][2]=-12.489;
tempO[1][3]=5.489;
tempO[2][0]=-9.042;
tempO[2][1]=-3.049;
tempO[2][2]=5.896;
tempO[2][3]=-5.750;
tempO[3][0]=6.970;
tempO[3][1]=-6.564;
tempO[3][2]=-5.647;
tempO[3][3]=-12.632;
```

- Pesos red neuronal AND
  - Entradas

```
templ[0][0]=5.990;
templ[0][1]=-3.100;
templ[1][0]=3.768;
templ[1][1]=5.023;
templ[2][0]=3.165;
templ[2][1]=-7.542;
templ[3][0]=-6.067;
templ[3][1]=0.193;
```

- Salidas

```
tempO[0][0]=-1.319;
tempO[0][1]=-12.283;
tempO[0][2]=8.930;
tempO[0][3]=13.489;
tempO[1][0]=1.096;
tempO[1][1]=-6.263;
tempO[1][2]=9.974;
tempO[1][3]=-14.757;
tempO[2][0]=-9.775;
tempO[2][1]=2.204;
tempO[2][2]=-9.214;
tempO[2][3]=5.069;
tempO[3][0]=4.618;
tempO[3][1]=0.137;
tempO[3][2]=-11.508;
tempO[3][3]=-10.109;
```

- Pesos red neuronal XOR
  - Entradas

```
tempI[0][0]=-4.082;
tempI[0][1]=4.337;
tempI[1][0]=3.413;
tempI[1][1]=-7.604;
tempI[2][0]=7.480;
tempI[2][1]=-3.334;
tempI[3][0]=-4.854;
tempI[3][1]=-4.575;
```

  - Salidas

```
tempO[0][0]=-4.538;
tempO[0][1]=0.354;
tempO[0][2]=-4.836;
tempO[0][3]=17.598;
```

```
tempO[1][0]=-12.748;
tempO[1][1]=3.751;
tempO[1][2]=1.666;
tempO[1][3]=-4.191;
tempO[2][0]=5.676;
tempO[2][1]=-4.541;
tempO[2][2]=-9.855;
tempO[2][3]=-3.323;
tempO[3][0]=-5.290;
tempO[3][1]=-13.266;
tempO[3][2]=7.598;
tempO[3][3]=-5.806;
```

Pruebas de las redes neuronales para operadores lógicos

Ahora bien, con el aplicativo modificado para trabajar con nuestros patrones tipo tabla de verdad, con las redes entrenadas con un error correctamente alcanzado, es indispensable probar la validez de estas redes antes de implementarlas en los dispositivos móviles. Es por eso que aquí se indican algunas pruebas realizadas para verificar el funcionamento exitoso o no de estas redes neuronales. Las pruebas a realizar son las esperables: se dibujará un vector de entrada de la forma [X , Y] y se esperará el resultado acorde al operador lógico trabajado. Los resultados se muestran en la siguiente Tabla 17:

Tabla 17 – Resultados de la red neuronal con pruebas de operadores lógicos en Java versión escritorio

Prueba	Dibujo	Entrada	Salida Obtenida	Salida Esperada	Resultado
OPERADOR OR		[0 , 0]	0	0	OK
OPERADOR OR		[0 , 1]	1	1	OK
OPERADOR OR		[1 , 0]	1	1	OK
OPERADOR		[1 , 1]	1	1	OK

OR					
OPERADOR AND		[0 , 0]	0	0	OK
OPERADOR AND		[0 , 1]	0	0	OK
OPERADOR AND		[1 , 0]	0	1	OK
OPERADOR AND		[1 , 1]	1	1	OK
OPERADOR XOR		[0 , 0]	0	0	OK
OPERADOR XOR		[0 , 1]	1	1	OK
OPERADOR XOR		[1 , 0]	1	1	OK
OPERADOR XOR		[1 , 1]	0	0	OK

Luego de realizadas las pruebas, se puede inferir que el entrenamiento de las redes neuronales ha sido suficiente para alcanzar un resultado y comportamiento acorde al de los operadores lógicos por el cual fueron entrenados. De esta forma, y teniendo los pesos que resultaron de este proceso, es tiempo de avanzar y volcar estos valores a un aplicativo móvil sencillo, que permita seleccionar a través de una interfaz los valores X e Y y realizar un reconocimiento y así poder determinar la factibilidad de uso en primer instancia en dichos dispositivos.

4.8 - DESARROLLO DE APLICACIÓN MÓVIL

Implementación de operadores lógicos en dispositivos móviles

Con los pesos de cada neurona de entrada y salida, cada uno de ellos volcados en matrices, se tienen las condiciones necesarias para comenzar a desarrollar el primer aplicativo para dispositivos móviles. El primer paso para realizar esta tarea consiste en

traspasar aquellas clases y métodos que se utilizaban en el programa original, hacia el código nativo de Android.

Por lo tanto valiéndonos de que una de las ventajas de la utilización de este algoritmo era su viabilidad y versatilidad técnica que nos proponía al estar desarrollado en Java, es que el pasaje se ve enormemente favorecido. Esto se traduce en facilidad en la creación de las aplicaciones móviles ya que las clases que realizan las operaciones terminan siendo casi idénticas entre el aplicativo Java de escritorio contra su implementación en Android.

Si bien en este trabajo no se indican detalles técnicos de la implementación en cuanto a la programación, debemos mencionar aquellas cuestiones que son utilizadas para volcar este algoritmo de back propagation en un celular. En primer lugar, cabe destacar las clases que fueron traídas del aplicativo Java de escritorio resumidamente: la clase “Matriz”, para el manejo operacional de los pesos que allí se almacenan (principalmente operaciones matemáticas de sumas, restas y productos), y la clase “Red” que almacena estas matrices, pero que esta vez no utilizaremos para entrenamiento sino solamente para reconocimiento, por lo tanto todos esos métodos pueden obviarse y no codificarse en Android.

Dada esta introducción hacia la implementación en Android del reconocimiento de una red neuronal, establezcamos nuestro punto de inicio: en al aplicativo Android lo primero que se debe hacer es generar las matrices que fueron enunciadas anteriormente conteniendo los pesos de entrada y salida. Esto bien puede hacerse por un archivo externo (manera por la cual se extrajieron los datos del aplicativo Java de escritorio), el cual permite utilizar distintos tipos de entrenamiento a futuro si se desean modificar, o bien definiendo en el código los pesos e inicializando cada matriz con estos valores. Para el fin demostrativo de nuestro aplicativo, con ingresar y establecer los valores como fueron debidamente entrenados es suficiente y no utilizaremos la obtención de un archivo físico ya que no es nuestra meta volver a entrenar estas redes neuronales, solo las estamos utilizando como prueba de factibilidad.

Con esta definición, y ya con las matrices en Android con sus pesos establecidos viene bien idear cómo será la interfaz para probar estas redes neuronales y su operatoria, recordando que se debe tener capacidad para poder probar los tres operadores lógicos.

Es por eso que ideamos una pantalla en donde se tengan por un lado los tres botones que representan cada uno su operación (OR, AND, XOR), y luego dos botones tipo "Checkbox" (botones seleccionables) para elegir los valores de X e Y, representando 0 y 1, encendido o apagado. Este diseño más un botón para calcular y realizar el reconocimiento, son suficientes para la operatoria del aplicativo. Ahora bien restan algunos mensajes informativos en el aplicativo informando resultados, errores u operaciones que el usuario debe realizar. Con el apartado funcional e informativo tenidos en cuenta, se establece la pantalla del aplicativo que se puede visualizar en la figura 57:

Figura 57 –Pantalla de aplicativo en Android para resolver operadores lógicos utilizando redes neuronales

En la figura 57 se puede ver lo anteriormente mencionado:

- Un título identificatorio del aplicativo
- Los tres botones para seleccionar la operación a utilizar con su leyenda respectiva. Aquí es necesario e importante indicar qué acción realiza cada botón. Al seleccionar cada uno de estos botones, lo que realmente se está haciendo es cambiar de red neuronal de trabajo. Recordemos que cada uno de estos

operadores tiene su propia red neuronal, por lo tanto al seleccionar cada uno de ellos, se deben llenar los pesos que les corresponden. De esta forma, en nuestro código no se tienen todas las redes neuronales preparadas al mismo momento, sino que al seleccionar el operador lógico se realiza la activación de cada una de ellas y el llenado correspondiente de las matrices de entrada y salida. Dicho de otra manera, solo se tienen en las neuronas de entrada y salida, los pesos de la red seleccionada.

- Un apartado con los botones seleccionables para los valores X e Y, que son la matriz objetivo a utilizar en el reconocimiento. Son botones tipo checkbox, indicando con una tilde cuando están seleccionados (valor 1), y sin la tilde cuando están apagados (valor 0).
- El modo actual de procesamiento. En primer instancia indica que “ningún operador está seleccionado”
- Finalmente debajo de todo, el botón de calcular para proceder al reconocimiento. Al apretar este botón cuando esté activo (es necesario tener un modo de operación seleccionado), comienza el proceso de simular la red neuronal, por el cual se multiplican las matrices, se evalúan las funciones de propagación y en general se propaga la red hacia adelante.

Concebida la pantalla única y principal de este aplicativo, resta ahora comprobar su operatoria y así cumplir con su propósito. Por lo tanto y teniendo en cuenta las consideraciones anteriormente descriptas, a continuación se indicará cómo se debe usar la aplicación para comprobar su funcionamiento.

Funcionamiento del aplicativo móvil: Operadores Lógicos

Primeramente se debe seleccionar uno de los tres modos de operación: AND, OR, XOR. Para el caso de esta demostración, se selecciona el operador “AND”, tal cual se muestra en la figura 58.

Figura 58 –Pantalla de aplicativo en Android para resolver operadores lógicos utilizando redes neuronales

El aplicativo indica que está funcionando en modo “AND” y ya habilitó el botón calcular. Este botón ya se encuentra disponible, porque los checkbox X e Y ya tiene su valor por defecto que es 0, por lo tanto ya puede procederse a calcular el procesamiento de la red para esas entradas. Sin embargo, para ejemplificar todas las funciones del aplicativo, seleccionaremos y activaremos el valor Y, quedando de la siguiente manera como se muestra en la figura 59:

Figura 59 –Pantalla de aplicativo en Android para resolver operadores lógicos utilizando redes neuronales

Con la operación elegida en “AND” y los valores seleccionados en [0 , 1], el botón de calcular sigue activo y no queda más que utilizarlo si son esas las condiciones que desean probarse. Seleccionando este botón, se procederá al reconocimiento en base a la red entrenada y los valores proporcionados, devolviendo el resultado más aproximado junto con su porcentaje de acierto.

Figura 60 –Pantalla de aplicativo en Android para resolver operadores lógicos utilizando redes neuronales

Realizado el cálculo, vemos en la figura 60 que el resultado es el esperado: 0 para una compuerta “AND” cuando alguno de sus elementos no está activo. El porcentaje es de casi 99%, correlato del ejemplificado en el programa de escritorio Java. Cabe destacar que los tiempos de procesamiento del aplicativo son instántaneos: no se presentan demoras ni dificultades en la obtención de los resultados.

Definida la operatoria, resta aplicar la comprobación masiva y minuciosa del aplicativo en términos de sus bondades en cuanto a los resultados que provee con respecto a su par versión escritorio. Los resultados que se esperan deberían ser idénticos a lo obtenidos en pruebas anteriores. Algún error o diferencia en estos puntos peligraría el uso de este algoritmo en los aplicativos móviles requiriendo de una revisión exhaustiva de sus motivos y razones.

Pruebas con Operadores Lógicos en Android que se muestran en la Tabla 18.

Tabla 18 – Resultados de la red neuronal con pruebas de operadores lógicos en Java versión Android.

Prueba	Entrada	Salida Obtenida	Salida Esperada	Resultado	Comentarios
OPERADOR AND	[0 , 0]	0	0	OK	Comparativo con aplicación de escritorio: 99% (Escritorio) - 98.80% (Android) – Diferencias por redondeo
OPERADOR AND	[0 , 1]	0	0	OK	Comparativo con aplicación de escritorio: 99% (Escritorio) - 98.95% (Android) – Diferencias por redondeo
OPERADOR AND	[1 , 0]	0	1	OK	Comparativo con aplicación de escritorio: 99% (Escritorio) - 98.91% (Android) – Diferencias por redondeo
OPERADOR AND	[1 , 1]	1	1	OK	Comparativo con aplicación de escritorio: 99% (Escritorio) - 98.71% (Android) – Diferencias por redondeo

Conclusiones con Operadores Lógicos en Android

Realizada esta primera intromisión en la aplicación de redes neuronales en dispositivos móviles, y en luz de los resultados obtenidos, podemos concluir que los mismos fueron altamente auspiciosos.

Nos gustaría destacar algunas cuestiones en cuanto a las prestaciones vistas, en términos de efectividad, sencillez y velocidad. Decimos efectividad ya que el reconocimiento realizado concordó exactamente con el utilizado en el aplicativo de escritorio como era de esperarse: no se tuvieron cambios ni alteraciones al respecto, los métodos de simulación funcionaron perfectamente, por lo tanto el algoritmo resulta fiable. Al hablar de sencillez, lo ya también mencionado de poder utilizar el código Java y adaptarlo rápidamente a un aplicativo sencillo pero suficiente para realizar un primer acercamiento al desarrollo final, nos proveen de gran versatibildad para implementar y realizar ajustes entre ambos sistemas de entrenamiento y simulación. Y en cuanto a la velocidad de procesamiento, los resultados también son exitosos ya que no se presenta ningún tiempo de espera ni de interface ni de tiempo de cómputo del dispositivo, sino por el contrario la experiencia de uso es completamente fluída, permitiendo probar todas las operaciones de manera muy veloz y eficiente, comprobando cada uno de los resultados posibles en pocos segundos.

Dichas estas consideraciones y con el optimismo presente y fundamentado, es que podemos continuar en el desarrollo de nuestro proyecto de redes neuronales para detectar billetes. Si bien el caso a tratar difiere mucho en cuanto al objeto a descifrar, el algoritmo de back propagation y sus componentes se presentaron acorde a lo esperado y perfectamente empleables para continuar.

De esta forma, prosigue entonces la adaptación y desarrollo del aplicativo en pos de nuestor prototipo final para reconocer billetes.

Aplicación de reconocimiento de billetes: Primeras consideraciones

Para ir adentrándonos en lo que debemos realizar, creemos necesario realizar un primer análisis de los puntos que se creen van a ser de conflicto o al menos generarán dudas sobre cómo resolver. Estos temas no son más que incógnitas que se presentan a lo largo del trabajo, ya sea por la naturaleza del problema en cuestón o por la técnica utilizada. Por lo tanto en este momento no debe escaparse ninguna cuestión que amenace ni conspire contra el éxito del proyecto.

El análisis que debe hacerse tiene que atravesar desde los algortimos utilizados, sus tamaños, métodos y composiciones, hasta la composición física del billete, sus formas, colores, tamaños, simbología y demás, pensando siempre en la implementación en un dispositivo móvil y teniendo presente la posibilidad del uso de otros algoritmos complementarios.

Para esto entonces, debajo enunciamos nuestros temas de principal interés para avanzar hacia nuestro objetivo:

- Tamaño de la matriz: el aplicativo Java originalmente funcionaba con una matriz de un tamaño de 5x7, e inclusive en nuestra prueba de factibildiad técnica y funcional, la adaptamos a una matriz de entrada de 1x2. Ahora bien es una incógnita cuál será el tamaño de nuestras matrices para reconocer billetes, si alcanzará el modelo de matriz bidimensional y demás.
- Composición de la matriz: el entrenamiento Java, se realiza basado en valores lógicos 0 y 1. Los billetes, en principio no se presentan de esta forma ya que no se los puede representar así. Se deberá evaluar el impacto de este punto y sus posibles soluciones en caso de que exista error.
- Entrenamiento de la red: determinar cómo se producirá el enternamiento de la red, si es necesario utilizar otras redes complementarias, con qué elementos se entrenarán, con qué nivel de error y demás asuntos que nos hemos ido topando en otras ocasiones a lo largo del trabajo y que nos han servido de experiencia para aplicarlos ahora en el desarrollo final.
- Composición del billete: habrá que determinar el tamaño de cada billete a utilizar o si este podrá ser o no dinámico, ya sea para su entrenamiento o su reconocimiento, teniendo en cuenta que para el reconocimiento se utilizará una cámara de un celular

también existirán consultas acerca de los colores obtenidos y luz de las imágenes. Este punto se relaciona intrínsecamente con el tamaño de la matriz a utilizar.

- Identificación del billete: junto con la forma de cada billete, se deberá estudiar y analizar los elementos para detectar un billete sobre otro, es decir las partes que lo identifican, sus numeros, dibujos, próceres y demás identificaciones que los distinguen de entre otros billetes.

Es apropiado decir que a lo largo del desarrollo de estos puntos, los mismos pueden y de hecho no van a seguir esta secuencia sitemática y aislada, sino que todo el proyecto los atravaserá a cada momento, ya sea a través de toma de decisiones por limitancias de un aspecto o el mismo aprovechamiento de ventajas de utilizar tal u otra combinación. En otras palabras, la elección de cada tema puede no ser la mejor en términos individuales, pero en términos del proyecto será la que se crea más conveniente pensando en el todo como un conjunto inseparable.

Aplicación de reconocimiento de billetes: Tamaño de la matriz

El primero de los puntos a analizar es abarcado por el tamaño de la matriz de entrada, representando una parte de la capa inicial de una red neuronal. La otra parte la darán sus valores posibles y pesos de entrenamiento. Este punto es de vital importancia porque a partir de aquí se configurará o empezará a configurar el funcionamiento de la red neuronal y sus características particulares para su correcto entrenamiento y posterior reconocimiento.

Por lo tanto en este tópico, lo más importantes a abordar corresponde primeramente a la cantidad de neuronas que tendrá la red como entrada, es decir traducido al algoritmo utilizado, el tamaño de matriz de entrada que recbirá sus pesos. Aquí se deberá definir o sentar base para posteriores definiciones, acerca de cuántas dimensiones son requeridas y el tamaño de cada arreglo interviniente, partiendo de que cada uno debe contener la información que representa a un billete.

De esta forma, para poder analizar y conferir correctamente, realizaremos diversos ensayos como a lo largo del trabajo para encaminar hacia una posible solución adaptable a nuestras necesidades.

Ahora bien, en este punto debemos como primer acercamiento, presentar los billetes a tratar e ir desarrollando pruebas sobre ellos. Por lo tanto, a continuación se visualizarán en las figuras 61, 62, 63, 64, 65 y 66 el primer conjunto de imágenes con billetes representados. Las mismas tienen un tamaño de 538 x 222 píxeles para proveer de una calidad bastante buena, en efecto son reducciones de imágenes de mayor definición llevadas a un tamaño manejable.

Figura 61 - Imagen de billete de 100 pesos (538 x 222) - BCRA

Figura 62 - Imagen de billete de 50 pesos (538 x 222) - BCRA

Figura 63 - Imagen de billete de 20 pesos (538 x 222) - BCRA

Figura 64 - Imagen de billete de 10 pesos (538 x 222) - BCRA

Figura 65 - Imagen de billete de 5 pesos (538 x 222) - BCRA

Figura 66 - Imagen de billete de 2 pesos (538 x 222) - BCRA

Con esta simple presentación hemos descubierto y abierto algunas situaciones: primeramente, solo hemos mostrado el frente de los billetes que a los fines prácticos del trabajo, solo serán utilizados de esta forma, por su frente y no realizaremos reconocimientos de su dorso, tareas que complicarían la aplicación del trabajo que se busca demostrar. Además a través del tamaño definido en estas imágenes y su composición, podemos rápidamente inferir que las matrices que contengan la información de estas imágenes tendrán que tener un tamaño mucho más grande al que venimos trabajando: recordemos, la red neuronal inicialmente era de 5x7 y nuestro aplicativo de demostración era de 1x2. Acá, cada imagen está representada en 538x222 = 119436 píxeles o valores para representar esta imagen. Y por si fuera poco, aquellas matrices que utilizábamos contenían valores 1 y 0: ahora los píxeles de estas imágenes no contienen valores 1 y 0, solo podemos descomponerlas en sus reprentaciones en la nomenclatura RGB (valores de rojo, verde y azul que componen el color). Y otro punto que debe tenerse en cuenta, es que en esta primera etapa estaremos analizando imágenes digitales de billetes que no fueron sacadas por cámaras de fotos, por lo tanto no tienen problema de luz ni demás cuestiones intrínsecas a una foto tomada por una cámara.

Aplicación de reconocimiento de billetes: Composición de la matriz

Realizada la introducción al apartado físico de los billetes y su tamaño inicial en píxeles, debemos evaluar cómo convertilos en información de carga para nuestra red neuronal a entrenar. Como hemos venido trajando, la red que utilizaremos trabaja con

valores lógicos 0 y 1 por lo cual es necesario llevar nuestras imágenes de billetes a este formato. Para tal fin, comienzan a barajarse, estudiarse y analizarse diversos enfoques.

Una primer herramienta podría ser generar la imagen de los billetes en blanco y negro puro, donde cada pixel que termine siendo negro represente nuestro valor 0 y donde cada pixel que se determine como color blanco valga 1. Para esto es necesario definir con qué criterios se convertirá cada color a blanco o negro. Un algoritmo sencillo que utilizan los programas de edición de imágenes es el siguiente:

Se sumariza el valor máximo RGB alcanzable para definir un color que no es más que 255 (R) + 255 (G) + 255 (B) = 765. Como se quiere llevar a un valor bicotómico, se divide por 2, resultando en 382,5. Con este valor, se define el umbral de conversión: aquellos colores que su suma RGB sea menor o igual a 382, se convertirán en negro y aquellos que la superen, se convertirán en blanco. Realicemos algunas imágenes de ejemplo:

Escala de grises: A partir del quinto gris que sumariza más de 382, al convertirlo a una imagen monocromática.

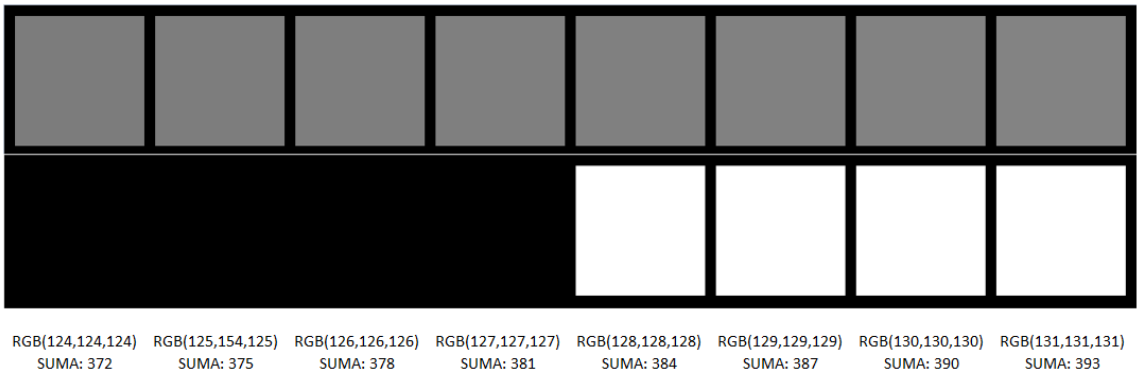


Figura 67 – Escala de Grises - Wikipedia

Escala de rojos: mismo comportamiento ocurre para colores intensos y puros como ser un rojo pleno (255,0,0). Solamente cuando la sumatoria de sus colores sobrepasan el valor 382, se convierten en blanco monocromático (naranja original)

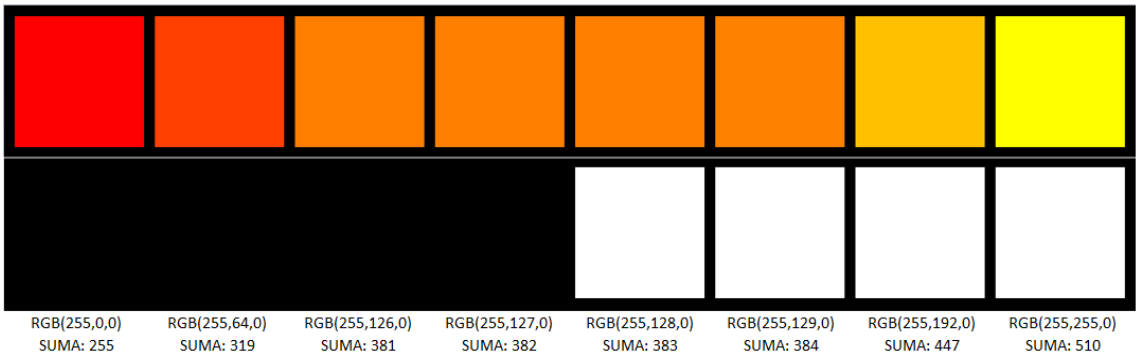


Figura 68 – Escala de rojos - Wikipedia

Multicolor: esta imagen sirve para ver gráficamente que ante colores extremadamente similares, debido al límite establecido de 382,5, se tienen resultados totalmente diferentes al convertirlos monocromáticamente.

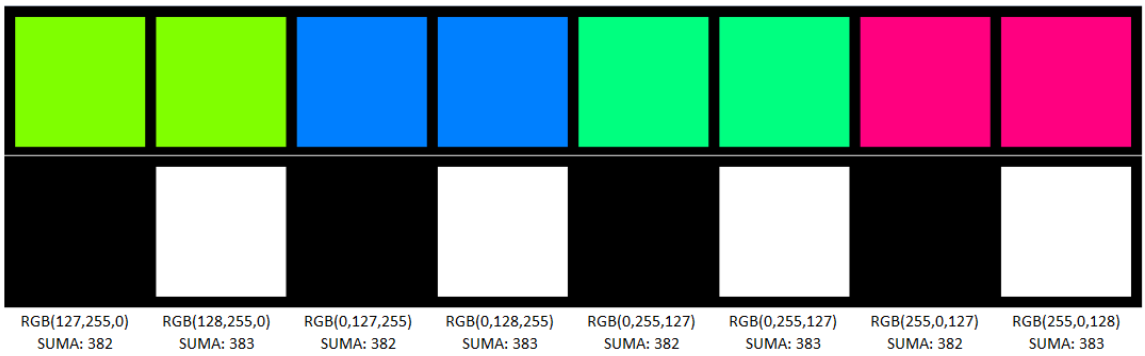


Figura 69 – Escala Multicolor - Wikipedia

Esta última imagen nos sirve de gran ayuda evidenciar un gran problema al trabajar con imágenes que son tomadas por cámaras de celular: los colores y las luces de las imágenes capturadas varían por un sinfín de motivos, ya sea si se tomó al aire libre o en un lugar cerrado, con luz natural, artificial o la ausencia de ella, y hasta inclusive de modelos de equipos que tiene nsus propias cámaras con su propia configuración del color. Adaptar esos colores a que trabajen y terminen realizando la misma derivación del color monocromático de las imágenes a entrenar, supone un problema mayúsculo.

Finalmente aplicaremos esta binarización de colores a las imágenes anteriormente presentadas para intentar producir el entrenamiento de nuestra red neuronal. Por

consecuente, las imágenes llevadas a colores monocromáticos resultan de esta forma (ver Figuras 70, 71, 72, 73 y 74) :

Figura 70 - Imagen de billete de 50 pesos B&N (538 x 222)

Figura 71 - Imagen de billete de 20 pesos B&N (538 x 222)

Figura 72 - Imagen de billete de 10 pesos B&N (538 x 222)

Figura 73 - Imagen de billete de 5 pesos B&N (538 x 222)

Figura 74 - Imagen de billete de 2 pesos B&N (538 x 222)

Con este conjunto se analiza y observa lo siguiente:

- Lo más nítido y visible del billete parecen ser los rombos que indican el valor de cada uno de ellos.
- Los números superiores izquierdos tienen problemas con el difuminado/brillo que tienen. Algunos se distinguen pero no presentan una claridad absoluta.
- Las imágenes de los próceres puede perderse entre las imágenes del fondo, e inclusive algunos no tienen todos los rasgos de sus caras visibles
- Las inscripciones verticales de sus valores se pierden con los paisajes de fondo que lleva el billete

Con este conjunto de imágenes, estamos en condiciones de realizar un primer entrenamiento de nuestra red neuronal. Intentaremos aplicar esta matriz de 538 x 222 binarizada con 1 y 0, determinando si nuestro algoritmo es capaz de trabajarla.

Aplicación de reconocimiento de billetes: Entrenamiento de la primer matriz

Con las imágenes anteriormente presentadas, realizaremos un primer intento de entrenamiento y posterior reconocimiento para conocer si es posible continuar operando con una imagen tan grande.Para esto es necesario que apliquemos las modificaciones necesarias en el código Java que poseemos.

Primeramente se tiene que modificar el tamaño de las matrices de trabajo (cuestión que ya hemos hecho para la calculadora lógica) que en vez de 5x7 sea de 538x222.

Luego se debe proceder a modificar el llenado de las matrices de entrada. Actualmente las mismas se llenan a través de un lienzo de dibujo, y en este caso precisamos cargar un archivo físico que contiene la imagen del billete. Esta imagen debe leerse y transformarse píxel a píxel en los valores que trabaja la matriz: 0 (píxel blanco) y 1 (píxel negro). Junto con cada imagen debe indicarse el patrón (resultado) al que corresponde, como ya hicimos cuando dibujábamos los números sobre el lienzo o cuando se completaban los operadores lógicos. En esta ocasión, completaremos como nombre de patrón el valor que representa cada billete, así entonces tendremos los siguientes patrones: 2p, 5p, 10p, 20p y 50p.

Debido a que la información que contienen las matrices de entrada es muy grande, es decir unos 119436 elementos, no es posible visualizar o entender esta información de la misma manera que utilizábamos anteriormente para comprobar que las mismas estén bien cargadas. Solamente por motivos de comprobación de que la información se haya cargado correctamente, hemos guardado estas matrices en archivos de texto para luego proceder a abrirlos a un zoom alejado donde el ojo humano puede interpretar las distintas figuras que componen el billete. Por ejemplo para el billete de 2 pesos, a un zoom 100% la información que contiene la matriz se ve de la siguiente manera (ver figura 75):

Figura 75 – Matriz billete de 2 pesos.

Si bien puede detectarse en la figura 75 que donde aparecen los “1” en la imagen se corresponden a la figura del número 2 en el margen superior izquierdo del billete, es necesario hacer un esfuerzo con la vista para visualizarlo correctamente. Para facilitar esto y ayudar a la comprobación de los datos, utilizamos un zoom mucho mayor. El resultado se puede visualizar en la figura 76:

Figura 76 – Matriz billete de 2 pesos en contraste con imagen de billete de 2 pesos.

Claramente puede verse en la figura 76 que la ubicación de los valores “1” y “0” generan el correcto dibujo del billete de dos pesos fácilmente identificable. Este proceso se aplica con el resto de las imágenes y queda comprobado que la carga de las matrices ha sido correcta y que las mismas se encuentran preparadas para el siguiente paso del entrenamiento.

Generados los patrones con sus valores de entrada, el siguiente paso en nuestro software es realizar el entrenamiento de la red neuronal. Para ello, continuaremos utilizando los valores que fueron utilizados en el entrenamiento de la red neuronal para los operadores lógicos, a saber:

- Error mínimo: 0,001
- Cantidad máxima de iteraciones: 100000
- Factor de aprendizaje: 1.0

Con estos parámetos se procede a lanzar el entrenamiento y analizar su comportamiento.

Incialmente se observa que el proceso demora mucho tiempo ya que para cada iteración, el entrenamiento está demorando alrededor de 8 segundos en realizarlo, por lo tanto de no alcanzar el error mínimo en alguna de sus iteraciones, estaríamos ante un procesamiento aproximado (calculando linealmente) de: 8 segundos c/iteración X 100000 iteraciones = 800000 segundos -> 13333 minutos -> 222 horas -> 9.25 días.

Efectivamente, el error no pudo ser alcanzado y en nuestra prueba, el entrenamiento estuvo corriendo más de 9 días hasta indicar el mensaje que no pudo alcanzar el error solicitado. Si bien evidentemente el procesamiento tomó mucho tiempo, no sería una impedancia o desventaja, por el contrario, presupone una red mayormente entrenada: a mayor cantidad de iteraciones, menor tiende a ser el error cuadrático global de la red. Además, recordemos que el entrenamiento de la red es una operación única: una vez finalizado y calculados los pesos de cara neurona de la red, estos pueden ser utilizados en futuras aplicaciones de reconocimiento sin necesidad de generar nuevos entrenamientos.

Ahora bien, debemos evaluar si este entrenamiento por 9 días nos será de utilidad o no. Eso solamente lo podremos dilucidar a través del proceso de reconocimiento de la

imagen. Para esta etapa, debemos alimentar a nuestra simulación con una imagen para que nuestra red evalúe a qué patrón se corresponde, tal como hace por ejemplo la aplicación de operaciones lógicas. Estas imágenes pueden ser las mismas que utilizamos para entrenar la red u otras. Cabe destacar, que la aplicación final no busca trabajar con imágenes fijas y reales, si no con fotos que serán dinámicas y distintas. Pero para poder comprobar que la red esté trabajando correctamente a este nivel y evitar problemas más complejos, es necesario partir desde los casos más sencillos y manejables para así ir logrando avanzar a través de bases sólidas y comprobadas.

Por lo tanto para este intento de reconocimiento utilizaremos los mismos archivos de imágenes que aplicamos para el entrenamiento, para evaluar cuáles son sus resultados. Se realiza la prueba con la imagen de 2 pesos y nos arroja los resultados que se visualizan en la figura 77:

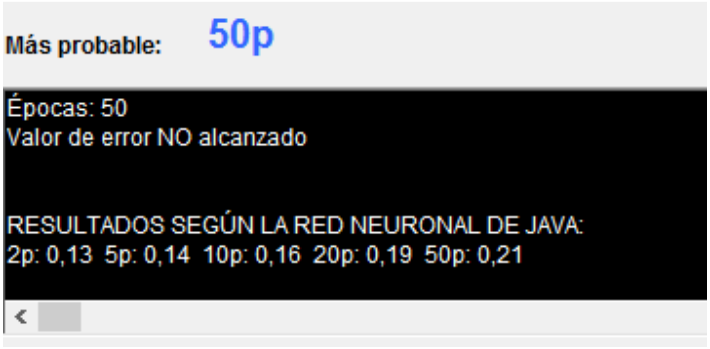


Figura 77 – Resultado de aplicación con ingresando imagen de billete de 50 pesos.

Los resultados visualizados en la figura 77 claramente no fueron los esperados y por el contrario, resultaron extremadamente malos y desalentadores. Ninguna de las imágenes pudo ser reconocida y los valores porcentuales fueron totalmente desacertados. De no haber tenido casos de éxito en las anteriores pruebas, estos algoritmos quedarían descartados.

Se plantea entonces evaluar qué pudo haber llevado a estos fallos y visiblemente la gran diferencia ha sido el tamaño de la red y elementos utilizados. Por lo tanto para dar otra oportunidad al desarrollo y continuar con este enfoque, reduciremos ampliamente el tamaño de las imágenes para evaluar si el comportamiento mejora y se obtienen los

resultados esperados. Para ello reduciremos la imagen en un 25% de su valor anterior logrando así las siguientes dimensiones: imágenes de 128x56.

Al utilizar imágenes de este tamaño, se revisa y valida que la fisonomía del billete y sus partes identificables lo continúen siendo, de esta forma podremos tener herramientas para diferenciar y generar un correcto entrenamiento. El conjunto de imágenes generadas se muestra en la figura 78:

Figura 78 – Imágenes de billetes de 2, 5, 10, 20 y 50 pesos en tamaño 128X56

Como puede apreciarse en la figura 78, la calidad de los billetes disminuyó drásticamente pero en ellos todavía pueden observarse sus partes características aunque con dificultad, por ejemplos el número sobre el margen superior izquierdo del billete de 50 no se alcanza a distinguir correctamente el 5 de la decena, pero esto puede no ser un problema debido a que las imágenes que se tomarán como foto sufrirán el mismo achicamiento y proceso de binarización del color para llevarlo a una figura simular aquí presentada.

Procedemos a entrenar esta nueva red con este nuevo conjunto de imágenes, intentando alcanzar el error habitual:

- Error mínimo: 0,001
- Cantidad máxima de iteraciones: 100000
- Factor de aprendizaje: 1.0

Esta vez el error fue alcanzado muy rápidamente, solo hicieron falta 193 repeticiones para alcanzarlo lo que nos alienta a un posible entrenamiento más profundo en caso de tener buenos resultados durante el reconocimiento.

A continuación se realizan las pruebas de rigor en cuanto al reconomiento de las imágenes con las mismas imágenes que la red fue entrenada. Los valores se muestran en la Tabla 19:

Tabla 19 – Resultados de reconocimiento de imágenes en red entrenada

Billete	Resultado obtenido	Conclusión
	Más probable: 2p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,96 5p: 0,02 10p: 0,01 20p: 0,00 50p: 0,02	OK
	Más probable: 5p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,00 5p: 0,96 10p: 0,01 20p: 0,01 50p: 0,00	OK
	Más probable: 10p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,01 5p: 0,02 10p: 0,96 20p: 0,00 50p: 0,01	OK
	Más probable: 20p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,00 5p: 0,03 10p: 0,00 20p: 0,97 50p: 0,00	OK
	Más probable: 50p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,02 5p: 0,01 10p: 0,00 20p: 0,00 50p: 0,96	OK

En resumen, todos los resultados fueron correctos y con un alto porcentaje de acierto. A continuación realizaremos una batería de pruebas con imágenes modificadas. Este conjunto de imágenes alteradas, constará de las mismas imágenes de los billetes con las que fueron entrenadas pero con más o menos píxeles de color negro, para lograr evaluar el comportamiento de la red ante pequeñas variaciones de las imágenes (ver tabla 20).

Tabla 20 – Resultados de reconocimiento con imágenes con distorsiones en red entrenada.

Billete	Resultado obtenido	Conclusión
	Más probable: 2p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,96 5p: 0,02 10p: 0,01 20p: 0,00 50p: 0,02	OK
	Más probable: 2p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,96 5p: 0,02 10p: 0,01 20p: 0,00 50p: 0,02	OK
	Más probable: 2p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,86 5p: 0,01 10p: 0,39 20p: 0,00 50p: 0,02	OK
	Más probable: 10p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,04 5p: 0,01 10p: 0,89 20p: 0,01 50p: 0,14	ERROR

La red neuronal tuvo un comportamiento más que aceptable ya que solamente recién en el cuarto intento después de haber borrado varias partes del billete y escrito encima del mismo, se produjo un error en el reconocimiento.

Ahora realizaremos un estudio de cuál es el comportamiento en cuanto a la localización de los píxeles de manera global y en conjunto de la imagen, es decir, partiendo de la imagen original que se utilizó en el entrenamiento, se realizarán pruebas con imágenes modificadas tomando esas iniciales y corriendo todos sus píxeles en dirección X (derecha o izquierda) e Y (arriba o abajo), Los resultados se muestran en la siguiente Tabla 21.

Tabla 21 – Resultados de reconocimiento con imágenes modificadas en red entrenada

Billete	Resultado obtenido	Conclusión
 X: +0; Y: -1	Más probable: 50p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,00 5p: 0,17 10p: 0,09 20p: 0,00 50p: 0,37	ERROR
 X: +1; Y: 0	Más probable: 5p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,62 5p: 0,95 10p: 0,00 20p: 0,01 50p: 0,02	ERROR
 X: +1; Y: +1	Más probable: 5p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,03 5p: 0,68 10p: 0,00 20p: 0,05 50p: 0,06	ERROR

Rápidamente concluímos que con tan solo mover las imágenes un píxel de donde fueron entrenadas, se producen errores gravísimos aún manteniendo toda la estructura de la imagen sin modificar. Al ojo humano no parece haber diferencia entre las imágenes pero a nivel de la red neuronal, todo su conjunto de valores ha sido modificado y reemplazado por otro valor no esperado. Esto nos plantea un problema muy importante y nos realza la importancia de tener una correcta ubicación o encontrar mecanismos para resolverla ya que ante imágenes fotográficas, las posiciones de las figuras de los billetes jamás serán exactas.

Cabe realizar una última prueba para determinar conclusiones en cuanto a la performance de la red contra imágenes fotográficas y así determinar si existe algún otro punto de interés además del anteriormente mencionado de la posición de cada píxel. Procedemos a tomar fotogafías de billetes, convertirlos a imágenes blanco y negro, reducirlos y analizar (ver Tabla 22):

Tabla 22 – Resultados de reconocimiento con imágenes modificadas en red entrenada

Billete	Resultado obtenido	Conclusión
	Más probable: 20p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,00 5p: 0,01 10p: 0,11 20p: 0,88 50p: 0,17	ERROR
	Más probable: 2p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,99 5p: 0,01 10p: 0,01 20p: 0,00 50p: 0,08	ERROR

No hace falta continuar realizando pruebas para determinar que las fotos procesadas tampoco están siendo correctamente reconocidas lo que acrescienta el problema del reconocimiento que en gran parte está relacionado al posicionamiento antes enunciado. Una forma de combatir o minimizar el error por posicionamiento sería entrenar la red con varias imágenes iguales y con distinto posicionamiento, con el fin de así generar mayores patrones de entrada y por lo tanto mayores posiblidades de acierto al tener un universo más grande de elecciones. Pero el abuso de este enfoque se interpreta más como una resolución por fuerza bruta, donde todas las posiblidades se conocen y prueban de antemano, que a través de una red neuronal entrenada.

Por lo tanto se analiza, piensa y reflexiona en buscar qué partes del billete pueden estar más o menos afectadas en cuanto a la posición que se tome la fotografía y que a su vez identifiquen al billete de entre otros. Por consiguiente, las zonas que se interpretan como de mayor identificación del billete son: el número en el margen superior izquierdo y los rombos ubicados hacia su derecha como se visualiza en la figura 79.

Figura 79 – Billete de 5 pesos resaltando número y rombos

En cuanto al número podemos visualizar con respecto a los demás billetes, analizando cada especie, que se presentan en diversos colores, tamaños y claramente en cantidad de cifras: un billete de 2 pesos contiene una sola contra uno de 10 que contiene 2 cifras por ejemplo. Idealmente en este caso, debería tomarse un rectángulo de dimensiones reducidas que contenga solamente el número del billete e insertarlo en la matriz de entrada de la red neuronal previas transformaciones y conversiones que fueran necesarias. Pero esto implicaría gestionar correctamente un tamaño de imagen que abarque para todos los tipos de billetes y que tenga en cuenta las anteriores variaciones antes mencionadas.

Figura 80 – Números en billetes

Los espacios en blanco juegan un papel preponderante en la identificación del billete a través de los números y por lo tanto, aumenta el riesgo de un incorrecto posicionamiento.

Por otro lado, los rombos parecen tener una fisonomía un tanto más homogénea en donde su caracterización hacia el billete dependerá de la cantidad que existan. Por lo tanto, un billete de 2 pesos contiene 6 imágenes de rombos contra uno de 100 que tan solo termina conteniendo 1. Sí cabe mencionar que al no estar sobre un margen del

billete o cerca de tal, es necesario encontrarle una posición relativa dentro del mismo. Realizando un cálculo rápido, los rombos comienzan aproximadamente después de la primer cuarta parte del billete, lo cual es un dato importante para encontrarlo dentro del billete.

Figura 81 – Rombos en billetes

Tanto el uso de los números como de los rombos, establece una primer limitante o condición necesaria para este proyecto: los billetes solamente podrán ser reconocidos desde su cara frontal y en su sentido izquierda-derecha, es decir sin estar dados vueltas o de "cabeza". A los fines demostrativos de este trabajo, se actuará teniendo en cuenta esta condición, dejando abierto a futuros trabajos la mejora en este apartado con otros tratamientos de la imagen.

Comentados estos dos puntos, se elige proseguir y continuar el desarrollo y el estudio a través de los rombos que componen la imagen.

Aplicación de reconocimiento de billetes: Entrenamiento de matriz con rombos

A lo largo del anterior punto del trabajo se realizaron diversas tareas que fueron encaminando el proyecto a este punto. Comenzando con las primeras pruebas de la red neuronal se determinó que el tamaño de la matriz a trabajar debería reducirse para mejorar su efectividad. A su vez se encontraron problemas en cuanto al posterior reconocimiento de las imágenes, que determinaron que la red no sea utilizable bajo las condiciones con las cuales fue entrenada. Principalmente se encontraron problemas a la hora de reconocer imágenes con sus píxeles en posición distinta a la que fueron entrenados, resultando en valores dados como válidos erróneamente, y los mismos inconvenientes para las pruebas en base a imágenes tomadas de fotografías: un cambio en las matrices de entrenamiento es necesario.

Todo esto se suscitó en realizar un análisis en cómo convertir estos billetes en una imagen reconocible bajo esta problemática, llegando así a tener que analizar y determinar

que el próximo entrenamiento deberá realizarse a través de imágenes que contengan los rombos que valorizan a cada billete.

De esta forma se comienza nuevamente un camino de hipótesis con el fin de descubrir otros problemas a resolver con el fin de poder lograr una solución utilizable. Por consecuente, es necesario definir las especificaciones con la que se procederá a realizar este nuevo entrenamiento de la red neuronal. Primariamente se define utilizar las siguientes imágenes para entrenamiento que están compuestas de 27x54 píxeles. Un punto a posterior será resolver cómo obtener estos rombos de una imagen más grande y diversamente proporcionada, pero eso será motivo de resolución luego en caso de que esta red prometa mejores resultados.

Figura 82 – Rombos en billetes

Con el conjunto de imágenes se procede a realizar el entrenamiento pero debido a que la cantidad de píxeles y elementos se ha reducido considerablemente, se optará por intentar mejorar la precisión de la red significativamente, utilizando los siguientes parámetros:

- Error mínimo: 0,00001
- Cantidad máxima de iteraciones: 100000
- Factor de aprendizaje: 1.0

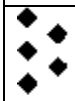
```
Entrenando la red por favor espere...¡Red entrenada con éxito!
Épocas: 764
Valor de error alcanzado
Error Global: 4.972959631457994E-5
```

Figura 83 – Resultado de entrenamiento de la red neuronal

El valor de error global fue alcanzado rápidamente, por más que se haya buscado mejorar la precisión. Queda ahora validar a través del reconocimiento de las imágenes, la validez del mismo. Para ello, no hay otra opción más que revalidarlo a través del esquema de pruebas anteriormente realizado: pruebas iniciales con las mismas imágenes

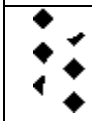
entrenadas, en segundo término pruebas con imágenes modificadas, y por último comprobaciones a partir de imágenes fotográficas procesadas. Dicho eso, se comienza con el experimento con las mismas imágenes utilizadas en el entrenamiento. Los resultados se muestran en la siguiente Tabla 23.

Tabla 23 – Resultados utilizando patrones seleccionados.

Billete	Resultado obtenido	Conclusión
	Más probable: 2p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 1,00 5p: 0,00 10p: 0,00 20p: 0,00 50p: 0,00	OK
	Más probable: 5p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,00 5p: 1,00 10p: 0,00 20p: 0,00 50p: 0,00	OK
	Más probable: 10p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,00 5p: 0,00 10p: 1,00 20p: 0,00 50p: 0,00	OK
	Más probable: 20p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,00 5p: 0,00 10p: 0,00 20p: 1,00 50p: 0,00	OK
	Más probable: 50p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,00 5p: 0,00 10p: 0,00 20p: 0,00 50p: 1,00	OK

Los resultados fueron los esperados, con el nivel de exactitud solicitado (1,00) debido al margen de error que fue entrenada la red. Ahora bien, este punto fue meramente para validar un comportamiento que ya conocíamos, ahora se procede a modificar las imágenes y comprobar cómo responden ante ese cambio, principalmente cuando se trate de posicionamientos (ver Tabla 24).

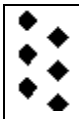
Tabla 24 – Resultados utilizando patrones modificados.

Billete	Resultado obtenido	Conclusión
	Más probable: 2p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,99 5p: 0,00 10p: 0,00 20p: 0,00 50p: 0,00	OK
	Más probable: 2p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 1,00 5p: 0,00 10p: 0,00 20p: 0,00 50p: 0,00	OK
	Más probable: 2p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,76 5p: 0,05 10p: 0,05 20p: 0,00 50p: 0,00	OK
	Más probable: 10p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,00 5p: 0,00 10p: 1,00 20p: 0,00 50p: 0,00	OK

Nuevamente no se presentan problemas cuando las imágenes son levemente modificadas. En efecto todavía se pudo comprobar a qué billete corresponde aún cuando todos sus rombos fueron alterados o reducidos en su gran mayoría. Llegado a este punto, la prueba que continúa es la desencadenante del mayor problema sobre el posicionamiento, por lo tanto se tomarán las mismas imágenes y se las moverá en sus ejes bidimensionales para comprobar el comportamiento de la red, a saber (ver Tabla 25).

Tabla 25 – Resultados utilizando patrones modificados.

Billete	Resultado obtenido	Conclusión
 X: -1; Y:0	Más probable: 2p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,98 5p: 0,00 10p: 0,00 20p: 0,00 50p: 0,00	OK

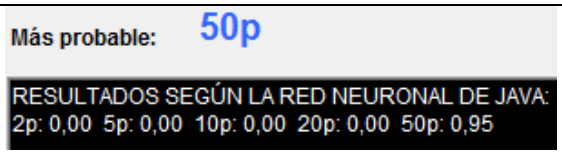
 X: -1; Y:+2	Más probable: 2p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,36 5p: 0,02 10p: 0,00 20p: 0,03 50p: 0,00	OK, con baja probabilidad
 X: +2; Y: +2	Más probable: 10p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,09 5p: 0,00 10p: 0,13 20p: 0,10 50p: 0,09	ERROR
 X: +1; Y: +0	Más probable: 20p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,00 5p: 0,06 10p: 0,00 20p: 0,94 50p: 0,00	OK
 X: -2; Y: +2	Más probable: 2p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,47 5p: 0,06 10p: 0,00 20p: 0,00 50p: 0,00	ERROR
 X: -5; Y: -5	Más probable: 5p RESULTADOS SEGÚN LA RED NEURONAL DE JAVA: 2p: 0,00 5p: 0,84 10p: 0,00 20p: 0,02 50p: 0,04	ERROR

Finalizada esta prueba, profundizamos y revalorizamos la complicación que plantea el problema de imágenes corridas de su posición original a las que fueron entrenadas. Si bien se han obtenido algunos buenos resultados, sin lugar a dudas la red neuronal tiene grandes complejidades para manejar estas situaciones lo cual conllevan a que este hito deba ser tratado de manera delicada y evaluando el uso de herramientas auxiliares para solucionar el conflicto.

Finalmente, se realiza una batería de pruebas con imágenes recortadas y convertidas a partir de fotografías, aún a sabiendas de la existencia o continuidad del

problema de posicionamiento para conocer y evaluar si existe alguna condición más vigente. Los resultados se muestran en la siguiente Tabla 26.

Tabla 26 – Resultados utilizando patrones con imágenes obtenidas de fotografías manuales.

Billete	Resultado obtenido	Conclusión
		OK
		OK, con escasa probabilidad

Se realizaron varias pruebas con diferentes fotografías de billetes, pero estas dos aquí expuestas son las más representativas: se alcanzó una excelente probabilidad de reconocimiento para el billete de 50 pesos y una escasísima casi nula probabilidad de acierto contra el billete de 10 pesos. Es importante o interesante remarcar y ver esto ya que a futuro es posible que se deba definir un umbral aceptable o porcentaje mínimo de precisión para devolver una respuesta, es decir, desde qué punto o condiciones se puede tomar como válido el resultado de un reconocimiento.

Aplicación de reconocimiento de billetes: Obtención de los rombos dentro de los billetes

En el apartado anterior se trabajó en el entrenamiento y reconocimiento de imágenes que contenían los rombos que justamente sirven para que los ciegos puedan identificar los billetes. Pero estas imágenes fueron tratadas y trabajadas manualmente para llegar a las figuras que se utilizaron para la red neuronal. Tanto la conversión a blanco y negro, como el recorte y achique de las imágenes se produjeron a mano con fines de evaluación.

Ahora bien, es evidente que este proceso no podrá realizarse en una aplicación que debe automáticamente reconocer qué billete está tomando con su cámara, por lo

tanto se da inicio a un estudio orientado a la obtención de estos rombos y sus características, que no es más que lo que se abarcará en este capítulo: la obtención automática de los rombos contenidos en el frente de un billete.

El primer punto a evaluar es la posición en la que se encuentran los rombos en cada billete en proporción a toda la imagen como también el tamaño que ocupan en proporción en el billete, tanto sea horizontal o verticalemente. Para ello, tomando cada imagen de cada billete se buscará cuáles son las coordenadas XY mínimas y máximas donde comienzan o terminan los rombos para comparar así contra el tamaño total del billete (en píxeles) y calcular qué proporción abarcan. Esto se realizará con cada billete para evaluar si existen diferencias entre cada uno de ellos.

Para esclarecer las medidas a tomar y cálculos a realizar, se indica referenciada la siguiente imagen a modo de ejemplo (figura 85):

Figura 85 – Coordendas mínimas y máximas de rombos en billetes de 2 pesos

Partiendo de la imagen del billete de 2 Pesos con un tamaño de 538 píxeles de ancho x 222 píxeles de alto, se busca el punto mínimo (X1;Y1) y máximo (X2;Y2) donde intervienen los rombos. Esto arroja lo siguiente:

Punto mínimo: 126;13

Punto máximo: 152;62

Estos valores no tienen sentido de ser si no son utilados en proporción con el tamaño total de la imagen o del billete, por lo tanto se calcula el comienzo y fin relativo del billete tanto a lo ancho como a lo alto.

Proporcional X1 = 126 / 538 * 100 = 23.42%

Proporcional Y1 = 13 / 222 * 100 = 5.85%

Proporcional X2 = 152 / 538 * 100 = 28.25%

Proporcional Y2 = 65 / 222 * 100 = 29.28%

Con estos puntos y sus proporciones, solo nos resta calcular el tamaño que ocupan los rombos en toda la imagen también en su proporción, por lo tanto lo que se debe calcular es la distancia punto a punto de la siguiente manera:

Tamaño ancho proporcional -> X2 – X1 -> 152 – 126 = 26 -> 26 / 538 * 100 = 4.83%

Tamaño largo proporcional -> Y2 – Y1 -> 65 – 13 = 52 -> 52 / 222 * 100 = 23.42%

A partir de este procedimiento repetido para cada uno de los billetes se podrán establecer parámetros mínimos y máximos para programar la obtención automática de la zona que comprende los rombos en cada billete. Estos cálculos y análisis se repiten por los 6 billetes, arrojando la siguiente tabla de resultados para utilización (ver Tabla 27):

Tabla 27 – Resultados utilizando billetes para obetener zona de rombos

Billete	Mínimo X	Mínimo Y	Máximo X	Máximo Y	Ancho Proporcional	Alto Proporcional
2	23.42%	5.85%	28.25%	29.28%	4.83%	23.42%
5	23.42%	5.85%	28.07%	25.23%	4.65%	19.38%
10	23,98%	7,21%	28,62%	23,42%	4,65%	16,22%
20	24,35%	7,66%	29,55%	20,72%	5,20%	13,06%
50	21,38%	5,41%	26,58%	15,32%	5,20%	9,91%
100	25,46%	9,46%	27,88%	15,77%	2,42%	6,31%

Como se puede observar, existen recuadros marcados en color amarillo. Esto se hizo para resaltar el valor más importante a la hora de definir los parámetros de tamaño. Para los casos de mínimos, se selecciona el punto mínimo de aparición como el más importante. Para los casos de máximos, por el contrario lo que interesa qué billete tiene más lejos sus rombos. Y por último, el tamaño proporcional, donde importará cuál es el

tamaño máximo de recorte que debe hacerse al billete para poder analizar todo los tipos de billetes.

Rápidamente con este cuadro podemos deducir que mínimamente debemos hacer un recuadro en el billete que sea de un 5.20% del ancho x un 22.07% de alto, de esta forma nos estaríamos asegurando que ningún rombo no entre en la imagen a evaluar. Cabe destacar que será necesario presentar también el delta o porcentaje de error que podemos tolerar, con lo que este recorte a hacer realmente no tendrá este tamaño exacto si no que se adicionará un plus para evitar problemas con billetes distintos o mal recortados en su totalidad. Estos deltas se analizarán luego, realizando pruebas sobre un conjunto de billetes más numerosos para evidenciar los posibles rangos que deban manejarse. Lo mismo ocurrirá con los puntos mínimos y máximos, también deben tener un error esperado que debemos mitigar.

De cualquier modo, realizando un seguimiento de los valores obtenidos en el cuadro anterior, vemos algunas oscilaciones importantes en cuanto a los puntos de comienzo y fin, si comparamos billete a billete. Para ser más gráficos, trazemos una línea imaginaria que atraviese todos los billetes tomando como referencia el billete de 2 pesos (ver figura 86):

Figura 86 – Comparativa de posición de rombos en clasificación de billetes

En los visualizado en la figura 86, queda demostrado y evidencia que con billetes alineados a la misma altura, existen grandes diferencias tomando como base el billete de 2 pesos. El cuadro anterior reflejaba esto mismo, pero visualmente la diferencia es más apreciable.

Con esta información se deduce que el billete de 50 pesos tiene sus rombos más a la izquierda es decir más hacia el comienzo del billete que los demás, mientras que el billete de 100 pesos al tener un solo rombo, por el contrario lo hace más alejado hacia la derecha o centro del billete. En cuanto a alturas las mismas también son un poco diversas

yendo del 5% al 10%. Estas variaciones hacen que tomar una determinación sobre cuál debe ser la proporción a tomar sea difícil de magnificar.

Con las mediciones obtenidas y su correspondiente análisis procedemos a especificar el tamaño objetivo y la posición relativa con que se realizará la obtención automática de los rombos en la imagen. Para facitiliar la comprensión, aplicaremos los proporcionales a valores absolutos con el fin de comprobar el rectángulo a recortar. Para ello mantenemos las dimensiones de 538 x 222.

Inicialmente, se tiene que el menor punto en X tiene como valor 21,38% y el máximo valor en X es 29,55%. Por lo tanto, para una imagen del tamaño mencionado esto se traduce en:

- X1 = 21,38% de 538 = 115,02
- X2 = 29,55% de 538 = 158,98

Con estos dos puntos, ya podemos conocer el ancho de la imagen: 158,98 – 115,02 = 43.96 píxeles. Resta la conversión de este ancho a su valor proporcional, por lo tanto aplicamos:

- 43.96 píxeles / 538 píxeles * 100 = 8,17% del ancho total del billete

Este 8,17% lo validamos contra el ancho máximo proporcional que habíamos obtenido en nuestra tabla, que era de 5,20% y vemos que estamos sobrados en casi un 3% en cuanto al ancho calculado máximo. Es decir que en caso de que la imagen no cuadre correctamente en cuanto a sus rombos, primero deberemos evaluar el posicionamiento correcto del recorte y luego ver si realmente es necesario seguir agrandando el ancho de la imagen que en principio se presenta holgada.

Se deben realizar los mísmos cálculos para la altura del recorte, relativos y absolutos utilizando los valores mínimo y máximos obtenidos:

- Y1 = 5,41% de 222 = 12,01
- Y2 = 29,28% de 222 = 65,00

Con estos dos puntos, ya podemos conocer el alto de la imagen: 65,00 – 12,01 = 52,99 píxeles. Se calcula su proporción:

- 52,99 píxeles / 222 píxeles * 100 = 23,87% del alto total del billete

El mayor alto relativo que habíamos obtenido era de 23,42%, el cual está bastante próximo al número recientemente calculado, lo que avecina que deba ajustarse este número promoviendo algún margen de error. Con estos valores, el recorte del billete quedaría de la siguiente forma que si visualiza en la figura 87:

Figura 87 – Zona de recorte de billete de 2 pesos para la obtención de rombos

Un recuadro de 44 píxeles X 53 píxeles, iniciando en el punto X = 115 e Y = 12. Como se puede apreciar rápidamente, para esta imagen la altura del cuadro ha quedado demasiado justa, no así el ancho que es bastante holgado pero debido a las diferentes oscilaciones que existen contra otros billetes, es necesario que así sea. Finalmente las imágenes recortadas resultantes de cada billete se visualizan en la figura 88:

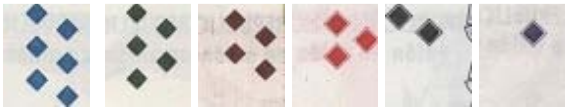


Figura 88 – Imágenes de rombos recortadas de billetes

De esta manera también se puede observar que el billete de 50 pesos es el que representaba el mínimo valor en X y cómo el billete de 20 pesos por el contrario representa el valor mayor X: ambos están sobre el límite de cada lado. Y debido a esta gran diferencia en el ancho, es que en billetes como el de 50 pesos, se pueden apreciar pedazos del billete que no corresponden a las figuras de los rombos sino al otro recuadro contiguo que contiene la figura del prócer.

Aplicación de reconocimiento de billetes: Encuadre de los rombos

En el capítulo anterior se trabajó sobre qué proporciones y ubicaciones tienen los rombos sobre la imagen de los billetes. Este análisis arrojó valores de base para comenzar a realizar la obtención automática o recorte de los mismos.

Pero quedó un punto sin resolver que no fue comentado: cuando un billete sea tomado a través de una cámara fotográfica de un dispositivo móvil, el encuadre del billete no será perfecto como en cual se realizaron estos análisis, muy por el contrario la imagen del billete tendrá mucho contenido que no corresponde al billete sino al fondo donde esté siendo apoyado, ya sea una mesa o simplemente la mano de quien lo sostiene.

Esto supone un problema importante para poder realizar el trabajo de recorte de los rombos y su automatización. Dado que la finalidad del trabajo es demostrar el uso de redes neuronales en dispositivos móviles junto al reconcimiento de imágenes en tiempo real, generar otro algoritmo.

Aplicación de reconocimiento de billetes: Binarización de los rombos

Bajo este tópico y teniendo ya la imagen del billete recortada conteniendo los rmbos a identificar, es necesario realizar algún proceos para convertir este recorte en elementos para las matrices de entrenamiento y reconocimiento de la red neuronal. Es decir que partiendo de una imagen como a continuación, lograr transfomarlo en información binaria (unos y ceros) para alimentar estas matrices.

Figura 89 – Imagen de rombos en billete de 10 pesos.

Se presenta ahora el dilema de cómo transformar esta imagen (figura 89) a información de entrada para la red. Para ello, el primer enfoque será volver a la teoría de convertir la imagen en blanco y negros para poder tomar estos 4 rombos de la imagen. Valiéndonos de esta figura, realizamos la conversión a blanco y negro de la misma, con el siguiente resultado que se muestra en la figura 90:

Figura 90 – Imagen de rombos en billete de 10 pesos en blanco y negro.

Como puede observarse en la figura 90, la conversión a blanco y negro sobre una imagen tomada por una cámara, devuelve una imagen no muy clara, en efecto aparecen más elementos de los que se necesitan evaluar. Si bien los rombos están visibles, los mismos están en color negro: contrariamente a lo que sucedía con los billetes de prueba que se venían utilizando. Esto se debe a diversos factores, por empezar que la luz de la cámara con la que la fotografía fue tomada juega un papel importante, haciendo que los rombos tengan un color oscuro más cerca del negro que del blanco, como así también el uso que tiene el billete, lo que lleva a que sus colores estén despintados y por lo tanto también más oscuros o con presencia de manchas, líneas o marcas que afectan a la conversión de la imagen.

Por lo tanto utilizar este enfoque de blancos y negros parece no ser muy viable cuando trabajemos con imágenes de fotografía, por lo que dejaríamos de lado este proceso para buscar alguno alternativo que nos pueda conseguir mejores resultados.

Es el momento de evaluar la aplicación del algoritmo de Canny explicado en el estado de la revisión de arte de este documento. El mismo resumidamente, se encarga de detectar los bordes de una imagen evaluando su contexto, no así como nos ocurrió recién con el tratamiendo de blancos y negros, que no evalúa en conjunto la imagen. A través de este algoritmo se buscará entonces encontrar cada uno de los rombos tratando de evitar aquellas líneas, marcas o manchas que no corresponden con la figura a tratar.

Para esto se consigue una librería Java OpenSource del algortimo de Canny y se realizan los desarrollos corespondientes para su adaptación y utilización en Android. El resultado de su aplicación para la imagen anteriormente tratada se visualiza en la figura 91:

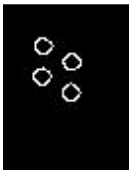


Figura 91 – Imagen de rombos en billete de 10 pesos aplicando algoritmo de Canny.

El resultado es mucho más limpio y orientado a la información que se necesita conocer. Sin embargo los rombos han perdido su forma característica a partir del proceso de obtención de sus bordes. Esto puede o no conducir a un problema, que en caso de continuar con este procesamiento podremos experimentar.

Si bien en esta primer prueba el algoritmo marcó claramente las figuras y no tuvo en cuenta los otros objetos como sucedía con la conversión a blanco y negro, el rombo o ahora figura más parecida a un círculo, está vacío. Como era de esperarse, dada su aplicación, el algoritmo de Canny solamente obtiene los bordes de aquellas figuras que a través del contraste y luminosidad pueden detectarse dentro de una imagen. Ahora bien, para aplicar a la red neuronal, estimamos que estos pocos píxeles pueden no alcanzar para realizar un reconocimiento correcto.

Lo ideal a obtener sería una combinación de lo que nos proveía la transformación a blanco y negro, con rombos marcados y rellenos en su interior, junto con la limpieza y obtención de Canny. Por lo tanto Canny como algoritmo único queda a medio camino, nos provee una buena herramienta en términos de solucionar los problemas de luminosidad, foco, contraste y diferencia de colores, pero nos agrega el problema de tener imágenes sin relleno y pocos valores para binarizar.

Por lo tanto antes de investigar e incursionar si existe alguna alternativa para combinar estas opciones, es necesario comprobar que Canny funciona de esta forma para todos los billetes. Se realiza a continuación una generación y obtención de los rombos para cada tipo de billete con el fin de determinar si es una herramienta confiable.

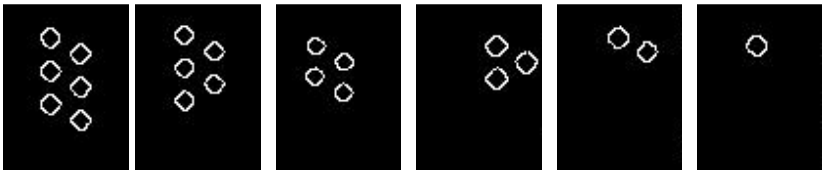


Figura 92 – Imagen de rombos en billete de 2, 5, 10, 20, 50 y 100 pesos aplicando algoritmo de Canny

El procesamiento Canny para todas las imágenes de los billetes (ver figura 92) fue satisfactorio, salvo por la figura del billete de 20 pesos (3 rombos), donde uno de sus figuras no está completamente cerrado. Agrandando la imagen se puede observar claramente que falta un píxel para completar el rombo/cículo que está a la derecha de la imagen (ver figura 93):

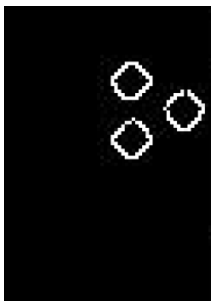


Figura 93 – Imagen de rombos en billete 20 pesos aplicando algoritmo de Canny

Si bien este problema apareció en pocas conversiones Canny, es necesario destacar que puede suceder y que por lo tanto en estas ocasiones, un algoritmo convencional de llenado de polígonos irregulares que por ejemplo ya proveen Java/Android, no es posible de utilizar debido a que necesitan que la figura sea completamente cerrada, es decir poder definir un punto de inicio y fin conectados.

Por lo tanto se plantea idear un algoritmo de desarrollo propio y específico, capaz de poder rellenar estas figuras de la mejor manera posible. Para ello, se deberán tener en cuenta los siguientes puntos:

- Encontrar las figuras irregulares e individualizarlas como tal
- Encontrar los límites de cada polígono individualmente

- Rellenar el contenido de estos polígonos, intentando formar una figura totalmente completa

El primer paso de encontrar las figuras irregulares que identifican al billete se realiza inicialmente obteniendo el conjunto de puntos blancos que la conversión de Canny arrojó. Obtener el conjunto de puntos significa conocer de cada punto, su posición X y su posición Y en la imagen. Mostrémoslo con un ejemplo en la figura 94 a través de la imagen del billete de 50 pesos:

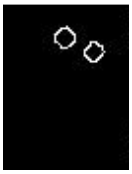


Figura 94– Imagen de rombos en billete 50 pesos aplicando algoritmo de Canny

Esta imagen (figura 94) está compuesta por dos figuras bien marcadas separadas entre sí. El primer paso es obtener solamente aquellos puntos que se ven como un blanco casi perfecto gracias a la conversión por Canny. Los puntos a obtener se leen de arriba hacia abajo, de izquierda a derecha. Por tal motivo, el primer punto que encontramos en esta imagen está en la siguiente posición X: 32 Y: 11. El algoritmo de búsqueda sigue recorriendo y en la siguiente fila vuelve a encontrar más puntos en blanco X: 29,30,31,32 todos con Y: 12. Así se van evaluando y obteniendo todos los puntos en blanco que están en la imagen y guardando sus valores en distintos arreglos/arrays. Esta imagen del billete de 50 pesos, en total tiene 76 puntos en blanco que son localizados y almacenados para posterior análisis (ver figura 95).

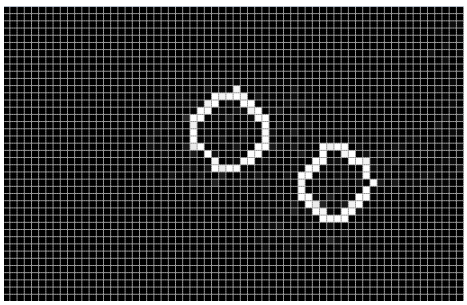


Figura 95– Análisis ampliado de imágen de rombos en billete 50 pesos aplicando algoritmo de Canny

Una vez obtenidos todos los puntos, es cuestión de analizar qué polígonos deben formar. Para eso nos valdremos de una ventaja esencial que tenemos a través de conocer cómo están diagramadas las figuras. Nos referimos puntualmente al hecho de que las figuras estén separadas entre sí, nos facilita para poder encontrar que un punto pertenece a tal u otro polígono. Por lo tanto en este siguiente paso de encontrar los límites que componen cada figura, debemos recorrer cada punto en blanco que obtuvimos y evaluar si debe pertenecer a una nueva figura o si puede pertenecer a alguna figura ya existente. Esto se verá más claro en el siguiente procedimiento.

Nuevamente debemos iterar de arriba hacia abajo y de izquierda a derecha pero esta vez ya directamente sobre el conjunto de puntos blancos obtenidos. Como indicamos anteriormente, el primer punto blanco para el billete de 50 pesos es el X: 32 Y: 11. Este punto al ser inicial, evidentemente no puede evaluarse si corresponde a algún polígono ya que en la primera iteración no va a existir ningún polígono, por lo tanto este punto automáticamente crea el primer polígono al cual se le corresponde el punto X: 32 Y: 11. Rápidamente, se continúa con la próxima iteración donde comienza lo interesante.

El siguiente punto blanco que se tiene es el X: 29 Y: 12. Ahora bien si nos centramos solamente en la información que tenemos, nuestra imagen sería de esta forma (ver figura 96):

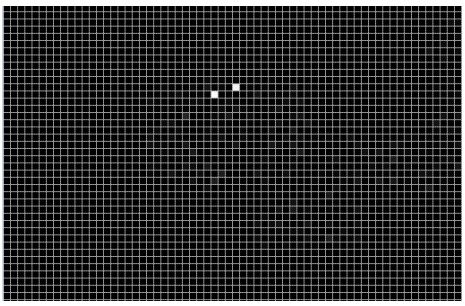


Figura 96 – Análisis de imágen de rombos en billete 50 pesos aplicando algoritmo de Canny

Un solo punto inicial que el es de arriba, y nuestro nuevo punto que estamos evaluando, debajo a su izquierda. Reiterando lo que dijimos al principio, nuestra principal ventaja es saber y conocer que los distintos rombos que componen al billete, se encuentran visiblemente separados unos de los otros y esta condición nos será útil de

explotar. Por lo tanto valiéndonos de esta condición, podemos realizar la siguiente inferencia: cuando un punto esté separado de otro a más de una distancia “D”, ese punto será parte de otro polígono, y por el contrario, cuando un punto está a menos de esta distancia “D”, indicará que ese punto forma parte del mismo polígono. Es necesario remarcar que este análisis de distancia entre puntos ahora parece muy sencillo porque solamente tenemos dos puntos, pero el algoritmo debe ser capaz de ir recorriendo y realizando esta evaluación contra todos los puntos que se vayan agregando a cada polígono hasta encontrar a cual pertenece o concluir que se trata de otro polígono nuevo.

Ahora bien, es necesario definir esta distancia “D” con un valor concreto. Si revisamos las imágenes obtenidas de Canny, vemos que la distancia horizontal (X) que podemos encontrar generalmente es de 4 píxeles.

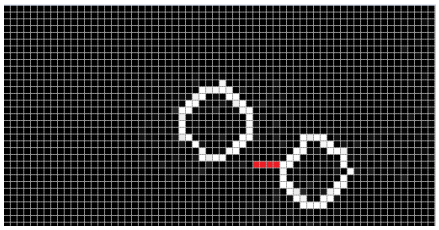


Figura 97 – Imagen de billete de 50 pesos con 4 píxeles de separación en X.

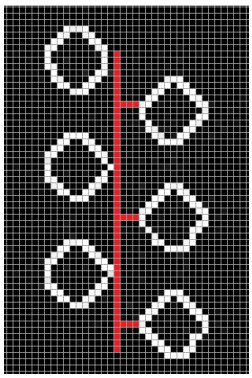


Figura 98 – Imagen de billete de 2 pesos con 4 píxeles de separación en X.

A partir de este número podríamos inferir que se podría utilizar una distancia “D” horizontal de hasta 4 píxeles, pero debido a que ese número está sobre el límite, se decide evaluando todas las otras imágenes que se resuelve que es más conveniente perder puntos del propio polígono, que adicionar un punto de otro polígono como propio.

Por lo tanto, se reduce esta distancia a “D” a un valor de 3 píxeles para el tamaño de las imágenes trabajadas.

De esta forma, evaluando los dos puntos en cuestión, el agoritmo realiza la comprobación de si el punto nuevo a leer está dentro del área +3 -3 de todos los puntos que ya se tengan como parte del polígono. Gráficamente para este caso:

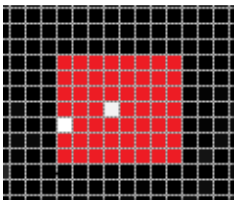


Figura 99 – Análisis de puntos para obtener área

Como se visualiza en la figura 99, el punto blanco de la izquierda, de la segunda iteración está dentro del cuadrado de 7x7 que se generó y por lo tanto se considera que forma parte del polígono y se agrega al mismo. El proceso se repite en cada iteración, buscando y evaluando si el punto leído pertenece está a una menor distancia tanto horizontal como verticalmente para ser considerado parte de algún polígono. Cuando llega al primer punto del otro polígono, la situación es la que se refleja en la figura 100:

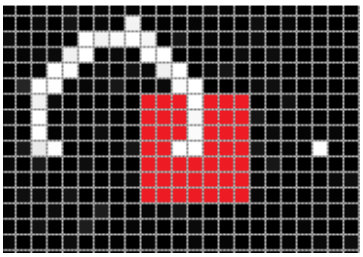


Figura 100 – Análisis de puntos para obtener área

Cuando la iteración alcance al punto blanco de la derecha y se evalúen todos los otros puntos que ya son parte de un polígono, no se obtendrá ninguna cercanía: en efecto el punto más cercano dibujado en la imagen no alcanza a este punto y por lo tanto en esta iteración se genera un nuevo polígono que se agrega al arreglo de polígonos que tienen la información.

El proceso continúa hasta leer todos los puntos blancos que habían sido detectados, y como resultado de esto se busca obtener al menos la cantidad mínima de polígonos para representar los rombos del billlete. Si bien se viene hablando de la importancia de generar este tipo de formas, para el paso siguiente no nos importará tanto que se haya o no formado un polígono irregular, sino poder haber detectado y por sobre todas las cosas, haber agrupado los distintos puntos que es la información que servirá en el próximo paso.

Por lo tanto, una vez encontrados los puntos y categorizados/agrupados en distintos polígonos o arreglos, es momento de utilizarlos para completar el llenado de estas figuras. Todo este proceso sirvió para informatizar las imágenes que se tenían inicialmente

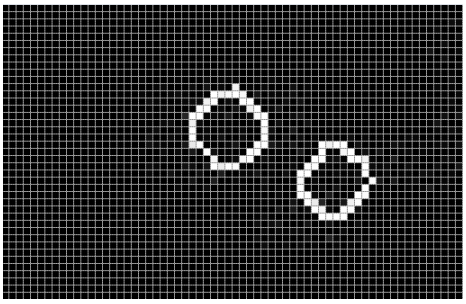


Figura 101 – Puntos categorizados/agrupados.

Con la separación y sectorización de cada punto, ahora queda nada más recorrer cada elemento de los arreglos que contienen los polígonos, evaluándolos de manera independiente y no en relación a los elementos externos, es decir, se recorrerá cada grupo de puntos obtenidos de manera individual para proceder al relleno de los mismos.

De esta manera partiendo de esa premisa, si se obtuvieron correctamente dos polígonos informatizados con la información pertinente, el llenado de esta figura será realizado de la siguiente manera. Nuevamente se leerán de arriba hacia abajo y de izquierda a derecha, los puntos que componen el polígono. Pero esta vez, no se buscará distancias contra otros puntos, si no que se buscará su punto límite en la figura. Pasemos a ilustrarlo en la figura 102:

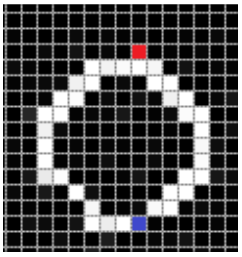


Figura 102 – Límites para la búsqueda de los puntos.

Nuevamente partiendo del punto inicial en X:29 Y:11 (marcado en rojo), el algoritmo debe encontrar cuál es el punto que comparte en X = 29, pero con Y mayor, es decir el punto mínimo y máximo para la posición X = 29. De esta forma conociendo ambos puntos, que en este caso serían X:29 Y:11 (color rojo) y X:29 Y:22 se puede trazar una línea hacia abajo que complete o rellene la figura como se refleja en la figura 103:

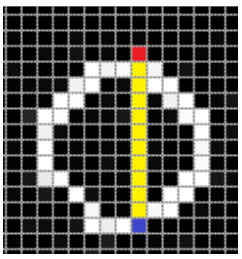


Figura 103 – Trazado de línea entre límites para la búsqueda de los puntos.

De esta forma, se comienza a completar y rellenar la figura a través de recorrer todos los puntos obtenidos. Existirán casos donde para ese valor de X (horizontal), se tendrá un solo punto, en ese caso claramente no habrá relleno porque ese punto ya lo completa. Mismo ocurrirá cuando se lean los puntos de abajo que hacen de límite: serán los puntos más hacia debajo de la figura en esa posición de X, y por lo tanto no necesitarán más relleno que ese (ya que han sido procesados por su punto más arriba).

Cuando finalicen todas las iteraciones de este polígono, se tendrá finalmente la forma buscada: completamente rellena (ver figura 104).

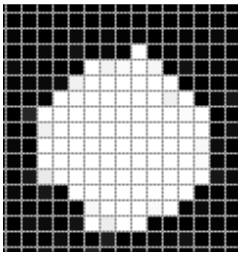


Figura 104 – Figura rellena recorriendo puntos obtenidos aplicando algoritmo de Canny.

De la misma manera, una vez terminado el primer grupo de puntos, se prosigue a evaluar individualmente el segundo grupo de puntos y por lo tanto se obtiene la representación de la figura del billete binarizada correctamente (ver figura 105):

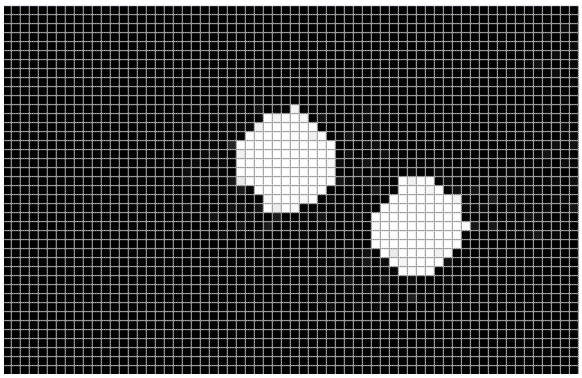


Figura 105 – Puntos obtenidos al procesar billete de 50 pesos aplicando algoritmo de Canny

Con este tipo de imágenes procesadas estamos cada vez más cerca de poder utilizarlas y volcarlas a nuestra red neuronal para comenzar el entrenamiento y posterior reconocimiento. Antes de eso, es bueno evaluar el caso del billete de 20 pesos que presentaba la anomalía de no tener uno de sus puntos cerrados, y por lo tanto no poder cerrar un polígono como tal (ver figura 93).

Este conjunto de puntos , cuando sea procesado por todos los algoritmos antes descriptos, desembocará en la siguiente representación (ver figura 106):

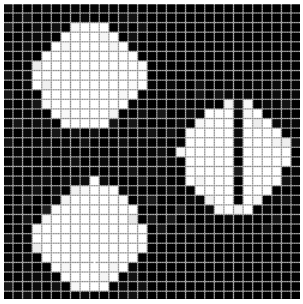


Figura 106 – Conjunto de puntos obtenidos al procesar zona de rombos en billete de 20 pesos

Tres polígonos correctamente detectados pero con uno de ellos que no pudo rellenar con totalidad sus elementos. Hacemos mención del polígono a la derecha, que al no poder haber obtenido desde Canny ese punto de cierre, nuestro algoritmo no pudo rellenarlo completamente. Si bien pueden existir soluciones, como por ejemplo al no tener 2 elementos en Y, completar hasta la misma altura que el más próximo o a través de una aproximación cuadrática, pudiendo inferir en otros tipos de problema, se simplifica y define que no es necesario hacer ninguna otra tratativa ya que en efecto, para esa figura se tiene un porcentaje de relleno del 89% (11 puntos de 99 que componen ese polígono no fueron detectados), el cual es altamente representativo. De igual modo, mejoraremos aún más este porcentaje con las aplicaciones que se realizarán posteriormente.

Si recordamos las imágenes iniciales que fueron procesadas por Canny y las recientemente obtenidas, tenemos imagénes del siguiente tipo (ver figura 107):

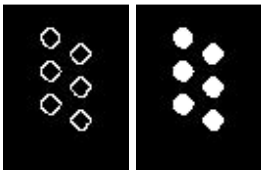


Figura 107 – Comparativa entre imágenes procesadas utilizando algoritmo de Canny y las obtenidas utilizando proceso de búsqueda por límites.

De la figura 107 se visualizan las mismas imágenes con un tamaño de 64 x 85 píxeles, y un gran espacio en negro o sin información. Por este motivo, y para lograr una estandarización de todas las imágenes, se plantea un nuevo procedimiento para reducir el tamaño de la imagen a los píxeles intervinientes, de tal manera que los puntos de

información tengan mayor impacto y presencia en el contenido total. Para lograr eso, la idea a poner en práctica es la siguiente: recortar la imagen resultante a partir del punto en X e Y mínimo. Localizados estos puntos, realizar un recorte fijo de tantos píxeles de ancho y alto, para normalizar todas las imágenes y así la presencia de los polígonos tomará mayor protragonismo en la totalidad de la imagen, donde tanto los pesacios llenos como vacíos servirán para determinar qué billete se corresponde.

Bajo estas condiciones, se evalúa cuáles son los tamaños fijos a adoptar de ancho y alto para equalizar todas las imágenes. Se parte analizando el billete de 2 pesos que por tener mayor cantidad de figuras, es el que define el tamaño mínimo que deberán tener todas las imágenes para poder representarse. Por lo tanto, se toma la imagen convertida y binarizada del billete de 2 pesos, y se busca su punto en XY mínimo.

Tomando la imagen binarizada del billete de 2 pesos (ver figura 75), y realizando un encuadre exacto de sus puntos, nos arroja una muestra de tamaño 26x53 píxeles como se visualiza en la figura 108:

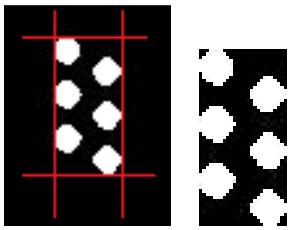


Figura 108– Zona de rombos en billete de 2 pesos.

De esta forma, pudiendo solamente identificar el punto superior izquierdo de cada billete que procesemos, al realizar un recorte de 26 píxeles a la derecha y 53 de largo, nos aseguraremos obtener todos los rombos que lo componen. Un dato no menor y de nuestra ayuda, es que todos los billetes poseeen su primer rombo, es decir su rombo más alto, siempre comenzando del lado izquierdo. Esto permite emplear este método sin dificultades.

Cabe destacar que a lo largo del trabajo se han utilizado otras técnicas y otros tipos de encuadre, como ser un enfoque a dos posiciones donde tiene importancia tanto el punto inicial como el punto final, y a partir de ahí se realiza el encuadre (muchos programas de manejo de imágenes realizar esta tarea justamente en la función llamada

"encuadre"). Pero esto nos podía causar un problema ya que en varias pruebas dependiendo de cómo se tome la foto, en algunos billetes la cercanía de los rombos contra la franja divisoria arrojaba puntos extraños que no pertenecían a nuestro análisis por lo que optamos por utilizar este método, basados en el supuesto que mencionamos anteriormente de tener una imagen de proporciones conocidas en un tamaño absoluto standard, nos permitió esta resolución tomando un extracto fijo de la imagen solamente partiendo del valor inicial que por su dibujo en el billlete se presenta más limpio y sin dibujos cerca que puedan afectar a los algoritmos.

Figura 109 – Encuadre de billete de 20 pesos.

Falta solamente un paso para que estas imágenes estén ya disponibles para servir de entrenamiento en nuestra red. Con el tamaño de 26x53, todavía nos arroha una combinación de 1378 píxeles para entrenar nuestra red matricial, lo cual por las pruebas y ensayos anteriores, continúa siendo un número muy grande. Por lo tanto para poder evaluar si nuestra red puede lograr un funcionamiento correcto con un entrenamiento adecuado, se lleva a un tamaño aún más chico a estas imágenes binarizadas. Entonces el objetivo es llevar la imagen al tamaño mínimo que pueda ser utilizado para representar cada figura, teniendo en cuenta que en el proceso no se pierda ninguna información. Es sabido que al achicar una imagen o agrandarla, la misma se deforma perdiendo sus cualidades originales: cuando se produce un achicamiento lo que se pierde son píxeles, unos píxeles les ganan a los otros y permanecen mientras que otros terminan siendo borrados o adoptados por sus inmediatos consecutivos. Así y todo logramos luego de realizar varias pruebas, conseguir un tamaño acorde a esta representación mínima para cada billete que e no es más ni menos que un tamaño de 3x6 píxeles. Esto se realiza a través de procesamientos de escalado de imagen, que aseguran que la pérdida de información nos siga siendo representativas para nuestra red neuronal. A continuación el resultado de esta transformación de las imágenes (ver figura 110), con un zoom del

1600% debido a que una imagen de 3x6 píxeles en un documento no es visible para el ojo humano, pero para nuestro favor, el procesamiento lo realizará un algoritmo de inteligencia artificial:

Figura 110 – Imágenes resultandos desde 2 pesos a 50 pesos.

Finalmente, con este tipo de imágenes estamos en condiciones de realizar el entrenamiento de nuestra red final para intentar lograr nuestro cometido, de poder tomar una fotografía a través de un celular y que el mismo pueda procesar e identificar cuál es el valor que representa.

Aplicación de reconocimiento de billetes: Entrenamiento de matriz con rombos

Con las imágenes ya obtenidas y tratadas, se acerca el momento cúlmine de nuestro trabajo. Los pasos posteriores que restan son solamente los de entrenamiento y reconocimiento. En el medio de estos procesos, también está la conjunción de trabajar con la red neuronal en el celular, junto a algunos agregados propios de la obtención de imágenes a través de la cámara y demás cuestiones, que no son más que problemas o condiciones que tienen que sortearse o que quedarán como mejoras técnicas y/o funcionales.

Por lo tanto, el entrenamiento la red se realiza de manera similar al que fue producido anteriormente cuando se evaluaron los rombos de figuras más grandes que solamente habían sido tratados en blanco y negro. A diferencia con las ejecuciones anteriores, ahora se poseen imágenes mucho más pequeñas de 3x6 resultando en una matriz de 18 posiciones. Nuestro conjunto de elementos serán los billetes de 2 pesos a 50 pesos.

Si bien el entrenamiento consiste en el mismo tipo de procesamiento, es decir, se cargan una a unas la imágenes, identificándolas con el valor que representan, en esta ocasión haremos algo diferente con respecto al parámetro de iteración. En vez de estar buscando por un error de error esperable, dejaremos la red entrenando por 24 horas para

ver qué valor de error alcanza y cuántas iteraciones realiza. Luego de 24 horas de entrenar las 5 imágenes de 3x6 de cada billete, los resultados fueron lo siguientes

- Error mínimo alcanzado: 10^{-7} = 0,0000001
- Cantidad de iteraciones realizadas: 74.574.123

Claramente consideramos que el entrenamiento es extremadamente satisfactorio luego de 1 día de estar procesando las imágenes, de hecho el nivel de error manejado es ampliamente superior al que se estuvo trabajando en procesos anterioes.

Con el resultado del entrenamiento, lo que obtenemos son matrices de pesos de salida que sirven al algoritmo de reconocimiento para luego poder determinar de qué billetes se tratan. Estas matrices resultantes, son extraídas de nuestro ambiente de entrenamiento de la red neuronal a un archivo de texto plano de fácil manera, tal como hicimos con nuestra red inicial de operadores lógicos. En efecto, lo estrictamente vinculado al proceso de la red neuronal, no posee diferencias con lo realizado con la red de operadores lógicos, solamente cambian el volumen de información y el medio para terminar realizando el reconocimiento (cámara celular). Cabe mencionar el peso del archivo resultante que contiene la información del entrenamiento de la red neuronal es de 12kb solamente. Desde otro punto de vista, se puede decir que toda la información del entrenamiento de la red neuronal consta en esos escacos 12kb. Veremos más adelante si esta minúscula base de información podrá satisfacer nuestras necesidades y experiencias.

Aplicación de reconocimiento de billetes: Preparación de la aplicación móvil

Generadas las matrices con los pesos de salida entrenados de cada imagen, las mismas ya se encuetran disponibles para ser imporatadas en nuestra aplicación Android para reconocimiento de billetes. Como es sabido, también por la utilización en los operadores lógicos, la aplicación ya cuenta con las clases y métodos correspondientes tanto para tomar estas matrices entrenadas como para realizar el proceso de reconocimiento. Lo que está faltando en este punto, es la forma de alimentar con una imagen o matriz de 3x6 nuestro método de reconocimiento.

Para comenzar la obtención de la imagen y codificación en Android para tal fin, debemos tener en mento los temas abordados en capítulos anteriores con respecto al

procesamiento y estudio que se les dio a las imágenes en ambientes aislados y específicos. Ahora es momento de todos esos procesos, algoritmos y ajustes volcarlos en la aplicación. A grandes rasgos, la aplicación deberá realizar lo siguiente:

- Obtener una imagen completa del billete a través de la cámara
- Realizar un encuadre del billete mismo para conocer los límites que lo contienen
- Reducir el tamaño a un tamaño absoluto, que mantiene las proporciones adecuadas
- Realizar un primer recorte que obtiene la zona donde se encuentran los rombos del billete
- Procesar la imagen con Canny para obtener una imagen en blanco y negro con el contorno de las figuras del billete
- Recorrer la imagen en búsqueda de los polígonos obtenidos
- Rellenar los polígonos para que la información sea utilizable
- Encuadrar el billete tomando su vértice superior izquierdo
- Realizar recorte de tamaño fijo
- Reducir la imagen al tamaño final de 3x6

Como se menciona, estos son en resumen los procesos que debe hacer la aplicación para dejarla disponible para su posterior reconocimiento. A continuación, iremos entrando en detalle en cada una de estas tareas comentando su resolución y problemas encontrados. Junto a cada paso, también se irán incluyendo las imágenes que componen el proceso.

El primer punto menciona la obtención de una imagen completa del billete. No es casualidad la aclaración "completa" en esta tarea. La obtención de la imagen a través del celular, técnicamente en Android no carece de ninguna novedad o desafío tecnológico. De hecho Android permite de manera standard, realizar obtenciones de imágenes con aplicaciones nativas o incluso de descarga en su PlayStore. A los fines de este trabajo, las pruebas y desarrollos fueron realizados con la aplicación nativa exclusivamente.

Ahora bien, el motivo por qué mencionamos e hicimos hincapié de que la imagen debía ser completa está relacionado a las restricciones técnicas que posee este trabajo, que no son de relevancia para lo que se estudia y quiere demostrar. Por ello que debido a

que no es finalidad del trabajo seguir incluyendo algoritmos de procesamiento de imágenes, para poder procesar la foto del billete se exigen las siguientes condiciones:

- La foto tomada debe contener el frente del billete: esto se deduce fácilmente siguiendo la línea de trabajo que se vino realizando, ya que todos los procesamientos y entrenamientos terminan en la obtención y procesamiento de los rombos del billete que no se encuentran al dorso del mismo. También el mismo tiene que estar sin haber sido girado de pies a cabeza en cuanto a su posición horizontal.
- La foto debe contener la totalidad del billete: es decir que debe ser tomada a una distancia relativamente media que abarque toda la figuras, e inclusive la misma debe ser obtenida sobre un fondo negro. Esto debido a que uno de los primeros pasos con la imagen obtenida es llevarla a un encuadre rectangular con las proporciones esperadas que tienen los billetes. Realizar esto de manera automática teniendo otras figuras en la imagen y no teniendo la totalidad del billete, conlleva otros tipos de problemas propios de un trabajo formal extenso. Por lo tanto esta restricción, tiene mucho potencial para mejorarse y reducirse, a través de algoritmos de reconocimientos de figuras que hoy en día existen y que son muy buenos para este tipo de tareas.

Comentadas estas limitantes, la imagen será obtenida sobre un fondo negro a una distancia de entre 20 a 30 cm, como se muestra en la figura 111.

Figura 111 – Imagen de billete de 5 pesos tomada a 30 centimetros de distancia con fondo negro.

Paso siguiente sería obtener el encuadre correcto de esta imagen para poder comenzar a trabajarla en los demás procesos, pero antes de eso cabe mencionar algunas otras complicaciones físicas que la aplicación solucionará.

Primeramente, la aplicación detectará si la imagen fue obtenida de forma horizontal o vertical, con la cámara dada vuelta o no, ya que se desconoce la posición de las manos con que cada persona toma el celular. Al obtener una foto con el celular, se guarda una información llamada que es la metadata de la imagen obtenida. En ella se incluyen datos interesantes, como ISO, resolución, fecha original, etc y para nuestro interés se incluye la orientación con la que obtenida. Basados en esa orientación, se realiza entonces también un giro de la imagen complementando para que siempre la misma ingrese a ser procesado de manera horizontal.

Con la imagen obtenida, se procede a realizar el primer encuadre de la foto total. El algoritmo para realizar tal tarea es bastante simple, por eso es que se ayuda estrictamente de la restricción de haber sido tomada completa sobre un fondo negro. La finalidad del mismo es encontrar el punto mínimo y máximo XY para realizar el recorte correspondiente. Para obtener este resultado, lo realizad de la siguiente manera: barre la imagen píxel por píxel primero buscando el punto mínimo XY evaluando un umbral de luminosidad que contenga ese píxel. Cuando encuentre el primer píxel que no es negro, evalúa si realmente se trata del comienzo de una imagen o un píxel aislado por problemas de la obtención de la foto y demás. Esto se hace evaluando en este caso, los píxeles contiguos hacia abajo y la derecha, tomando una muestra de 10 píxeles por cada dirección. Si la luminosidad se mantiene en más de un 50%, estamos en presencia de un punto que corresponde al billete. De la misma manera pero en el sentido inverso, se realiza la obtención del punto máximo XY.

Con estos dos valores, se procede al recorte de la imagen exclusivamente del billete, descartando todo el resto, teniendo como resultado una imagen del siguiente tipo (ver figura 112):

Figura 112 – Imagen de billete de 5 pesos tomada a 30 centimetros de distancia con fondo negro recortada

El paso posterior es de simple ejecución, consiste solamente en achicar la imagen al tamaño de estudio principal que era 513x216 píxeles. Hablamos solamente de achicar debido a que la resolución de imagen con la que fue obtenida la foto por la cámara del celular, es altamente mayor a nuestro objetivo. Este achique y encuadre siempre respetan las proporciones esperadas, caso esto no ocurriese todas las inferencias posteriores serían inapropiadas, infudadas y por consecuente erróneas.

Con la imagen neta del billete, llevada a nuestro tamaño absoluto de estudio procedemos también a hacer el recorte simple de la parte que nos interesa. Nuevamente gracias al estudio realizado junto con las pruebas e inferencias anteriores, se determinó que el recorte debería realizar en las posiciones relativas:

- X1 = 21,38%
- X2 = 29,55%
- Y1 = 5,41%
- Y2 = 29,28%

Nuevamente estos pasos son bastante simples, ya que se utilizan librerías de manejo habitual sobre gráficos e imágenes, sin problemas ni técnicos ni conceptuales. El resultado que se obtiene se muestra en la figura 114:

Figura 114 – Zona de rombos obtenida de imagen de billete de 5 pesos tomada a 30 centimetros de distancia con fondo negro.

En la figura 114 podemos ver la imagen que contiene los rombos del billete, disponible para procesar con Canny.

Luego de este paso, comienza un proceso un poco más complejo ya que es momento de la aparición de Canny para que haga su cometido, proveernos de una imagen limpia con rombos a binarizar. Para realizar esto, se reescribieron algúnos métodos de la clase Java para Canny que se utilizó en las pruebas de concepto, para que los mismos sean utilizables en un dispositivo móvil. Esto se debe a que existen librerías optimizadas para aplicaciones de escritorio como otras para desarrollo de aplicaciones móviles, por tanto se hizo una reingeniería en este aspecto que no tiene mucho sentido continuar ahondando en detalles ya que no es la finalidad del trabajo. Ahora bien, con Canny funcionando solamente resta indicarle nuestros umbrales de trabajo, lo que nosotros consideramos como la sensibilidad del método para identificar y trazar líneas. Con umbrales pequeños, cualquier imperfección o mancha, Canny posiblemente lo dibujaría como una línea, es por eso que se utilizan valores límites altos de histéresis para que Canny solamente tome las partes bien marcadas del billete. Luego de varias pruebas de ajuste, se determinan que los valores a utilizar son 14 para el mínimo y 18 para el máximo. Finalmente, se está en condiciones de procesar la imagen y obtener la siguiente figura (ver figura 115):

Figura 115 – Imagen de procesamiento aplicando algoritmo de Canny en zona de rombos obtenida de imagen de billete de 5 pesos tomada a 30 centimetros de distancia con fondo negro.

Como puede observarse en la figura 115, los 5 rombos se transforaron en figuras casi circulares como hemos venido experimentando, nítidamente marcadas y separadas unas de otras.

Ahora es momento de aplicar lo enunciado en el apartado “Aplicación de reconocimiento de billetes: Binarización de los rombos” donde se explicó el algoritmo utilizado para generar y recorrer estas figuras obtenidas, generando un arreglo o conjunto de polígonos irregulares para posterior llenado de los mismos, es decir nuestro proceso llamado de binarización.

Este algoritmo es reutilizable 100% ya que técnicamente no son más que iteraciones sobre matrices de píxeles contienendo información true/false de si el píxel es blanco o negro. Como se comentó oportunamente, el algoritmo evalúa punto por punto de la imagen, para ver si pertenece o no alguna figura que ya se haya evaluado. Esto lo hace a partir de evaluar la proximidad con aquellos otros puntos, caso ser un punto que no está próximo a otra figura, se está en presencia de un punto que forma parte de un nuevo polígono hasta ahora no procesado por la iteración. Este paso carece de una ejemplificación gráfica con respecto al billete, ya que solamente se puede evidenciar evaluando los valores que va tomando el programa en tiempo de ejecución, es decir a través de realizar un debug del programa paso por paso de cada iteración.

Para este caso, el procesamiento de la imagen arrojará que contiene 5 polígonos listos para poder rellenar, que es el paso inmediatamente posterior. Como se comentó en el capítulo mencionado sobre la binarización, el proceso de relleno es también una iteración de izquierda a derecha y de arriba hacian abajo, pero solamente recorriendo los puntos que componen a cada polígono. Es decir, primero se toman todos los puntos del polígono número , y se busca el valor mínimo y máximo de cada punto X que se esté recorriendo para trazar una línea entre ellos. De esta forma se va produciendo el llenado de las imágenes, hasta tener una imagen como la que se muestra en la figura 116:

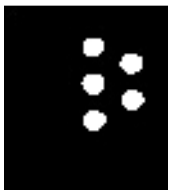


Figura 116 – Imagen de procesamiento de relleno de rombos.

Una figura con polígonos irregulares con tendencia a circulares completamente rellenas y binarizadas correctamente. De esta forma, la binarización queda completa, solo restan ajustes de tamaño para llevarlo a las dimensiones que se necesitan.

Pero primero para hacer el recorte final, si revisamos los pasos hay que realizar un encuadre correcto de la representación del billete para poder hacer el recorte final correctamente. Este encuadre se hace partiendo del punto inicial XY con menor valor, y luego recortando un valor absoluto ya establecido según estudios, de 24x44, obteniendo una imagen como la que se visualiza en la figura 117.

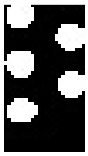


Figura 117 – Imagen de procesamiento de relleno de rombos recortada.

Una imagen perfectamente adecuada para nuestra red neuronal, conteniendo la información solamente de los rombos del billete, perfectamente identificables, sin alteraciones ni mezclas entre las figuras que la componen.

Por último, dadas todas estas condiciones, la imagen se reduce al tamaño de entrenamiento de la red neuronal de 3x6 píxeles, bajo el simple método de la clase Java Bitmap.createScaledBitmap. A continuación, se muestra la imagen en su tamaño original de 3x6, que difícilmente sea visible o interpretable para el ojo humano bajo el formato de este documento, aunque a su lado se indica la misma imagen con un zoom de 800% para poder visualizarlo como se muestra en la figura 118.

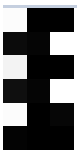


Figura 118 – Imagen de entrenamiento en la red neuronal ampliada (Zoom 800%).

Finalmente, la foto obtenida originalmente no requiere de más procesamientos: hemos llegado al punto que se propuso al entrenar la red neuronal, tener una imagen de

3x6 píxeles conteniendo la información binarizada de los rombos del billete. Ahora solo queda el último apartado de este trabajo y por consiguiente, su desenlace.

Aplicación de reconocimiento de billetes: Procesamiento y reconocimiento de imágenes

En este punto se ha llegado al momento cúlmine de este trabajo formal. La red neuronal ya se encuentra lista y entrenada dentro del dispositivo móvil Android. Luego se ha obtenido una imagen a través de la cámara del celular de un billete y se ha procesado a través de varios algoritmos para lograr el mismo tipo de entrada con que se realizó el entrenamiento. Solamente resta introducir esta imagen final en el procesamiento de nuestra red para que la misma nos indique si puede determinar y con qué grado de acierto, a qué billete corresponde según el entrenamiento recibido.

Para ello, nuestra aplicación tomará la imagen y la procesará a través del método con nombre "masProbable" (el cual se espefició oportunamente en este trabajo) y arrojará si pudo o no reconocer la imagen. El procedimiento desde el celular para realizar esta tarea es muy sencillo y consta de los siguientes pasos:

Iniciar la aplicación del Trabajo Final de Redes Neuronales y utilizar la opción de “Detector de billetes”.

Figura 119 – Pantalla de inicio de aplicación desarrollada.

Al elegir esta opción, inmediatamente el sistema lanzará la aplicación de la cámara. Obtener la imagen y aguardar el procesamiento que también se produce de manera automática. Luego se indicará el resultado por pantalla acompañado de un audio recordando la primer iniciativa de que esta aplicación sirva para personas con dificultad en la visión, entonces el resultado se les provee por medio de un audio.

Figura 120 – Resultado de detección de billete de 5 pesos en aplicación desarrollada.

En la figura 120 se visualiza el resultado del reconocimiento de la aplicación, simplemente se indica el valor obtenido a través del reconocimiento y el % de acierto que se obtuvo. Esta prueba había sido realizada justamente con un billete de 5 pesos, por lo tanto el valor del reconocimiento fue correcto.

Explicada la forma de proceder para poder utilizar la aplicación, se comienza a realizar una batería de pruebas con todos los billetes para poder determinar conclusiones sobre la aplicabilidad de esta solución. Los resultados se muestran en la tabla 28.

Tabla 28 – Resultados de las pruebas en aplicación desarrollada.

Prueba	Billete probado	Billete obtenido	% de acierto	Observaciones
1	2 Pesos	2 Pesos	92.12%	OK
2	2 Pesos	2 Pesos	87.51%	OK

3	2 Pesos	2 Pesos	88.91%	OK
4	2 Pesos	2 Pesos	98.55%	OK
5	2 Pesos	2 Pesos	96.75%	OK
6	5 Pesos	5 Pesos	95.41%	OK
7	5 Pesos	5 Pesos	67.43%	OK –% bajo
8	5 Pesos	10 Pesos	35.12%	ERROR – debido a incorrecta obtención de los rombos
9	5 Pesos	10 Pesos	22.12%	ERROR – debido a incorrecta obtención de los rombos
10	5 Pesos	5 Pesos	75.53%	OK –% medio
11	10 Pesos	10 Pesos	90.12%	OK
12	10 Pesos	10 Pesos	88.40%	
13	10 Pesos	20 Pesos	45.12%	ERROR – debido a encuadre incorrecto
14	10 Pesos	10 Pesos	75.58%	OK - % medio
15	10 Pesos	10 Pesos	98.03%	OK
16	20 Pesos	10 Pesos	15.95%	ERROR – debido a incorrecta obtención de los rombos
17	20 Pesos	20 Pesos	90.33%	OK
18	20 Pesos	20 Pesos	87.54%	OK
19	20 Pesos	20 Pesos	93.22%	OK
20	20 Pesos	20 Pesos	97.90%	OK
21	50 Pesos	50 Pesos	89.48%	OK
22	50 Pesos	50 Pesos	90.25%	OK
23	50 Pesos	50 Pesos	90.85%	OK
24	50 Pesos	50 Pesos	92.24%	OK
25	50 Pesos	50 Pesos	80.00%	OK

El resultado de estas pruebas como muestra la tabla ha sido muy alentador. De un total de 25 muestras, solo 4 han sido erróneas y otras 3 han tenido baja % de acierto. Con esto podemos evidenciar la siguiente situación: cuando ha habido error, el porcentaje de acierto ha sido bajo o al menos muy distantes de aquellos que fueron proceados de

manera correcta. Lo que nos da una pauta que tal vez sea necesario indicar un algunas restricciones para mejorar la indicación del resultado otenido por el reconocimiento.

Por lo tanto para filtrar y evitar algunos resultados incorrectos, se configura que la aplicación no puede dar una respuesta como correcta si no tiene más de un 80% de seguridad que su procesamiento de reconocimiento sea correcto. En caso que ocurra un reconocimiento por debajo de este valor, se procederá a indicar que no pudo ser correctamente identificado, aunque igualmente se mostrará el valor más probable que no se consideró como válido. Mantener este valor informado se utiliza para continuar estudiando las situaciones en donde no se puede identificar correctamente el billete y dejar posibilidad a mejoras en este apartado.

Con respecto a los errores ocurridos, se debieron a que las fotos fueron tomadas de manera inadecuada. Antes de explicar con detalle por qué se produce esto, destacar que ninguno de los problemas de incorrecto reconocimiento se debió a un problema del algoritmo de reconocimiento, es decir al trabajo de la red neuronal, por el contrario los errores se debieron a falencias en el proceso de binarización de las imágenes. Ocurre que en lo posible, la foto debe ser tomada de un billete que no presente sus características deformadas o alteradas. Algunos de los billetes que se utilizaron para las pruebas contenían dobleces y marcas de desgaste, provocando que por ejemplo al momento de obtener los rombos, la proporción del encuadre se haya corrido teniendo así partes del billete que no solamente eran los rombos, y también manchas de color sobre los mismos rombos imposibilitaron su correcta lectura. También un problema de encuadre se presentó, al haber tomado la foto sobre una superficie negra brillosa donde el flash de la cámara rebotó produciendo que la imagen se recorte inicialmente de manera incorrecta, creyendo el algoritmo que la parte del billete poseía otra extensión.

Figura 121 – Imagen de billete de 10 pesos tomada manualmente utilizando flash.

En la figura 121 se puede ver el brillo del flash sobre el lienzo negro. Para evitar estos problemas, es recomendable tomar la foto sobre un lienzo negro opaco que en el caso de utilizar la cámara con flash no intervenga en la toma de la foto y encuadre del billete.

También es necesario destacar que la cantidad de pruebas no se limitó a esas 25 solamente, por lo contrario cientos y cientos (sino miles) de pruebas se han realizado a lo largo del trabajo, solo que documentar cada una de ellas sería desviar la atención sobre otros puntos no importantes del proyecto. A su vez, se han realizado diversas pruebas alentando al fallo, es decir, tomando imágenes de manera incorrecta sobre superficies incorrectas, no siguiendo los supuestos establecidos, pero claramente este tipo de casos fueron sacados de los 25 mostrados ya que la intención es justamente respetar estas condiciones para conocer las bondades del reconocimiento y algoritmos aplicados, basados y empoderados sobre los principios que se han establecido.

Así y todo, adelántonos un poco a las conclusiones del trabajo, los resultados dan margen a la mejora. Si bien en el uso de la aplicación y sobre características poco mutables los resultados se muestran por demás de satisfactorios, queda mucho camino para recorrer para disminuir estas condiciones, para paliarlas lo más posible y encontrar caminos alternativos. Procesos de mejora puede haber en cada punto que el trabajo ha ido desarrollando, desde la elección de la red neuronal, su entrenamiento, los algoritmos y procedimientos para el manejo de imágenes, y su finalización con el reconocimiento. Pero si de algo estamos seguros es que el trabajo provee de un análisis de muchas de las aristas o factores que se ven involucrados en este tipo de proyectos, lo cual puede servir

para cualquier otra persona o grupo de trabajo que se vea involucrado en esta clase de aplicaciones.

5 – CONCLUSIONES

La motivación para la realización de este trabajo fue desde el inicio poder detectar una problemática que afecte a una parte de la sociedad y poder contribuir en una búsqueda de la solución para ellos. Dentro de esa búsqueda surgió la dificultad que conllevan diariamente las personas con discapacidad visual para el uso del dinero en efectivo.

Durante el trabajo estudiamos y describimos el concepto de discapacidad y más puntualmente el de discapacidad visual y las problemáticas a las que se enfrentan las personas que la padecen.

Luego realizamos un estudio de nuestro sistema de billetes con la identificación para personas no videntes y lo comparamos con algunos otros países. Detectamos que en nuestro sistema de billetes existe una marca identificatoria por relieve pero con el desgaste se termina borrando. Otros países como Chile o los pertenecientes a la Unión Europea dieron una solución práctica variando el tamaño del billete según el valor.

Después de detectar la problemática comenzamos a examinar soluciones, investigaciones y trabajos sobre la misma y las enunciamos en el trabajo. Luego nos introducimos en la búsqueda de herramientas, tecnologías y modelos para desarrollar nuestra solución. En esa búsqueda realizamos diferentes estudios y comparativas que nos fueron guiando y llevando a la solución propuesta.

Creemos que nuestra solución es un prototipo que debe ser mejorado pero que hemos cumplido con nuestro objetivo de brindar un aporte a la sociedad. El mismo estará a disposición de quien lo requiera para poder trabajarlo abiertamente. Las mejoras deberán centrarse en la toma de las fotografías en condiciones adversas y en la adecuación a los nuevos billetes que fueron surgiendo desde nuestro estudio hasta el momento.

Como último enunciado en esta conclusión, debemos aclarar que si bien es muy interesante y llena de motivación poder brindarles una solución a las personas que sufren discapacidad visual y conviven con esta problemática, la misma debe ser una política de fondo y como en otros países, que el mismo billete sea desarrollado de tal manera que brinde el reconocimiento sin necesidad de visualizarlo.

6 – REFERENCIAS

BIBLIOGRAFIA

- **Discapacidad:** https://es.wikipedia.org/wiki/Discapacidad
- **Discapacidad:** http://www.who.int/topics/disabilities/es/
- **Ceguera y discapacidad visual:** http://www.who.int/es/news-room/fact-sheets/detail/blindness-and-visual-impairment
- **Red neuronal artificial:** https://es.wikipedia.org/wiki/Red_neuronal_artificial
- **Redes Neuronales: Conceptos Básicos y Aplicaciones:** https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf
- **¿Qué es Android?:** https://www.xatakandroid.com/sistema-operativo/que-es-android
- **La historia de Android:** https://www.android.com/intl/es-419_mx/history/#/marshmallow
- **10 Entornos de programación para desarrollar apps Android sin Java:** https://www.yeeply.com/blog/entornos-programacion-desarrollar-apps-android/
- **10 razones para usar Android:** http://www.tudosisgeek.com/10-razones-para-usar-android-ventajas/
- **JAVA:** https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)
- **Qué es Java y qué necesitas para comenzar:** https://desarrolloweb.com/articulos/iniciacion-java-caracteristicas-programacion-ideeclipse.html

- **Java Garbage Collection Basics**::http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

- **¿Qué es la máquina virtual de Java o Java Virtual Machine? :** https://www.campusmvp.es/recursos/post/que-es-la-maquina-virtual-de-java-o-java-virtual-machine.aspx

- **Algoritmo de Canny: Wikipedia – Algoritmo de Canny:** https://es.wikipedia.org/wiki/Algoritmo_de_Canny

- **Looktel Money Reader:** http://www.looktel.com/moneyreader

- **MCT Money Reader en Google Play:** https://play.google.com/store/apps/details?id=com.mctdata.ParaTanima

- **PROYECTO EXPERIMENTAL INTI:** https://www.inti.gob.ar/noticiero/2015/noticiero465.htm

- **Leaves Recognition Project:** https://sourceforge.net/projects/lrecog/

- **Detección de bordes en una imagen:** http://www4.ujaen.es/~satorres/practicas/practica3_vc.pdf

- **BackPropagation Network for Image Recognition (BP Simplified Demo):** https://www.codeproject.com/Articles/19323/Image-Recognition-with-Neural-Networks

- **Aplicación de redes neuronales en java – Pablo Borbón:** http://pabloborbon.com/2011/06/07/aplicacion-de-redes-neuronales-en-java/