‘SEP SES TNM

INSTITUTO TECNOLÓGICO DE CHIHUAHUA II

SOFTWARE DE RECONOCIMIENTO ÓPTICO DE CARACTERES

TESIS

PARA OBTENER EL GRADO DE

MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA

DAVID FERNANDO ROMO MARÍN

DIRECTOR DE TESIS

M. I. S. C. JESÚS ARTURO ALVARADO GRANADINO

CHIHUAHUA CHIH 8 DE MARZO DEL 2018

D i c t a m e n

Chihuahua, Chihuahua, 05 de agosto del 2020

M.C. MARÍA ELENA MARTÍNEZ CASTELLANOS
COORDINADORA DE POSGRADO
Presente. -

Por medio de este conducto el comité tutorial revisor de la tesis para obtención de grado de Maestro en Sistemas Computacionales, que lleva por nombre "SOFTWARE DE RECONOCIMIENTO ÓPTICO DE CARACTERES", que presenta el (la) C. DAVID FERNANDO ROMO MARÍN, hace de su conocimiento que después de ser revisado ha dictaminado la APROBACIÓN del mismo.

Sin otro particular de momento, queda de Usted.

Atentamente

La Comisión de Revisión de Tesis.

M.I.S.C. JESÚS ARTURO ALVARADO GRANADINO
Director de Tesis

DR. HERNÁN DE LA GARZA GUTIÉRREZ
Co-Director

M.C. ARTURO LEGARDA SÁENZ
Revisor

DRA. MARISELA IVETTE CALDERA FRANCO
Revisor

INDICE

I. INTRODUCCIÓN

1.1 Introducción

Cuando se tiene información en forma de documento impreso y se desea capturarla para su posterior procesamiento mediante una computadora, existen dos opciones; la primera consiste en introducirla a través del teclado, labor larga y pesada, la otra posibilidad es automatizar esta operación por medio de una aplicación de OCR (software especializado en el reconocimiento óptico de caracteres) adecuado, que reduciría considerablemente el tiempo de entrada de datos (Ordoñez, 2009).

El uso de una aplicación OCR permite extraer el contenido textual de documentos en formato de imagen, a fin de realizar diversas funciones de gestión documental, tales como la búsqueda de cierta información, la extracción de información y traducir el documento escrito contenido en una imagen a un archivo con formato de texto plano.

Una persona promedio escribe con un teclado de computadora aproximadamente a la velocidad de 35 palabras por minuto y en una página hay alrededor de 500 palabras, por lo que escribir los datos de una imagen de un archivo .pdf o .jpg, tomaría alrededor de 15 minutos, lo que representa una tarea laboriosa (Jakóbczak, 2015).

En la vida laboral y escolar se generan continuamente grandes cantidades de información escrita, tipográfica o manuscrita semanalmente. La introducción de caracteres utilizando una aplicación OCR, evita la entrada por teclado e implica un importante ahorro de tiempo y un aumento de la productividad (Paellasoft Software Development, 2019).

Existe un gran número de aplicaciones OCR. Además de reconocer palabras de imágenes, también son útiles para el reconocimiento de matrículas, facturas y códigos de barra (Jakóbczak, 2015). Esta tecnología tiene especial aplicabilidad en áreas informáticas, administrativas y de documentación. Algunas manejan archivos con diversos formatos de imagen (pdf, jpg, tiff, etc.). Esto es debido fundamentalmente a que la base documental de este tipo de aplicaciones comúnmente se obtiene tras realizar un proceso de digitalización (por ejemplo, el escaneo) de los documentos impresos.

El software que se desarrolla en esta investigación permitirá el reconocimiento óptico de caracteres por medio de técnicas de inteligencia artificial, automáticamente, a partir de un archivo pdf. Se encargará de reconocer símbolos que pertenecen a un determinado alfabeto, para luego almacenarlos en forma de datos en un archivo .txt.

1.2 Planteamiento del Problema

En la División de Estudios del Posgrado e Investigación del Instituto Tecnológico de Chihuahua II en el área de Sistemas Inteligentes se pretende desarrollar un traductor de español a inglés y viceversa, el cual consta de las siguientes etapas (Ver Figura 1.1):

1. Captura de texto en formato de imagen.
2. Módulo de Reconocimiento Óptico de Caracteres.
3. Módulo de Reconocimiento de Nombres Propios.
4. Módulo de Clasificación por tipo de palabras.
5. Módulo Traductor.

Figura 1.1. Esquema de Traducción Automática.

La escuela requiere de un sistema que le permita capturar datos a partir de una imagen automáticamente, y así poder traducir documentos de español a inglés y viceversa, con la finalidad de poder utilizarlo posteriormente, como lo puede ser la renta de la aplicación.

Este trabajo de tesis se basará en la creación del módulo de Reconocimiento Óptico de Caracteres y a partir de ese módulo se construirá el resto del proyecto. La continuación del proyecto será responsabilidad de otro alumno de la Institución. El siguiente alumno le agregará un traductor al proyecto por medio de técnicas de procesamiento de lenguaje natural y además la convertirá en una aplicación móvil.

Por lo tanto, se plantean las siguientes preguntas de investigación:

¿Cómo capturar los datos que están inmersos en una imagen para que posteriormente sea convertida a un archivo .txt?

¿Se podrá construir un módulo OCR que acepte una imagen de entrada y a la salida genere un archivo .txt?

1.3 Alcances y Limitaciones

Alcances

- El software es una aplicación de escritorio que se enfoca en capturar el texto que está escrito en una imagen escaneada y lo almacenara en un archivo .txt.
- Los idiomas a detectar serán el español e inglés.
- Detectará letra impresa.

Limitantes

- La falta de experiencia en programación y el hecho de que se desconocen las técnicas de inteligencia artificial como el reconocimiento de patrones y las redes neuronales hace que el desarrollo del proyecto tome más tiempo que lo planeado.
- No detectará letra manuscrita.
- Se cuenta con 4 semestres para obtener el grado de maestría.

1.4 Justificación

- Con el módulo implementado se logra ahorrarse 15 minutos por cada página al transcribirla a un documento .txt.
- Este módulo forma parte de un proyecto más grande (construcción del traductor).
- El Instituto Tecnológico de Chihuahua II es una escuela pública y no cuenta con los recursos para poder adquirir una aplicación OCR y tampoco pagar una cuota anual por concepto de soporte. Además, es difícil que se autorice la compra de este tipo de aplicaciones.
- Si por alguna razón se decidiera comprar la aplicación, las empresas que lo venden no te proporcionan el código fuente, por lo tanto, sería inútil para la escuela, ya que lo requiere para poder hacerle modificaciones en un futuro. Además, si la empresa desapareciese, la institución dejaría de tener dicho soporte.

1.5 Objetivo General

Desarrollar un módulo aplicando las técnicas de inteligencia artificial como las Redes neuronales y Reconocimiento de Patrones, para el reconocimiento óptico de caracteres de una imagen escaneada y lo almacene en un archivo .txt, con la finalidad de ahorrar tiempo, evitar errores en la captura y eficientizar el proceso.

Objetivos Específicos

- Investigar los tipos de aplicaciones OCR para conocer cuales hay en existencia en el mercado.
- Determinar las técnicas para el reconocimiento de caracteres de un archivo pdf con el propósito de escoger la técnica más óptima.
- Conocer el proceso del reconocimiento óptico de caracteres con la finalidad de saber cómo se va a realizar el proyecto.
- Aprender la programación necesaria para desarrollar el proyecto.
- En caso de que la imagen tenga imperfecciones o ruido, entonces se deberán eliminar, ya que se necesita una imagen clara y limpia.
- Segmentar el archivo a capturar y clasificar cada parte para obtener una mejor detección de caracteres.
- Reconocer los caracteres por medio de técnicas de inteligencia artificial y archivarlos en un archivo .txt para finalizar el proyecto.

1.6 Hipótesis

Con las técnicas de Reconocimiento de Patrones y Redes Neuronales se logrará capturar el texto, almacenarlo en un archivo .txt y procesarlo, lo que disminuye el tiempo de captura y eficientiza el proceso.

A continuación, se presentan los elementos que se tomaron a considerar al realizar este proyecto**:**

- Tiempo de captura.
- Incrementar la cantidad de páginas en una hora.
- Eficientizar el proceso.

II. ESTADO DEL ARTE

El reconocimiento óptico exacto de la escritura latina impresa, ahora se considera en gran parte un problema solucionado. La exactitud excede el 97%, requiriendo la revisión humana para corrección de errores.

Actualmente está en proceso de investigación el reconocimiento alfanumérico de los escritos a mano. Se ha obtenido una exactitud aproximadamente de entre 80 a 90% en caracteres, pero esto contiene decenas de errores por cada página, haciendo la tecnología útil solamente en contextos muy limitados. El reconocimiento del texto impreso en letra cursiva y en otras lenguas sigue en desarrollo (especialmente en los que tienen un número muy grande de caracteres).

Los algoritmos de OCR de vanguardia hoy en día usan técnicas de redes neuronales y aprendizaje automático (Machine Learning) como ConvNets, LTSM y la mayoría de ellos son técnicas cerradas y patentadas. Las redes convolucionales + BiDirectional LSTM proporcionan un muy buen rendimiento en OCR (Ordoñez, 2009).

Los sistemas que reconocen el texto impreso a mano han gozado de éxito comercial estos últimos años. Entre éstos están dispositivos de asistencia personales digitales (PDA) tales como los Palm OS. Apple es pionero en esta tecnología. Los algoritmos usados en estos dispositivos toman ventaja del orden, velocidad y dirección de las líneas o segmentos individuales en su entrada son conocidos. También, el usuario puede haber aprendido habilidades nuevas para utilizar solamente formas específicas de la letra (por ejemplo, un triángulo sin su base correspondería a la letra A). Estos métodos no se pueden utilizar en software que escaneen documentos en papel, por lo que el reconocimiento exacto de documentos impresos a mano sigue siendo en gran parte un problema abierto al desarrollo. Se pueden alcanzar índices de exactitud del 80 al 90% en caracteres impresos a mano, pero esta exactitud todavía se traduce en docenas de errores por cada página, haciendo la tecnología útil solamente en contextos muy limitados. Esta variedad de OCR ahora se conoce comúnmente en la industria como ICR, o el Reconocimiento Inteligente de Caracteres.

El reconocimiento del texto cursivo es un campo de investigación activo, con medidas de reconocimiento incluso más baja que el del reconocimiento de texto impreso a mano. Índices más altos del reconocimiento de la escritura cursiva general no serán probablemente posibles sin el uso de la información del contexto o gramatical. Por ejemplo, el reconocimiento de palabras enteras de un diccionario es más fácil, que intentando analizar caracteres individuales de la escritura. La lectura de la línea del monto de un cheque, es un ejemplo donde usar un diccionario más pequeño especializado en escritura de números, puede aumentar tarifas del reconocimiento enormemente. El conocimiento de la gramática de la lengua puede también ayudar a determinar si una palabra probablemente sea un verbo o un sustantivo, por ejemplo, permitiendo mayor exactitud.

Para problemas más complejos del reconocimiento, se usan los sistemas de reconocimiento inteligente de caracteres, pues las redes neuronales artificiales que los componen, trabajan indiferentes a las transformaciones lineales y no lineales del proceso de reconocimiento. Una variante del OCR es el OMR (Optical Mark Recognition) que se utiliza para reconocimiento de marcas. Una aplicación sería la corrección automática de exámenes de tipo test, en los que la respuesta correcta se rodea con un círculo, tipo PSU (Prueba de Selección Universitaria).

Hay algunos métodos comunes para reconocer caracteres impresos y manuscritos. Rahman y Fairhurst en 1998, explotaron la clasificación de expertos múltiples para proporcionar nuevos enfoques para el procesamiento de datos impresos. Cuatro conocidos algoritmos de reconocimiento de caracteres escritos a mano (esquema ponderado binario (BWS), esquema de frecuencia ponderada (FWS), clasificadores de patrones por momentos (MPC), perceptrón multicapa y propagación de retorno (MLP)) se utilizan para reconocer caracteres impresos en bases de datos específicas, con fuentes limitadas y una precisión del 97.16%. Se presenta un sistema para la identificación y el reconocimiento de textos manuscritos y mecanografiados a partir de imágenes de documentos utilizando modelos ocultos de Markov (HMM) por Huaigu en el 2011.

Moussa en el 2008, propone la identificación automática multilingüe del árabe y el latín tanto en manuscrito como impreso. Este método se basa en el análisis global de texturas, mediante la extracción de características multidimensionales fractales. El sistema que propone se aprueba para 1000 prototipos con varios tipos y tamaños de fuentes. La tasa de discriminación de precisión es de aproximadamente del 96.64% mediante el uso de KNN, y del 98.72% mediante el uso de RBF.

Dhandra presenta un enfoque basado en momentos invariantes modificados, para el reconocimiento de caracteres ingleses multifondos. El trabajo trata caracteres ingleses aislados que están normalizados a un tamaño de 33 × 33 píxeles y la imagen se adelgaza. Para el tamaño y la invariancia de la traducción, se evalúan los momentos invariantes modificados sugeridos por Palaniappan. El sistema se entrena y se prueba en 7 estilos de fuente diferentes con 7280 imágenes en tamaños a partir de 8 a 72 y la tasa de éxito es 99.65% (B.V. Dhandra, 2008).

Lakshmi y Cols en 2009 propusieron un nuevo algoritmo de esqueletización que utiliza una estructura modificada de Bloque de adyacencia de bloques (BAG) para reconocer caracteres. Se prueba el rendimiento computacional en tres fuentes populares y tamaños de un guion indio, Telugu, y se demuestra que el método de hecho amplifica la falta de uniformidad entre los diferentes caracteres y la similitud entre los mismos caracteres en diferentes fuentes (C.V. Lakshmi, 2009).

Se hizo un sistema para desarrollar un algoritmo para el reconocimiento de vocales Kannada aisladas impresas a máquina y números de diferentes tamaños y estilos de fuente que utilizan momentos invariantes modificados y son invariables con respecto a rotación, escala y traducción por Hangarge Mallikarjun y Dhandra. Se adoptó una clasificación mínima para el clasificador de vecinos más cercanos. El algoritmo propuesto se experimenta en 1800 imágenes de vocales

y 1000 imágenes de números. Los resultados experimentales confirman la precisión de reconocimiento a partir del 97.7% para las vocales y el 98.92% para los números (P.S. Hangarge Mallikarjun, 2010).

Una forma de modelado contextual iterativo que aprende modelos de caracteres directamente del documento que está tratando de reconocer es propuesto por Kae en el 2011. Estos modelos aprendidos se utilizan tanto para segmentar los caracteres como para reconocerlos en un proceso incremental e iterativo. Los resultados muestran una precisión del 98.1% al reconocer un documento en inglés en una fuente dominante. La velocidad de este proceso iterativo es de aproximadamente de 8 a 12 horas (Kae, 2011).

Se han realizado varias investigaciones para diferentes idiomas para resolver este problema y reconocer caracteres multifondos. Slimane representó un sistema de reconocimiento árabe multifondos y de tamaños múltiples. Este sistema se basa en Hidden Markov Model Toolkit (HTK) y Bernoulli HMM (BHMM), es decir, HMM en el que las funciones de densidad de mezcla de Gauss convencionales se sustituyen por funciones de probabilidad de mezcla de Bernoulli. Varias pruebas evaluaron la precisión en la base de datos APTI. Las imágenes de prueba presentadas en el sistema son las que se representan con una fuente en los tamaños 6, 8, 12, 18 y 24; se logra una precisión del 98.3%. Al probar 5 fuentes, la precisión se reduce alrededor del 10-20% (F. Slimane, 2011).

Sukhiha en el 2013, propone un sistema de reconocimiento manuscrito e impreso que utiliza operaciones morfológicas. Se extrae un conjunto de características estructurales prominentes para distinguir con precisión un carácter del otro. El proceso de clasificación utiliza un clasificador de árbol de decisión, donde en cada nodo las reglas de decisión se definen mediante algunas operaciones morfológicas, hasta que se realiza la realización final. Los árboles de decisión se han optimizado para el rendimiento según los algoritmos de clasificación. Los resultados obtenidos son prominentes y la precisión del sistema es, en promedio, del 95% para el texto escrito a mano y para el texto impreso en una fuente, se obtiene una precisión del 99%.

Siriteerakul (2013) propuso e investigó el rendimiento de un sistema de clasificación que utiliza un histograma de gradiente orientado, como una característica de imagen con la máquina de vectores de soporte, como una herramienta de clasificación para reconocer caracteres mixtos de tailandés e inglés. Los experimentos se realizaron en los conjuntos de datos proporcionados por NECTEC que consta de más de 600,000 imágenes impresas de caracteres individuales de 142 clases distintas y se puede lograr una precisión del 97%.

Rani propone un sistema de múltiples tamaños para la mejor detección de caracteres. Los experimentos con características de Gabor basadas en la frecuencia direccional y las características del gradiente basadas en la información del gradiente de un personaje individual para identificarlas como Gurumukhi o inglés cuentan con 2431 muestras y 4862 muestras de prueba encontradas en 17 fuentes diferentes en tamaños de 10 a 28 y se logra una precisión del 96.47% y 98.08% para las características Gradiente y Gabor, respectivamente (R. Rani, 2013).

Las restricciones contextuales de la HMM mejoran significativamente el rendimiento de identificación sobre el método con base convencional modelo de mezcla gaussiana (GMM). La identificación del tipo se usa luego para estimar las tasas de muestreo de cuadros y el ancho del marco de las secuencias de características para el sistema HMM OCR para cada tipo de forma independiente. Este enfoque dependiente del tipo para calcular la frecuencia de muestreo del cuadro y el ancho del cuadro muestra una mejora significativa en la precisión del OCR sobre los enfoques independientes del tipo (Samadiani, 2015).

Tomando en cuenta todas las investigaciones que se analizan alrededor del tema de tesis, es importante recalcar que, a pesar de los extensos estudios realizados para reconocer caracteres, los métodos existentes tienen un problema importante, ya que son muy sensibles a las fuentes, los tamaños y el modo de caracteres (cursiva, negrita y regular) en la fase de prueba. Este problema particularmente causa una mayor tasa de error cuando usamos los métodos existentes para reconocer caracteres en fuentes diferentes en lugar de las muestras de entrenamiento. Por lo tanto, el tamaño de la base de datos de capacitación de los métodos existentes es demasiado grande para que tengan un representante por fuente y tamaño durante la etapa de capacitación. Además, la cantidad de fuentes disponibles aumenta con el tiempo para satisfacer diferentes gustos. Por lo tanto, los métodos OCR existentes no pueden reconocer estas muestras en fuentes nuevas y deben cambiarse. Pero si hubiera un sistema de OCR resistente a los cambios en las fuentes de las muestras, no habría ninguna falla en el reconocimiento de las muestras.

III. MARCO TEÓRICO

En este apartado se describe la teoría en la que se basa la investigación.

3.1 El lenguaje de Programación Java

Java es un lenguaje de propósito general, concurrente, basado en clases y orientado a objetos. Está diseñado para ser lo suficientemente simple como para que muchos programadores puedan lograr fluidez en el lenguaje. El lenguaje de programación Java está relacionado con C y C ++, pero está organizado de forma bastante diferente, con una cantidad de aspectos de C y C ++ omitidos y algunas ideas de otros lenguajes incluidos. Se pretende que sea un lenguaje de producción, no un lenguaje de investigación, por lo que, como sugirió C. A. R. Hoare en su artículo clásico sobre diseño de lenguaje, el diseño ha evitado incluir características nuevas y no probadas.

Fue originalmente desarrollado por James Gosling, de Sun Microsystems (la cual fue adquirida por la compañía Oracle), y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems.

Además, Java está fuertemente y estáticamente tipado. Esta especificación distingue claramente entre los errores de tiempo de compilación que pueden y deben detectarse en tiempo de compilación, y los que ocurren en tiempo de ejecución. El tiempo de compilación normalmente consiste en traducir programas a una representación de código de bytes independiente de la máquina. Las actividades en tiempo de ejecución incluyen la carga y el enlace de las clases necesarias para ejecutar un programa, la generación de código de máquina opcional y la optimización dinámica del programa, y la ejecución real del programa.

Java es un lenguaje de nivel relativamente alto, en el que los detalles de la representación de la máquina no están disponibles a través del lenguaje. Incluye administración automática de almacenamiento, generalmente utilizando un recolector de basura, para evitar los problemas de seguridad de la desasignación explícita. Las implementaciones de alto rendimiento recogidas de basura pueden tener pausas acotadas para admitir la programación de sistemas y las aplicaciones en tiempo real. El lenguaje no incluye construcciones inseguras, como accesos a arreglos sin verificación de índice, ya que tales construcciones inseguras causarían que un programa se comporte de una manera no especificada. (James Gosling, 2015).

El lenguaje se basa en pensar que hay en el mundo real objetos y esos objetos tienen un tipo, o clase. Por ello el lenguaje se basa en clases, que describen cómo son los objetos. Por ejemplo, el lenguaje tiene una clase que describe archivos, una que describe cadenas de texto, o bien nosotros podemos crear clases, como por ejemplo la clase Persona que describe los datos que interesan de una persona.

Por ello siempre para comenzar a trabajar con un programa java hay que crear una clase (ver ejemplo siguiente):

```
public class TablaMultiplicar{
}
```

Además, se deben de cumplir las siguientes características:

- La clase se debe de llamar exactamente igual que el archivo que la contiene.
- La clase que se llama igual que el archivo debe de estar precedida de la palabra *public*.

Cuando se intenta ejecutar una clase java, lo que hace la máquina virtual es llamar a un método especial llamado main que debe de estar dentro de la clase a ejecutar:

```
public class TablaMultiplicar{
    public static void main(String arg[]){
    }
}
```

Y es dentro de la función main donde escribiremos el código que queremos que se ejecute:

```
public class TablaMultiplicar{
    public static void main(String arg[]){
        int numero = Integer.parseInt(arg[0]);
        for(int i = 1 ; i<=10 ; i++){
            System.out.println(""+numero+" * "+i+" = "+(i*numero));
        }
    }
}
```

3.2 El lenguaje de programación Python

Python presenta una serie de ventajas que lo hacen muy atractivo, tanto para su uso profesional como para el aprendizaje de la programación (Andres Marzal, 2014). Entre las más interesantes desde el punto de vista didáctico tenemos:

- Python es un lenguaje muy expresivo, es decir, los programas Python son muy compactos: un programa Python suele ser bastante más corto que su equivalente en lenguajes como C. (Python llega a ser considerado por muchos un lenguaje de programación de muy alto nivel).
- Python es muy legible. La sintaxis de Python es muy elegante y permite la escritura de programas cuya lectura resulta más fácil que si utilizáramos otros lenguajes de programación.
- Python ofrece un entorno interactivo que facilita la realización de pruebas y ayuda a despejar dudas acerca de ciertas características del lenguaje.
- El entorno de ejecución de Python detecta muchos de los errores de programación que escapan al control de los compiladores y proporciona información muy rica para detectarlos y corregirlos.
- Python puede usarse como lenguaje imperativo procedimental o como lenguaje orientado a objetos.
- Posee un rico juego de estructuras de datos que se pueden manipular de modo sencillo. Estas características hacen que sea relativamente fácil traducir métodos de cálculo a programas Python.

Los lenguajes de programación no permanecen inmutables a lo largo del tiempo: evolucionan. Python no es una excepción. A partir de la experiencia con una versión del lenguaje y de la influencia que ejercen otros lenguajes sobre los programadores, hay una presión constante por hacer que el lenguaje ofrezca nuevas capacidades o simplifique el modo en el que se expresan ciertos cálculos. Python fue diseñado inicialmente por Guido van Rossum a partir de su experiencia colaborando con otros en el desarrollo de un lenguaje experimental: ABC. La World Wide Web aparecía al poco de crearse la primera versión de Python y ayudaba a poner en contacto a miles de programadores en todo el mundo. La elegancia de Python, unida a la aparición de un nuevo medio de comunicación entre especialistas, hicieron que un lenguaje que no provenía de la academia o la industria tuviera un éxito inusitado. Hablamos de los años 90 del pasado siglo, década en la que fue tomando fuerza el concepto de «software libre».

Una activa comunidad de desarrolladores liderada por Guido van Rossum (quien sigue teniendo la última palabra en todas las decisiones) va mejorando el lenguaje progresivamente. Cada nueva versión se marca con una serie de números separados por puntos. Lee, si quieres, el cuadro titulado «Versiones» para entender más sobre la codificación tradicional de versiones de productos software.

3.3 Proceso del Reconocimiento Óptico de Caracteres

Para cualquier sistema de reconocimiento de caracteres hay varias etapas principales, como se muestra en la figura 3.1 y 3.2.

Figura 3.1. Proceso del OCR.

Figura 3.2 Desarrollo del OCR.

Paso 1 - Escaneo Óptico:
A través del proceso de escaneo se captura la imagen digital del documento original. En OCR, se utilizan escáneres ópticos que consisten en un mecanismo de transporte y un dispositivo sensor que convierte la intensidad de la luz en niveles de grises. Los documentos impresos consisten en impresión en negro sobre fondo blanco. Al realizar la imagen multinivel OCR se convierte en una imagen en blanco y negro de dos niveles. Este proceso conocido como umbral se realiza en el escáner para ahorrar espacio de memoria y esfuerzo de cálculo. El proceso de umbralización es importante ya que los resultados del reconocimiento dependen totalmente de la calidad de la imagen de dos niveles. Se usa un umbral fijo donde los niveles de gris por debajo de este umbral son negros y los niveles anteriores son blancos. Para documentos de alto contraste con fondo uniforme, un umbral fijo previamente elegido puede ser suficiente. Sin embargo, los documentos encontrados en la práctica tienen un rango bastante amplio. En estos casos, se requieren métodos más sofisticados de umbralización para obtener buenos resultados. Los mejores métodos de umbralización varían la adaptación del umbral a las propiedades locales del documento, como el contraste y el brillo. Sin embargo, tales métodos generalmente dependen del escaneo multinivel del documento que requiere más memoria y capacidad de cómputo.

Paso 2 - Segmentación de Localización:
La segmentación determina los constituyentes de una imagen. Es necesario ubicar las regiones del documento que tienen datos impresos y se distinguen de las figuras y los gráficos. Por ejemplo, al realizar la clasificación automática de correo a través de los sobres, la dirección debe ubicarse y separarse de otras impresiones, como los sellos y los logotipos de la empresa, antes del reconocimiento. Cuando se aplica al texto, la segmentación es el aislamiento de caracteres o palabras. La mayoría de los algoritmos de OCR segmenta las palabras en caracteres aislados que se reconocen individualmente. Por lo general, la segmentación se realiza aislando cada componente conectado. Esta técnica es fácil de implementar, pero surgen problemas si los caracteres están juntos o fragmentados y constan de varias partes. Los principales problemas en la segmentación son: (a) extracción de caracteres conmovedores y fragmentados, (b) distinguir el ruido del texto, y (c) malinterpretar los gráficos y la geometría con el texto y viceversa.

Paso 3 - Preprocesamiento:
Los datos brutos que dependen del tipo de adquisición de datos están sujetos a una serie de pasos de procesamiento preliminares para que se puedan utilizar en las etapas descriptivas del análisis de caracteres. La imagen resultante del proceso de escaneo puede contener cierta cantidad de ruido. Según la resolución del escáner y el umbral intrínseco, los caracteres pueden mancharse o romperse. Algunos de estos defectos pueden causar tasas de reconocimiento deficientes y se eliminan a través del preprocesador suavizando los caracteres digitalizados. El suavizado implica relleno y adelgazamiento. El llenado elimina pequeños cortes, huecos y agujeros en los caracteres digitalizados, mientras que la reducción reduce el ancho de la línea. La técnica más común para suavizar mueve una ventana a través de una imagen binaria de carácter y aplica ciertas reglas al contenido de la ventana. El preprocesamiento también incluye la normalización

junto con el suavizado. La normalización se aplica para obtener caracteres de tamaño uniforme, inclinación y rotación. La rotación correcta se encuentra a través de su ángulo. Para páginas giradas y líneas de texto, las variantes de la transformada Hough se usan comúnmente para detectar sesgos.

Por lo tanto, el componente de preprocesamiento tiene como objetivo producir datos que sean fáciles de operar para los sistemas OCR. Es una actividad importante que debe realizarse antes del análisis de datos real. Los principales objetivos del preprocesamiento pueden señalarse como: (a) reducción de ruido, (b) normalización de los datos, y (c) compresión en la cantidad de información que se debe conservar. En el resto de esta subsección, los objetivos antes mencionados de los objetivos de preprocesamiento se discuten con las técnicas correspondientes.

En otras palabras, se producen imágenes de documentos limpios que son fáciles para que el reconocimiento de caracteres sea exacto (con menor cantidad de ruido). Se debe de considerar el tamaño de la matriz que posee la información de las características de cada carácter. Además, se convierte el texto a escala de grises.

Paso 4 - Segmentación:
Aquí la imagen del carácter es segmentada en sus subcomponentes. La segmentación es importante porque la extensión que uno puede alcanzar en la separación de las diversas líneas en los caracteres afecta directamente la tasa de reconocimiento. Aquí se usa la segmentación interna que aísla líneas y curvas en los caracteres cursivamente escritos. Las estrategias de segmentación de caracteres se dividen en tres categorías: (a) segmentación explícita, (b) segmentación implícita, y (c) estrategias mixtas.

a) En la segmentación explícita, los segmentos se identifican en función de las propiedades de carácter. El proceso de cortar la imagen del carácter en componentes significativos se logra a través de la disección. La Disección analiza la imagen del carácter sin utilizar una clase específica de información de forma. El criterio para una buena segmentación es el acuerdo de las propiedades generales de los segmentos con los esperados para los caracteres válidos. Los métodos disponibles basados en la disección de la imagen del carácter utilizan espacios en blanco y tono, análisis de proyección vertical, análisis de componentes conectados y puntos de referencia. La segmentación explícita puede someterse a evaluación utilizando el contexto lingüístico.
b) La estrategia de segmentación implícita se basa en el reconocimiento. Busca en la imagen los componentes que coinciden con las clases predefinidas. La segmentación se realiza utilizando la confianza de reconocimiento, incluida la corrección sintáctica o semántica del resultado global. En este enfoque, se emplean dos clases de métodos: métodos que hacen que algún proceso de búsqueda y métodos que segmentan una representación característica de la imagen. La primera clase intenta segmentar caracteres

en unidades sin el uso de algoritmos de disección basados en características. La imagen se divide sistemáticamente en muchas piezas superpuestas sin tener en cuenta el contenido. Estos métodos se originan a partir de esquemas desarrollados para el reconocimiento de palabras impresas en máquina. El principio básico es usar una ventana móvil de ancho variable para proporcionar secuencias de segmentaciones tentativas que son confirmadas por OCR. La segunda clase de métodos segmenta la imagen implícitamente por clasificación de subconjuntos de entidades espaciales recopiladas de la imagen como un todo. Esto puede hacerse a través de cadenas de Markov ocultas o enfoques no basados en Markov. El enfoque no de Markov se deriva de los conceptos utilizados en la visión artificial para el reconocimiento de objetos ocluidos [1, 3]. Este enfoque basado en el reconocimiento utiliza la relajación probabilística, el concepto de regularidades y singularidades y el emparejamiento hacia atrás.

c) Las estrategias mixtas combinan la segmentación explícita e implícita de una manera híbrida. Se aplica un algoritmo de disección a la imagen del carácter, pero la intención es sobre el segmento, es decir, cortar la imagen en suficientes lugares como para que los límites de segmentación correctos estén incluidos entre los cortes realizados. Una vez que esto esté asegurado, se busca la segmentación óptima mediante la evaluación de subconjuntos de los cortes realizados. Cada subconjunto implica una hipótesis de segmentación y la clasificación se utiliza para evaluar las diferentes hipótesis y elegir la segmentación más prometedora. El problema de segmentación se formula como la búsqueda de la ruta más corta de un gráfico formado por una imagen de documento de nivel binario y gris. Las probabilidades ocultas de la cadena de Markov obtenidas a partir de los caracteres de un algoritmo de disección se utilizan para formar un gráfico. La ruta óptima de este gráfico mejora el resultado de la segmentación por disección y el reconocimiento oculto de la cadena de Markov. Las estrategias mixtas arrojan mejores resultados en comparación con los métodos de segmentación explícitos e implícitos. Los mecanismos de detección y corrección de errores a menudo están integrados en los sistemas. El uso inteligente del contexto y la confianza del clasificador generalmente conduce a una mayor precisión.

Paso 5 - Representación:

La representación de la imagen juega uno de los roles más importantes en cualquier sistema de reconocimiento. En el caso más simple, el nivel de gris o las imágenes binarias se alimentan a un reconocedor. Sin embargo, en la mayoría de los sistemas de reconocimiento para evitar una complejidad adicional y aumentar la precisión de los algoritmos, se requiere una representación más compacta y característica. Para este propósito, se extrae un conjunto de características para cada clase que ayuda a distinguirlo de otras clases mientras permanece invariante a las diferencias características dentro de la clase. Los métodos de representación de imágenes de caracteres generalmente se clasifican en tres grupos principales: (a) transformación global y series expansión, (b) representación estadística, y (c) representación geométrica y topológica.

En conclusión, el objetivo principal de la representación es extraer y seleccionar un conjunto de características que maximice la tasa de reconocimiento con la menor cantidad de elementos. La extracción y selección de características se define [1, 3] como la extracción de la información más representativa de los datos en bruto, lo que minimiza la variabilidad dentro del patrón de clase al tiempo que mejora la variabilidad entre patrones de clase.

Paso 6 - Extracción de Características:
El objetivo de la extracción de características es capturar las características esenciales de los símbolos. La extracción de características se acepta como uno de los problemas más difíciles de reconocimiento de patrones. La forma más directa de describir el carácter es mediante una imagen de trama real. Otro enfoque es extraer ciertas características que caracterizan los símbolos, pero deja los atributos sin importancia. Las técnicas para la extracción de tales características se dividen en tres grupos: (a) distribución de puntos, (b) transformaciones y expansiones de series, y (c) análisis estructural. Los diferentes grupos de características se evalúan según su sensibilidad al ruido, deformación, facilidad de implementación y uso. Los criterios utilizados en esta evaluación son: (a) robustez en términos de ruido, distorsiones, variación de estilo, traducción y rotación, y (b) uso práctico en términos de velocidad de reconocimiento, complejidad de implementación e independencia. Algunas de las técnicas de extracción de características comúnmente utilizadas son la coincidencia de plantillas y la correlación, las transformaciones, la distribución de puntos y el análisis estructural.

Otra tarea importante asociada con la extracción de características es la clasificación. La clasificación es el proceso de identificar a cada carácter y asignarle una clase de carácter correcta. Las dos categorías importantes de enfoques de clasificación para la OCR son los métodos teóricos y estructurales de decisión. En el reconocimiento teórico de decisión, la descripción del carácter se representa numéricamente en el vector de características. También puede haber características de patrón derivadas de la estructura física del carácter que no se cuantifican tan fácilmente. Aquí, la relación entre las características puede ser importante a la hora de decidir sobre la pertenencia a una clase. Por ejemplo, si sabemos que un carácter consiste en un trazo vertical y uno horizontal, puede ser 'L' o 'T'. La relación entre dos trazos es necesaria para distinguir los caracteres. Los principales enfoques para el reconocimiento teórico de decisiones son los clasificadores de distancia mínima, los clasificadores estadísticos y las redes neuronales. En reconocimiento estructural, los métodos sintácticos son los enfoques más prevalentes.

Paso 7 - Entrenamiento y reconocimiento:
Los sistemas de reconocimiento de caracteres utilizan ampliamente las metodologías de reconocimiento de patrones que asignan muestras desconocidas a clases previamente definidas. Los sistemas OCR usan ampliamente las metodologías de reconocimiento de patrones que asignan una muestra desconocida a una clase predefinida. Los OCR se investigan en cuatro enfoques generales de reconocimiento de patrones, como se sugiere en: (a) comparación de

plantillas, (b) técnicas estadísticas, (c) técnicas estructurales, y (d) redes neuronales artificiales. Estos enfoques no son ni necesariamente independientes ni desarticulados entre sí. Ocasionalmente, una técnica de OCR en un enfoque también puede considerarse como un miembro de otros enfoques. En todos los enfoques anteriores, las técnicas de OCR utilizan estrategias holísticas o analíticas para las etapas de capacitación y reconocimiento. La estrategia holística emplea enfoques descendentes para reconocer el carácter completo eliminando el problema de segmentación. El precio de este ahorro computacional es limitar el problema de OCR a un vocabulario limitado. Además, debido a la complejidad introducida por la representación de un solo carácter o trazo, la precisión del reconocimiento disminuye. Por otro lado, las estrategias analíticas emplean un enfoque ascendente desde el trazo o el nivel de carácter y van hacia la producción de un texto significativo. Los algoritmos de segmentación explícitos o implícitos son necesarios para esta estrategia, no solo al agregar complejidad adicional al problema sino también al introducir un error de segmentación en el sistema. Sin embargo, con la cooperación de la etapa de segmentación, el problema se reduce al reconocimiento de caracteres simples o trazos aislados, que pueden manejarse para un vocabulario ilimitado con altas tasas de reconocimiento.

Paso 8 - Postprocesamiento:

Algunas de las actividades de postprocesamiento comúnmente utilizadas incluyen la detección y corrección de grupos y errores. Al agrupar, los símbolos en el texto están asociados con cadenas. El resultado del reconocimiento de símbolo simple en el texto es un conjunto de símbolos individuales. Sin embargo, estos símbolos generalmente no contienen suficiente información. Estos símbolos individuales están asociados entre sí formando palabras y números. La agrupación de símbolos en cadenas se basa en la ubicación de los símbolos en el documento. Los símbolos que están lo suficientemente cerca se agrupan. Para fuentes con proceso de agrupamiento de tono fijo es fácil ya que se conoce la posición de cada carácter. Para caracteres tipográficos, la distancia entre caracteres es variable. La distancia entre las palabras es significativamente mayor que la distancia entre los caracteres y la agrupación es por lo tanto posible. Los problemas ocurren para los caracteres escritos a mano cuando el texto está sesgado. Hasta agrupar cada carácter se trata por separado, el contexto en el que aparece cada carácter no se ha explotado. Sin embargo, en problemas avanzados de reconocimiento óptico de texto, el sistema que consiste únicamente en el reconocimiento de caracteres individuales no es suficiente. Incluso los mejores sistemas de reconocimiento no darán una identificación correcta del 100% de todos los caracteres. Solo algunos de estos errores se detectan o corrigen mediante el uso del contexto. Hay dos enfoques principales. El primero utiliza la posibilidad de secuencias de caracteres que aparecen juntos. Esto se hace usando reglas que definen la sintaxis de la palabra. Para diferentes idiomas, las probabilidades de dos o más caracteres que aparecen juntas en secuencia se pueden calcular y se utilizan para detectar errores. Por ejemplo, en el idioma inglés, la probabilidad de que k aparezca después de h en una palabra es cero y si se detecta dicha combinación, se asume un error. Otro enfoque es el uso de diccionarios, que es el método de detección y corrección de errores más eficiente. Dada una palabra en la que hay un error y la

palabra se busca en el diccionario. Si la palabra no está en el diccionario, se detecta un error y se corrige cambiando la palabra a la palabra más similar. Las probabilidades obtenidas de la clasificación ayudan a identificar el carácter erróneamente clasificado. El error transforma la palabra de una palabra legal a otra y tales errores son indetectables por este procedimiento. La desventaja de los métodos del diccionario es que las búsquedas y las comparaciones consumen mucho tiempo. (Arindam Chaudhuri, 2016).

3.4 Técnicas utilizadas en los OCR

Las técnicas que se utilizarán en este proyecto son: el reconocimiento de patrones y las redes neuronales.

3.4.1 El reconocimiento de patrones

El reconocimiento de patrones además de ser una rama de Machine Learning, es un estudio de cómo las máquinas pueden observar el entorno, aprender a distinguir patrones de interés de sus orígenes y tomar decisiones sensatas y razonables. El único propósito de este método es el clasificar un grupo de patrones conocido como conjunto de pruebas en dos o más clases de categorías. Esto es logrado al calcular las categorías del conjunto en prueba comparándolo con un conjunto de entrenamiento (previo) o training set. Un clasificador dado mide la distancia entre varios puntos dados (compara), para saber cuáles puntos son más cercanos a la meta en un modelo parametrizado.

Un patrón de entrada se compara con la representación almacenada. La identidad está determinada por la selección de la plantilla con la mayor cantidad de coincidencias. El reconocimiento de patrones consiste de varios elementos (ver Figura 3.3):

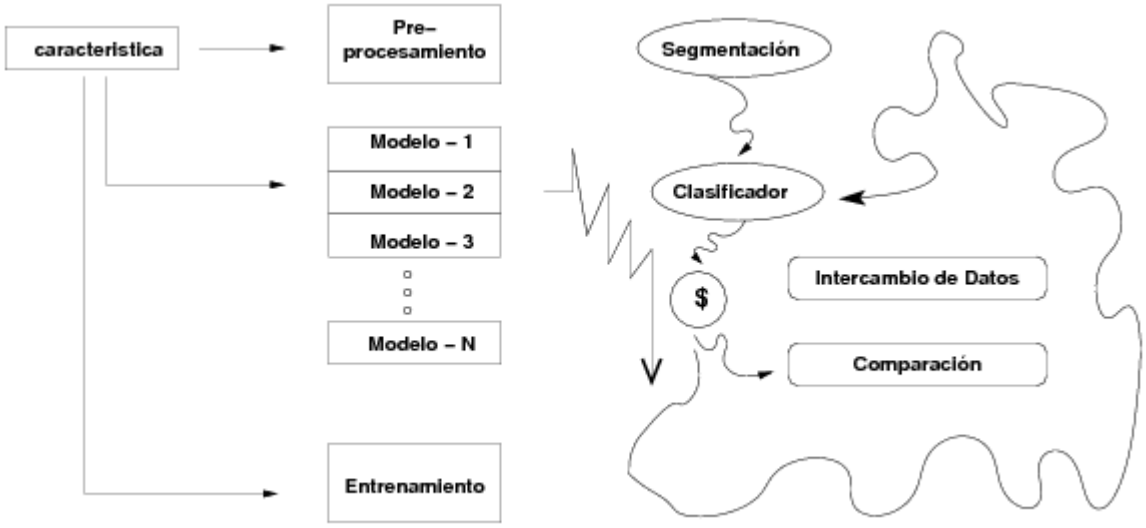


Figura 3.3 Elementos del Reconocimiento de Patrones.

Sensor: Se le llama así al mecanismo que sensa los datos o la entrada de la información.

Segmentación y Agrupamiento: La operación de segmentación ocurre cuando el sistema determina que un elemento, objeto o muestra finaliza y da comienzo a otro. Los patrones individuales deben ser segmentados y focalizados.

Extracción de Características: La meta del extractor es caracterizar un objeto con medidas o cualidades cuyos "valores" tienden a ser similares. Para objetos en la misma categoría las diferencias son mínimas y por lo tanto las características son invariables y poco relevantes a cambios en datos leídos por el sensor.

Clasificación: El objetivo en la operación de clasificación es utilizar un "vector" con las características provistas por el extractor para asignar el objeto (patrón) de la entrada a una categoría.

Preprocesamiento y Segmentación: Este paso es necesario para simplificar las siguientes operaciones en el reconocimiento sin que se pierda la información relevante al modelo de patrón.

Procesamiento a posteriori: Es utilizado para recomendar decisiones y acciones que dependen de un costo o riesgo particular. En el procesamiento a posteriori se utiliza la descarga o resultado del clasificador para recomendar una acción. En teoría esta etapa produce la tasa de aciertos o errores y califica al clasificador. Por lo tanto y en este sentido el objetivo del post procesador es buscar un mínimo de errores y fallas.

Aprendizaje o Entrenamiento: Cualquier método que incorpora información sobre un ejemplo o conjunto entrenamiento en el diseño del clasificador, necesariamente emplea algún tipo de aprendizaje, razón básica para considerar nociones sobre aprendizaje. (Samadiani, 2015)

3.4.2 Redes Neuronales Artificiales

La Red Neuronal Artificial (ANN, por sus siglas en inglés) es un paradigma de procesamiento de información inspirado en la forma en que los sistemas nerviosos biológicos, como el cerebro, procesan la información. El elemento clave de este paradigma es la nueva estructura del sistema de procesamiento de la información. Se compone de una gran cantidad de elementos de procesamiento altamente interconectados (neuronas) que trabajan al unísono para resolver problemas específicos. Las redes neuronales, como las personas, aprenden con el ejemplo. Una red neuronal se configura para una aplicación específica, como reconocimiento de patrones o clasificación de datos, a través de un proceso de aprendizaje. Las redes neuronales son una colección de células muy simples y masivamente interconectadas. Las células están dispuestas de una manera que cada célula deriva su entrada desde una o más células. Está vinculada a través de conexiones ponderadas a una o más células.

La arquitectura más usada en la actualidad de una red neuronal (como la presentada en la figura 3.4) que consiste en:

- Una primera capa de entradas, que recibe información del exterior.
- Una serie de capas intermedias (ocultas), encargadas de realizar el trabajo de la red.
- Una capa de salidas, que proporciona el resultado del trabajo de la red al exterior.

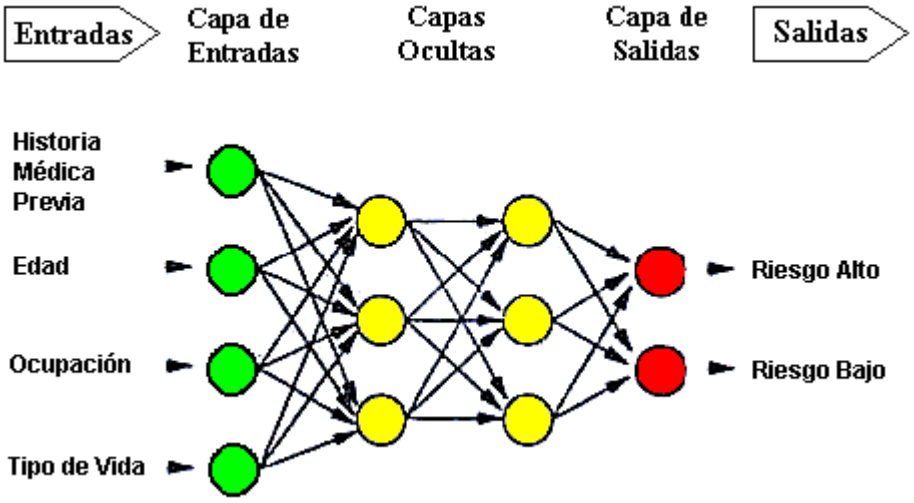


Figura 3.4 Esquema de una Red Neuronal.

Cada neurona de la red es una unidad de procesamiento de información; es decir, recibe información a través de las conexiones con las neuronas de la capa anterior, procesa la información, y emite el resultado a través de sus conexiones con las neuronas de la capa siguiente, siempre y cuando dicho resultado supere un valor de "umbral".

El procesamiento de la información llevado a cabo por cada neurona Y, consiste en una función (F) que opera con los valores recibidos desde las neuronas de la capa anterior (X_i, generalmente 0 o 1), y que tiene en cuenta el peso sináptico de la conexión por la que se recibieron dichos valores (W_i). Así, una neurona dará más importancia a la información que le llegue por una conexión de peso mayor que no a aquella que le llegue por una conexión de menor peso sináptico.

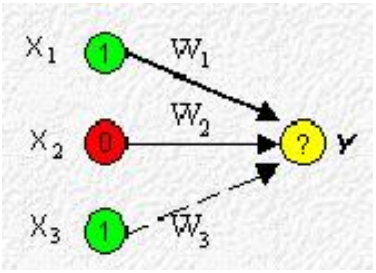


Figura 3.5 Conexiones de diferente peso sináptico convergen en Y (W1 > W2 > W3).

Un modelo simple de la función F sería:

$$F = X_1W_1 + X_2W_2 + \ldots + X_iW_i$$

Si el resultado de la función F es mayor que el valor umbral (U), la neurona se activa y emite una señal (1) hacia las neuronas de la capa siguiente. Si el resultado es menor que el valor umbral, la neurona permanece inactiva (0) y no envía ninguna señal:

$$X_1W_1 + X_2W_2 + \ldots + X_iW_i \leq U \leftrightarrow Inactivación \leftrightarrow Y = 0$$

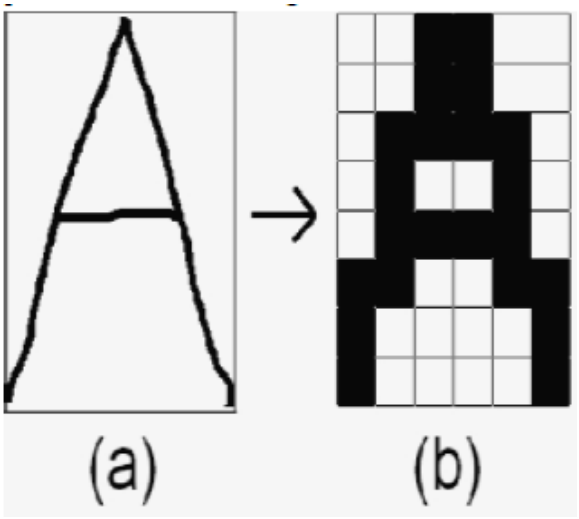
$$X_1W_1 + X_2W_2 + \ldots + X_iW_i > U \leftrightarrow Activación \leftrightarrow Y = 1$$

De esta forma, definido un conjunto inicial de pesos en las conexiones, al presentar un "estímulo" (conjunto de ceros y unos que representa un dato, perfil u objeto) a la capa de entradas, cada neurona en cada capa realiza la operación descrita anteriormente, activándose o no, de manera que al final del proceso las neuronas de la capa de salidas generan un resultado (otro conjunto de ceros y unos), que puede coincidir o no con el que se desea asociar el estímulo.

En el *entrenamiento* de una red neuronal tanto el *peso sináptico* de las conexiones como el *valor de umbral* para cada neurona se modifican (según un algoritmo de aprendizaje), con el fin de que los resultados generados por la red *coincidan* con (o se aproximen a) los resultados esperados.

Para poder aprender, las redes neuronales se sirven de un *algoritmo de aprendizaje*. Estos algoritmos están formados por un *conjunto de reglas* que permiten a la red neuronal *aprender* (a partir de los *datos* que se le suministran), mediante la modificación de los *pesos sinápticos* de las conexiones entre las neuronas (recordar que el umbral de cada neurona se modificará como si fuera un peso sináptico más).

Digitalización de imágenes: el proceso de digitalización es importante para las redes neuronales. En este proceso, la imagen de entrada se muestrea en una ventana binaria que forma la entrada al sistema de reconocimiento. Un ejemplo de este proceso se muestra en la figura a continuación (Ver Figura 3.6):



001100
001100
011110
010010
011110
110011
100001
100001

Figura 3.6 Ejemplo de Digitalización de Imagen.

En la figura, el alfabeto A se digitaliza en 6x8 = 48 celdas, cada una con un solo color negro o blanco. Esto se hace para que la computadora entienda el formulario.

Y en el proceso de digitalización, a la celda con color negro se le asigna un valor +1 y a la celda con color blanco se le asigna un valor 0, para darle una estructura binaria. Y esto crea una matriz de imagen binaria I [3]. Esto hace que la imagen de entrada invariante de las dimensiones reales de la arquitectura de la red neuronal estudiada.

3.5 Explicación de una Red Neuronal Convolucional Simple

Las redes neuronales funcionan de la siguiente manera (Ver figura 3.7):

Primero se empieza con una imagen de entrada, a la cual se le aplican detectores de características (feature detectors) para crear mapas de características (feature maps), y todo eso constituye la capa convolucional. Luego se le aplica la función de activación, que puede ser ReLU o cualquier otra.

Después se le aplica la capa de Pooling donde se reduce el tamaño de la imagen y se elimina la información adicional. Más tarde se le aplica el flattening, donde se acomoda toda la información proveniente del pooling a manera de vector.

Finalmente, en la capa Fully Connected se procesa toda la información y se hace una especie de votación para determinar a qué clase pertenece la imagen. (Eremenko, 2017).

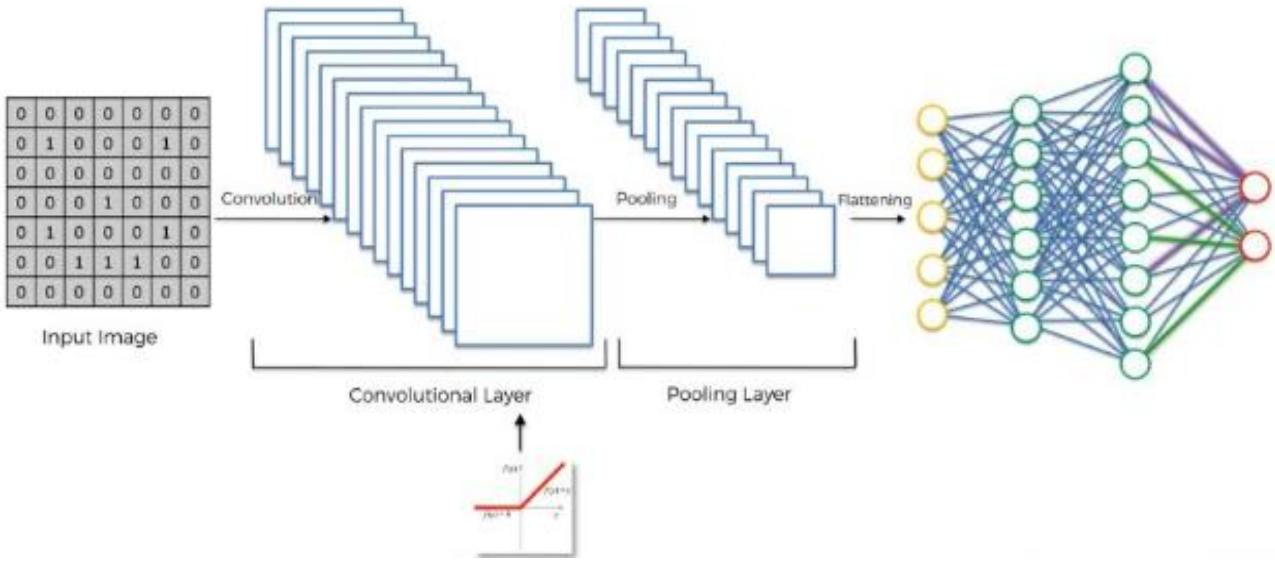


Figura 3.7 Red Neuronal Convolucional Simple.

3.6 Redes Neuronales Recurrentes

Las Redes Neuronales Recurrentes (RNR) son un grupo de redes neuronales especializadas en procesar datos secuenciales, x(1)…, x(t), de longitud variable y con la capacidad de incrementar

el tamaño (longitud) de las secuencias, algo inviable para redes sin especialización basada en secuencias. (Numerentur, 2018).

Opera con tiempo, es decir, por ciclos; acepta un vector de entrada y actualiza su estado oculto a través de funciones de activación no lineales, lo usa para hacer una predicción de su salida. El estado oculto puede almacenar información como representaciones distribuidas de alta dimensión (en oposición al HMM- Hidden Markov Mode, modelo oculto de Markov) y su dinámica no lineal pueden implementar grandes cálculos para realizar tareas de modelado y predicción para secuencias de cadenas complejas (Ver Figura 3.8).

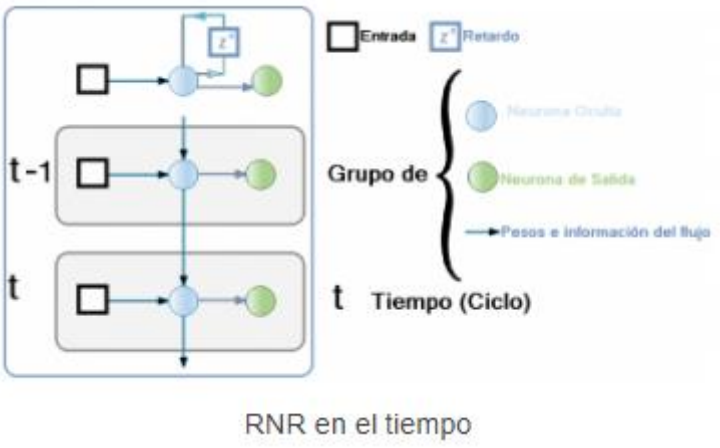


Figura 3.8 Red Neuronal Recurrente en el tiempo.

Las RNR se caracterizan por el tipo de neurona, el esquema de conexión de las mismas (topología) y el algoritmo de aprendizaje empleado para adaptar su función de cálculo a las necesidades del problema a tratar.

Su principal característica es que deben tener, al menos, un circuito de retroalimentación. Esto nos ofrece dos topologías básicas:

- **Una capa:** La red puede consistir en una sola capa de neuronas, cada neurona alimenta con su señal de salida a las entradas de las otras neuronas. En esta estructura no hay bucles de Auto-retroalimentación (Auto-retroalimentación se refiere a una situación donde la salida de una neurona es la entrada de la misma). En el modelo tampoco existen neuronas ocultas (Ver Figura 3.9).

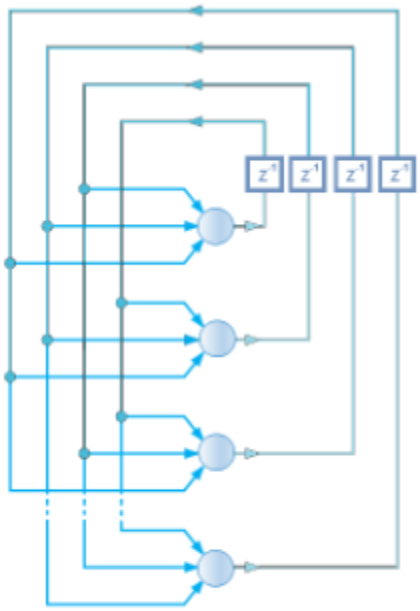


Figura 3.9 Red Neuronal Recurrente de una capa

- **Capa Oculta:** En este modelo aparecen las neuronas ocultas y las conexiones de retroalimentación provienen también de las salidas de las neuronas ocultas (Ver Figura 3.10).

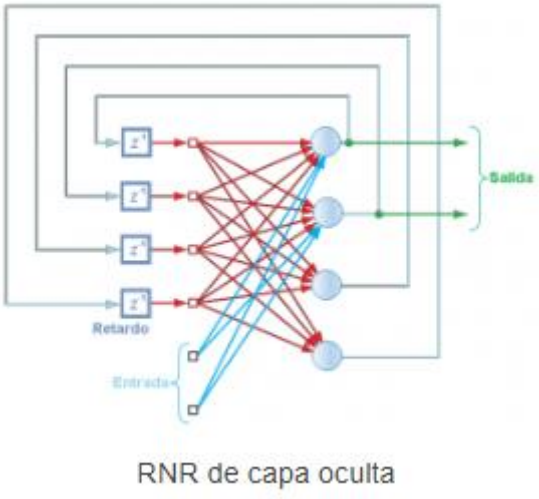


Figura 3.10 Red Neuronal Recurrente de capa oculta.

En ambos modelos se encuentran ramas particulares compuestas de elementos de retardo de tiempo (denominados Z), esto da como resultado una dinámica no lineal de comportamiento. También podemos verla de forma unidireccional con una línea de retardo y desplegada en el tiempo con dos ciclos.

Con base en la función del modelo de conexión las RNR pueden ser:

- **Totalmente recurrentes:** Son aquellas que cada neurona puede estar conectada a cualquier otra y sus conexiones recurrentes son variables, y
- **Parcialmente recurrentes:** Son aquellas que sus conexiones recurrentes son fijas. Estas últimas son la forma usual para reconocer o reproducir secuencias. Generalmente tienen la mayoría las conexiones hacia adelante, pero incluyen un conjunto de conexiones retroalimentadas.

Existen dos formas básicas de aplicar la retroalimentación:

1. Es aplicada a una única neurona dentro de la red, y
2. Engloba una o más o capas ocultas o toda la red que permiten:
    a) Memoria asociativa o por contenido,
    b) Auto-asociación, y
    c) Reconstrucción dinámica de un proceso caótico.

En un breve resumen podríamos decir que su principal ventaja estriba en la posibilidad de almacenar (memorizar) una representación de la historia reciente de la secuencia, lo que permite, a diferencia de lo que ocurre con las redes neuronales no recurrentes, que la salida ante un determinado vector de entrada pueda variar en función de la configuración interna actual de la red. Son más potentes que las redes neuronales hacia adelante (feed forward), debido a su memoria y a que el procesamiento de datos es dinámico.

Su principal inconveniente estriba en la dificultad para entrenarlas, sobre todo en el aprendizaje de contextos relativamente grandes. Inicialmente el modelo de entrenamiento basado en gradientes parece el más aplicable, porque se pueden calcular de forma ligera mediante el algoritmo BPTT (BackPropagation Through Time) pero fallan al entrenar la red con familias que necesitan muchos pasos de tiempo (ciclos). Para superar esta limitación se han propuesto modelos como el LSTM (Long Short Term Memory) y el que utiliza unidades repetidas segmentadas denominadas GRU (Gated Recurrent Units).

3.7 Instalación de Keras

Keras es una API de aprendizaje profundo (Deep learning) escrita en Python, que se ejecuta en la parte superior de la plataforma de aprendizaje automático (machine learning) TensorFlow. Fue desarrollado con un enfoque en permitir la experimentación rápida. Keras es una biblioteca

de Redes Neuronales que está especialmente diseñada para posibilitar la experimentación en más o menos poco tiempo con redes de Aprendizaje Profundo. Sus fuertes se centran en ser amigable para el usuario, modular y extensible (equipo Keras, 2020).

Para instalar Keras se deben seguir los siguientes pasos:

1. Buscar anaconda prompt en el buscador del menú de Windows y proceder a abrir anaconda prompt.
2. Una vez abierta anaconda prompt se procede a ejecutar "conda install –c conda-forge keras", el programa le preguntara que si desea seguir con la instalación. Finalmente se escribe la tecla "y".

3.8 Instalación de Tensorflow

TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos (Dean, 2018).

Para instalar Tensorflow se deben seguir los siguientes pasos:

1. Abrir la página de https://sites.wustl.edu/jeffheaton/t81-558/, seleccionar Course Content/Sessions(Github), y descargar como zip todos los archivos. Descomprimir los archivos de la carpeta zip.
2. Seleccionar t81_558_class01_intro_python.ipynb.
3. Buscar en inicio "Anaconda Prompt" y escribir "conda create --name tensorflow python=3.6" darle enter y después escribir "y". Después escribir actívate tensorflow y darle enter.
4. Escribir conda install jupyter y darle enter. Luego escribir los siguientes paquetes, escribir "y" y darle enter.

    - conda install scipy
    - pip install --upgrade sklearn
    - pip install --upgrade pandas
    - pip install --upgrade pandas-datareader
    - pip install --upgrade matplotlib
    - pip install --upgrade pillow
    - pip install --upgrade requests
    - pip install --upgrade h5py
    - pip install --upgrade psutil

- pip install --upgrade tensorflow==1.12.0
- pip install --upgrade keras==2.2.4

5. Escribir “python” y darle enter.
6. Escribir “import tensorflow as tf”
7. Escribir print(tf.__version__) y verificar que la version sea 1.12.0 , luego escribir quit.
8. Escribir “actívate tensorflow”, luego escribir “python -m ipykernel install --user --name tensorflow --display-name "Python 3.6 (tensorflow)".
9. Escribir “dir” buscar el archivo que se llame t81_558_deep_learning-master y darle enter.
10. Ejecutar Jupyter y abrir el archivo llamado t81_558_class01_intro_python.ipynb.
11. Seleccionar kernel, después Python 3.6(tensorflow) y ejecutar el kernel.

3.9 Importar Opencv y numpy

OpenCV es una biblioteca de código abierto muy utilizada para el desarrollo de aplicaciones de visión artificial. En OpenCV se encuentran muchos algoritmos de tratamiento de imagen y de clasificación de patrones que sirven para crear aplicaciones de visión artificial. OpenCV está desarrollada originalmente en C++. Sin embargo, dado la dificultad de aprendizaje de C++ se ha desarrollado un recubrimiento de Opencv para que pueda ser utilizado desde Python.

Numpy es una biblioteca de cálculo numérico que se integra con OpenCV en numerosos aspectos. Por otro lado, MatPlotLib es una biblioteca de presentación gráfica de resultados para Numpy que resulta útil por motivos obvios. (Vélez, 2017)

Para instalar Opencv y Numpy se deben seguir los siguientes pasos:

Una vez instalado Anaconda hay que instalar Opencv. Para instalar una biblioteca desde Anaconda hay que abrir un terminal (en el caso de Windows una consola de Anaconda) y teclear:

```
conda search opencv
```

Este comando mostrará las versiones de Opencv disponibles en los repositorios oficiales de Anaconda.

Esta vez, en el listado se encuentra Opencv 3.1 y 3.2. Entonces, hay que elegir un repositorio de Opencv 3.2 válido para nuestra plataforma. Por ejemplo, si nuestra plataforma es Linux,

Windows o Mac de 64 bits, para instalar OpenCV 3.2 podemos utilizar el repositorio "conda-forge/opencv3". Para instalar utilizando este repositorio escribiremos:

```
anaconda install -c conda-forge opencv3
```

Una vez instalado Opencv podemos comprobar que funciona correctamente abriendo una consola de python e importando la biblioteca. Para ello, desde la consola de Anaconda tecleamos: python. En ese momento se presentará un mensaje de bienvenida a Python. En ese momento podremos teclear los siguientes comandos. Si no se observan mensajes de error es que todo ha ido bien, pero si aparecen mensajes de error deberán leerse y entenderse para buscar una solución.

```
>>> import cv2
>>> import numpy as np
>>> import matplotlib
```

IV. ANALISIS Y DISEÑO DEL SOFTWARE

4.1 Técnicas de recopilación de información

La técnica para la recopilación de información que se utilizó fue la entrevista. Para ello se discutieron los temas con el director de tesis asignado (Ing. Arturo Alvarado). Él fue el que proporcionó gran parte de la información.

4.2 Captura, definición y validación de requisitos

Captura: Al entrevistar al director de tesis se determinó la necesidad de construir un OCR, definiéndose los requisitos que debería contar el programa.

Los requisitos son:

- Crear una aplicación de escritorio con la finalidad de capturar texto escrito en una imagen escaneada y se almacenará como texto plano en un archivo .txt.
- De preferencia usar el lenguaje Python.
- Detectar únicamente la letra impresa (no manuscrita).
- Los idiomas a detectar serán el español e inglés.
- Usar redes neuronales y visión artificial en la aplicación.
- Escribir una tesis que trate sobre el proyecto a realizar.

Definición: La escuela necesita un sistema que le permita capturar datos a partir de una imagen automáticamente, y así poder traducir documentos instantáneamente, con la finalidad de poder utilizarlo para otros asuntos más importantes dentro de la institución.

Se realizará únicamente el módulo de Reconocimiento Óptico de Caracteres y a partir de ese módulo se construirá el resto del proyecto. La continuación del proyecto será realizada por otro alumno de la Institución. El siguiente alumno le agregará un traductor al proyecto por medio de técnicas de procesamiento de lenguaje natural y además la convertirá en una aplicación móvil.

También se obtuvo que detectará únicamente la letra impresa escrita en el idioma inglés y en español, ya que otro alumno se encargará de agregar un traductor. El lenguaje de programación es opcional, pero el profesor tiene una preferencia con el lenguaje Python, ya que hay más información de inteligencia artificial en el lenguaje mencionado.

Validación de Requisitos: Finalmente se validó la información y se hicieron las preguntas necesarias para determinar si había comprendido bien en qué consistía el proyecto. La información se validó con el director de tesis (Ing. Arturo Alvarado).

4.3 Requisitos funcionales y no funcionales.

Funcionales

- Procesamiento de imágenes, ya que se planean capturar y usar imágenes de caracteres.
- Uso de redes neuronales que se encargarán de ser entrenadas para el reconocimiento de caracteres.
- Reconocimiento de caracteres, que es el resultado final del proyecto.
- Almacenamiento de caracteres en un archivo .txt, para que al final se almacene la información procesada.
- Carpeta conteniendo los tipos de letras que se van a reconocer gracias a la red neuronal.

No Funcionales

- El lenguaje de programación que se utilizará no es tan determinante debido a que se puede desarrollar en cualquier lenguaje de alto nivel.
- La interfaz gráfica de usuario, ya que su contenido puede variar.

4.4 Prototipo

El siguiente diagrama muestra los requisitos que debe llevar el Software de Reconocimiento Óptico de Caracteres (Ver Figura 4.1)

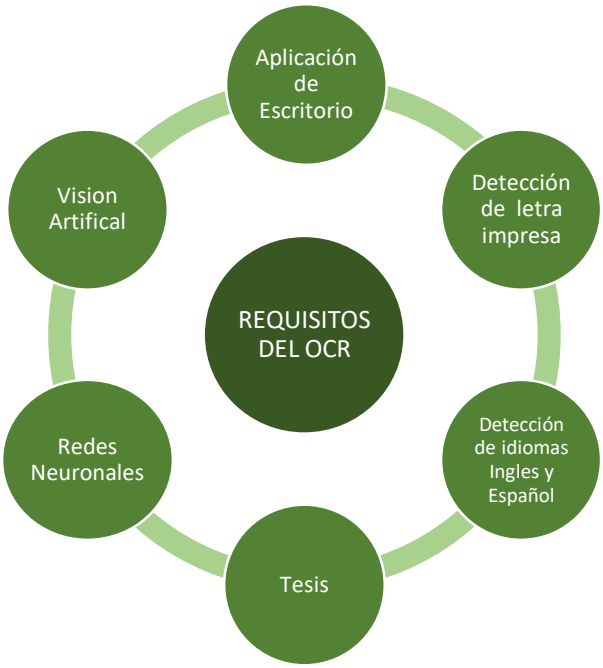


Figura 4.1 Requisitos del Software de Reconocimiento Óptico de Caracteres.

En la siguiente imagen se muestran los pasos necesarios que se necesitan para realizar un software de reconocimiento óptico de caracteres (Ver Figura 4.2).

Figura 4.2 Proceso del OCR

4.5 Problemas y soluciones al hacer el análisis

Al hacer el análisis se tuvo que reflexionar mucho, ya que no se había pensado a profundidad el problema. Por lo cual se consideró todos los temas de las unidades vistas en clase y la información de la clase de seminario.

Uno de los problemas más importantes es que no se tenía un prototipo para la creación de un software de reconocimiento óptico de caracteres. Por lo cual se tuvo que pensar en uno y crear su diseño.

4.6 Diseño

A. Diseño de Entrada

Mi entrada es un archivo pdf que contiene texto impreso.

B. Diseño de Salida

Mi salida es un archivo de texto en donde se guardarán las cadenas de caracteres correspondientes a la imagen de entrada.

C. Diseño de Interfaz de Usuario

A Continuación, se muestra el diseño de la interfaz gráfica de usuario que se tomó como base para realizar el proyecto (Ver Figura 4.3).

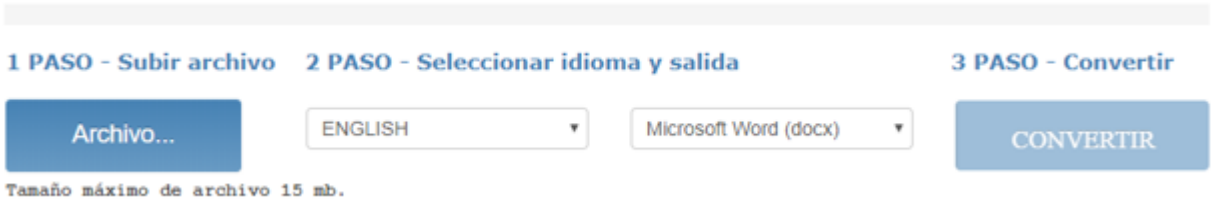


Figura 4.3 Diseño base de la interfaz gráfica de usuario.

Y finalmente se muestra la interfaz gráfica de usuario que se realizó en la alternativa 1 (Ver Figura 4.4) y en la alternativa 2 (Ver Figura 4.5):

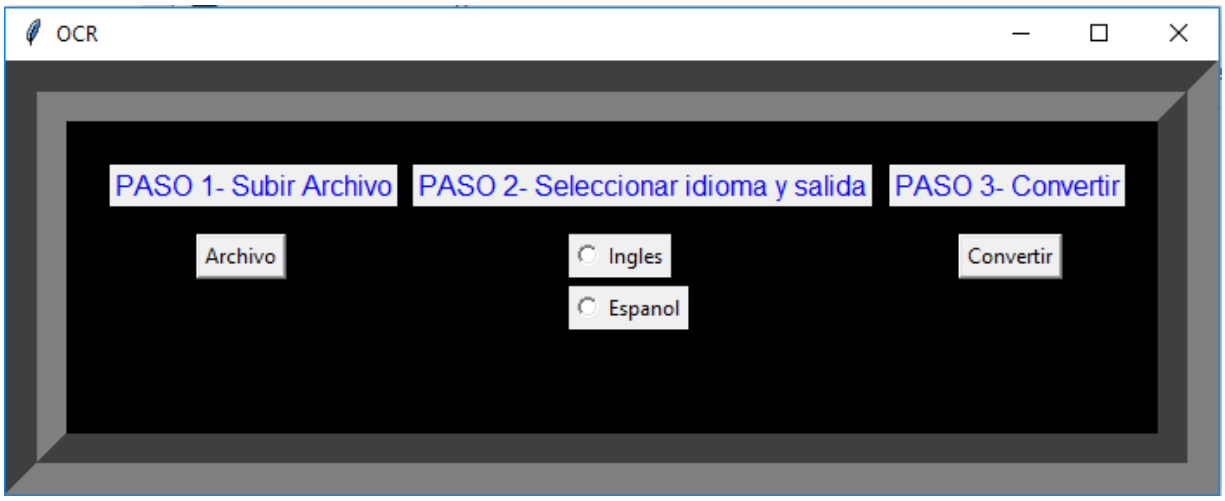


Figura 4.4 Diseño base de la interfaz gráfica de usuario que se realizó en la alternativa 1.

Figura 4.5 Diseño base de la interfaz gráfica de usuario que se realizó en la alternativa 2.

D. Carpeta conteniendo el repositorio de imágenes de los tipos de letras a reconocer
El repositorio de imágenes se va a guardar en una carpeta, para ser utilizada por la red neuronal.

V. DESARROLLO DEL SOFTWARE

En esta sección se describirá como se desarrolló el proyecto OCR. Debido a que se realizaron dos versiones para este proyecto (una en java y otra en python).

La versión en python no detectaba letras acentuadas, ni la letra ñ. Tampoco detectaba documentos pdf, en lugar de eso, detectaba gráficos en pyplot (donde se tenía que ingresar las letras directamente).

Finalmente, la opción que fue tomada en mejor consideración fue la versión realizada en java, ya que está cumplía mejor con los objetivos iniciales del proyecto.

A continuación, se mostrará la explicación de cada una de las alternativas:

VERSION EN PYTHON

5.1 Preprocesamiento de imágenes con Visión Artificial

Se intentaron varios filtros como la ecualización de histograma, sobel y gaussiano, pero no se obtuvieron resultados óptimos para el proyecto. Así que se tomó el texto original, se convirtió a una escala de grises, se aplicó binarización inversa y filtro otsu. Y se obtuvieron los resultados que se ven a continuación (Ver figura 5.1).

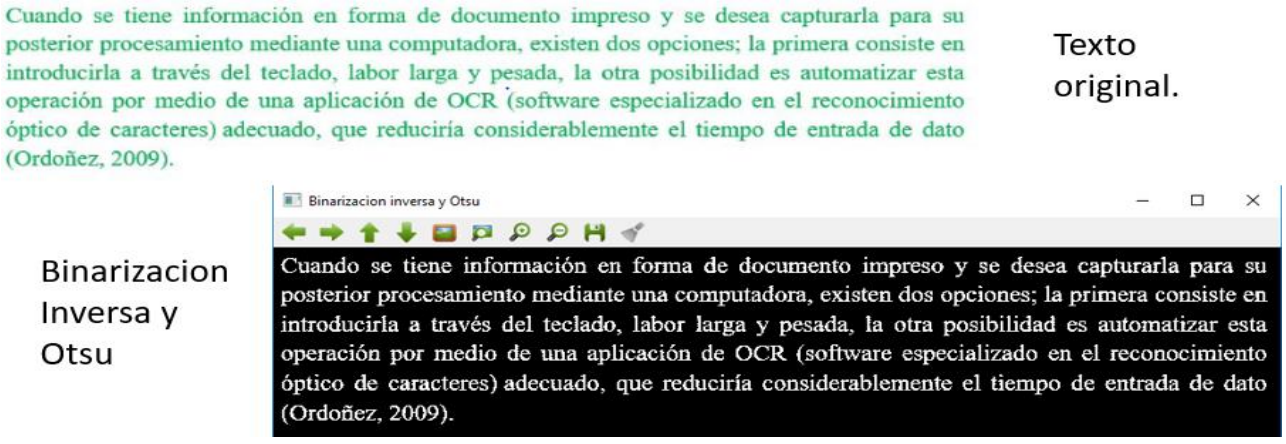


Figura 5.1 Preprocesamiento de imágenes Texto original y Salida.

5.2 Pruebas Realizadas con las Redes Neuronales Convolucionales

Se hicieron pruebas con diferentes redes neuronales para detectar letras:

- Primero se intentó la red neuronal convolucional simple (Ver figura 5.2). Al utilizar esta red para detectar 2 letras la tasa de exactitud fue del 100% (Ver figura 5.3). El problema de esta red neuronal es que, al subir la cantidad de letras a detectar, la tasa de exactitud va disminuyendo considerablemente. Por ejemplo, cuando detecta 4 letras la tasa de exactitud es aproximadamente del 80%.

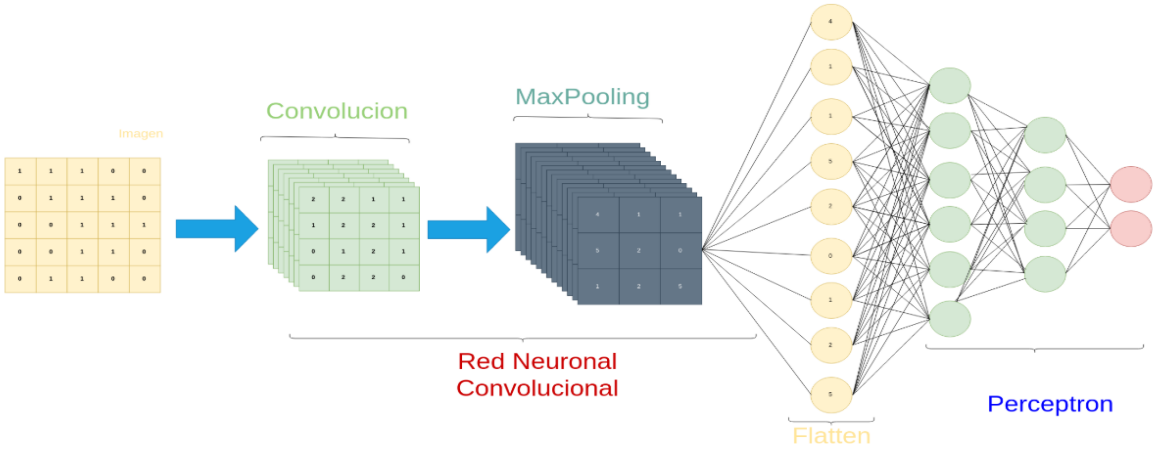


Figura 5.2 Imagen de una red neuronal convolucional simple.

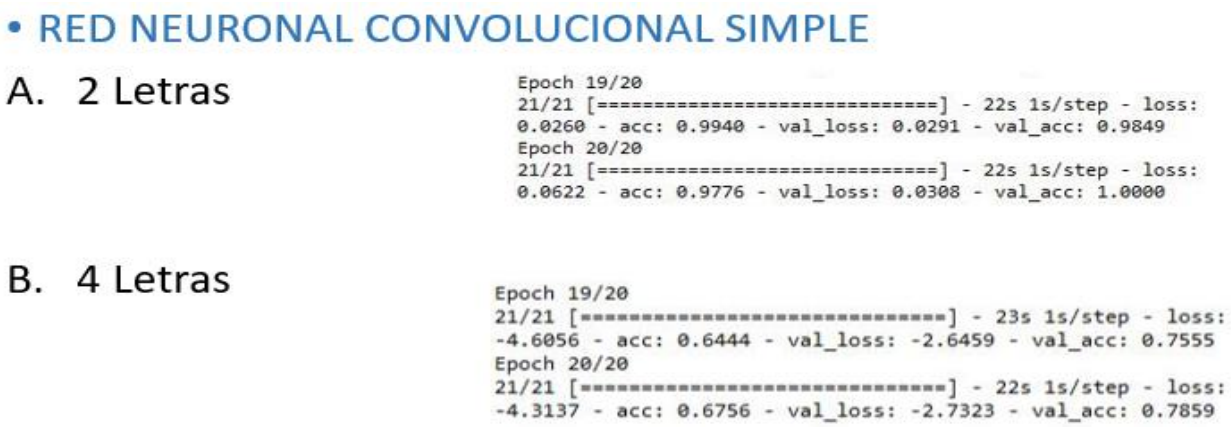


Figura 5.3 Resultados de la Red Neuronal Convolucional Simple.

En conclusión, no se considera que esta sea la red neuronal más óptima para desarrollar el proyecto.

- También se realizaron pruebas con una Alexnet (Ver Figura 5.4). La Alexnet se caracteriza por tener 5 capas convolucionales, 5 de maxpooling y 3 fully connected.

  Se intentaron diferentes optimizadores, funciones de activación y pérdidas. También se intentó cambiar el número de épocas y con diferentes data-sets.

  Con esta red neuronal se obtuvo el mismo problema. Para detectar 2 letras, la tasa de exactitud era alta, pero a la hora de subir la cantidad de letras, la tasa de exactitud bajaba considerablemente. Pero el problema no era tan acentuado como con la red neuronal convolucional simple.

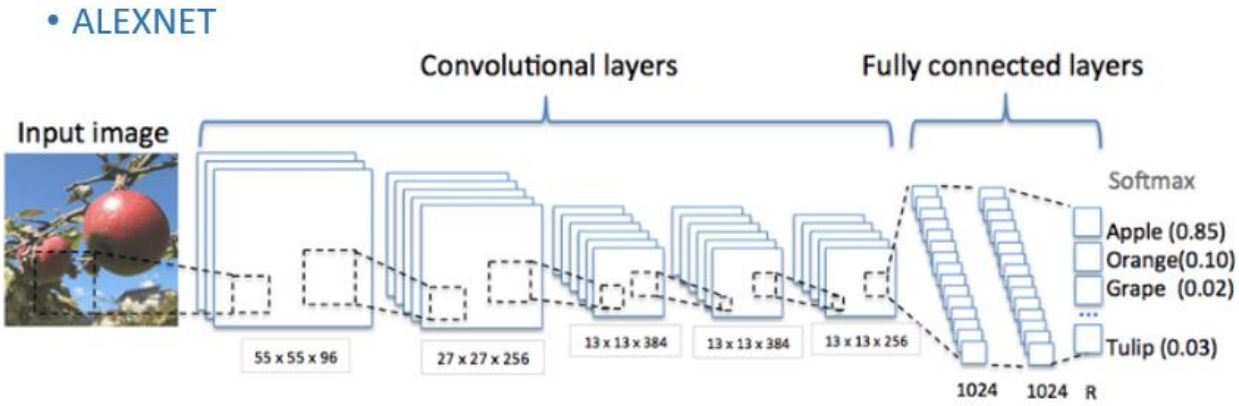


Figura 5.4 Red neuronal convolucional Alexnet.

- El tercer intento fue con una VGG16 (Ver figura 5.5). La VGG16 se caracteriza por tener 13 capas convolucionales, 5 de maxpooling y 3 fully connected.

  Al utilizar esta red para detectar 4 letras la tasa de exactitud fue del 65.6%. El problema de esta red neuronal es que la tasa de exactitud sigue siendo considerablemente baja.

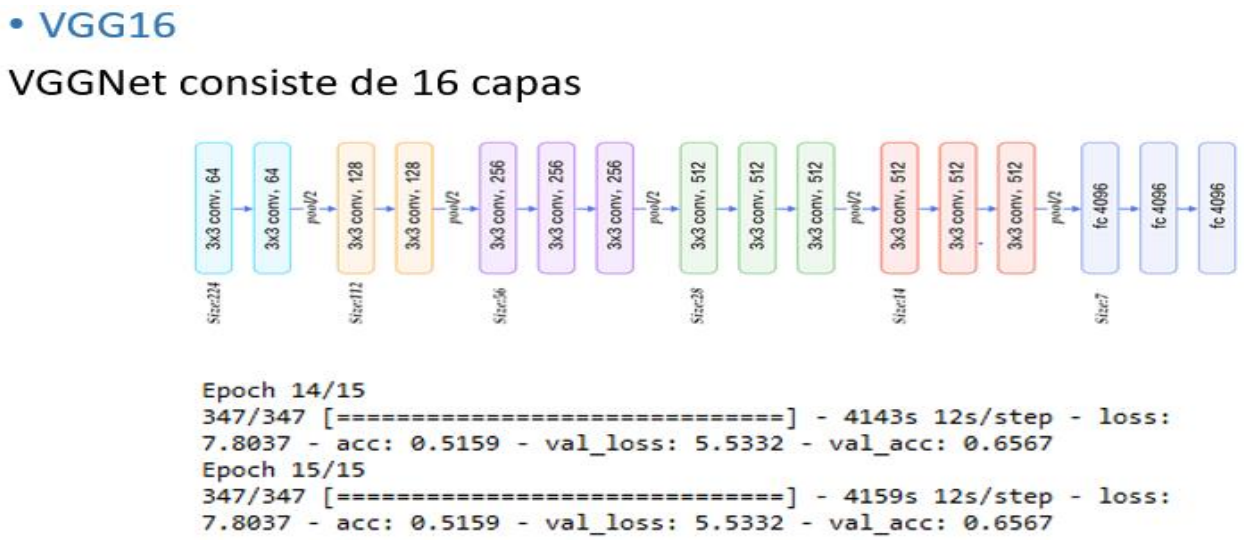


Figura 5.5 Red neuronal convolucional VGG16.

5.3 Resultados finales de las redes neuronales

Matplotlib es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación python y su extensión matemática numpy. Matplotlib.pyplot es una colección de funciones de estilo de comando que hacen que matplotlib funcione como MATLAB.

Al intentar diferentes redes neuronales, finalmente se decidió utilizar 2 capas de convolución, 2 de maxpooling y 2 capas bidireccionales GRU (red neuronal recurrente). En la figura 5.6 se muestra una red neuronal de una máquina de resonancia magnética que es similar a la que se usó en este proyecto.

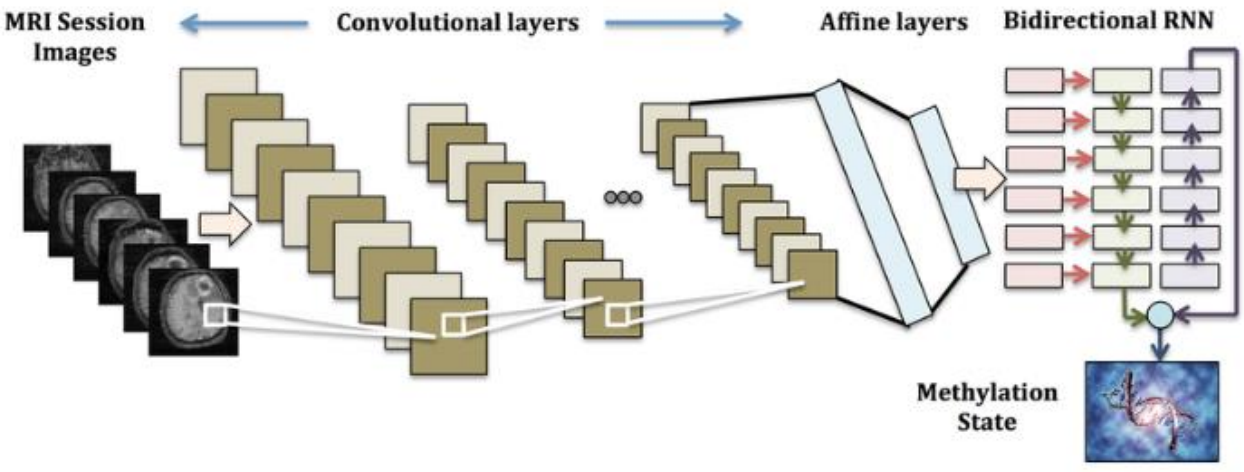


Figura 5.6 Red neuronal de una Maquina de Resonancia Magnética.

Cuando se probó esta red neuronal para que identificara la palabra *Carlos* de la imagen en pyplot (Ver figura 5.7), finalmente dio como resultado la palabra *Carlos* (Además el programa mostró otros 2 posibles resultados), dando una muy alta tasa de exactitud (Ver figura 5.8).

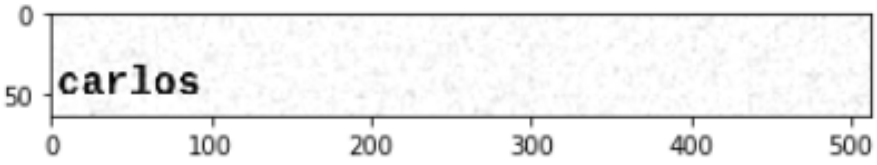


Figura 5.7 Imagen en pyplot donde se utiliza el reconocimiento de caracteres.

```
predit_a_image(a, top_paths = 3)

['carlos', 'tcarlos', 'ocarlos']
```

Figura 5.8 Tres posibles resultados del reconocimiento de la palabra *Carlos*.

VERSION EN JAVA

A continuación, se mostrará la explicación de cada una de las clases que se tienen en este proyecto. También se podrá observar los cálculos de la red neuronal.

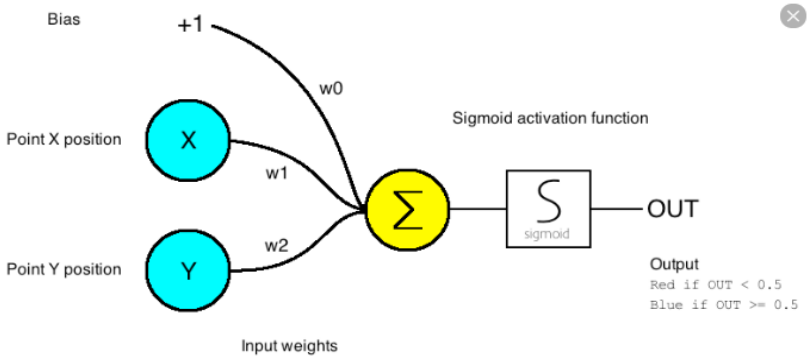
5.3 Calculos del Perceptrón

En la sección 5.13 (Clase Neurona) se puede ver el código que realiza todos los cálculos de la red neuronal, así como también se mostrará una breve explicación.

Se debe entender que los cálculos de esta red neuronal solo contienen 2 entradas, pero en realidad la neurona contiene 1024 entradas (32x32 pixeles). Esto se hizo con el propósito de simplificar los cálculos. También se debe recordar que es una neurona por cada carácter (una para mayúsculas y otra para minúsculas).

Cálculos del Perceptrón

Caso
A = 1
B = 0
S = 0



Entrada (Valor y peso)

Wa = 0.5
Wb = 0.3
Wu = 0.1

Wu , Wa y Wb son aleatorios (del 0 al 1)

Wu * 1
Wa * a
Wb * b
R

Función de Activación (R) = S

Función Procesar

Wu * Umbral Wa * a Wb * b

(0.1 * 1) + (0.5 * 1) + (0.3 * 0) = 0.6 Salida

Función Evaluar

Aplica la función

Sigmoide = $\frac{e \wedge t}{e \wedge t + 1} = \frac{e \wedge 0.6}{e \wedge 0.6 + 1} = 0.6456$

Heavyside
t > 0
Res = 1
t ≤ 0 = 0

e = salida obtenida - salida esperada
e = 0.6456 - 0 > 0.1

Paso de Entrenamiento

	Wa + (e * a)		Wu + (e * u)
Wa =	0.5 + (0.6456 * 1)	Wu =	0.1 + (0.6456 * 1)
Wa =	1.1456	Wu =	0.7456

	Wb + (e * b)
Wb =	0.3 + (0.6456 * 0)
Wb =	0.3

5.5 Clase Glifo

En la clase Glifo se guarda una imagen de la letra, el carácter detectado y las dimensiones.

```
public class Glifo {

    private BufferedImage imagen;
    private String caracter;
    private Interprete interprete;
    private Rectangle bounds;

    public Glifo(Rectangle bounds) {
        this.bounds = bounds;
        caracter = "";
        interprete = Interprete.getInstancia();
        this.imagen = new BufferedImage(Interprete.gx, Interprete.gy, BufferedImage.TYPE_BYTE_BINARY);
    }

    public Glifo(Rectangle bounds, String caracter) {
        this(bounds);
        this.caracter = caracter;
    }

    public Glifo(Rectangle bounds, BufferedImage img) {
        this(bounds);
        imagen = interprete.recortar(img);
        imagen = interprete.ajustar(img);
    }
```

Figura 5.9 Constructores de la clase glifo.

Se puede observar que se encuentran 3 constructores (Ver figura 5.9): En el primer constructor se piden las dimensiones y se crea una imagen binaria. El segundo constructor hace lo del

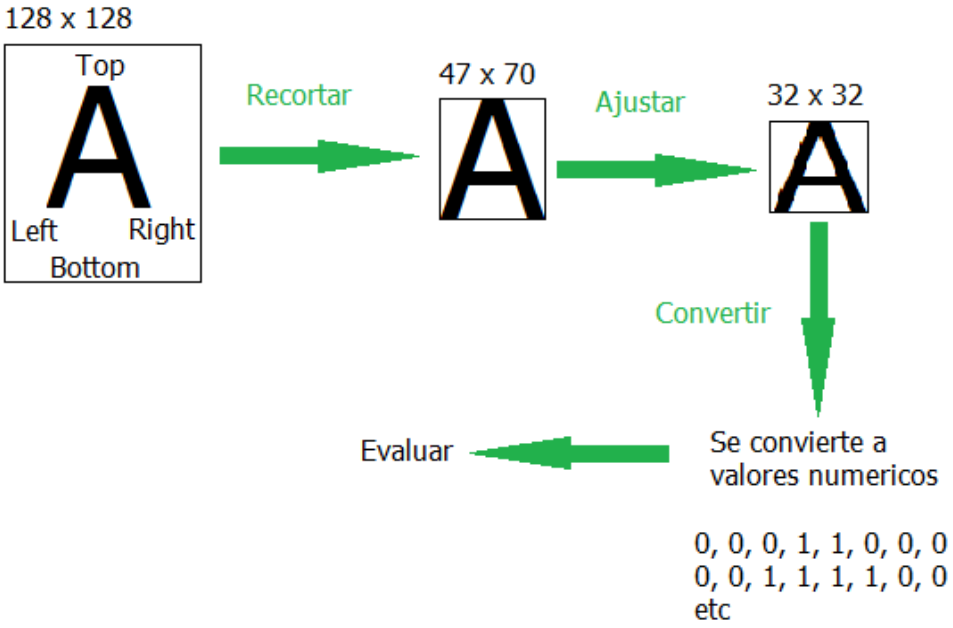
anterior y además asigna el carácter. El tercer constructor hace lo del primer constructor y además toma una imagen, la ajusta y la recorta para adaptarse a las dimensiones dadas.

```
public Rectangle getBounds() {
    return bounds;
}

public void setBounds(Rectangle bounds) {
    this.bounds = bounds;
}

public BufferedImage getImagen() {
    return imagen;
}

public void setImagen(BufferedImage imagen) {
    this.imagen = imagen;
}

public String getCaracter() {
    return caracter;
}

public void setCaracter(String caracter) {
    this.caracter = caracter;
}
```

5.10 Getters y setters de la clase glifo.

En la imagen 5.10 se muestran los getters y los setters de las dimensiones, de la imagen y del carácter.

5.6 Redimensionamiento del Tamaño de la Imagen



5.11 Proceso del redimensionamiento del tamaño de la imagen.

Cada glifo o cada carácter pasa por varias etapas: recortar, ajustar, convertir y evaluar (Ver figura 5.11). En la clase interprete (sección 5.7) se muestra el código que explica cada una de estas etapas.

5.7 Clase Interprete

```
private List<Neurona> neuronas;
public static final int gx = 32, gy = 32;
public static List<String> caracteres;
```

5.12 Sección del código de la clase interprete.

En esta parte del código (Ver figura 5.12) se encuentra una lista de neuronas (también se podría definir como una red neuronal). Las dimensiones del glifo son 32 x 32 pixeles. De la misma manera se encuentran todos los caracteres posibles.

```
public Interprete() {
    instancia = this;

    caracteres = Arrays.asList(
            "a", "b", "c", "d", "e", "f", "g", "h", "i", "j",
            "k", "l", "m", "n", "ñ", "o", "p", "q", "r", "s",
            "t", "u", "v", "w", "x", "y", "z", "á", "é", "í",
            "ó", "ú",
            "A", "B", "C", "D", "E", "F", "G", "H", "J",
            "K", "L", "M", "N", "Ñ", "O", "P", "Q", "R", "S",
            "T", "U", "V", "W", "X", "Y", "Z", "Á", "É", "Í",
            "Ó", "Ú",
            "0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
            "!", "@", "#", "$", "%", "^", "&", "*", "(", ")",
            "/", "?", "+", "=");
    File red = new File("red_neuronal.dat");
    if (red.exists()) {
        neuronas = Helper.deserializar(red);
        System.out.println("Red cargada");
    } else {//si no existe
        neuronas = new ArrayList<>();
        entrenar();
    }
}
```

5.13 Caracteres de la red neuronal.

En la figura 5.13 se muestran cada uno de los caracteres del idioma español (incluyendo letras acentuadas y la letra ñ) y del idioma inglés que serán identificados en el OCR.

Cabe mencionar que el programa no puede identificar la diferencia entre la letra l (minúscula) y la letra I (mayúscula), ya que son muy parecidas.

La red neuronal (ya entrenada) se guardará en un archivo en el disco duro llamado red_ neuronal.dat. Si este archivo existe, se descomprimirá la red neuronal. Si no existe el archivo, se crea un nuevo arreglo vacío de neuronas y se manda a entrenar.

```
private void entrenar() {
    List<Font> fuentes = new ArrayList<>();
    fuentes.add(new Font("arial", Font.PLAIN, 54));
    fuentes.add(new Font("arial narrow", Font.PLAIN, 54));
    fuentes.add(new Font("Calibri", Font.PLAIN, 54));
    fuentes.add(new Font("Georgia", Font.PLAIN, 54));
    fuentes.add(new Font("Times New Roman", Font.PLAIN, 54));
    fuentes.add(new Font("Book Antiqua", Font.PLAIN, 54));
    fuentes.add(new Font("Centaur", Font.PLAIN, 54));
    fuentes.add(new Font("Garamond", Font.PLAIN, 54));
    fuentes.add(new Font("Tahoma", Font.PLAIN, 54));
    fuentes.add(new Font("Trebuchet MS", Font.PLAIN, 54));
    fuentes.add(new Font("Lucida Sans", Font.PLAIN, 54));
    fuentes.add(new Font("Verdana", Font.PLAIN, 54));
    fuentes.add(new Font("Vrinda", Font.PLAIN, 54));
    fuentes.add(new Font("Comic Sans MS", Font.PLAIN, 54));
    fuentes.add(new Font("verdana", Font.PLAIN, 30));
    Map<Font, Map<String, Map<Object, Double>>> convertidos = new HashMap<>();
```

5.14 Lista de fuentes de la red neuronal.

Con la función entrenar se entrena la red neuronal (Ver figura 5.14). Además, se agregan las fuentes que van a ser entrenadas con su tamaño.

En caso de que se quiera agregar otra fuente, lo único que se tiene que hacer es borrar la estructura de datos y mandar a entrenar la red neuronal.

```
fuentes.parallelStream().forEachOrdered(fuente -> {
    caracteres.parallelStream().forEachOrdered(caracter -> {
        BufferedImage tmp = new BufferedImage(
                128, 128, BufferedImage.TYPE_BYTE_BINARY);
        Graphics2D g = tmp.createGraphics();
        g.setColor(Color.white);
        g.fillRect(0, 0, tmp.getWidth(), tmp.getHeight());
        g.setColor(Color.black);
        g.setFont(fuente);
        g.drawString(caracter, 30, 80);
        tmp = recortar(tmp);
        tmp = ajustar(tmp);
        if (convertidos.containsKey(fuente)) {
            convertidos.get(fuente).put(caracter, convertir(tmp));
        } else {//si aun no existe esa fuente
            convertidos.put(fuente, Helper.map(new MapEntry<>(caracter, convertir(tmp))));
        }

    });
});
```

Figura 5.15 Segunda parte del código de la función entrenar.

En la figura 5.15 se puede observar en esta parte del código que se encuentran dos ciclos anidados, es decir, se recorre cada una de las fuentes y cada uno de los caracteres.

Se crea una imagen temporal de 128 x 128 pixeles, donde se va a dibujar el carácter. Luego se crea un objeto que permite dibujar en esa imagen temporal, poniendo el fondo de color blanco y la letra de color negro. Se dibuja el carácter. Y finalmente se manda a recortar, ajustar y convertir (Cada una de estas funciones serán explicadas más adelante).

```
caracteres.parallelStream().forEachOrdered(ci -> {
    Neurona n = new Neurona(ci);
    System.out.println("-Entrenando " + n.getId());
    List<Caso> casos = new ArrayList<>();
    fuentes.parallelStream().forEachOrdered(fuente -> {
        caracteres.parallelStream().forEachOrdered(cj -> {
            casos.add(new Caso(convertidos.get(fuente).get(cj), ci.equals(cj) ? 0d : 1d));
        });
    });
    n.entrenar(casos);
    System.out.println("-Entrenamiento finalizado");
    neuronas.add(n);
});
guardar();
```

5.16 Tercera parte del código de la función entrenar.

En la figura 5.16 se puede ver que se recorre el arreglo de caracteres y se crea una neurona por cada carácter. A cada neurona le va corresponder un identificador (id). Más tarde se crea una lista de casos, donde se agregan todas las fuentes.

Después se vuelven a recorrer todas las fuentes y todos los caracteres para comprobar que sea el carácter correcto de la neurona (para ello marcara con un cero si es correcto y un 1 si es incorrecto).

Al final con esos casos se entrena la neurona y se añade la neurona a la red neuronal. También se imprimirá en consola “Entrenamiento Finalizado”.

```
public BufferedImage recortar(BufferedImage original) {
    int left = original.getWidth(), top = original.getHeight(), right = 0, bottom = 0;
    for (int y = 0; y < original.getHeight(); y++) {
        for (int x = 0; x < original.getWidth(); x++) {
            int pixel = original.getRGB(x, y);
            if (pixel != Color.white.getRGB()) {
                if (x < left) {
                    left = x;
                }
                if (x > right) {
                    right = x;
                }
                if (y < top) {
                    top = y;
                }
                if (y > bottom) {
                    bottom = y;
                }
            }
        }
    }
    return original.getSubimage(left, top, right - left, bottom - top);
}
```

5.17 Función recortar de la clase interprete.

La función recortar sirve para delimitar el tamaño del glifo (Ver figura 5.17). Ejemplo: la letra A tiene un tamaño de 128x128 pixeles, al recortarla, se corta a partir de las esquinas inferior, superior, derecha e izquierda y finalmente te da un tamaño de 47x70 pixeles. (Ver sección 5.6 Redimensionamiento del Tamaño de la Imagen).

Como se puede ver en el código recorre el eje de las *x* y el eje de las *y*, se declara el pixel, y si el valor más lejano de color negro es el último en ese mismo color, se guarda el valor de left, right, top, bottom (izquierda, derecha, superior e inferior).

```
public BufferedImage ajustar(BufferedImage original) {
    BufferedImage tmp = Scalr.resize(original, Scalr.Method.QUALITY, gx, gy);
    BufferedImage res = new BufferedImage(gx, gy, BufferedImage.TYPE_BYTE_BINARY);
    Graphics g = res.createGraphics();
    int x = (gx - tmp.getWidth()) / 2, y = (gy - tmp.getHeight()) / 2;
    g.setColor(Color.white);
    g.fillRect(0, 0, gx, gy);
    g.drawImage(tmp, x, y, tmp.getWidth(), tmp.getHeight(), null);
    return res;
}
```

5.18 Función ajustar de la clase interprete.

Con la función ajustar, se redimensiona la imagen para que quede de otro tamaño, respetando la proporción de la imagen para que no se deforme (para ello se crea una imagen temporal). Ver figura 5.18. Ejemplo: la letra A tiene un tamaño de 47x70 pixeles, al ajustar la imagen se redimensiona a 32x32 pixeles. (Ver sección 5.6 Redimensionamiento del Tamaño de la Imagen).

```
public String interpretar(Glifo g) {
    Map<Object, Double> resultados = new HashMap<>();
    for (int i = 0; i < neuronas.size(); i++) {
        Neurona n = neuronas.get(i);
        resultados.put(n.getId(), n.evaluar(convertir(g.getImagen())));
    }

    Map.Entry<Object, Double> entry = resultados.entrySet().parallelStream().min((o1, o2) -> {
        return Double.compare(o1.getValue(), o2.getValue());
    }).get();
    String res = (String) entry.getKey();
    g.setCaracter(res);
    return res;
}
```

5.19 Función interpretar de la clase interprete.

En la función interpretar se convierte un glifo a un carácter (Ver figura 5.19). Para ello se crea un arreglo donde se guarden los resultados de cada neurona y se recorre el arreglo de neuronas. Después se lee una neurona y se evalúa el glifo (la imagen) de cada neurona con respecto al id (el id de la neurona es lo que indica cual es el carácter). Con la función min va a devolver el valor más cercano a 0 de los resultados de las neuronas.

```
public static Map<Object, Double> convertir(BufferedImage imagen) {
    Map<Object, Double> res = new HashMap<>();
    int i = 0;
    for (int y = 0; y < gy; y++) {
        for (int x = 0; x < gx; x++) {
            double c = imagen.getRGB(x, y) == Color.black.getRGB() ? 1 : 0;
            res.put(i, c);
            i++;
        }
    }
    return res;
}
```

Figura 5.20 Función convertir de la clase interprete.

En la función convertir (Ver figura 5.20) se recorre la imagen punto a punto en las coordenadas *x*, y *y* se convierte a valores numéricos (para ello utiliza un operador ternario que convierte los pixeles negros en 1 y los pixeles blancos a cero). (Ver sección 5.6 Redimensionamiento del Tamaño de la Imagen).

```
public void guardar() {
    Helper.serializar(neuronas, new File("red_neuronal.dat"));
    System.out.println("red actualizada");
}
```

Figura 5.21 Función guardar.

En la función guardar de la figura 5.21, se guardan todas las neuronas en la estructura de datos llamada "red_neuronal.dat" (se guardarán en el disco duro) y se imprime en consola "red actualizada". Si se borra ese archivo y se le da click en entrenar, se volverá a entrenar la red neuronal y se creará una nueva estructura de datos. Esto último se puede realizar en caso de que se quiera agregar otra fuente.

5.8 Clase Procesador

En la clase procesador se toma una foto del pdf y se separan los glifos

```
public class Procesador {

    private List<Glifo> glifos;
    private BufferedImage pagina;

    public Procesador(BufferedImage pagina) {
        this.pagina = pagina;
        procesar();
    }

    public List<Glifo> getGlifos() {
        return glifos;
    }

    public BufferedImage getPagina() {
        return pagina;
    }

    public void setPagina(BufferedImage pagina) {
        this.pagina = pagina;
        procesar();
    }
```

Figura 5.22 Getters y setters de la clase procesador.

En la imagen 5.22 se muestran los getters y los setters de la lista de glifos y de página.

```
private void procesar() {
    pagina = Preprocesador.escalaGrises(pagina);
    pagina = Preprocesador.binarizacion(pagina);
    glifos = new ArrayList<>();
    int w = pagina.getWidth();
    int h = pagina.getHeight();
    BufferedImage tmp = new BufferedImage(w, h, BufferedImage.TYPE_BYTE_BINARY);
    tmp.createGraphics().drawImage(pagina, 0, 0, null);

    List<Glifo> linea = new ArrayList<>();
```

Figura 5.23 Primera parte del código de la función procesar.

La función procesar convierte una imagen en escala de grises y luego la binariza (Ver figura 5.23). Además, se guarda un arreglo de glifos y se obtiene el ancho y la altura de la imagen.

Luego se crea una imagen temporal para editar la imagen original y se crea un arreglo donde se guardarán los glifos por línea en el texto.

```
List<Glifo> linea = new ArrayList<>();
Rectangle inicio = null;
for (int y = 0; y < h; y++) {
    for (int x = 0; x < w; x++) {

        int pixel = tmp.getRGB(x, y);
        if (pixel == Color.black.getRGB()) {
            Glifo actual = separarGlifo(x, y, tmp, pagina);
            actual.interpretar();
            if (inicio == null) {
                inicio = actual.getBounds();
            } else {
                Rectangle rac = actual.getBounds();
                Rectangle r = new Rectangle(inicio.x, rac.y, rac.width, rac.height);
                if (!inicio.intersects(r)) {
                    procesarLinea(linea);
                    glifos.addAll(linea);
                    linea = new ArrayList<>();
                    inicio = actual.getBounds();
                }
            }
            linea.add(actual);
        }
    }
}
procesarLinea(linea);
glifos.addAll(linea);
```

Figura 5.24 Segunda parte del código de la función procesar.

En la figura 5.24 se muestra el código donde se recorre la imagen en *x* y *y*, pixel por pixel. Si se detecta un pixel en color negro quiere decir que hay un glifo. Más tarde se utiliza la función separar glifo para guardarlo y separar las letras, y se interpreta para saber qué letra es.

Con inicio == null se detecta que se está al inicio de una línea. Luego se comprueba que estamos en la misma línea, sino se llama a la función procesar línea. Todo eso se hace para que al final se agreguen todos los glifos a la lista de glifos.

```
glifos.parallelStream().forEach(g -> {
    if (g.getCaracter().equals("")) {
        g.interpretar();
    }
});
```

Figura 5.25 Tercera parte del código de la función procesar.

Finalmente se vuelven a interpretar los glifos para mostrar un mejor resultado (Ver figura 5.25).

```
private void procesarLinea(List<Glifo> linea) {
    Glifo salto = new Glifo(new Rectangle(), System.lineSeparator());
    Glifo espacio = new Glifo(new Rectangle(), " ");
    linea.sort((o1, o2) -> {
```

Figura 5.26 Función procesar línea

La función procesar línea ordena a los glifos de acuerdo a como estén escritos y agrega espacios y saltos donde sea necesario. Ver figura 5.26

La función sort de java ordena los caracteres de acuerdo a la posición del glifo.

```
private Glifo separarGlifo(int x, int y, BufferedImage tmp, BufferedImage pagina) {/
    List<Point> pixeles = new ArrayList<>();
    floodFill(pixeles, x, y, tmp);
```

Figura 5.27 Función separar glifo

La función separar glifo utiliza la función de floodFill para separar cada uno de los glifos de la imagen principal. Ver figura 5.27.

5.9 Clase Decodificador

La clase decodificador sirve para leer el pdf y convertirlo a imagen.

```
public Decodificador(File f) {
    this.ruta = f.getPath();
    paginas = new ArrayList<>();
    try (final PDDocument document = PDDocument.load(f)) {
        PDFRenderer pdfRenderer = new PDFRenderer(document);
        for (int page = 0; page < document.getNumberOfPages(); page++) {
            BufferedImage bim = pdfRenderer.renderImageWithDPI(page, 300, ImageType.RGB);
            paginas.add(bim);
        }
        document.close();
```

Figura 5.28 Clase Decodificador.

En la figura 5.28 se muestra que se crea un arreglo para guardar las páginas. Se convierten las páginas del pdf en imágenes con una resolución de 300 puntos por pulgada (o se renderiza la imagen a 300 puntos por pulgada).

5.10 Clase Preprocesador

En la clase preprocesador se hace la parte de visión artificial. Se preprocesa el pdf.

```
public static BufferedImage escalaGrises(BufferedImage imagen) {
    BufferedImage tmp = new BufferedImage(imagen.getWidth(), imagen.getHeight(), BufferedImage.TYPE_BYTE_GRAY);
    tmp.createGraphics().drawImage(imagen, 0, 0, null);
    imagen.createGraphics().drawImage(tmp, 0, 0, null);
    return tmp;
}

public static BufferedImage binarizacion(BufferedImage imagen) {
    BufferedImage tmp = new BufferedImage(imagen.getWidth(), imagen.getHeight(), BufferedImage.TYPE_BYTE_BINARY);
    tmp.createGraphics().drawImage(imagen, 0, 0, null);
    imagen.createGraphics().drawImage(tmp, 0, 0, null);
    return tmp;
}
```

Figura 5.29 Código que trata sobre el preprocesamiento de imágenes.

Como preprocesamiento de imágenes, primero la imagen del pdf se convierte a escala de grises (dejando al carácter con un aspecto borroso con una escala de grises) y finalmente pasa por una etapa de binarización (en esta parte el carácter toma mejor forma, ya que se aclara la imagen). Ver figura 5.29.

En la imagen de abajo se puede ver un ejemplo del preprocesamiento que se lleva a cabo en la letra A. Ver figura 5.30.

Figura5.30 Imagen de una letra preprocesada.

5.11 Clase Caso

La clase Caso va obtener las entradas y la salida esperada.

```
public Caso(Map<Object, Double> entradas) {
    this.entradas = entradas;
    this.salida = 0d;
}

public Caso(Map<Object, Double> entradas, Double salida) {
    this.entradas = entradas;
    this.salida = salida;
}
```

Figura 5.31 Constructores del caso.

En la figura 5.31 se puede observar los constructores del caso.

```
public Map<Object, Double> getEntradas() {
    return entradas;
}

public void setEntradas(Map<Object, Double> entradas) {
    this.entradas = entradas;
}

public Double getSalida() {
    return salida;
}

public void setSalida(Double salida) {
    this.salida = salida;
}
```

Figura 5.32 Getters y Setters de la clase caso.

En la figura 5.32 se muestran los getters y los setters de las entradas y salidas.

5.12 Clase Funciones

En la clase Funciones se pueden ver los cálculos matemáticos que se realizan con la función Sigmoide y HeaviSide. Ver figura 5.33.

```
public class Funciones {

    public static final Function<Double, Double> sigmoide = (Double t) -> {
        return Math.pow(Math.E, t) / (Math.pow(Math.E, t) + 1);
    };
    public static final Function<Double, Double> heavySide = (t) -> {
        return t > 0 ? 1d : 0d;
    };
}
```

Figura 5.33 Funciones Sigmoide y Heavyside.

Más adelante se puede ver los cálculos matemáticos de cada una de las funciones:

Función Evaluar

Aplica la función Sigmoide = $\frac{e \wedge t}{e \wedge t + 1} = \frac{e \wedge 0.6}{e \wedge 0.6 + 1} = 0.6456$

$t > 0$

Heaviside $Res = 1$

$t \leq 0 = 0$

e = salida obtenida - salida esperada

$e = 0.6456 - 0 > 0.1$

Por fines prácticos, solo se utilizó la función sigmoide en este proyecto.

5.13 Clase Neurona

En la clase neurona se muestra el código de los cálculos de los perceptrones.

```
private Double salida;
private Map<Object, Entrada> entradas;
private String id;
```

Figura 5.34 Sección del código de la clase neurona.

En la figura 5.34 se muestra el código donde se guarda la salida, las entradas y el carácter respectivamente.

```
public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}
```

Figura 5.35 Getter y Setter del id.

En la imagen 5.35 se muestra el getter y el setter del id.

```
public void entrenar(List<Caso> casos) {
    casos.get(0).getEntradas().keySet().parallelStream().forEachOrdered((k)
```

Figura 5.36 Primera parte de la función entrenar de la clase neurona.

Para entrenar a la neurona se pide una lista de casos y de acuerdo a los casos se crean las entradas virtuales de la neurona. Ver figura 5.36.

```
int error;
do {
    error = 0;
    for (Caso m : casos) {
        m.getEntradas().entrySet().parallelStream().forEach(e -> {
            Entrada tmp = entradas.get(e.getKey());
            tmp.setEntrada(e.getValue());
        });
        procesar();
        Entrada umbral = entradas.get(UMBRAL);
        double shs = Funciones.sigmoide.apply(salida);
        if (Math.abs(shs - m.getSalida()) > 0.1) {
            double e = m.getSalida() - shs;
            umbral.setPeso(umbral.getPeso() + e);
            m.getEntradas().entrySet().parallelStream().forEach(el -> {
                Entrada tmp = entradas.get(el.getKey());
                tmp.setPeso(tmp.getPeso() + e * el.getValue());
            });
            error++;
        }
    }
} while (error > 0);
```

Figura 5.37 Segunda parte de la función entrenar de la clase neurona.

En la figura 5.37 se muestra el código donde se recorre el arreglo de casos y se asignan los valores de las entradas del caso a la neurona. Después se manda a la función procesar, se lee el umbral y se aplica la función sigmoide a la neurona. Luego, se compara el resultado de la función a la salida esperada del caso; si es mayor a 0.1 entonces se prosigue con los siguientes cálculos. Más tarde se calcula el error (salida obtenida – salida esperada), se cambia el peso del

umbral sumándole el error y acomodamos el valor de los pesos, sumándoles la multiplicación del error por el valor de la entrada.

Finalmente, se incrementa el contador del error para saber que se tiene que reentrenar y se repetirá el ciclo hasta que no haya errores.

En la sección 5.4 (Calculos de Perceptron) se puede visualizar todos los cálculos que se realizan en esta parte del código.

```
private void procesar() {
    salida = 0d;
    entradas.entrySet().parallelStream().forEachOrdered(e -> {
        Entrada tmp = e.getValue();
        salida += tmp.getEntrada() * tmp.getPeso();
    });
}
```

Figura 5.38 Función procesar.

La función procesar que se encuentra en la figura 5.38 realiza los siguientes cálculos:

Wu * Umbral		Wa * a	Wb * b	
(0.1 * 1)	+	(0.5 * 1) +	(0.3 * 0) = 0.6	Salida

En otras palabras, multiplica la entrada o el umbral por el peso.

```
public Double evaluar(Map<Object, Double> datos) {
    datos.entrySet().parallelStream().forEach(e -> {
        Entrada tmp = entradas.get(e.getKey());
        tmp.setEntrada(e.getValue());
    });
    procesar();
    return Funciones.sigmoide.apply(salida);
}
```

Figura 5.39 Función evaluar.

La función evaluar que se encuentra en la figura 5.39 realiza los cálculos de la función procesar y aplica la sigmoide como a continuación:

:

Función Evaluar

Aplica la función Sigmoide = $\frac{e \wedge t}{e \wedge t + 1} = \frac{e \wedge 0.6}{e \wedge 0.6 + 1} = 0.6456$

```
public Entrada() {
    entrada = 0d;
    peso = new Random().nextDouble();
}

public Entrada(Double entrada) {
    this.entrada = entrada;
    peso = new Random().nextDouble();
}

public Entrada(Double entrada, Double peso) {
    this.entrada = entrada;
    this.peso = peso;
}
```

Figura 5.40 Constructores de la entrada y el peso.

En el código que se encuentra en la figura 5.40 se encuentran los constructores de la entrada y el peso. Como se puede observar el peso siempre será aleatorio.

```
public Double getEntrada() {
    return entrada;
}

public void setEntrada(Double entrada) {
    this.entrada = entrada;
}

public Double getPeso() {
    return peso;
}

public void setPeso(Double peso) {
    this.peso = peso;
}
```

Figura 5.41 Getters y setters de la entrada y el peso.

En la figura 5.41 se puede ver los getters y los setters de la entrada y el peso.

5.14 Interfaz Gráfica de Usuario

```
private void btnCerrarActionPerformed(java.awt.event.ActionEvent evt) {
    if (!glifo.getCaracter().equals(txtCaracter.getText())) {
        glifo.setCaracter(txtCaracter.getText());
        Interprete interprete = Interprete.getInstancia();
        interprete.getNeuronas().parallelStream().forEach(neurona -> {
            Caso caso = new Caso(Interprete.convertir(glifo.getImagen()));
            if (neurona.getId().equals(glifo.getCaracter())) {
                caso.setSalida(0d);
            } else {
                caso.setSalida(1d);
            }
            neurona.entrenar(caso);
        });
        interprete.guardar();
    }
    setVisible(false);
    dispose();
}
```

Figura 5.42 Código de la interfaz gráfica de usuario.

En esta parte del código (ver figura 5.42) se recorre la red neuronal y se crea un caso para cada carácter. Si la neurona tiene un id igual al introducido por el usuario, se pondrá la salida del caso a cero para indicar que esa es la neurona correcta. (Se pondrá un cero si es correcto y un uno si es incorrecto).

Finalmente se reentrena neurona por neurona y se guarda la red neuronal.

VI. ¿COMO USAR EL SOFTWARE?

En esta sección se mencionará la manera en la que debe usarse el software. Como ya se había mencionado con anterioridad, se realizaron 2 versiones (una en python y otra en java). A continuación, se explicará cada una de las versiones:

VERSION EN PYTHON

6.1 Interfaz Gráfica de Usuario

La interfaz gráfica de usuario consta de 2 etapas (Ver figura 6.1):

El primer paso consiste en darle clic en el botón de *Archivo*. Una vez que se selecciona, se abrirá una carpeta que te permitirá elegir la imagen a procesar. Se selecciona la imagen y pasa por un módulo de preprocesamiento de imagen que más adelante se va a explicar.

En el segundo paso se le da clic en *Convertir* y se convierte el texto de la imagen .jpg a .txt.

Figura 6.1 Interfaz Gráfica de Usuario en Python.

VERSION EN JAVA

La interfaz gráfica de usuario consta de 2 etapas (Ver figura 6.2):

El primer paso consiste en darle clic en el botón de *Convertir PDF*. Una vez que se selecciona, se abrirá una carpeta que te permitirá elegir el pdf que se va a procesar. Se selecciona el pdf y más tarde se convierte la imagen a texto.

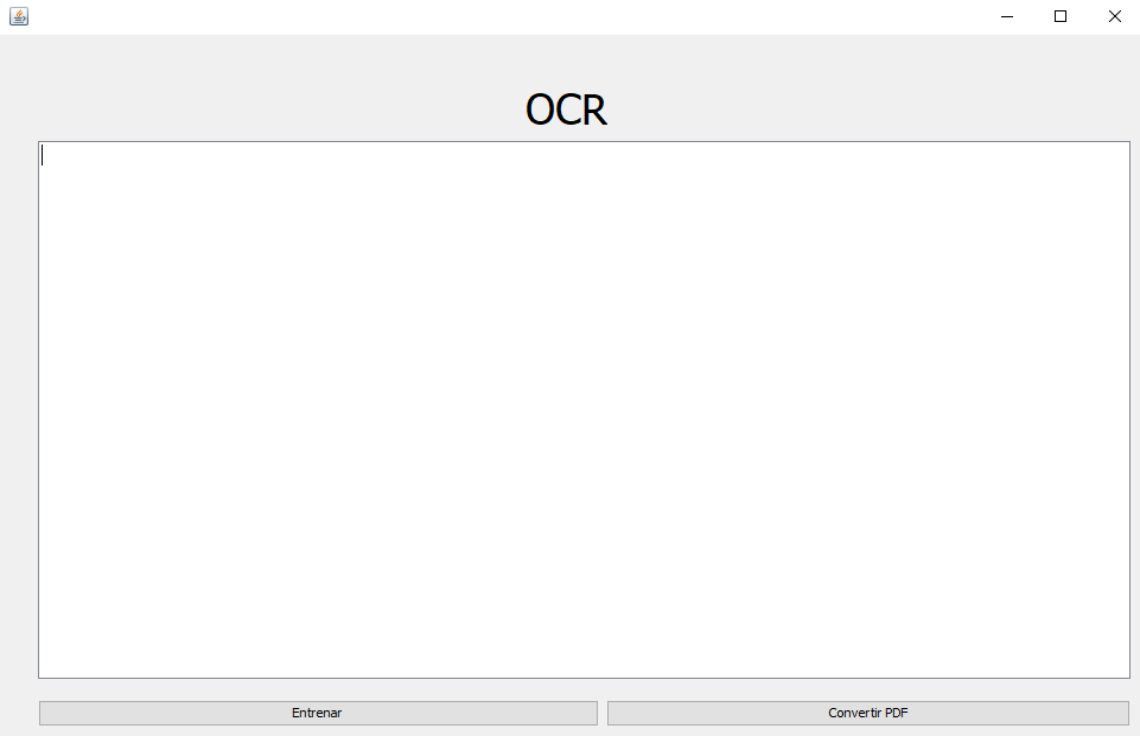


Figura 6.2 Interfaz Gráfica de Usuario en Java.

Ejemplo del programa ya en funcionamiento:

Se tiene el siguiente pdf en letra Calibri tamaño 12. (Ver figura 6.3)

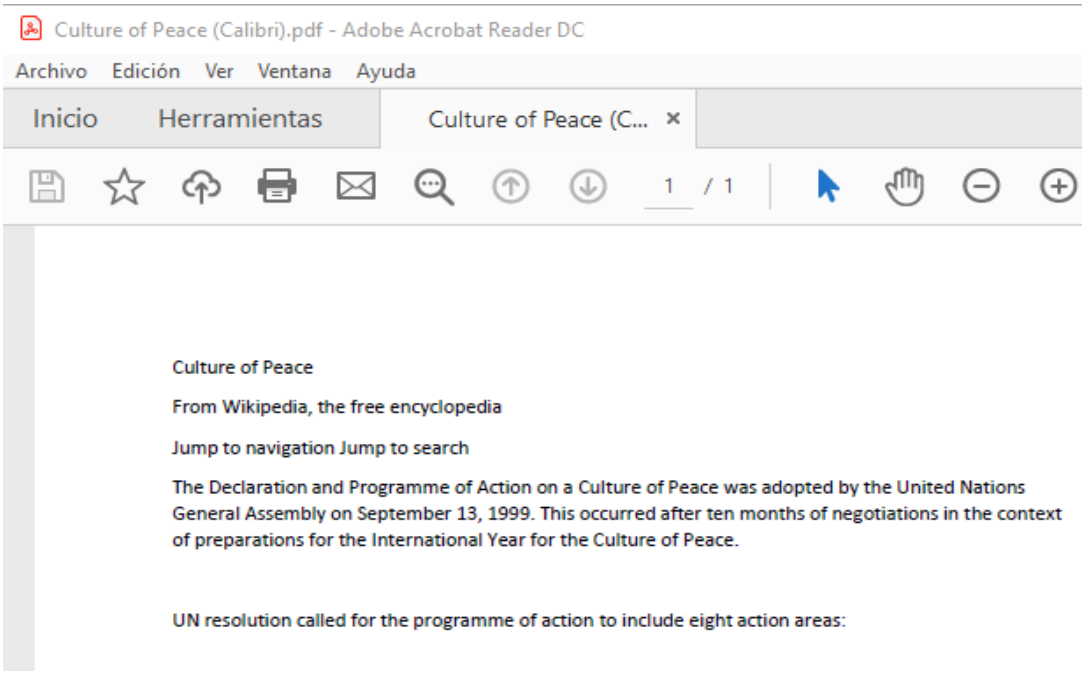


Figura 6.3 Ejemplo de un documento pdf.

Al darle click en *Convertir PDF*, el programa lo convertirá a texto. (Ver figura 6.4).

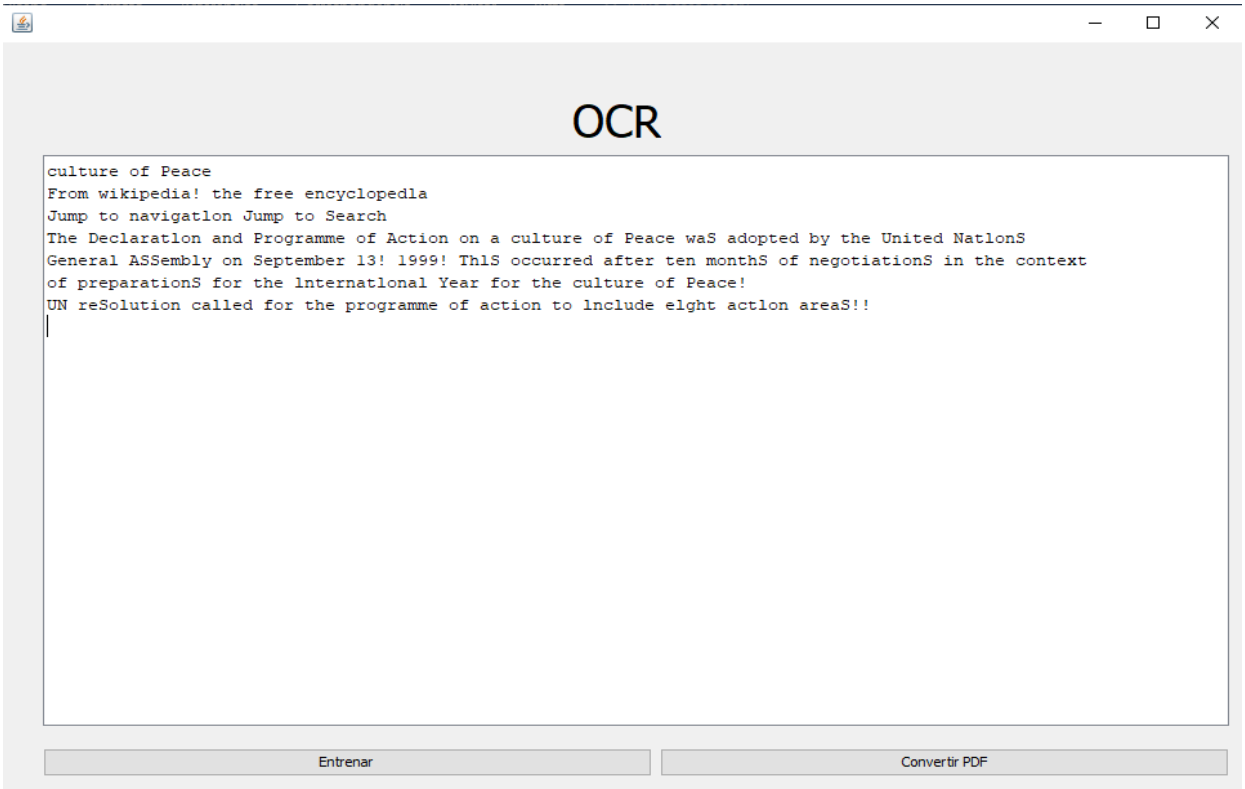


Figura 6.4 Texto del pdf.

Una vez que se muestre el resultado, es posible que haya errores en el contenido. Los errores se mostrarán con el signo de admiración (!).

Para corregir los errores se selecciona el botón de entrenar. Al seleccionarlo, se abrirá otra ventana (Ver figura 6.5). La ventana recorrerá cada una de las letras del pdf y pedirá que letras quieres que se corrijan. Se escribe la letra para cada uno de los errores marcados con el signo de admiración.

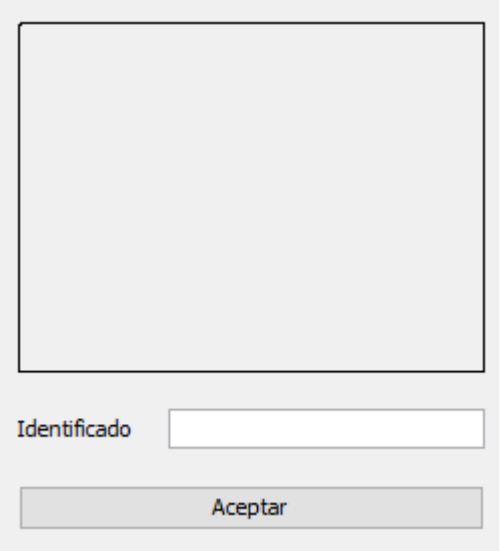


Figura 6.5 Ventana de Entrenar en Java.

VII. CONCLUSIONES

Con base a las pruebas que se realizaron se determinó que se puede realizar el proyecto de dos maneras: ya sea con perceptrones o con redes neuronales convolucionales y redes neuronales recurrentes.

Al crear el OCR, me di cuenta que la versión que realicé en python contenía errores, ya que no detectaba las letras acentuadas ni la letra ñ. Además de eso, el proyecto solo detectaba letras de imágenes en pyplot y tampoco podía distinguir imágenes en pdf (lo cual era una de los objetivos preestablecidos). Por estas razones el proyecto en python fue descartado y se decidió realizar otra versión en java donde se corrigieron todos los errores de la antigua versión.

El OCR es aplicable en áreas informáticas, administrativas y de documentación. Además de reconocer palabras de imágenes, también son útiles para el reconocimiento de matrículas, facturas y códigos de barras.

La idea de usar funciones extraídas para entrenar una red neuronal parece funcionar, aunque la tasa de éxito no es del 100%, sigue siendo buena. A pesar de la complejidad involucrada de las redes neuronales artificiales, el Deep Learning (aprendizaje profundo) ofrece varias ventajas para el reconocimiento y clasificación de patrones.

VIII. REFERENCIAS BIBLIOGRÁFICAS

Andres Marzal, I. G. (2014). *Introduccion a la programacion con Python 3.* Sevilla: Sapientai.

Arindam Chaudhuri, K. M. (2016). Optical Character Recognition for Different Languages with Sof Comput. Springer.

B.V. Dhandra, V. M. (2008). Multi-font English character recognition based on modified invariant moments.

C.V. Lakshmi, S. S. (2009). Pattern Recognition and Machine Intelligence. En *Lecture Notes in Computer Science* (págs. 393-399).

Dean, J. (2018). *Tensorflow.* Obtenido de www.tensorflow.org

equipo Keras. (2020). *Keras.* Obtenido de keras.io/about/

Eremenko, K. (2017). *Deep Learning A - Z. Hands on Artificial Neural Networks*. Obtenido de udemy.com/course/deeplearning/learn/lecture/6761092#overview

F. Slimane, S. K. (2011). International Conference on Document Analysis and Recognition (ICDAR). 1449-1453.

Jakóbczak, D. J. (2015). International Journal of Recent Research Aspects Vol. 2 Issue 3. p4-8. 5p.

James Gosling, B. J. (2015). The Java Language Specification.

Kae, D. S.-M. (2011). Learning on the fly: a font-free approach toward multilingual OCR. 289-301.

Numerentur. (11 de abril de 2018). *Redes Neuronales Recurrentes*. Obtenido de http://numerentur.org/rnr/

Ordoñez, J. P. (2009). Reconocimiento Optico de Caracteres con Redes Neuronales.

P.S. Hangarge Mallikarjun, B. D. (2010). Int. J. Comput. Appl. 126-130.

Paellasoft Software Development. (2019). *Paellasoft*. Obtenido de https://paellasoft.com/ocr/

R. Rani, R. D. (2013). International Conference on Document Analysis and Recognition (ICDAR),.

S. Ben Moussa, A. Z. (2008). International Conference on Pattern Recognition.

Samadiani, N. (2015). A neural network-based approach for recognizing multi-font printed English characters. *Journal of Electrical Systems and Information Technology*, 207-218.

Sukhiha. (2013). IEEE Second International Conference on Image Information Processing (ICIIP). 323-328.

Vélez, J. (26 de Julio de 2017). *Vision por Computadora*. Obtenido de http://visionporcomputadora.blogspot.com/2017/07/breve-introduccion-python-numpy-opencv.html