UNIVERSIDAD NACIONAL JOSÉ MARIÁ ARGUEDAS

FACULTAD DE INGENIERIA

ESCUELA PROFESIONAL DE INGENIERIA DE SISTEMAS

Presentado por:

MARIO HUARACA FLORES

"SISTEMA DE RECONOCIMIENTO DE CARACTERES MANUSCRITOS USANDO REDES NEURONALES CONVOLUCIONALES"

Asesor:

Ing. ROBERTO QUISPE QUISPE

TESIS PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO DE SISTEMAS

ANDAHUAYLAS – APURÍMAC – PERÚ

2019

Dedicatoria

El presente trabajo investigativo lo dedico principalmente a Dios, por ser el inspirador y darme fuerza para continuar en este proceso de obtener uno de mis anhelos más deseados que es obtener el título profesional de ingeniería de sistemas.

A mis padres, por su amor, trabajo y sacrificio en todos estos años, gracias a ustedes he logrado llegar hasta aquí y convertirme en lo que soy. Ha sido un orgullo y el privilegio de ser su hijo, son los mejores padres.

A mis hermanos y hermanas por estar siempre presentes, a acompañándome y por el apoyo moral, que me brindaron a lo largo de esta etapa de mi vida.

A todas las personas que me han apoyado y han hecho que el trabajo se realice con éxito en especial a aquellos que me abrieron las puertas y compartieron sus conocimientos conmigo.

Agradecimiento

Agradezco a Dios por bendecirme en la vida, por guiarme a lo largo de mi existencia, ser el apoyo y fortaleza en aquellos momentos de dificultad y de debilidad.

Gracias a mis padres: Anastasio Huaraca, Elisa Flores, por ser los principales promotores de mi sueño, por confiar y creer en mis expectativas, por los consejos, valores y principios que me han inculcado.

TABLA DE CONTENIDO

INDICE DE TABLAS

INDICE DE FIGURAS

RESUMEN

El reconocimiento y verificación de firmas en numerosos documentos, entre ellos los bancarios, son unos procedimientos que se están realizando de forma generalizada de manera manual. En este proyecto, se propone un método y una implementación del mismo que sea capaz de realizar estas tareas automáticamente y de forma satisfactoria, con unos costes en tiempo y memoria reducidos que lo hagan competitivo frente al tratamiento manual.

Para lograrlo, el sistema desarrollado será capaz de realizar un preproceso basado en el esqueletizado, etiquetará los puntos de la firma según su pendiente utilizando matrices de convolución, realizará un seguimiento y una extracción de los trazos etiquetados siguiendo un esquema de búsqueda con retroceso, normalizará el tamaño de los trazos y mediante un proceso de ajuste de distancias (distance matching) calculará el porcentaje de similitud entre las colecciones de trazos extraídas de dos firmas. En ese momento el sistema será capaz de reconocer una firma, buscando en la base de datos creada para tal fin, aquélla que más se le parezca, es decir, la que tenga mayor porcentaje de similitud.

Por último y para dotar al software de la capacidad de verificar firmas habrá que calcular el umbral a partir del cual se podrá decir que dos firmas son lo suficientemente parecidas como para pertenecer al mismo firmante.

El grado de fiabilidad logrado en los experimentos realizados hace esperar que las tareas de verificación y reconocimiento de firmas puedan ser automatizas en

gran medida, contribuyendo a la reducción del número de fraudes por falsificaciones.

Palabras clave: red neuronal artificial, inteligencia artificial subsimbólica, esqueletizado, matriz de convolucion, ajuste de distancia.

INTRODUCCION

Hasta la fecha, el problema del reconocimiento de firmas en documentos ha sido obviado o relegado a la responsabilidad de las personas. Esto podría atribuirse a la necesidad de altas capacidades de cómputo, a su complejidad técnica o a que no es el problema más importante para una institución.

El hecho de que durante muchos años no se dieran las condiciones óptimas para llevar a cabo estos proyectos, y en muchos casos el problema se haya apartado de las prioridades de las instituciones, no impide que la misma siga asumiendo riesgos, ya innecesarios, a la hora de tratar aquello que resulta vital para su negocio: las firmas de sus usuarios y su verificación. Las consecuencias de no asumir por parte de la entidad algún tipo de tecnología que, en mayor o menor medida, permita la autenticación de firmas fuera de la oficina donde se originan las mismas implica, entre otros, los siguientes factores negativos:

- Riesgos económicos en operaciones fraudulentas.
- Empobrecimiento de la imagen institucional ante los clientes.
- Demora en las operaciones.
- Elevados tiempos de espera para los usuarios.
- Coste de personal innecesario.

CAPITULO I

1 PROBLEMA DE INVESTIGACION

1.1 Descripción del problema

En la Oficina de Admisión de la Universidad Nacional José María Arguedas, se lleva el proceso de Inscripción para ser postulante para alguna de las seis carreras existentes como son la Escuela Profesional de Agroindustrias, la Escuela Profesional de Ingeniería de Sistemas, la Escuela Profesional de Administración de Empresas, Escuela Profesional de Contabilidad, Escuela Profesional de Ingeniería Ambiental y la Escuela Profesional de Educación Primaria Intercultural, cada carrera tiene establecida la cantidad de cuarenta (40) vacantes de ingreso.

En la etapa de inscripción de postulante se detalla las siguientes actividades a) preinscripción en la web (www.unajma.edu.pe), b) paga de derecho de postulante, c) Presentación de Documentos de inscripción, d) Asistir al Examen de Admisión y e) Emisión de la Constancia de ingreso, efectuada estas actividades y el ingresante contando con su constancia de Ingreso se apersona a la Oficina de Registro Académico formalizando su matrícula como alumno regular en la Escuela Profesional a la Cual ingreso es postulante.

Durante los procesos mencionados existe una omisión el mismo que se evidencia durante el cambio de condición del postulante es decir cuando alcanza la vacante el postulante y este se convierte en alumno regular, durante esta etapa no existe algún método de verificación en el cual se

verifique que el postulante es el mismo que se matricula como alumno regular.

El presente proyecto de investigación tiene como propósito llevar acabó la autenticación y validación del postulante mediante el proceso de ajuste de distancia, es decir se generará un aplicativo un “sistema de reconocimiento de caracteres manuscritos (firmas) usando redes neuronales convolucionales”.

1.2 Delimitación del problema

1.2.1 Alcances

Para realizar el presente trabajo se dispuso de un conjunto de datos que corresponden a grupos de postulantes tomados al azar de los que participaron de dicho proceso de admisión, teniendo en consideración que dichos documentos tienen grado de confidencialidad.

1.2.2 Limitaciones

No existió limitaciones para efectuar el entrenamiento de una Red Neuronal Artificial para efectuar la validación de la firma de dicho postulante.

1.3 Formulación del problema

1.3.1 Problema general

¿Es factible realizar la implementación de un sistema de reconocimiento de caracteres manuscritos (firmas) usando redes neuronales convolucionales para la validación del ingresante a la UNAJMA?

1.3.2 Problemas específicos

- ¿Es factible la Implementación de un software de extracción de características de archivo de imágenes y normalización del archivo objetivo?
- ¿Es factible la Implementación de un software de configuración y entrenamiento de la red neuronal para obtener los pesos y las vías de la red neuronal?
- ¿Es factible Diseñar el proceso de reconocimiento de caracteres dentro de la red neuronal?
- ¿Es factible la Implementación del prototipo de sistema de información para el reconocimiento manuscritos (firmas)?

1.4 Justificación

En la actualidad la firma manuscrita sigue siendo uno de los métodos de validación de documentos más empleado a nivel nacional e internacional.

Según National Check Fraud Center, Credit card Fraud Prevention Techniques and information menciona que “en las operaciones bancarias, la firma es uno de los métodos empleados en el protocolo de seguridad al realizar pagos con tarjeta de crédito y débito”.

En el presente trabajo se considerará el estudio de las técnicas biométricas de reconocimiento de firmas, así como también el análisis de algoritmos para comparar firmas.

De igual forma el uso de métodos de visión computacional que permita implementar un algoritmo matemático de comparación de firmas, y para el

presente trabajo de investigación se tomó como método de prueba de validación de firma el método de ajuste de distancia.

El presente proyecto de investigación tiene como propósito implementar una aplicación biométrica con la que probar el algoritmo matemático previamente programado para llevar acabó la validación de una firma de un postulante o ingresante a la Universidad Nacional José María Arguedas, mediante un “sistema de reconocimiento de caracteres manuscritos (firmas) usando redes neuronales convolucionales”.

1.5 Objetivos de la Investigación

1.5.1 Objetivo general

Implementar sistema de reconocimiento de caracteres manuscritos (firmas) usando redes neuronales convolucionales para validar los ingresantes a la UNAJMA.

1.5.2 Objetivos específicos

- Implementar el software de extracción de características de archivo de imágenes y normalización del archivo objetivo.
- Implementar el software de configuración y entrenamiento de la red neuronal para obtener los pesos y las vías de la red neuronal.
- Diseñar el proceso de reconocimiento de caracteres dentro de la red neuronal.
- Implementar el prototipo de sistema de información para el reconocimiento manuscritos (firmas).

CAPITULO II

2 ANTECEDENTES

2.1 Antecedentes de la investigación

2.1.1 Antecedentes a nivel internacional

La historia de las Redes Neuronales Artificiales proviene de varios campos que buscaron desarrollar la idea de encontrar un modelo matemático que asimilara la forma y el comportamiento de las neuronas humanas.

A lo largo de los siglos XIX y XX se empezó a desarrollar teorías de aprendizaje, visión y acondicionamiento por parte de un equipo de físicos, psicólogos y neurofisiólogos. Sin embargo, aún no se disponía de un modelo específico para la neurona.

En la década de los 40 se probó que las redes neuronales podían realizar cualquier operación aritmética o lógica. Sin embargo, no fue hasta la década de los 50 cuando se inventó el perceptrón, así como las reglas de aprendizaje. Desde ese momento se empezaron a desarrollar más algoritmos y la implementación de éstos creció, sin embargo, solo podían resolver una limitada serie de problemas. Esto, unido a la falta de computación digital por parte de los ordenadores, resultó en el abandono de este campo durante un tiempo.

El resurgir de la investigación en esta área fue gracias al algoritmo de retropropagación (Backpropagation algorithm). Desde entonces se han escrito muchos artículos y se han encontrado múltiples aplicaciones.

Actualmente, la gran capacidad computacional de los ordenadores permite explorar nuevos conceptos, arquitecturas y reglas de aprendizaje. Con el fin de entender el estado del arte en el campo de las redes neuronales, se van a explicar algunas de las aplicaciones de éstas actualmente.

- Reconocimiento de Caracteres: Esta idea tuvo mucho auge y se hizo muy popular hace algunos años, pues permite reconocer texto manuscrito.
- Compresión de imágenes: Las redes neuronales pueden recibir y procesar grandes cantidades de información a la vez, siendo esto útil en la compresión de imágenes.
- Mercado de Valores: Estas redes pueden examinar una gran cantidad de información de forma rápida y organizar todo de manera que se pueda hacer un adecuado estudio de proyección y predecir el valor de las acciones.
- Medicina: una de las áreas que más ha ganado la atención es en la detección de afecciones cardiopulmonares, es decir compara muchos modelos distintos para identificar similitud en patrones y síntomas de la enfermedad. Estos sistemas ayudan a los médicos con el diagnóstico por el análisis de los síntomas reportados y las resonancias magnéticas y rayos x. También se han usado para dispositivos analizadores de habla para ayudar a personas con sordera profunda, monitorización de cirugías, predicción de reacciones

adversas a un medicamento, entendimiento de causas de ataques epilépticos.

- Militar: Las redes neuronales juegan un papel importante en el campo de batalla, especialmente en aviones de combate y tanques que son equipados con cámaras digitales de alta resolución que funciona conectado a un computador que continuamente explora el exterior de posibles amenazas. De igual manera se pueden emplear para clasificar señales de radar o creación de armas inteligentes.
- Diagnóstico de Maquinas: A nivel industrial cuando una de estas máquinas presenta fallos automáticamente las apaga cuando esto ocurre.
- Análisis de Firmas: Las redes neuronales pueden ser empleadas para la comparación de firmas generadas. Esta ha sido una de las primeras aplicaciones implementada a gran escala en USA.
- A nivel Administrativo: Identificaciones de candidatos para posiciones específicas, optimización de plazas y horarios en líneas de vuelo, minería de datos.

En diversas actividades que realizamos a diario se nos exige, por un lado, identificarnos (decir quiénes somos) y por otro lado verificar nuestra identidad, es decir, dar una prueba de que en realidad somos quienes en determinado momento aseguramos ser, por lo que usualmente se implementan una variedad de metodologías de identificación que van desde la retención de documentos de identificación, la creación de tarjetas de seguridad, firmas manuscritas en libros de registro, pasando por

registros fotográficos, hasta el uso de rasgos biométricos entre los cuales cuentan el reconocimiento de huellas dactilares, retina del ojo, registro de voz, reconocimiento de rostro a partir de imágenes, entre otros. Todo esto se hace en pro de salvaguardar la identidad e intereses de los miembros de una determinada organización y de esta forma evitar acciones fraudulentas, como la falsificación, que en parte son la causa del desarrollo de tales prácticas y en general de delitos como la suplantación. Una práctica común, relacionada con lo anterior consiste en el registro de firmas manuscritas, para la validación de identidad en entidades como por ejemplo la registraduría, en el caso de entidades estatales y bancos, en el caso de entidades privadas; para la validación de transacciones, como retiros de dinero y cobranza de cheques, y es también en estas prácticas donde se ha hecho común la falsificación de firmas y donde surge la necesidad de crear sistemas de verificación que permitan validar la identidad de un usuario, partiendo del registro de una firma entrante, posterior comparación contra una base de datos de firmas registradas y análisis. Debido a que varias de las actividades de verificación se deben hacer en intervalos de tiempo relativamente cortos (0 a 5 min) es necesario que los sistemas de verificación requieran de poco tiempo para el procesamiento y análisis, además de fácil interpretación de resultados.

CAPITULO III

3 MARCO TEORICO

3.1 Aprendizaje Supervisado y Optimización

3.1.1 Linear Regression

El fin a conseguir mediante la regresión lineal es predecir el valor de $\boldsymbol{y}$ dado un vector con datos de entrada $\boldsymbol{x} \in \mathfrak{R}^{n}$ para ello, en un ejercicio propuesto como ejemplo se desea estimar el precio de una casa a partir de las características que la definen, por lo que y representa el precio de la casa y cada uno de los elementos $\boldsymbol{x_j}$ de $\boldsymbol{x}$ representan características que definen dicha casa, tales como superficie o número de habitaciones. Se representará el conjunto de características de la i-ésima casa como $\boldsymbol{x^{(i)}}$ y el precio como $\boldsymbol{y^{(i)}}$.

El objetivo pues, es encontrar una función que cumpla $\boldsymbol{y^{(i)} \approx h(x^{(i)})}$ para cada ejemplo del set de entrenamiento. En el caso de que se encuentre dicha función, a la que se le introducirán suficientes ejemplos de casas con sus precios, cabe esperar que sea capaz de predecir el precio, una vez dadas las características de la casa, aunque éste no sea conocido.

En pos de encontrar la función presentada anteriormente, se debe decidir cómo representarla. Para empezar, se utilizarán funciones lineales: $h_\theta(x) = \sum_j \theta_j x_{j=\theta^T x}$ Aquí, $h_\theta(\boldsymbol{x})$ representa una extensa familia de funciones parametrizadas en función de $\boldsymbol{\theta}$. El objetivo es, por tanto, encontrar una $\boldsymbol{\theta}$ tal que $h_\theta(\boldsymbol{x}^{(i)})$ sea lo más cercano a $\boldsymbol{y^{(i)}}$. Así pues, se busca una $\boldsymbol{\theta}$ que minimice:

$$\boldsymbol{J(\theta) = \frac{1}{2}\sum_{i}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = \frac{1}{2}\sum_{i}(\theta^T x - y^{(i)})^2}$$

Esta función se denomina coste y mide cuánto error se introduce en la predicción de $\boldsymbol{y^{(i)}}$ para una elección concreta de $\boldsymbol{\theta}$.

Como se ha dicho, se busca una $\boldsymbol{\theta}$ que minimice $J(\boldsymbol{\theta})$. Existe una gran variedad de algoritmos destinados a minimizar funciones, de hecho, en el apartado 4.3 se explicará con detalle uno de ellos. Sin embargo, por ahora se va a dar por supuesto que la mayoría de estos algoritmos necesitan que se les proporcione la función coste $\boldsymbol{J(\theta)}$ y el gradiente de dicha función $\nabla_\theta J(\boldsymbol{\theta})$ para cualquier valor de $\boldsymbol{\theta}$. Tras esto, el proceso de optimización restante para encontrar la mejor elección de $\boldsymbol{\theta}$ es llevado a cabo por el algoritmo de optimización.

El gradiente queda como:

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} = \begin{bmatrix} \sum_i x_1^{(i)}(h_\theta(x^{(i)}) - y^{(i)}) \\ \sum_i x_2^{(i)}(h_\theta(x^{(i)}) - y^{(i)}) \\ \vdots \\ \sum_i x_n^{(i)}(h_\theta(x^{(i)}) - y^{(i)}) \end{bmatrix}$$

Expresado en forma vectorial:

$$\nabla_\theta J(\theta) = \sum_i x^{(i)}(h_\theta\left(x^{(i)}\right) - y^{(i)})$$

Esta última ecuación es esencialmente la misma que la del gradiente para la regresión lineal, solo que ahora $\boldsymbol{h_\theta(x) = \sigma(\theta^T x)}$, por lo que, haciendo $\hat{\mathbf{y}} = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})\boldsymbol{-y}$ la equivalencia de la ecuación (3-5) sigue siendo válida.

3.1.2 Logistic Regression

Gracias a la regresión lineal se pueden predecir cantidades continuas como el precio de las casas como una función lineal que depende de los datos de entrada. Sin embargo, en algunas ocasiones resulta de interés predecir variables discretas, es decir, para predecir por ejemplo si un conjunto de píxeles representa un “0” o un “1”. Los problemas de este tipo son problemas de clasificación y la regresión logística es un algoritmo de clasificación bastante simple para aprender a tomar dichas decisiones.

En la regresión lineal, se probó a predecir los valores de $y^{(i)}$ usando una función lineal $\boldsymbol{y} = h_\theta(\boldsymbol{x}) = \boldsymbol{\theta}^T\boldsymbol{x}$. Este tipo de funciones no son las adecuadas para predecir valores binarios ($y^{(i)} \in \{0,1\}$), por lo que se usa una hipótesis diferente para predecir la probabilidad de pertenecer a una clase frente a la contraria. Concretamente, se usará una función de la forma:

$$P(y = 1|x) = h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \equiv \sigma(\theta^T x)$$

$$P(y = 0|x) = 1 - P(y = 1|x) = h_\theta(x)$$

La función $\sigma(z) \equiv \frac{1}{1+e^{-z}}$, también llamada función “logística”, contiene todos los valores de $\boldsymbol{\theta^T x}$ dentro del rango [0,1] para poder ser interpretado como probabilidad. El objetivo es encontrar un valor de θ tal que la probabilidad de $\boldsymbol{P}(\boldsymbol{y} = 1|x)$ sea alta cuando $\boldsymbol{x}$ pertenece a la clase “1” y pequeña cuando pertenezca a la clase “0”. La función de coste que se usara con un set de entrenamiento cuyos targets son binarios es:

$$J(\theta) = \sum_i (y^i \log\left(h_\theta\left(x^{(i)}\right)\right) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)})))$$

En la función anterior, solo uno de los dos términos de la sumatoria es distinto de cero para cada ejemplo del set de entrenamiento (dependiendo de la clase de $\boldsymbol{y}^{(i)}$ es "1" o "0"). Una vez que se tiene la función coste, es necesario aprender a clasificar los datos como "1" o "0" comprobando las probabilidades de pertenecer a cada una de las clases. Si $\boldsymbol{P}(\boldsymbol{y} = 1|x) > \boldsymbol{P}(\boldsymbol{y} = 0|x)$, entonces se clasifica como "1", "0" en caso contrario. esto último es lo mismo que comprobar $h_\theta(\boldsymbol{x}) > 0.5$.

Para minimizar la función coste se pueden utilizar las mismas herramientas que con la regresión lineal. Hay que proveer a la función optimizadora el cálculo del coste y del gradiente para cualquier valor de $\boldsymbol{\theta}$. En este caso, las componentes del gradiente se pueden calcular como:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_i x_j^{(i)} (h_\theta\left(x^{(i)}\right) - y^{(i)})$$

Escrito en forma vectorial, el gradiente se puede expresar como:

$$\nabla_\theta J(\theta) = \sum_i x_j^{(i)} (h_\theta\left(x^{(i)}\right) - y^{(i)})$$

Esta última ecuación es esencialmente la misma que la del gradiente para la regresión lineal, solo que ahora $\boldsymbol{h_\theta(x) = \sigma(\theta^T x)}$, por lo que, haciendo $\hat{\mathbf{y}} = \sigma(\boldsymbol{\theta}^T\boldsymbol{x})-\boldsymbol{y}$ la equivalencia de la ecuación (3-5) sigue siendo válida.

3.1.3 Softmax Regression

La regresión softmax, también llamada regresión logística multinomial, es una generalización de la regresión logística para el caso en el que se manejen más de dos clases. En la regresión logística, se asumió que los objetivos eran binarios $\boldsymbol{y^{(i)} \in \{0,1\}}$ y se utilizó el algoritmo anterior para

distinguir entre dos tipos de dígitos manuscritos. La regresión softmax permite que $y^{(i)} \in \{1, \dots, K\}$, donde K es el número de clases. En el caso que nos abarca, el set MNIST tiene 10 clases diferentes.

Dado una entrada $\boldsymbol{x}$, el objetivo sería estimar la probabilidad $\boldsymbol{P(y = k|x)}$ para cada valor de $k = 1, \dots, K$, por lo que la salida será un vector de K elementos cuya suma es igual a 1. Por tanto:

$$h_\theta(x) = \begin{bmatrix} P(y=1|X;\theta) \\ P(y=2|X;\theta) \\ \vdots \\ P(y=K|X;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} e^{\theta(j)^T x}} \begin{bmatrix} e^{\theta(1)^T x} \\ e^{\theta(2)^T x} \\ \vdots \\ e^{\theta(K)^T x} \end{bmatrix}$$

Aquí $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(K)} \in \boldsymbol{\Re^n}$ son los parámetros de nuestro modelo. Asimismo, el termino $\frac{1}{\sum_{j=1}^{K} e^{\theta(j)^T x}}$ normaliza la distribución de manera que la suma sea uno.

Por conveniencia, se usará $\boldsymbol{\theta}$ para denotar a todos los parámetros del modelo. Al implementar una regresión softmax, es útil representar $\boldsymbol{\theta}$ como una matriz de $n \times K$ obtenida de la concatenación en columnas de $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \dots, \boldsymbol{\theta}^{(K)}$, es decir:

$$\theta = \begin{bmatrix} | & | & & | & | \\ \theta^{(1)} & \theta^{(2)} & \dots & \theta^{(K)} \\ | & | & & | & | \end{bmatrix}$$

A continuación, se describirá la función coste que se usará para la regresión softmax. En la ecuación (3-11), 1{*} es una "función indicadora", de manera que 1{*a true statement*}=1 y 1{*a false statement*}=0. Por ejemplo 1{2+2=4}=1.. teniendo esto en cuenta, la función coste es:

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\{y^{(i)} = k\} log \frac{e^{\theta(k)^T x^{(i)}}}{\sum_{j=1}^{K} e^{\theta(j)^T x^{(i)}}}\right]$$

En la regresión softmax se tiene que:

$$P\left(y^{(i)} = k \middle| x^{(i)}; \theta\right) = \frac{e^{\theta^{(j)T} x}}{\sum_{j=1}^{K} e^{\theta^{(j)T} x}}$$

Debido a que no se puede obtener el mínimo de la función coste de forma analítica, es necesario utilizar una vez más un algoritmo de optimización iterativo. Este algoritmo necesita, a su vez el gradiente, que se puede expresar

como:

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m} \left[x^{(i)} (1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)};\ \theta)) \right]$$

En este caso, $\nabla_{\theta^{(k)}} J(\theta)$ es un vector tal que el j-ésimo elemento es $\frac{\partial J(\theta)}{\partial \theta_j^{(k)}}$, es decir, la derivada de *J(θ)* con respecto al j-ésimo elemento de $\theta^{(k)}$. con esto, ya se puede calcular el gradiente e introducírselo al algoritmo de optimización para minimizar *J(θ)*.

3.1.4 La Neurona Artificial

Si se considera un problema de aprendizaje supervisado en el que se tiene acceso a determinados ejemplos para el entrenamiento ($\boldsymbol{x}^{(i)}$, $y^{(i)}$), las redes neuronales permiten definir la forma de la función $h_{W,b}(\boldsymbol{x})$, la cual puede ser compleja o incluso no lineal.

Antes de describir redes neuronales, se empezará describiendo la red neuronal más sencilla posible, es decir, aquella que se corresponde con una sola “neurona”. El diagrama de una neurona es el siguiente:

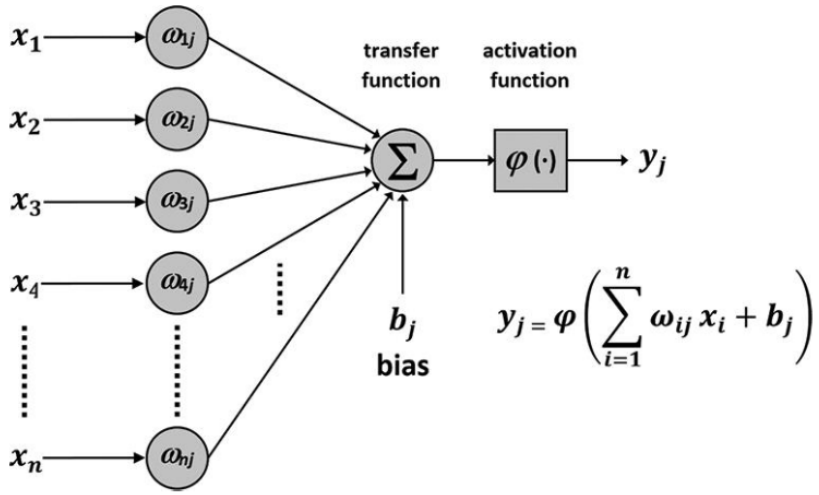


Figura 01: diagrama de una red neuronal artificial

Esta neurona es la unidad computacional que toma como entradas x_1, x_2, x_3, x_4, … , x_n y produce una salida de acuerdo con la ecuación $y = h_{W,b}(x) = F(W^T x + b) = f(\sum_{i=1}^{n} W_i x_i + b)$, donde f (φ en la Figura 01) es la función de activación y su dominio e imagen son $f: \mathfrak{R} \rightarrow \mathfrak{R}$. se tomara $f(*)$ como la función sigmoide:

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

Por tanto, nuestra neurona simple se corresponde exactamente con el algoritmo de regresión logística visto en el apartado 2.2.1.2

Aunque se va a utilizar la función sigmoide, cabe destacar que otra elección bastante común de f es la tangente hiberbólica o tanh:

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Además, investigaciones recientes han encontrado una función de activación diferente: la función lineal rectificada (rectified linear function o ReLU en inglés). Esta función trabaja bien en redes neuronales con Deep

Learning y es diferente de la sigmoide y la tanh pues no está limitada y no es continuamente diferenciable. Esta función viene dada por:

$$f(z) = \max(0, z)$$

A continuación, se muestran en una sola gráfica las tres funciones mencionadas anteriormente para su mejor visualización y comparación:

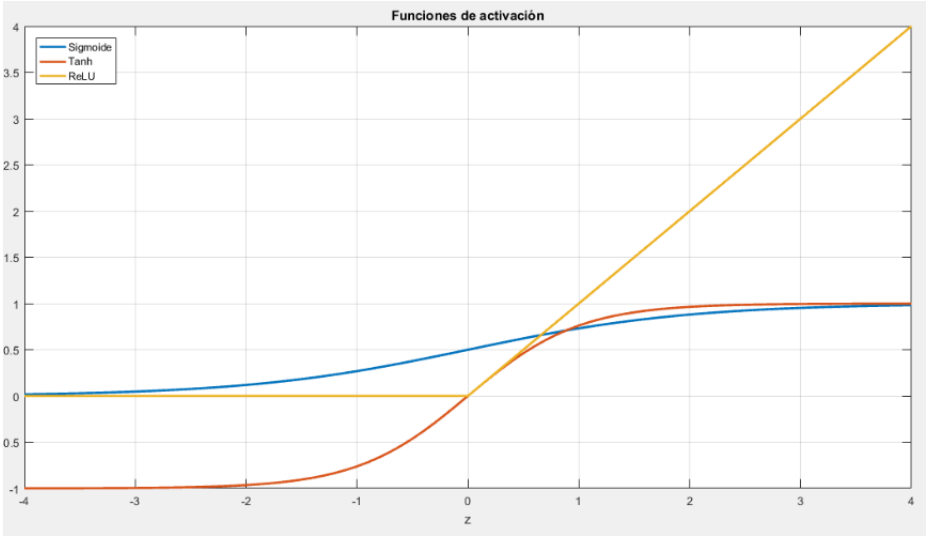


Figura 02: Comparación de funciones de activación

La función tanh (z) es una versión reescalada de la sigmoide, pues el rango de salida es de [−1,1] en vez de ser de [0,1]. La función lineal rectificada es una función lineal definida a trozos que satura el valor a 0 cuando la entrada z es menor que dicho valor.

Por último, comentar una igualdad que será de utilidad en el futuro: si $f(z) = 1/(1 + e^z)$ es la función sigmoide, entonces su derivada viene dada por $f'(z) = f(z)(1 - f(z)))$. Si, por el contrario, $f(z)$ es la función de tanh, entonces su derivada resulta ser $f'(z) = 1 - (f(z))^2$. La función lineal rectificada tiene un gradiente de 0 cuando $z \leq 0$ y 1 en cualquier otro caso.

3.1.5 Modelo de Red Neuronal

Una red neuronal se crea interconectando las neuronas simples que se vio en el apartado anterior de manera que la salida de una neurona puede ser la entrada de otra. He aquí una pequeña red neuronal:

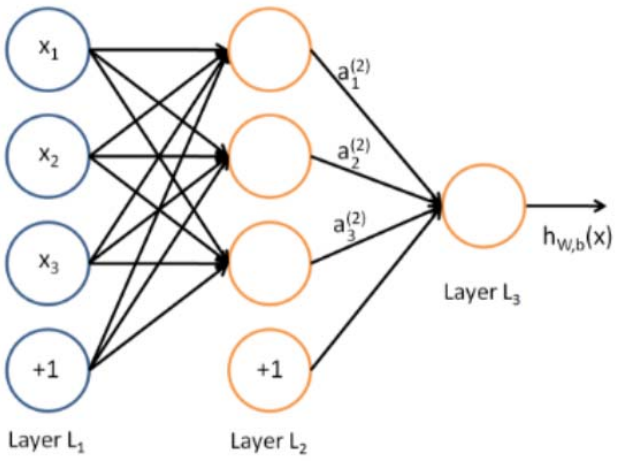


Figura 03: Red neuronal con varias neuronas

En la figura anterior, se han usado círculos para denotar también a las entradas de la red. Los círculos con un "+1" en su interior se corresponden con el término de bias. La capa de más a la izquierda se llama capa de entrada y la capa de más a la derecha, capa de salida (que en este caso solo tiene un nodo o neurona). La capa intermedia se llama capa oculta porque sus valores no se observan en el set de entrenamiento. La red de este ejemplo tiene 3 unidades de entrada (sin contar la bias), 3 unidades ocultas y 1 unidad de salida.

A partir de ahora se denotará el número de capas de nuestra red con n_l, por lo que, en el ejemplo anterior n_l = 3. A las distintas capas se las llamará L_l, de manera que L_1 se corresponde con la capa de entrada y la capa L_{nl} es la de salida. Dicha red tiene como parámetros $(\boldsymbol{W}, b)$ =$(\boldsymbol{W}^{(1)}, b^{(1)}, \boldsymbol{W}^{(2)},$

$b^{(2)}$), donde W_{ij}^{l} es el parámetro (o peso) asociado a la conexión entre la neurona j de la capa l y la neurona i de la capa l + 1 y b_i^l es la bias asociada con la neurona i en la capa l + 1. Por tanto, se tiene que $\boldsymbol{W}^{(1)} \in \Re^{3x3}$ y $\boldsymbol{W}^{(2)} \in \Re^{1\times3}$.

Se denotará $a_i^{(2)}$ como la activación (el valor de salida) de la i-esima neurona de la capa l. Dado un conjunto de parámetros fijos $\boldsymbol{W}$, b, la función $h_{W,b}(x)$ de la red neuronal da como resultado un número real. En concreto, la salida resultante de la red de la figura 03 viene dada por:

$$a_1^{(2)} = f\left(W_{11}^{(1)} + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}\right)$$

$$a_2^{(2)} = f(W_{21}^{(1)} + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$a_3^{(2)} = f\left(W_{31}^{(1)} + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_1^{(1)}\right)$$

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

Asimismo, se denotará z_i^l como la suma ponderada de todas las entradas a la i-ésima neurona de la capa l, término de bias incluido. Por ejemplo, $z_i^2 = \sum_{j=1}^{n} W_{ij}^{(1)}x_j + b_i^{(1)}$, es decir, $a_i^l = f(z_i^l)$. Cabe destacar que esto tiende a una notación aún más compacta si se extiende a forma vectorial, ya que ahora la ecuación (4-4) se puede sobre escribir como:

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f\left(z^{(2)}\right)$$

$$z^{(3)} = W^{(2)}a^2 + b^{(2)}$$

$$h_{W,b}(x) = a^3 = f(z^3)$$

A esto se le llama froward propagation (propagación hacia adelante). Además, se suele usar $a^{(1)} = x$ para denotar a los valores de la capa de entrada. Las actividades de la capa $l + 1$, dadas las actividades de la capa anterior, se puede calcular:

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$

$$a^{(l+1)} = f\left(z^{(l+1)}\right)$$

La ventaja de agrupar los parámetros en matrices y utilizar operaciones vectoriales y matriciales es que los cálculos se ejecutan de manera más rápida gracias al algebra lineal.

Lo expuesto hasta ahora sirve para la red de la figura 4-4, pero también es una generalización que puede ser aplicada a arquitecturas diferentes de redes neuronales, incluyendo aquellas con varias capas ocultas. La

opción mas común es una red neuronal de n_l capas en la que la capa 1 es la de la entrada, la n_l es la salida y cada una de las l capas está totalmente conectada a la siguiente. Con esta configuración, para calcular la salida de la red, se puede calcular las activaciones de la capa L_2, luego de la capa L_3 y asi sucesivamente hasta la capa L_{n_l} usando las ecuaciones anteriores que describen la forward propagation. Este es un ejemplo de una red tipo feedfordward, ya que no tiene ningún bucle ni ciclo.

Lsa redes neuronales pueden tener múltiples unidades o neuronas de salida, por ejemplo, a continuación, se muestra una red con dos capas ocultas L_2 y L_3 y una capa de salida L_4 con dos unidades de salida:

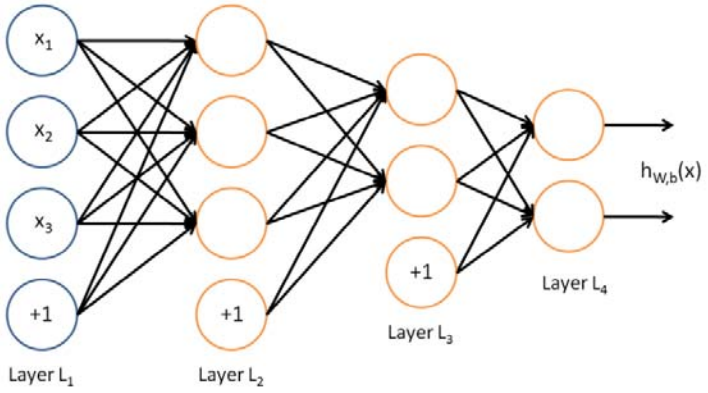


Figura 04: Red neuronal multicapa

Para entrar esta red, se necesitarían ejemplos de entrenamiento $(x^{(i)}, y^{(i)})$ donde $y^{(i)} \in \mathfrak{R}^2$. Este tipo de redes son útiles si existen múltiples salidas que interesen ser predichas. Por ejemplo, en una aplicación de diagnóstico médico, el vector x serviría como las características de entrada de un paciente. Y las diferentes salidas y_i podrían indicar la presencia o ausencia de diferentes enfermedades.

3.1.6 Backpropagation Algorithm

Si ahora se supone un set de entrenamiento $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ de m ejemplos. Se entrenará a la red neuronal usando el algoritmo de gradient descen (descenso por gradiente). Es más, para un ejemplo concreto (x, y), se define la función coste con respecto a ese ejemplo como:

$$J(W, b; x, y) = \frac{1}{2} ||\boldsymbol{h_{W,b}}(\boldsymbol{x}) - \boldsymbol{y}||^2$$

Dado un conjunto de m ejemplos de entrenamiento, se puede definir la función coste total como:

$$J(W, b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W, b; x^{(i)}, y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{nl-1}\sum_{i=1}^{sl}\sum_{j=1}^{sl+1}\left(W_{ji}^{(l)}\right)^2 = \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left|\left|\boldsymbol{h_{W,b}}(\boldsymbol{x}) - \boldsymbol{y}^{(i)}\right|\right|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{nl-1}\sum_{i=1}^{sl}\sum_{j=1}^{sl+1}\left(W_{ji}^{l}\right)^2$$

Las dos ecuaciones anteriores difieren ligeramente pues los pesos tienen un weigth decay aplicado y, por el contrario, el termino de bias no.

Dado un ejemplo de set de entrenamiento (x, y), lo primero a realizar es calcular todas las actividades a lo largo de la red, incluido el valor de salida de la función $h_{W,b}(x)$. Seguidamente, para cada nodo i de la capa l, seria conveniente calcular un "termino de error" $\delta_i^{(l)}$ que mida cuan responsable es dicho nodo del error cometido a la salida de la red. Para los nodos en la capa de salida, se puede medir directamente la diferencia ente el resultado y el valor real o target y definir dicha diferencia como $\delta_i^{(nl)}$, donde n_l es la capa de salida. Para las neuronas de las capas ocultas se

calculará $\delta_i^{(l)}$ según una medida ponderada de los términos de error de los nodos que se toman $a_i^{(l)}$ como entrada.

A continuación, se muestra en detalle el algoritmo de retro programación:

1. Calcular las actividades de las n_l capas de la red utilizando el método de feedforward visto en el apartado 4.3, concretamente en la ecuación (4-4)
2. Para la i-ésima unidad en la capa de salida, establecer:

$$\delta_i^{nl} = \frac{\partial}{\partial z_i^{(nl)}}\left(\frac{1}{2}\left\|\boldsymbol{y} - \boldsymbol{h}_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{x})\right\|^2\right) = -\left(y_i - a_i^{(n_l)}\right).f'(z_i^{(n_l)})$$

3. Para $l = n_l - 1,\ n_1 - 2,\ n_1 - 3, \ldots, 2$ establecer para cada nodo i de la capa l:

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)}\delta_j^{(l+1)}\right).f'(z_i^{(l)})$$

4. Calcular las derivadas parciales deseadas, que vienen dadas por:

$$\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b;x,y) = a_j^{(l)}\delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}}J(W,b;x,y) = \delta_i^{(l+1)}$$

Además, el algoritmo antes descrito se puede reescribir en forma matricial con el fin de acelerar el tiempo en el que se realiza los cálculos. Con este fin, se utilizará el operador producto elemento a elemento (también llamado producto Hadamard) y se denotará como "⊙". El algoritmo queda ahora como:

1. Calcular las actividades de las n_l capas de la red utilizando el método de feedforward visto en el apartado 4.3, concretamente en la ecuación (4-6)
2. Para la capa de salida, establecer:

$$\delta^{n_l} = -(y - a^{n_l}) \odot f'(z^{n_l})$$

3. Para $l = n_l - 1,\ n_1 - 2,\ n_1 - 3, \dots, 2$ establecer:

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{l+1}\right) \odot f'(z^l)$$

4. Calcular las derivadas parciales deseadas, que vienen dadas por:

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{l+1} (a^{(l)})^T$$

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{l+1}$$

Por último, el algoritmo completo para calcular el gradient descent expresado en pseudocodigo es el siguiente:

1. Establecer $\Delta W^{(l)} \coloneqq 0, \Delta b^{(l)} \coloneqq 0$ (matriz/vector de ceros) para todo l
2. Desde $i = 1\ hasta\ m$,
   a) Usar la retropropagación para calcular $\nabla_{W^{(l)}} J(W, b; x, y)$ y $\nabla_{b^{(l)}} J(W, b; x, y)$
   b) Establecer $\Delta W^{(l)} \coloneqq \Delta W^l + \nabla_{W^{(l)}} J(W, b; x, y)$
   c) Establecer $\Delta b^{(l)} \coloneqq \Delta b^{(l)} + \nabla_{b^{(l)}} J(W, b; x, y)$
3. Actualizar parámetros:

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)}\right) + \lambda W^{(l)}\right]$$

$$b^{(l)} = b^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta b^{(l)} \right) \right]$$

Para entrenar a la red neuronal, se puede realizar el método del gradient descent repetidamente para reducir la función coste $J(W, b)$

3.1.7 Redes Neuronales Convolucionales

Cuando se trabaja con imágenes, como en el caso de las redes neuronales convolucionales, se suele hacer con imágenes con resoluciones muy altas (por ejemplo, el standard actual es Full HD que consta de 1,980 x 1,080 píxeles). Esto conlleva un grave problema en las redes neuronales convencionales, que se han visto anteriormente. Éstas, al ser full-connected, tienen cada una de las neuronas que conforman la primera capa oculta conectada con todos y cada uno de los píxeles de la imagen de entrada a la vez. Si estas imágenes de entrada son de una resolución asumible, por ejemplo 28 x 28, el número de elementos que habrá que entrenar será lo suficientemente comedido como para que el sistema pueda funcionar correctamente, aunque todo el proceso será significativamente más lento. Sin embargo, si la resolución aumenta ligeramente (y no digamos ya a resoluciones actuales) los tiempos de entrenamiento y testeo se vuelven enormes.

A continuación, se expondrá la magnitud del problema anterior de manera cuantitativa. Si, por ejemplo, se usan imágenes de 96 x 96 se tendrían alrededor de 10^4 elementos de entrada y, si se asume que se quieren aprender 100 características, el número de elementos a entrenar sería del orden de 10^6. Esta enorme cantidad de elementos que deben ser

entrenados hace que la duración del entrenamiento sea extraordinariamente larga, dificultando, en gran medida, el uso de las redes neuronales convencionales para aplicaciones con imágenes.

Una vez se es consciente de la magnitud del problema, se hace evidente la necesidad de usar algún tipo diferente de sistema, que permita sortear esta cuestión. En los siguientes puntos de este tema, se verá cómo se resuelve este problema.

Una red neuronal convolucional es un tipo de red multicapa que consta de diversas capas convolucionales y de pooling (submuestreo) alternadas, y al final tiene una serie de capas full-connected como una red perceptrón multicapa. La entrada de una red capa convolucional suele ser, generalmente, una imagen $m\ x\ m\ x\ r$, donde m es tanto la altura como el ancho de la imagen de la imagen y *r* es el número de canales. Las capas convolucionales tienen k filtros (o kernels) cuyas dimensiones son $n\ x\ n\ x\ q$, donde n y q son elegidas por el diseñador (generalmente que suele ser igual a r). Cada filtro genera mediante convolución un mapa de rasgos o características de tamaño $(m - n + 1)x\ (m - n + 1)x\ p$, siendo p el número de filtros que se desean usar. Después cada mapa es sub-muestreado en la capa de pooling con la operación “mean pooling” o “max pooling” sobre regiones contiguas de tamaño $p\ x\ p$ donde p puede tomar valores desde 2 para imágenes pequeñas hasta, comúnmente, no más de 5 para imágenes grandes. Antes o después del submuestreo, se aplica una función de activación sigmoidal más un sesgo para cada mapa de rasgos.

La composición de bloques anterior se puede ver en la Figura, en la que está representada una disposición genérica de las capas.

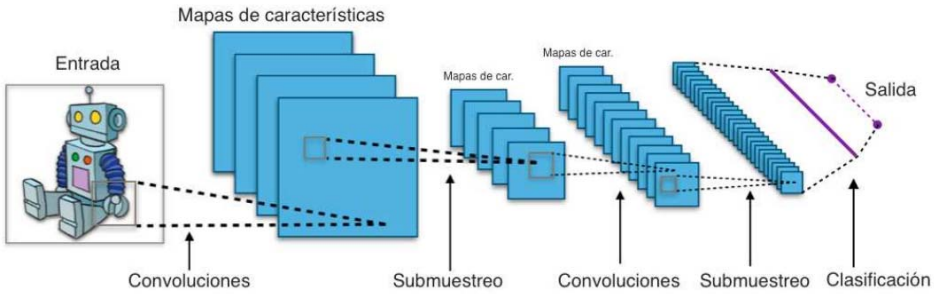


Figura 05: Esquema básico de una red neuronal convolucional.

Además, las redes convolucionales están diseñadas suponiendo que la entrada a la red es una imagen, lo cual permite codificar ciertas propiedades en la arquitectura, permitiendo ganar eficiencia y reducir la cantidad de parámetros usados en la red.

Capa de Entrada

La primera capa de esquema de una red convolucional se trata de la entrada a la red. Como se ha visto anteriormente, en las redes convolucionales se tiene que la entrada a la red es una imagen. Para mejor entendimiento usaremos la base de datos MNIST por lo que estas imágenes de entrada tendrán dimensiones de $28\ x\ 28$ y estará en escala de grises.

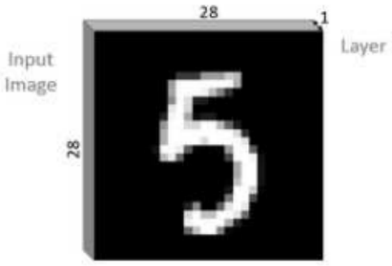


Figura 06: Entrada a la red convolucional

En concreto, se le pasaran a la red imágenes de entrada para llevar a cabo el proceso de entrenamiento y, una vez entrenada, se comprobara su correcto funcionamiento con imágenes de test.

Capa Convolucional

Es recomendable reducir la carga computacional del sistema, la capa convulocional toma un importante papel en este aspecto. Para ello, durante esta etapa de la red, se lleva a cabo una solución muy simple: se restringe el numero de conexiones posibles entre las neuronas de la capa oculta y los elementos de la imagen de entrada. De esta manera, cada neurona oculta solo estará conectada con un pequeño subconjunto de elementos de la imagen total. Esta idea esta inspirada en como funciona el sistema visual biológico, debido a que las neuronas que trabajan en el cortex visual poseen una serie de campos receptivos que responden solo ante unos estimulos localizados en una region o área específica.

Además, otra característica reseñable es que las imágenes naturales poseen la propiedad de ser “estacionarias”, esto quiere decir que las características o rasgos que existen en alguna parte determinada de la imagen pueden ser los mismos que otros que se encuentren en otra zona totalmente distinta.

Por otro lado, la convolución es una operación de productos y sumas entre la imagen de entrada y un filtro (o kernel) que genera un mapa de características. La ventaja es que el mismo filtro sirve para extraer el mismo rasgo en cualquier parte de la imagen, atendiendo al carácter estacionario de las imágenes que se ha comentado. Esto permite reducir

el número de conexiones y el número de parámetros a entrenar en comparación con una red multicapas full-connected.

En conclusión, la capa convolucional permite tanto reducir el número de elementos que conforman la red como detecta una serie de características que serán útiles a la hora de analizar la imagen. Considerando ello se puede precisar de que la capa de convolución es de gran utilidad en el análisis de imágenes permitiendo reducir la complejidad del sistema y, al tiempo, extraer rasgos útiles de estas que ayuden con su análisis.

La entrada de una capa convolucional es una imagen $m\ x\ m\ x\ r$, a esta imagen se le aplica un filtro de dimensiones $n\ x\ n\ x\ q$ y, finalmente, como resultado se obtiene una matriz de características de tamaño $(m - m + 1)\ x\ (m - n + 1)\ x\ p$. El comportamiento se consigue realizando el procedimiento explicado a continuación:

Cuando le llega la imagen de entrada a la red, se superponen el filtro y ésta y se calcula la convolución de dos dimensiones entre los respectivos elementos de la imagen y el kernel. Una vez se obtiene el resultado de la operación anterior, se almacena en una posición de la matriz de activación, como se pude observar en la Figura 07.

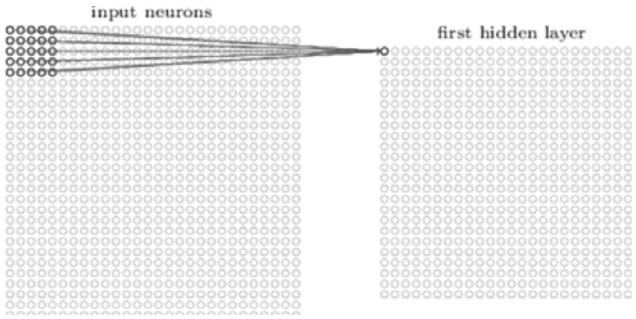


Figura 07: Operación básica de convolución

A continuación, se desplaza o desliza el filtro una posición a la derecha sobre la imagen y se vuelve a calcular la convolución, almacenando el resultado en la siguiente posición de la matriz de activación. De esta manera, este proceso iterativo se repite a lo largo de toda la imagen desplazándose de izquierda a derecha y bajando una unidad al llegar a un borde. Una vez se ha recorrido toda la imagen se obtiene la matriz de activación completa que contiene las características que se buscan en la imagen para cada filtro. Este proceso se puede observar con mayor claridad en la figura 08.

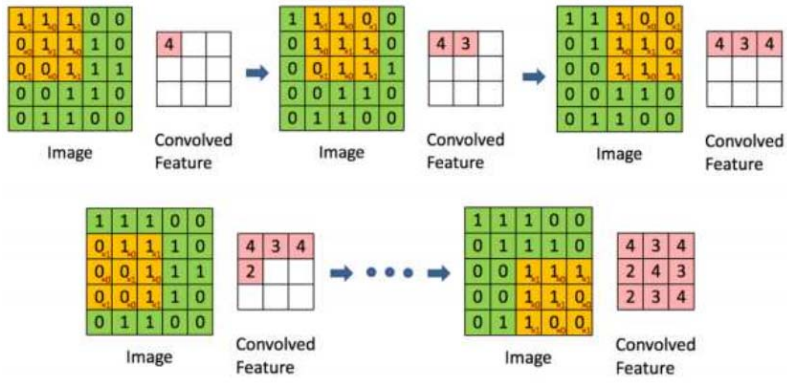


Figura 08: Obtención de la matriz de activación

Si, por ejemplo, tenemos una imagen de $28\ x\ 28\ x\ 1$ y le aplicamos 6 filtros de $5\ x\ 5\ x\ 1$, la matriz de activación resultante será de $(28 - 5 + 1) = 24\ x\ 24\ x\ 6$, como se muestra en la Figura 09.

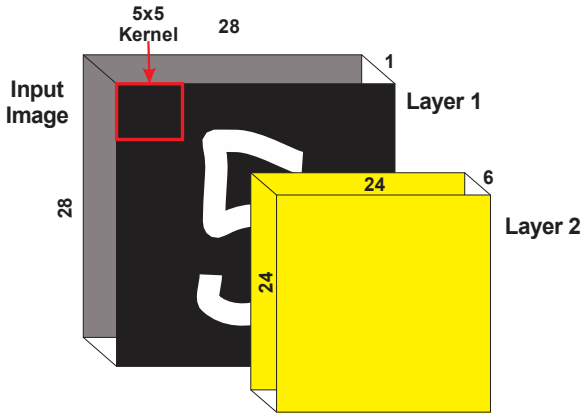


Figura 09: Resultado de aplicar la convolución sobre la imagen de entrada

Capa de Pooling

Siguiendo el esquema de la Figura 4-1, inmediatamente después de la capa de convolución se encuentra la capa de pooling. Después de haber obtenido las características en la capa de convolución, el siguiente paso en la lista es usarlas para la clasificación de las imágenes. En teoría, ya se podría usar cualquier tipo de clasificador para llevar a cabo esta tarea, sin embargo, este proceso aún puede ser muy costoso computacionalmente hablando. Por ejemplo, se considera lo siguiente: si la imagen de entrada es de $96\,x96$ píxeles, a la salida de la capa convolucional se tendrá que el número de elementos será de $(96-8+1)*(96-8+1)=7921$ y, si se tienen, por ejemplo, 400 rasgos, el vector resultante es de $89^2\,x\,400=3\,168\,400$ características. Una red con ese ingente número de características tendrá un comportamiento muy lento y, seguramente, se producirán errores de sobre-ajuste durante el aprendizaje.

Como se vio en el apartado 4.3, las imágenes tienen la propiedad de ser estacionarias y en la capa convolucional se ha creado un mapa de características. Así, el siguiente paso natural es ver en qué zona de la imagen son esos rasgos predominantes. Para realizar esta labor, es posible calcular la media o buscar el máximo valor de una característica a lo largo de una región de la imagen. La matriz resultante de la operación anterior resulta de unas dimensiones considerablemente menores que la matriz de características, obtenida en la capa de convolución.

Entonces, el objetivo de esta capa es el disminuir aún más la carga computacional del sistema y, al mismo tiempo, ayudar con la caracterización de la imagen obteniendo y localizando los rasgos predominantes en ella. Además, como se comentó en el párrafo, es importante destacar que existen dos tipos de pooling: meanpooling o average-pooling y max-pooling. En la Figura 10, pueden verse los efectos de aplicar un tipo de pooling u otro.

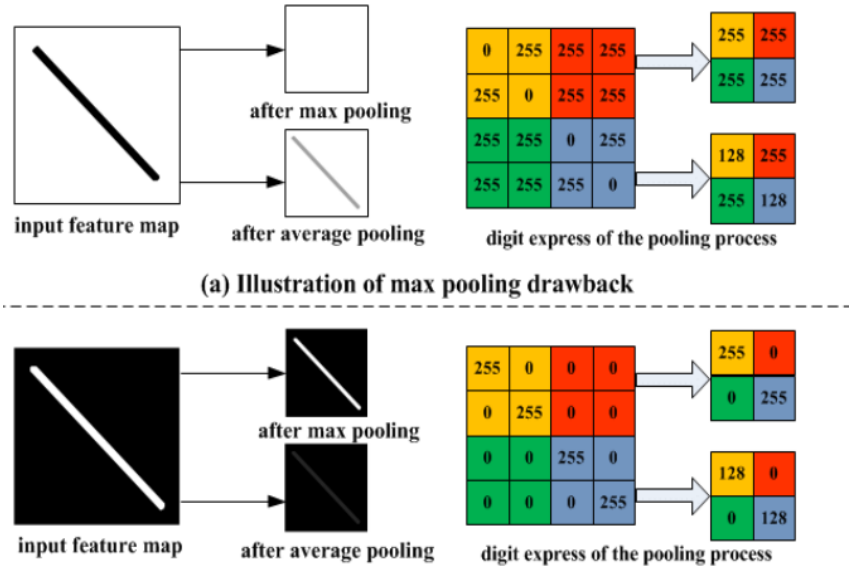


Figura 10: Resultado de aplicar diferentes tipos de pooling.

El resultado de aplicar la capa de pooling, en cuanto a disminución de dimensiones como se ha discutido anteriormente, se puede apreciar en la figura 10.

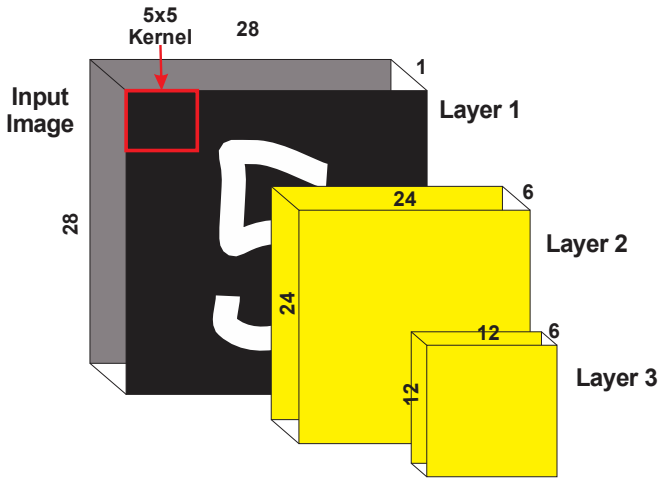


Figura 10: Salida de la capa de pooling.

El funcionamiento de la capa de pooling es el siguiente: en primer lugar, desde la esquina superior izquierda de la matriz de activación se selecciona una submatriz de tamaño $p\ x\ p$. A continuación, dependiendo del tipo de pooling se hará lo siguiente: si se trata de mean-pooling o average-pooling se cogen todos los elementos de la submatriz, se calcula su media y el resultado se guarda en la primera posición de la matriz de salida, si es max-pooling se busca el elemento de mayor valor que se encuentre en esa submatriz y se guarda en la primera posición de la matriz de salida. A continuación, se selecciona la siguiente submatriz que está a $p + 1$ posiciones a la derecha y se realiza la misma operación. Una vez llegado al borde derecho de la imagen, se vuelve a la izquierda de la imagen, se da un salto hacia abajo de $p + 1$ elementos y se vuelve a

recorrer la imagen de izquierda a derecha. Entonces, si la matriz de activación

creada por la capa de convolución es de tamaño $n\ x\ n\ x\ q$, la matriz de salida de la capa de pooling será de $\frac{n}{p}\ x\ \frac{n}{p}\ x\ q$. El procedimiento anterior puede verse en la Figura 11.

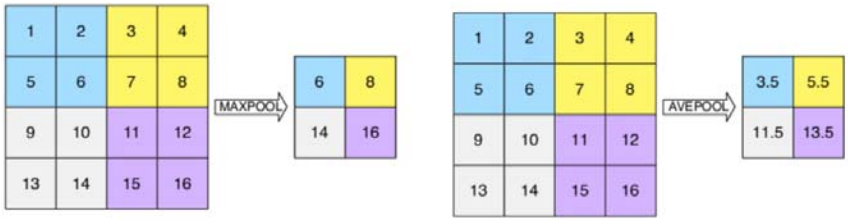


Figura 11: Operación de max-pooling y de average-pooling

Capa Full-Connected

Ésta se trata de la última capa del esquema de las redes neuronales convolucionales y se trata de un clasificador que determina a que clase pertenece la imagen de entrada, es decir, su trabajo en este proyecto es indicar que número 'cree' la red que se le ha proporcionado a la entrada.

La capa totalmente conectada se encuentra justo a continuación de la última capa de pooling. Está compuesta por un número de neuronas igual al número de clases, en el caso de este proyecto son 10 neuronas (son 10 clases: los números del 0 al 9). El esquema que sigue esta capa es el de modelos Neuronales, cuando se habló de las redes neuronales totalmente conectadas. Es decir, cada una de estas neuronas está a su

vez conectada con todos y cada uno de los elementos de las matrices de la capa inmediatamente anterior.

A la salida de esta capa se encontrará un vector de 10 componentes. Cada una de estas componentes representa la probabilidad que tiene la imagen de entrada de pertenecer a una determinada clase, lo que es lo mismo, la probabilidad que tiene de ser un número u otro. Por último, la red determinará cuál de estos elementos del vector es mayor y lo indica.

3.2 Hipótesis de investigación

3.2.1 Hipótesis general

H0: El uso de redes neuronales convulucionales no presenta grado de verosimilidad para el reconocimiento de Caracteres manuscritos (firmas)

H1: El uso de redes neuronales convolucionales presenta grado de verosimilidad para el reconocimiento de Caracteres manuscritos (firmas)

3.3 Variables e indicadores

3.3.1 Identificación de variables

en el presente proyecto de tesis denominado "***Sistema de reconocimiento de caracteres manuscritos usando redes neuronales convolucionales***".

Variable 1:

Redes neuronales convolucionales.

Variable 2:

Sistema de reconocimiento de caracteres manuscritos (firmas).

3.3.2 Clasificador de Variables.

Para este proyecto, se utilizará dos variables:

Variables independientes : Redes neuronales convolucionales.

Variables dependientes : Sistema de reconocimiento de caracteres manuscritos (firmas).

3.3.3 Definición de variables

- **Redes neuronales convolucionales**

  Por la función que cumplen en la hipótesis: Variable independiente.

  Por su naturaleza: Desactiva.

  Por el método de estudio: Cualitativo.

  Por la posesión de la característica: Discreta.

  Por los valores que adquiere: Politómica.

- **Sistema de reconocimiento de caracteres manuscritos (firmas)**

  Por la función que cumplen en la hipótesis: Variable dependiente.

  Por su naturaleza: Activa.

  Por la posesión de la característica: Continuo.

  Por los valores que adquiere: Politómica.

3.3.4 Operacionalizacion de variables

Variable independiente

- Variable independiente: Redes neuronales artificiales convolucionales.
- Definición conceptual: Las redes neuronales artificiales son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso.
- Definición operacional: Para tener una red neuronal la cual se ajuste de manera adecuada a los datos, se tiene que realizar varias

configuraciones como el número de capas ocultas, el número de neuronas por cada capa y las funciones de transferencias.

- Indicadores: Valor de validación (aceptación de resultado obtenido por la red), valor del error cuadrático medio (error de entrenamiento de la red).
- Valores: Numéricos (expresado en porcentaje).

Variable dependiente.

- **Variable dependiente:** reconocimiento de caracteres manuscritos (firmas).
    - Definición conceptual: grado de verosimilidad en el reconocimiento de caracteres manuscritos (firmas).
    - Definición operacional: Para realizar la predicción usando series de tiempo el cual es necesario que el conjunto de datos cumpla cierta hipótesis para que, en primer lugar, se busque un modelo estadístico que se ajuste a estos de manera adecuada y posteriormente realizar la predicción.
    - Indicadores: Cantidad (el número total de los datos), valor máximo (el mayor número de los datos), valor mínimo (el menor número de los datos).

3.4 Tipo y diseño de la investigación

Diseño experimental

La presente investigación está enfocada en un diseño experimental, puesto que la firma ingresada será modificada mediante matrices de transición en 0º, 45º, 90º y 135º grados, para posteriormente ser comparada con el

Dataset previamente guardada, para efectuar la comparación mediante el método de ajuste de distancia.

Población y muestra

Población: Se ha seleccionado mediciones de volumen de las firmas de los postulantes a la Universidad Nacional Jose Maria Arguedas – Andahuaylas.

Muestra: Debido a que los datos o se las firmas están almacenadas y son manipulados electrónicamente de los postulantes a las 6 carreras profesionales, fue necesario de establecer como población las firmas de los postulantes a la Carrera Profesional de Ingeniería de Sistemas y como muestra de validación los ingresantes a la misma.

3.5 Método de investigación

Se usará para el presente trabajo de investigación el método descriptivo puesto que se basa en observaciones y además se evidencia los cuatro factores psicológicos como son atención, sensación, percepción y reflexión.

3.6 Técnicas de instrumentos de acopio de datos

La recopilación de datos es importante para establecer las bases de parámetros para el entrenamiento de la red neuronal artificial y la búsqueda de diferente información documentaria que guarde relación con la investigación.

Métodos Pronóstico: Método de diseño de pronóstico con redes neuronales artificiales.

Técnicas: Algoritmo Back Propagation – Perceptrón multicapa.

Herramientas: Lenguaje de programación Python

Fuente: Los datos empleados fueron obtenidos de la biblioteca especializada de la Escuela Profesional de Ingeniería de Sistemas de la Universidad Nacional José María Arguedas.

3.7 Técnicas de análisis de datos

Para la implementación de las redes neuronales artificiales para evaluar el grado de verosimilidad del texto manuscrito (firmas), no fue necesario realizar el pre-procesamiento de datos del volumen de postulantes, dado que los datos se encontraron libres de ruido (no contiene valores nulos ni valores con formato inconsistente). La confiabilidad del resultado estuvo dada por el valor del test validación y el algoritmo de validación de verosimilidad.

Selección de Pruebas Estadísticas

Por el diseño de la investigación utilizado no hubo pruebas estadísticas. Los resultados obtenidos de la red neuronal artificial sirvieron para verificar el porcentaje del test de validación y verosimilidad.

CAPITULO IV

4 DISEÑO METODOLOGICO

4.1 Descripción

Para llevar a cabo un reconocimiento de formas es necesario realizar un modelado del proceso de razonamiento y aprendizaje, además se debe tener en cuenta que existe una imposibilidad intrínseca de alcanzar resultados exactos.

El modelado del proceso de percepción es difícil de formalizar (es raro que un “experto” pueda verbalizar sus habilidades). Además, requiere una extraordinaria plasticidad: aprendizaje (inconsciente) mediante exposición reiterada de los problemas a resolver (y de sus soluciones).

Para el procesamiento de la percepción existen dos posibles enfoques como es el caso de inteligencia artificial y reconocimiento de formas.

4.2 Preproceso

Dependiendo del algoritmo que se elija en la selección de características, en la elección del clasificador o en la interpretación de resultados puede ser necesario emplear distintos tipos de preproceso para incrementar el índice de fiabilidad o la velocidad de reconocimiento

4.2.1 Limpieza y eliminación de ruido

En muchos de los casos cuando se escanea la firma, por error se pueden introducir ruido, es p0r ello que se hace necesario considerar los trazos más significativos de la firma manuscrita y desestimar los detalles

adicionales, los ruidos serán obviados por la etapa de extracción de trazos, aunque un preproceso en este sentido servirá para incrementar el rendimiento en etapas posteriores.

La eliminación de ruido gaussiano o ruido uniforme, puede ser tratado con un filtro lineal, con una máscara determinada que elimine los píxeles solitarios o con un filtro no lineal como puede ser el filtro de la mediana que consiste en reemplazar el nivel de gris de cada píxel por la mediana de los valores de grises de los vecinos

4.2.2 Esqueletizado

Dado que lo que se busca es la dirección y la forma que tiene la firma y no tanto sus pequeños detalles como las pequeñas manchas, signos de puntuación o el grosor del bolígrafo que además, es un factor que puede variar significativamente de una muestra a otra, parece razonable pensar que un adelgazamiento de la firma podría ser de gran ayuda.

La esqueletización o adelgazamiento es una técnica muy usada en el reconocimiento de patrones. Consiste en eliminar píxeles de la imagen hasta que queda un esqueleto de un píxel de grosor. Idealmente, este esqueleto contiene toda la información relevante de la región, conservando el mismo número de regiones de la imagen, y debe ser similar a la región original. Esto resulta complicado de conseguir y puede introducir ciertos “artefactos” de ruido no presentes en la firma original. De hecho, aunque ha habido muchos intentos de

resolver el problema, no hay todavía una solución completamente satisfactoria.

La idea del algoritmo es que la esqueletización se puede ver como la eliminación de capa tras capa de píxeles de frontera, es decir, como erosiones repetidas, pero teniendo cuidado de no eliminar los píxeles que mantienen las relaciones de conexión. El algoritmo que se ha utilizado es el de Zhang y Suen que supone que los píxeles del objeto están aquellos etiquetados como uno, que tienen al menos uno de los ocho vecinos a cero.

Para ello, teniendo en cuenta que la imagen está definida como una matriz de 1 o 0, en la que 1 representa un pixel de la imagen (o un pixel de color negro) y 0 representan un espacio en blanco, empezamos por calcular los pixeles vecinos al pixel a evaluar cómo se muestra en la siguiente imagen:

P9*(i-1, j-1)*	P2*(i-1, j)*	P3*(i-1, j+1)*
P8*(i, j-1)*	P1*(i, j)*	P4*(i, j+1)*
P7*(i+1, j-1)*	P6*(i+1, j)*	P5*(i+1, j+1)*

Figura12: Matriz de pixeles vecinos

En esta imagen podemos ver que el pixel evaluado es el que nombraremos como P1 y que sus vecinos son todos aquellos pixeles que colindan con este.

Primera sub-iteración: un píxel del contorno se marca para borrarlo si cumple las siguientes condiciones y en caso de cumplirlas marcamos el pixel para borrar:

a) $2 \leq N(p_1) \leq 6$

b) $S(p_1) = 1$

c) $p_2 * p_4 * p_6 = 0$

d) $p_4 * p_6 * p_8 = 0$

donde N(p_1) es el número de vecinos nulos de p1, en una 8-vecindad, es decir:

N(p_1) = p2 +p3 + p4 + p5 + p6 +p7 +p8 + p9

y S(p1) es el número de transiciones 0-1 en la secuencia ordenada p2, p3, p4, p5, p6, p7, p8, p9

Una vez evaluados todos los pixeles, se borran aquellos marcados.

Segunda sub-iteración: aquí tomamos como entrada la imagen resultante de la **primera sub-iteración,** evaluamos cada pixel en base a las un píxel del contorno se marca para borrarlo si cumple las siguientes cuatro siguientes condiciones:

e) $2 \leq B(p_1) \leq 6$

f) $A(p_1) = 1$

g) $p_2 * p_4 * p_8 = 0$

h) $p_2 * p_6 * p_8 = 0$

Una vez evaluados todos los pixeles, procedemos a borrar todos aquellos marcados para borrar. Como podemos apreciar, ambas iteraciones difieren únicamente en las condiciones 3 y 4.

Por tanto, cada iteración de este algoritmo queda esquematizado en el siguiente pseudocódigo:

(1) Para todos los píxeles del contorno, marcarlo como candidato a ser borrado si cumple las cuatro condiciones del **primera sub-iteración.**

(2) Una vez recorridos todos los puntos borrar (poner a 0) todos los píxeles marcados en el paso anterior.

(3) Para todos los píxeles del contorno, marcarlo como candidato a ser borrado si cumple las cuatro condiciones del **Segunda sub-iteración**

(4) Una vez recorridos todos los puntos borrar (poner a 0) todos los píxeles marcados en el **Segunda sub-iteración**.

(5) Volver al paso 1 mientras se sigan marcando píxeles.

Mediante este algoritmo *Zhang y Suen*, lograremos que la información de los trazos se mantiene y que la distorsión producida por el ruido se reduce o por lo menos no introduce mucho error en los trazos o no más que las variaciones que pudiera introducir el autor en posteriores firmas.

4.3 Técnicas de etiquetado de trazos

Mediante este procedimiento se detectará los pixeles que forman los trazos, y etiquetarlos según la inclinación del trazo, de esta manera se conseguirá extraer los trazos verticales, horizontales, etc. Existiendo diversas formas de abordar este procedimiento como son por ejemplo a) Redes neuronales, b) Perceptrón multicapa, c) Modular networks y d) Matrices de convolucion, para el presente trabajo de investigación usaremos matrices de convolución.

4.3.1 Matrices de convolución

Las matrices de convolución permiten realizar en el dominio del espacio filtrados en el que el resultado de un píxel después de aplicarle una función que se conoce como impulsional depende únicamente de su valor y del de sus vecinos.

De esta manera, suponiendo la siguiente función impulsional h de tamaño 3x3 igual a la matriz que sigue:

$$h = \begin{bmatrix} h1 & h2 & h3 \\ h4 & h5 & h6 \\ h7 & h8 & h9 \end{bmatrix}$$

la imagen de salida I´ resultante de aplicar el filtrado espacial viene por la expresión:

$$\boldsymbol{I'(x,y) = I(x,y) * h}$$

$$= h_1 I(x-1, y-1) + h_2 I(x, y-1) + h_3 I(x+1, y-1)$$

$$+ h_4 I(x-1, y) + h_5 I(x, y) + h_6 I(x+1, y)$$

$$+ h_7 I(x-1, y+1) + h_8 I(x, y+1) + h_9 I(x+1, y+1)$$

Si no se hubiera esqueletizado como etapa de preproceso en el análisis de la firma manuscrita, se hubiese tenido que ir añadiendo matrices de convolución de por ejemplo 9x9 para ir detectando los trazos de distinto grosor, e inclinación.

Gracias a que ya se ha esqueletizado se puede reducir el problema a pasar por la imagen de origen cuatro matrices de convolución de 2x2 con las cuatro direcciones:

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

Matriz convolución 0°

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

Matriz convolución 45°

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

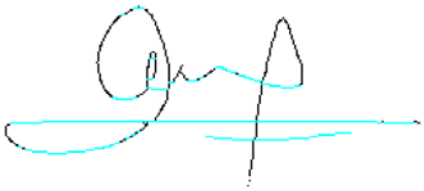
Matriz convolución 90°

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

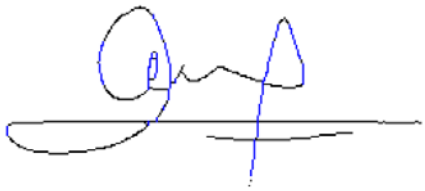
Matriz convolución 135°

Figura 13: Matrices de Convolución 0°, 45°, 90° y 135°

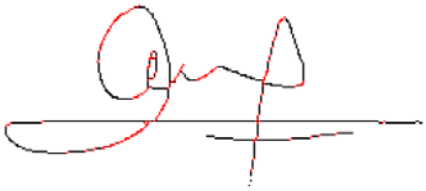
Obteniéndose las siguientes cuatro salidas:



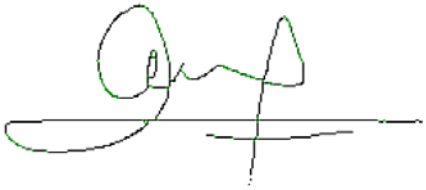
Matriz convolución 0°



Matriz convolución 45°



Matriz convolución 90°



Matriz convolución 135°

Figura 14: Resultados de las matrices de Convolución 0°, 45°, 90° y 135°

Como podemos observar, un mismo pixel puede haber sido etiquetado con cuatro inclinaciones distintas, en otras palabras, un mismo píxel puede formar parte de varios trazos. Esto es interesante cuando en la firma existe cruces de trazos, evitando de esta manera perdida de información.

Otro hecho importante en este proceso es que se produce una simplificación y por tanto una pérdida de información respecto al original, ya que los trazos se han aproximado a rectas de 0º, 45º, 90º y 135º. Esto podría suponer aparentemente un inconveniente, sin embargo, será precisamente este hecho el que nos permita decir que dos trazos de distintas firmas de la misma persona sean iguales a pesar de que sean ligeramente diferentes.

Por ejemplo, en el caso de aplicar la matriz de convolución 0º sobre la firma original, podemos apreciar en la Figura 15 que el subrayado está

etiquetado como una línea horizontal y sin embargo presenta una cierta curvatura.

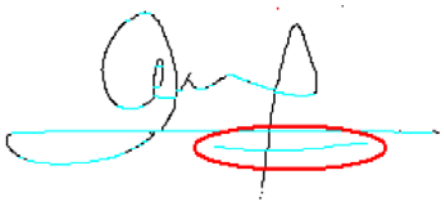
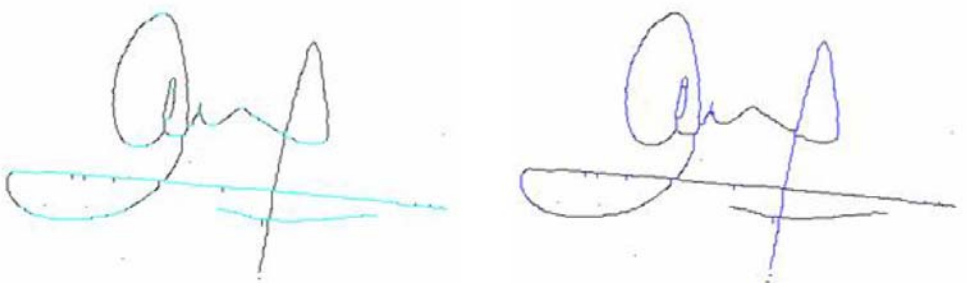


Figura 15. Aproximación realizada

Lo peculiar de la aproximación, es que aunque en otras firmas del mismo autor este subrayado sea más horizontal o ligeramente más curvo (hasta los $\beta°$ siendo $\beta°$ = 22,5º - $\alpha°$ donde α es la inclinación del trazo original) seguirá siendo etiquetado como trazo horizontal, lo que proporciona una gran flexibilidad ante pequeñas variaciones de los trazos. De esta forma se consigue que si se tienen como entradas dos firmas iguales de la misma persona y una de ellas tenga una ligera inclinación respecto a la otra, el sistema seguirá clasificándolas de igual manera, sin necesidad de tener que rotarlas respecto a su eje de inercia. Lo que otorga una cierta robustez frente a pequeñas inclinaciones, típicas del proceso de adquisición de imágenes y respecto a las variaciones de los trazos habituales en la escritura manuscrita.



Resultado matriz 0º Resultado matriz 90º

Figura 16. Robustez frente a la pequeña rotación

4.4 Extracción de características

Las características que se buscan son los trazos que hay en la firma con una orientación de 0º, 45º, 90º y 135º. Para ello y una vez que ya tenemos los píxeles etiquetados con su orientación debemos hacer un proceso de seguimiento y extracción de los segmentos de color de los ejemplos anteriores. A continuación, se introducen los conceptos de conectividad 4 y 8, que son necesarios para este trabajo.

4.4.1 8 vecinos y relación de proximidad

Los vecinos próximos es un camino de secuencia de pixeles que se puede recorrer completamente saltado de un pixel a su vecino, y el vecino de ésta final. Por tanto, existe dos posibles definidos de región conexa, según el tipo de relación de vecindad que se establezca.

Vecinos de un pixel: Un pixel p de coordenadas (x,y) tiene cuatro vecinos horizontales y verticales cuyas coordenadas son: (x+1,y) (x-1,y) (x,y-1) (x,y+1)

- Estos pixels se denominan 4-vecinos de p o N4(p).
- Se encuentran a una distancia unitaria del pixel p.

Los cuatro vecinos diagonales de p,Nd(p) tienen por coordenadas: (x+1,y+1) (x-1,y-1) (x+1,y-1) (x-1,y+1).

Estos Nd(p) junto con N4(p) son los 8-vecinos de p o N8(p).

Algunos de los vecinos de un pixel pueden estar fuera de la imagen si el pixel p está en un borde de la misma.

$$\begin{array}{ccc} & n & \\ n & p & n \\ & n & \end{array} \quad 4-vecinos \qquad \begin{array}{ccc} n & n & n \\ n & p & n \\ n & n & n \end{array} \quad 8-vecinos$$

Figura 17. Relación de vecindad

Conectividad: Una región 4-conexa es un conjunto de píxeles del mismo color, en que cualquier par de píxeles puede ser unido mediante un camino de 4-vecinos pertenecientes a la región. La definición de región 8-conexa es análoga, pero teniendo en cuenta a los 8 vecinos de un píxel dado.

Sea V el conjunto de los valores de intensidad para pixel que se quieren considerar adyacentes. Se pueden considerar tres tipos de conectividad.

- Conectividad-4: 2 pixels p,q con valores en V están 4-conectados si q está en el conjunto N4(p).
- Conectividad-8: 2 pixels p,q con valores en V están 8-conectados si q está en el conjunto N8(p).
- Conectividad mixta: 2 pixels p,q con valores en V están m-conectados si:
    - q está en el conjunto N4(p), ó
    - q está en Nd(p) y $N4(p) \cap N4(q) = \Phi$.
- La conectividad mixta sirve para eliminar conexiones múltiples.

4.5 Algoritmo de seguimiento y extracción de trazos

En el caso del proyecto estos problemas se resuelven usando un doble bucle para

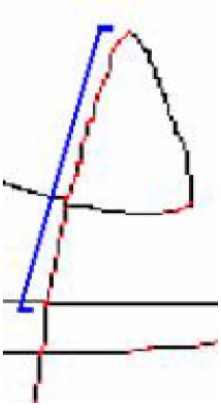
recorrer la imagen de arriba abajo y de izquierda a derecha. Cuando encuentra un píxel etiquetado del color que queremos extraer el trazo mira a sus vecinos en un determinado orden, que veremos a continuación, y los sigue hasta completar ese trazo, marcándolos como visitados, al finalizar

el trazo, continua en el punto en el que se había quedado. De esta manera se soluciona el problema de la recursividad y de visitar varias veces el mismo píxel.

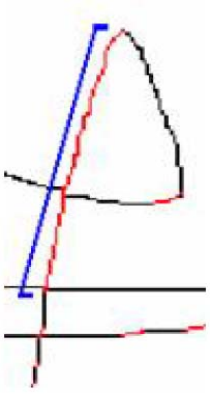
Al haber empleado cuatro matrices de convolución, una por cada inclinación, hay que aplicar también cuatro veces el algoritmo de seguimiento y extracción de los trazos con la ayuda de los colores. Por lo tanto tenemos una imagen con el fondo en blanco, los píxeles de la firma en negro y los píxeles de la firma que forman parte de un trazo con la inclinación adecuada de color.

4.5.1 Seguimiento de trazos sensibles a pixeles sin etiquetar

Al realizar la convolución puede ocurrir que queden píxeles sin etiquetar (en negro) sin embargo, el objetivo es extraer el trazo completo como un solo elemento de longitud igual al segmento de color azul en la Figura 30 y no como una serie de pequeños trazos rojos (45º).



Ejemplo píxeles sin etiquetar



Resultado seguimiento sensible a píxeles sin etiquetar

Figura 18: Proceso de etiquetado

Para solucionarlo, el algoritmo comenzará por un píxel rojo y lo seguirá hasta

encontrarse con uno negro, seguirá el negro hasta una distancia prudencial que llamaremos horizonte (por ejemplo 6 píxeles), en ese caso, si encuentra un píxel a rojo, etiquetará los anteriores píxeles negros a rojos y continuará. En caso de no encontrar ningún píxel rojo, parará y extraerá el trazo hasta el último píxel rojo encontrado, marcando como visitados los anteriores puntos para no volver a visitarlos en posteriores iteraciones, optimizando esta manera el algoritmo recursivo descrito en el punto anterior, consiguiendo de esta forma una complejidad O(n) siendo n el número de píxeles de la firma.

4.5.2 Prioridad en las transiciones

Con referencia a la notación, que codifica la tabla de direcciones, el algoritmo de seguimiento no permite o da prioridad a aquellas transiciones que más interese en cada situación. Por ejemplo, para los trazos de 45º (color rojo) las transiciones prioritarias tanto cuando se esté leyendo píxeles rojos como cuando se esté saltando píxeles negros y teniendo presente que el algoritmo siempre recorre la imagen de arriba a abajo y de izquierda a derecha, el orden a seguir es:

$$\{SW, W, S, SE, NW, E, N, NE\}$$

$$\begin{matrix} NW & N & NE \\ W & V & E \\ SW & S & SE \end{matrix}$$

Tabla 01: Tabla de direcciones

La tabla completa de prioridades en el seguimiento del trazo es:

endiente	**Color**	**Orientación**	**Secuencia de búsqueda**
0º	Agua	W	{W, SW, NW, N, S, NE, SE, E}

0º	Agua	E	{E, NE, SE, N, S, SW, NW, W}
45º	Rojo	SW	{SW, W, S, SE, NW, E, N}
90º	Azul	S	{S, SE, SW, E, W, NE, NW, N}
135º	Verde	SE	{SE, E, S, SW, NE, W, N}

Tabla02: Seguimiento de trazo completo

Para cada inclinación tenemos un caso, salvo para los 0º que puede darse la situación que se recorra el trazo de izquierda a derecha (en dirección E) o de derecha a izquierda (dirección W). Por lo tanto la tabla completa de prioridad de transiciones viene determinada por la inclinación (el color a seguir) y por la dirección en la que se espera encontrar el siguiente píxel del trazo.

Para la pendiente 0º comenzamos por ejemplo con la orientación W, si el siguiente píxel de color que se encuentra está en la posición E, significa que el trazo es de izquierda a derecha, ya que el único píxel de color que es vecino del actual está en la dirección opuesta. En consecuencia, para buscar el siguiente píxel se cambiará a orientación E y viceversa.

Otro detalle a destacar es que tanto la pendiente 45º como la de 135º deja por visitar una orientación, en concreto, la opuesta de la buscada, para 45º la orientación NE y para 135º la orientación NW. Esto es debido a que puede darse el siguiente caso:

Figura de Ejemplo de seguimiento Figura de Posible error

Figura 19: Seguimiento de trazo

Si partimos de la Figura (Ejemplo de seguimiento) y aplicamos la secuencia de búsqueda apropiada para 45º (color rojo) obtenemos como resultado la Figura (Posible error) y nos encontramos en el píxel verde, si la secuencia terminara buscando un píxel rojo en la dirección opuesta a la SW, es decir en la NE, lo encontraría y comenzaría a subir el algoritmo, en lugar de seguir bajando por los negros, encontrándose en un callejón sin salida y por tanto con el final del trazo. Para evitar que el algoritmo se detenga, en estos casos particulares, debido al esqueletizado de los segmentos inclinados, eliminamos de la secuencia de búsqueda la orientación opuesta de la que se busca.

4.5.3 Búsqueda con retroceso

Otro elemento a tener en cuenta, es la necesidad de buscar el camino adecuado en los cruces de los trazos, debido principalmente a la distorsión producida en el proceso de esqueletizado.

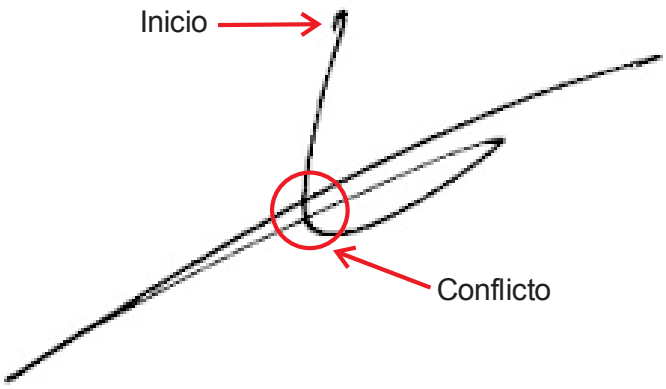


Figura 20: Seguimiento de trazo e identificación de conflictos.

En este caso, a pesar de ser el mismo trazo, el algoritmo del esqueletizado ha producido un escalonamiento en el cruce. Para solucionarlo se puede recurrir a un esquema de búsqueda con retroceso sobre los diferentes caminos posibles, en este caso, hacia arriba y hacia abajo.

En cada iteración sólo se considera un sucesor, e igualmente se eliminan los nodos que sean una “vía muerta”. Cuanto mejor sea el criterio utilizado para limitar el número de estados considerados, más eficiente será el proceso de búsqueda, por eso se ha creado la tabla de transiciones

4.6 Calculo de la caja de feret

A partir de este momento se comienza a trabajar directamente con la lista de trazos y no con la representación gráfica de la firma. De esta manera, ya no es necesario recorrer la imagen píxel a píxel, mejorando el rendimiento y reduciendo la complejidad que ya no será factor de n siendo n el número de píxeles de la firma como hasta ahora, sino que la complejidad de las próximas operaciones será función de n’ siendo n’ el número de trazos localizados, en el caso de este punto, referente al cálculo

de la dimensión total del conjunto de los trazos caja de Feret, la complejidad es O(n').

El siguiente paso en el algoritmo es el cálculo de la caja de Feret (Feret Box), que se define como el rectángulo que circunscribe al conjunto de los trazos alineado respecto a los ejes cartesianos. Para calcularlo, es necesario recorrer los n' trazos de la firma y almacenar las coordenadas de aquellos que tengan como origen o destino la mayor y menor X, y la mayor y menor Y (Figura caja de feret sobre los trazos). Al realizar este proceso sobre los trazos de la firma se consigue que las pequeñas manchas o el ruido no afecten a los siguientes procesos de escalado y normalización.

Una representación gráfica del proceso hasta este momento:

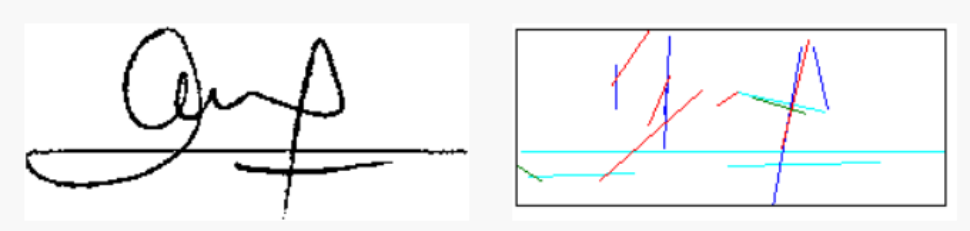


Figura de la Imagen original

Resultado de la caja de Feret sobre los trazos

Figura 21: Resultado de la casa de feret.

4.7 Normalización de los trazos

El rectángulo negro de la Figura caja de Feret sobre trazos, delimita la dimensión útil de la firma, y respecto a estas dimensiones se realiza un proceso de normalización. De esta forma se consigue que los trazos sean independientes a las dimensiones iniciales de la firma, incluso cuando se producen deformaciones de estiramiento.

El proceso de normalización es el siguiente:

Se llamará x' a la base del rectángulo negro de la Figura 39 e y' a la altura del mismo rectángulo, p1 representará al punto superior izquierdo del rectángulo y p2 al inferior derecho.

Entonces:

$$\forall Puntos(x,y) \epsilon TrazosFirma$$

$$X = Round\left(\frac{x - p_{1.}x}{x'} * 100\right)$$

$$Y = Round\left(\frac{y - p_{2.}x}{y'} * 100\right)$$

Por tanto la complejidad de la normalización de los trazos es también O(n').

Gráficamente lo que se ha conseguido hasta el momento es:

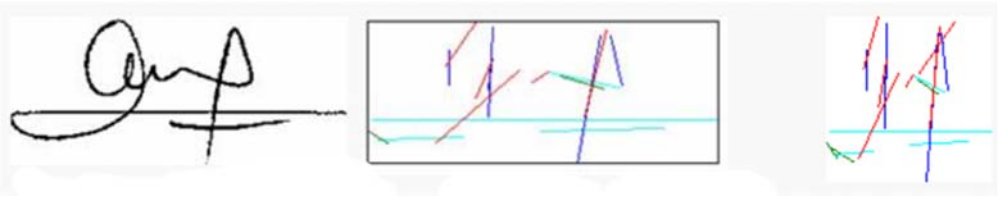


Figura. Imagen original Figura. Extracción de trazos Figura. Normalizado

Figura 22: Proceso de Normalización.

4.8 Reconocimiento basado en trazos

Como resultado de las etapas anteriores, las firmas se describen ahora mediante una lista de sus trazos que son independientes al tamaño original de la firma y al grosor del bolígrafo utilizado.

Un clasificador es un sistema que plantea el problema del reconocimiento como la asignación de los objetos, denominados patrones, a algunos de los grupos representativos, o clases, considerados en el problema. Cada clase representa, por tanto, un tipo diferente de objeto.

En este capítulo se describirá el proceso de comparación entre dos firmas y cómo un sencillo clasificador euclídeo permitirá obtener buenos resultados.

Para resolver el problema de comparar dos conjuntos de trazos y poder extraer algún tipo de medida que permita su posterior interpretación, se pueden utilizar varias técnicas usuales en el reconocimiento de caracteres manuscritos, que se exponen a continuación.

4.8.1 Ajuste de curvas

4.8.1.1 Ajuste de Distancias

Para el presente trabajo de investigación se utilizó el método basado en fronteras de decisión con redes neuronales y ajuste de distancia, cuyo método consiste, en normalizar el tamaño de los trazos de las dos imágenes a comparar, seleccionar ciertos puntos de control a una distancia normalizada para obtener un trazo con un determinado número de puntos significativos y calcular la suma de las distancias entre los puntos correspondientes de las dos imágenes.

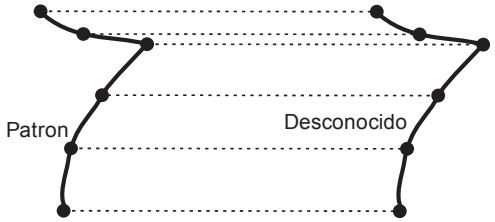


Figura 22: Ajuste de Distancias

El principal inconveniente de este método es que no es muy robusto a las transformaciones del patrón y la solución suele estar fuertemente asociada a las características de escritura de un solo usuario. Aunque precisamente, este inconveniente en el reconocimiento de caracteres

puede ser de gran ayuda en el reconocimiento de firmas, ya que es un problema con una fuerte dependencia con la caligrafía de cada sujeto

4.8.1.2 Ajuste elástico

La expresión ajuste elástico o elastic matching suele usarse habitualmente en el reconocimiento de escritura manuscrita para describir el método anterior, pero en el que la comparación se realiza de una manera más dinámica.

Un método de ajuste de curvas (curve matching), conocido como ajuste elástico (elastic matching), utiliza como medida, la distancia entre puntos del patrón o la plantilla con los puntos de la imagen a comparar. La elección de los puntos que corresponden con la plantilla se realiza de forma dinámica y no estática como en el método anterior, eligiéndose el punto que se encuentra a menor distancia de uno dado de la otra

Figura 23: Ajuste de Elástico

No hay necesidad de tener exactamente el mismo número de puntos en la plantilla que en la imagen a reconocer, los puntos redundantes son eliminados mediante un algoritmo de comparación. Aunque el método no está limitado a un número fijo de puntos en cada trazo, se suele realizar una normalización de la distancia entre puntos para reducir la redundancia y el tiempo de reconocimiento.

4.8.2 Combinación de clasificadores

También existe la posibilidad de combinar diferentes clasificadores para producir un clasificador más estable y robusto. Diferentes clasificadores se complementan y producen uno más completo, algunos son más tolerantes a la traslación, otros a la rotación y otros a distintos estilos de escritura.

Para combinar varios clasificadores hay que utilizar algún tipo de ponderación, darle diferentes pesos a los resultados obtenidos por cada uno de ellos.

Otra aproximación sería enviar la salida de los diferentes clasificadores a una red de neuronas para que tomara la decisión final. El principal inconveniente de la combinación de clasificadores es su coste computacional.

4.8.3 Algoritmo elegido

El algoritmo elegido es una modificación de los sistemas de ajuste de distancias (distance matching) aprovechando las simplificaciones realizadas hasta el momento. En especial, cada trazo curvo se ha aproximado a una recta, es decir, se puede representar por su punto de inicio y de fin y por la pendiente del trazo original (no de la recta que lo representa), por eso mantenemos la información del color que hace referencia a la inclinación de los trazos antes de la normalización.

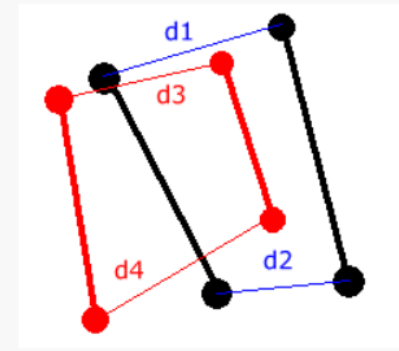


Figura 24: Ajuste basado en las distancias

De esta manera, tenemos que hacer un ajuste o matching únicamente entre los dos puntos de referencia de la plantilla o patrón con los dos puntos del elemento a identificar.

Por tanto, puede decidirse que un trazo está en la plantilla (firma con la que se compara) cuando existe un trazo de ese mismo color cuya distancia d3 y d4 es menor a una distancia D que se determinará de forma empírica.

El algoritmo obedece al siguiente pseudocódigo:

(a) Si el número de trazos de la imagen a reconocer es igual al número de trazos de la plantilla ± 10 pasar al paso 2, en otro caso el resultado (% similitud) = 0.0%

(b) Para todo trazo de la plantilla, buscamos en la imagen a reconocer un trazo de ese mismo color (misma pendiente).

(c) Si la distancia de sus extremos d3 y d4 (Figura. Ajuste basado en distancias) es menor a la distancia D determinada a-priori, eliminamos ese trazo de la imagen a reconocer para evitar que de un segundo positivo con otro trazo del patrón e incrementamos el valor de una variable llamada aciertos.

De esta manera, se puede calcular con muy poco esfuerzo de cómputo, el porcentaje de los trazos de la plantilla que tienen su pareja correspondiente en la imagen a reconocer. Cuanto mayor sea ese porcentaje de trazos encontrados significará que más parecidas son esas dos imágenes.

$$Resultado\ (\%\ similitud) = \frac{aciertos}{n°\ trazos\_planilla} * 100$$

4.8.4 Determinación de los parámetros: longitud de trazo (LT), distancia de ajuste (D) y umbral de verificación (UV).

Hasta el momento nos hemos encontrado con tres variables importantes cuyo valor habrá que determinar empíricamente en un proceso de aprendizaje.

Una de ellas, es la longitud mínima de los trazos (LT) que se considerarán significativos, necesario para evitar que el ruido o pequeñas modificaciones de la firma afecten en el resultado de la caracterización.

La segunda variable a determinar es la distancia (D) en el proceso de ajuste o matching y el tercer parámetro (UV) es necesario para el proceso de verificación de firmas, que consiste en comparar dos muestras y decidir con un cierto grado de confianza si pertenecen a la misma persona o no, de esta manera si se comparan dos firmas y se obtiene un índice de similitud superior a un cierto umbral (UV) se puede decir que son del mismo autor.

4.8.5 Proceso de aprendizaje

Para encontrar la pareja de valores que hacen óptimo el reconocimiento y la verificación, se ha utilizado una base de datos de 40 firmas

pertenecientes a 10 personas, agrupadas de cuatro en cuatro, es decir 4 firmas del sujeto 1, 4 firmas del sujeto 2, etc.

CAPITULO V

5 RESULTADOS

5.1 Esqueletizacion

Imagen de Entrada, archivo: firma7.jpg

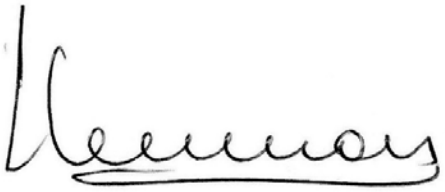


Imagen de salida, archivo: temp.png

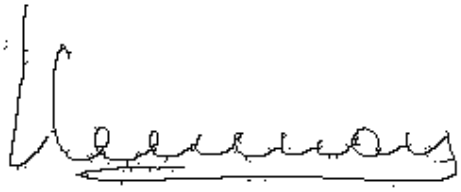


Figura 25: Resultado del proceso de esqueletizado.

5.1.1 Código:

Función: esqueletizar

Propósito: Reduce el grosor de la firma a solo 1 pìxel, manteniendo su forma

Argumentos de entrada: img: Imagen de la firma a esqueletizar que después se guarda como un temporal.

```
def esqueletizar(img):
    w, h = img.size
    pixels = list(img.getdata())

        # Create black and white pixel bitmap image
    nimg = Image.new('1',img.size,-1)

        # Convert source image to black and white pixels
    bwdata = _getbwdata( img )

        # Run the algorithm until no further modifications are required
    is_modified = True
    while is_modified:

            bwdata, modified1 = do_step(bwdata,img.size,step1_func,w,h)
            bwdata, modified2 = do_step(bwdata,img.size,step2_func,w,h)

            is_modified = modified1 | modified2

            print (is_modified, modified1, modified2)

        # Push the data to image
```

```
    nimg.putdata( bwdata )
    nimg.save("temp.png")
            #fp.close()
def _isbw(col):
    c = 240

    if col[0] < c and col[1] < c and col[2] < c:
        col = C_BLACK
    else:
        col = C_WHITE
    return col

def _getcoord( size, pos ):
    x,y = pos
    w,h = size
    i = (y * w) + x
    return i

def _getbw( imgdata, size, pos ):
    return imgdata[ _getcoord(size,pos) ]

def _setbw( imgdata, size, pos, col ):
    imgdata[ _getcoord(size,pos) ] = col
def _getbwdata( img ):
    d = list(img.getdata())
    print(d)
    for i, c in enumerate(d):
        d[ i ] = _isbw( c )
    return d


#############################################################################
#   Algorithm implementation
#############################################################################

# step1_func = lambda parr: p2 + p4 + p6 > 0 and p4 + p6 + p8 > 0
# step2_func = lambda parr: p2 + p4 + p8 > 0 and p2 + p6 + p8 > 0
step1_func = lambda parr: parr[0] + parr[2] + parr[4] > 0 and parr[2] + parr[4] + parr[6] > 0
step2_func = lambda parr: parr[0] + parr[2] + parr[6] > 0 and parr[0] + parr[4] + parr[6] > 0

def do_step(imgdata, size, func,w,h):
    was_modified = False
    for j in range(1,h-1):
        for i in range(1,w-1):
            p1 = _getbw( imgdata, size, ( i,  j   ) )
            p2 = _getbw( imgdata, size, ( i,  j-1 ) )
            p3 = _getbw( imgdata, size, ( i+1,j-1 ) )
            p4 = _getbw( imgdata, size, ( i+1,j   ) )
            p5 = _getbw( imgdata, size, ( i+1,j+1 ) )
            p6 = _getbw( imgdata, size, ( i,  j+1 ) )
            p7 = _getbw( imgdata, size, ( i,  j+1 ) )
            p8 = _getbw( imgdata, size, ( i-1,j   ) )
            p9 = _getbw( imgdata, size, ( i-1,j-1 ) )

            # nimg.putpixel( (i,j),  )
            # nimg.putpixel( (i,j), p1 )
            A_Val  = (p2 == 0 and p3 == 1) + (p3 == 0 and p4 == 1)
            A_Val += (p4 == 0 and p5 == 1) + (p5 == 0 and p6 == 1)
```

```
        A_Val += (p6 == 0 and p7 == 1) + (p7 == 0 and p8 == 1)
        A_Val += (p8 == 0 and p9 == 1) + (p9 == 0 and p2 == 1)

        B_Val = sum([p2,p3,p4,p5,p6,p7,p8,p9])
        parr = [p2,p3,p4,p5,p6,p7,p8,p9,p2]

        if p1 == C_BLACK:
            if 2 <= B_Val <= 6:
                if A_Val == 1:
                    if func(parr):
                        _setbw( imgdata, size, (i,j), C_WHITE )
                        was_modified = True
                        # imgdata.putpixel( (i,j), C_WHITE )
    return (imgdata, was_modified)
```

5.2 Binarizado

Imagen de Entrada, archivo: temp.png

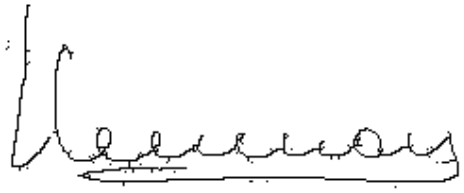


Imagen de salida, archivo: temp2.png

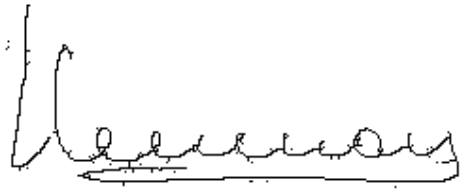


Figura 26: Resultado del proceso de Binarizado.

5.2.1 Código:

Función: binarizar

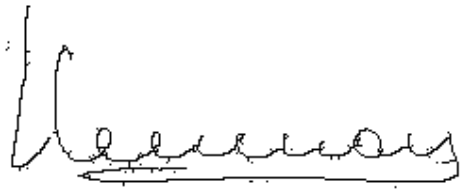
Propósito: Separar el fondo de la firma para un mejor tratamiento

Variables clave: gray: imagen leída desde la ruta de la aplicación que después de la binarizacion se guarda como un temporal.

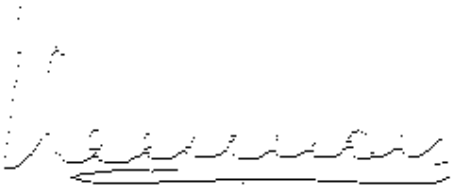
```
def binarizar():
    gray = cv2.imread('temp.png', cv2.IMREAD_GRAYSCALE)
    t, dst = cv2.threshold(gray, 170, 255, cv2.THRESH_BINARY)
    cv2.imwrite("temp.png",dst)
```

5.3 Matriz de Convolución en ángulos de 0º, 45º, 90º, 135º

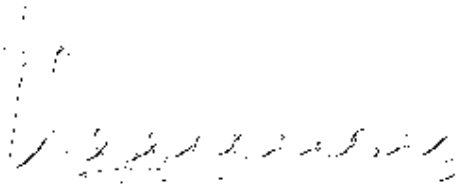
Imagen de Entrada, temp2.png :



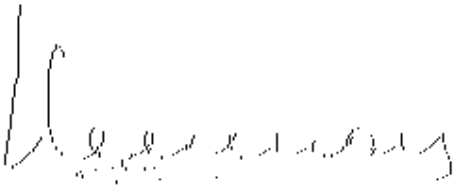
Imágenes en cada Angulo:
Angulo 0º, archivo: temp0.png



Angulo 45º, archivo: temp45.png



Angulo 90º, archivo: temp90.png



Angulo 135º, archivo: temp135.png

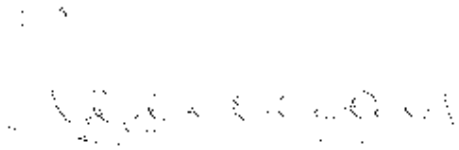


Figura 27: Resultados de las matrices de convolución en ángulos de 0º, 45º, 90º, 135º

5.3.1 Código:

Función: angulos

Propósito: Obtiene los ángulos de trazo de la firma en 0º, 45º, 90º y 135º

Variables clave: im: lee la Imagen de la firma a tratar

ángulos: lista que guarda las imágenes de los ángulos de la firma

```
def angulos():
   ruta = "temp.png"
```

```
im = cv2.imread(ruta)
#Generamos la matriz para 0 grados
matriz0 = [[1.0, 1.0],
      [0.0, 0.0]]
#Generamos la matriz para 90 grados
matriz90 = [[1.0, 0.0],
      [1.0, 0.0]]
#Generamos la matriz para 135 grados
matriz135 = [[1.0, 0.0],
         [0.0, 1.0]]
#Generamos la matriz para 45 grados
matriz45 = [[0.0, 1.0],
         [1.0, 0.0]]
#Asignamos los kernel a trabajar
kernel_0 = np.asarray(matriz0)
kernel_1 = np.asarray(matriz90)
kernel_2 = np.asarray(matriz135)
kernel_3 = np.asarray(matriz45)
#Creamos los filtros respectivos
img0 = cv2.filter2D(im, -1, kernel_0)
img90 = cv2.filter2D(im, -1, kernel_1)
img135 = cv2.filter2D(im, -1, kernel_2)
img45 = cv2.filter2D(im, -1, kernel_3)

im1 = cv2.copyMakeBorder(im,0,0,0,0,cv2.BORDER_REPLICATE)
im2 = cv2.copyMakeBorder(im,0,0,0,0,cv2.BORDER_REPLICATE)
im3 = cv2.copyMakeBorder(im,0,0,0,0,cv2.BORDER_REPLICATE)
im4 = cv2.copyMakeBorder(im,0,0,0,0,cv2.BORDER_REPLICATE)

for x in range(np.size(img0, 0)):
   for y in range(np.size(img0, 1)):
      if (img0[x,y][0]==0):
         im1[x,y]=[0,255,0]

for x in range(np.size(img90, 0)):
   for y in range(np.size(img90, 1)):
      if (img90[x,y][0]==0):
         im2[x,y]=[255,0,0]

for x in range(np.size(img135, 0)):
   for y in range(np.size(img135, 1)):
      if (img135[x,y][0]==0):
         im3[x,y]=[0,100,255]

for x in range(np.size(img45, 0)):
   for y in range(np.size(img45, 1)):
      if (img45[x,y][0]==0):
         im4[x,y]=[120,255,0]



#cv2.imshow("salidaoriginal.png",img135)
angulos=[]
angulos.append(img0)
angulos.append(img45)
angulos.append(img90)
angulos.append(img135)
```

```
    cv2.imwrite("temp/temp135.png", img135)
    cv2.imwrite("temp/temp90.png", img90)
    cv2.imwrite("temp/temp45.png", img45)
    cv2.imwrite("temp/temp0.png", img0)
    return angulos
```

5.4 Comparación de Firmas

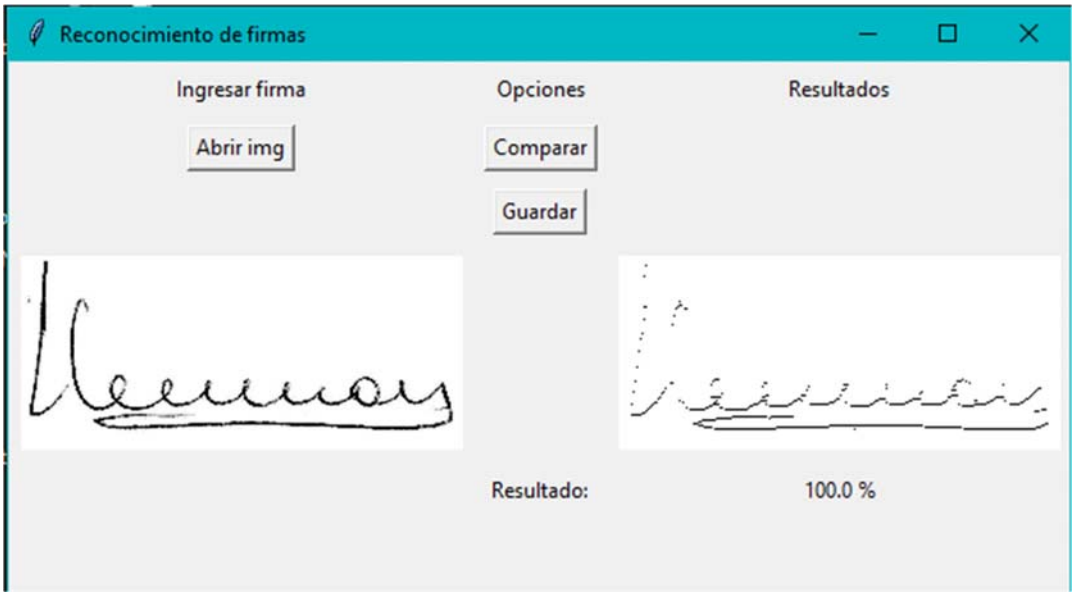


Figura 28: Caja de comparación de firmas.

5.4.1 Código:

Función: compararDist

Propósito: Compara las similitudes de dos imágenes a través del algoritmo ORB, obteniendo las distancias de similitud de las imágenes, cuanto mas tiende a 0, mas similares son.

Argumentos de entrada: img1, img2: Imágenes a comparar

Argumentos de salida: prom: promedio de similitud de las imágenes

```
def compararDist(img1,img2):
    #Iniciar detector SIFT
    img1 = np.asarray(img1)
    img2 = np.asarray(img2)
    img1=cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
    img2=cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
    orb = cv2.ORB_create()
    # Encuentra los puntos clave y descriptores con SIFT
    kp1, des1 = orb.detectAndCompute(img1,None)
    kp2, des2 = orb.detectAndCompute(img2,None)
    # crea BFMatcher object
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    # Match descriptors.
    matches = bf.match(des1,des2)
```

```
    #print(matches)
    # Clasifícalos en el orden de su distancia.
    matches = sorted(matches, key = lambda x:x.distance)
    prom=0
    for match in matches:
        prom+=match.distance

    if(len(matches)!=0):
        prom=prom/len(matches)
    else:
        prom=100
    # Dibuja las primeras 30 coincidencias.
    return prom
```

Función: comparar

Propósito: Funcion que compara la firma de entrada con las firmas en la basen de datos, usando la función compararDist, e imprime el resultado en la interfaz.

Variables clave: imgEntrada: Firma de entrada.

```
def comparar():

    global imgEntrada
    ang=procesarImg(imgEntrada)
    imagesBD=load_images_from_folder('bd')
    angImg=load_images_from_folder('temp')
    mayor=100
    imgnum=0
    for imgAng in angImg:
        cont=0
        for img in imagesBD:
            value = compararDist(imgAng, img)
            if(value<10):
                mayor=value
                imgnum=cont
                imgR = ImageTk.PhotoImage(img)
                imgResult['image']=imgR

            cont+=1
    if(mayor<10):
        print(mayor)
        mayor=100-mayor
        lblPorcentaje['text']=str(mayor)+" %"
    else:
        lblPorcentaje['text']=str(0)+" %"
    ventana.mainloop()
    #print(imgnum)
```

5.5 Funciones Anexas

Función: numero_imagenes

Propósito: Cuenta el número de imágenes existentes en un directorio.

Argumentos de entrada: folder: directorio en el cual se contará.

```
def numero_imagenes(folder):
   global contadorImg
   for filename in os.listdir(folder):

      contadorImg+=1
```

Función: load_images_from_folder

Propósito: carga las imágenes existentes en un directorio en una lista de imágenes.

Argumentos de entrada: folder: directorio en el cual se trabajará.

Argumentos de salida: imagesBD: Lista de imágenes del directorio.

```
def load_images_from_folder(folder):
   imagesBD=[]

   for filename in os.listdir(folder):
      img = Image.open(os.path.join(folder,filename))
      imagesBD.append(img)

   return imagesBD

numero_imagenes('bd')
```

Función: procesarImg
Propósito: Procesa la imagen por las 3 etapas de tratamiento propuestas: esqueletizado, binarizado y extraccion de angulos.
Argumentos de entrada: imgEn: imagen a ser procesada
Variables clave: angulos: lista que guarda las imagenes de los angulos de la firma.

```
def procesarImg(imgEn):
   esqueletizar(imgEn)
   binarizar()
   return angulos()
#Funcion:   comparar
#Proposito:  Compara la iamgen guardada en la variable global imgEntrada con las imagenes de la base de datos.
#Argumentos de entrada:    imgEn: imagen a ser procesada
#Argumentos de salida:      porcentaje de similitud con las imagenes de la base de datos
```

Función: redim

Propósito: redimensiona una imagen a un tamaño de 250x110 pixeles para su tratamiento.

Argumentos de entrada: img: imagen a ser redimensionada

```
def redim(img):
   img = img.resize((250,110))
imgEntrada=NONE
```

Función: abrirImg

Propósito: Permitir seleccionar una imagen para su analisis gráficamente y la guarda en la

variable global imgEntrada

Variables clave: imgentrada: variable global donde se guarda la imagen ingresada.

```
def abrirImg():
    Tk().withdraw() # we don't want a full GUI, so keep the root window from appearing
    filename = askopenfilename() # show an "Open" dialog box and return the path to the
selected file
    #print(imgEntry)
    imgEnt = Image.open(filename)
    new_img = imgEnt.resize((250,110))

    filename=filename[0:len(filename)-4]
    print(filename)
    new_img.save(filename+'.png','png')
    imgEnt.close()
    img = PhotoImage(file=filename+'.png')
    global imgEntrada
    print(img)
    imgEntrada = Image.open(filename+'.png')

    #imgEntry2 = Label(frm, width=250, height=250,image=img).grid(row=3,column=0)

    imgEntry['image']=img
    ventana.mainloop()
```

Función: guardarImg
Propósito: Permitir añadir una firma a la base de datos
Variables clave: imgEntrada: Variable global donde se guardo la imagen ingresada, y que en esta función es guardada a la base de datos.

```
def guardarImg():
    global imgEntrada
    print(imgEntrada)
    ang=procesarImg(imgEntrada)
    global contadorImg
    cv2.imwrite("bd/"+str(contadorImg+1)+"_135.png", ang[0])
    cv2.imwrite("bd/"+str(contadorImg+1)+"_90.png", ang[1])
    cv2.imwrite("bd/"+str(contadorImg+1)+"_45.png", ang[2])
    cv2.imwrite("bd/"+str(contadorImg+1)+"_0.png", ang[3])
    contadorImg+=1
```

CAPITULO VI

6 DISCUSIÓN

Con los resultados anteriores, la funcionalidad de reconocimiento, es decir, buscar en una base de datos de firmas aquella que más se parezca a una dada, el sistema ha obtenido unos resultados muy buenos diferenciando siempre entre las firmas del mismo autor y el resto.

Sin embargo, para el problema de la verificación, en la que se debe de dar un criterio de confianza al decir que dos firmas son iguales, hay que determinar un umbral de similitud (UV) a partir del cual el sistema dirá que son iguales. En este caso, es fácil intuir que a pesar de los buenos resultados obtenidos en la validación.

Actualmente la base de datos de firmas sobre la que se realiza el reconocimiento consta simplemente de un directorio (llamado BD) donde se almacena una imagen de cada patrón de cada clase en formato .bmp, este modelo es apropiado para los casos en los que el número de firmas sea pequeño, ya que se compara la firma que se quiere reconocer, con todas las firmas que forman parte de la base de datos. Sin embargo, el sistema podría ser escalable si organizamos las firmas por el número total de trazos, o jerárquicamente por la cantidad de trazos horizontales, verticales, etc. De esta manera, cada firma se compararía con las firmas similares, mejorando los tiempos de respuesta del sistema, si ha esto se le añade que en lugar de almacenar directamente los bitmaps o mapas de bits se almacenaran en la base de datos las listas de los trazos que la componen, además de reducir significativamente las necesidades de

espacio en disco, supondría una importante mejora en tiempo pues no sería necesario extraer los trazos de la firma patrón, que es el proceso más costoso (esqueletizado, cuatro filtros de matrices de convolución y cuatro seguimiento y extracción de los trazos, cálculo de comparacion y normalización).

CONCLUSIONES

Cada firma digitalizada a 250x110 pixeles, en blanco y negro ocupa en disco menos de 3Kb. Permitiendo crear bases de datos de miles de firmas de forma sencilla, incluso si se quisiera mejorar este parámetro se podría almacenar únicamente la información de los trazos reduciéndose el tamaño necesitado por firma a apenas 1Kb. mejorándose también y de forma significativa, el rendimiento en tiempo, al ser innecesario la ejecución de los algoritmos necesarios para etiquetar y extraer los trazos de la firma patrón, que son los procesos más costosos, como se ha visto anteriormente.

Durante cada fase del proceso y en el peor de los casos se mantienen en archivo temporal la firma, la original y la copia sobre la que se realizan las modificaciones, como todas las operaciones se realizan con una profundidad de color de 24bits, cada firma ocupa en memoria menos de 1 Kb.

RECOMENDACIONES

Existen numerosas líneas de investigación para trabajos futuros en el mundo del reconocimiento biométrico mediante firma manuscrita, puesto que es un campo muy amplio y como se ha mencionado varias veces a lo largo del documento, está en auge en la sociedad de la información.

Se recomienda efectuar investigaciones referidas a la comparación de eficiencia en la identificación de firmas manuscritos, entre los algoritmos se encuentran (a) ajuste de distancias, (2) Alineamiento temporal dinámico DTW y (3) Algoritmos ocultos de Márkov HMM

REFERENCIAS BIBLIOGRAFIA

Heaton, J. (2008). Introduction to Neural Networks for Java. St. Louis: Heaton Research, Inc.

Heaton, J. (2008). Introduction to Neural Networks with C#. U.S.: Heaton Research, Inc.

Heaton, J. (2010). Programming Neural Networks with Encog 2 in Java. St. Louis: Heaton Research, Inc.

Joseph Howse (2013). OpenCV Computer Vision with Python, PACKT PUBLISHING, open source comunity experience distilled.

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P. (2010). Metodología de la Investigación. México: McGRAW - HILL.

Hu, Y. H., & Hwang, J.-N. (2002). Handbook of Neural Network Signal Processing. U.S.: CRC Press LLC.

Isasi Viñuela, P., & Galván León, I. M. (2004). Redes de neuronas artificiales: Un enfoque práctico. España: Pearson.

ANEXO

Código main.py

```
from tkinter import *
import os
from PIL import Image
import numpy as np
import cv2
from matplotlib import pyplot as plt
import sys
from PIL import Image,ImageTk
from pprint import pprint as pp
from PIL import ImageOps
from tkinter.filedialog import askopenfilename
##from SSIM_PIL import compare_ssim

from numpy import array

ventana=Tk()
ventana.title("Reconocimiento de firmas")
#ventana.iconbitmap("img/food.ico")
frm=Frame(ventana)
frm.pack(fill="both")
frm.config(width="600",height="300")
frm.grid_propagate(False)

#variables

folder="img"
contadorImg=0
C_BLACK = 0
C_WHITE = 1

imgEntradaDir="img/entry.png"
```

```
#funciones

def _isbw(col):
   c = 240

   if col[0] < c and col[1] < c and col[2] < c:
      col = C_BLACK
   else:
      col = C_WHITE
   return col

def _getcoord( size, pos ):
         x,y = pos
         w,h = size
         i = (y * w) + x
         return i

def _getbw( imgdata, size, pos ):
         return imgdata[ _getcoord(size,pos) ]

def _setbw( imgdata, size, pos, col ):
         imgdata[ _getcoord(size,pos) ] = col
def _getbwdata( img ):
   d = list(img.getdata())
   print(d)
   for i, c in enumerate(d):
      d[ i ] = _isbw( c )
   return d


#########################################################################
#        Algorithm implementation
```

```
#####################################################################

# step1_func = lambda parr: p2 + p4 + p6 > 0 and p4 + p6 + p8 > 0
# step2_func = lambda parr: p2 + p4 + p8 > 0 and p2 + p6 + p8 > 0
step1_func = lambda parr: parr[0] + parr[2] + parr[4] > 0 and parr[2] + parr[4] + parr[6]
> 0
step2_func = lambda parr: parr[0] + parr[2] + parr[6] > 0 and parr[0] + parr[4] + parr[6]
> 0

def do_step(imgdata, size, func,w,h):
        was_modified = False
        for j in range(1,h-1):
                for i in range(1,w-1):
                        p1 = _getbw( imgdata, size, ( i,  j   ) )
                        p2 = _getbw( imgdata, size, ( i,  j-1 ) )
                        p3 = _getbw( imgdata, size, ( i+1,j-1 ) )
                        p4 = _getbw( imgdata, size, ( i+1,j   ) )
                        p5 = _getbw( imgdata, size, ( i+1,j+1 ) )
                        p6 = _getbw( imgdata, size, ( i,  j+1 ) )
                        p7 = _getbw( imgdata, size, ( i,  j+1 ) )
                        p8 = _getbw( imgdata, size, ( i-1,j   ) )
                        p9 = _getbw( imgdata, size, ( i-1,j-1 ) )

                        # nimg.putpixel( (i,j),  )
                        # nimg.putpixel( (i,j), p1 )
                        A_Val  = (p2 == 0 and p3 == 1) + (p3 == 0 and p4 == 1)
                        A_Val += (p4 == 0 and p5 == 1) + (p5 == 0 and p6 == 1)
                        A_Val += (p6 == 0 and p7 == 1) + (p7 == 0 and p8 == 1)
                        A_Val += (p8 == 0 and p9 == 1) + (p9 == 0 and p2 == 1)

                        B_Val = sum([p2,p3,p4,p5,p6,p7,p8,p9])
                        parr = [p2,p3,p4,p5,p6,p7,p8,p9,p2]
```

```
                        if p1 == C_BLACK:
                                if 2 <= B_Val <= 6:
                                        if A_Val == 1:
                                                if func(parr):
                                                        _setbw( imgdata, size, (i,j),
C_WHITE )
                                                        was_modified = True
                                                        # imgdata.putpixel( (i,j), C_WHITE
)
        return (imgdata, was_modified)
def numero_imagenes(folder):
    global contadorImg
    for filename in os.listdir(folder):

        contadorImg+=1

def load_images_from_folder(folder):
    imagesBD=[]

    for filename in os.listdir(folder):
        img = Image.open(os.path.join(folder,filename))
        imagesBD.append(img)

    return imagesBD

numero_imagenes('bd')
def esqueletizar(img):
    w, h = img.size
    pixels = list(img.getdata())

        # Create black and white pixel bitmap image
```

```
    nimg = Image.new('1',img.size,-1)

        # Convert source image to black and white pixels
    bwdata = _getbwdata( img )

        # Run the algorithm until no further modifications are required
    is_modified = True
    while is_modified:

            bwdata, modified1 = do_step(bwdata,img.size,step1_func,w,h)
            bwdata, modified2 = do_step(bwdata,img.size,step2_func,w,h)

            is_modified = modified1 | modified2

            print (is_modified, modified1, modified2)

        # Push the data to image
    nimg.putdata( bwdata )
    nimg.save("temp.png")
    nimg.save("temp2.png")
        #fp.close()
def binarizar():
    gray = cv2.imread('temp.png', cv2.IMREAD_GRAYSCALE)
    t, dst = cv2.threshold(gray, 170, 255, cv2.THRESH_BINARY)
    cv2.imwrite("temp.png",dst)
    cv2.imwrite("temp3.png",dst)
def angulos():
    ruta = "temp.png"

    im = cv2.imread(ruta)
    #Generamos la matriz para 0 grados
    matriz0 = [[1.0, 1.0],
```

```
        [0.0, 0.0]]
#Generamos la matriz para 90 grados
matriz90 = [[1.0, 0.0],
        [1.0, 0.0]]
#Generamos la matriz para 135 grados
matriz135 = [[1.0, 0.0],
            [0.0, 1.0]]
#Generamos la matriz para 45 grados
matriz45 = [[0.0, 1.0],
            [1.0, 0.0]]
#Asignamos los kernel a trabajar
kernel_0 = np.asarray(matriz0)
kernel_1 = np.asarray(matriz90)
kernel_2 = np.asarray(matriz135)
kernel_3 = np.asarray(matriz45)
#Creamos los filtros respectivos
img0 = cv2.filter2D(im, -1, kernel_0)
img90 = cv2.filter2D(im, -1, kernel_1)
img135 = cv2.filter2D(im, -1, kernel_2)
img45 = cv2.filter2D(im, -1, kernel_3)

im1 = cv2.copyMakeBorder(im,0,0,0,0,cv2.BORDER_REPLICATE)
im2 = cv2.copyMakeBorder(im,0,0,0,0,cv2.BORDER_REPLICATE)
im3 = cv2.copyMakeBorder(im,0,0,0,0,cv2.BORDER_REPLICATE)
im4 = cv2.copyMakeBorder(im,0,0,0,0,cv2.BORDER_REPLICATE)

for x in range(np.size(img0, 0)):
    for y in range(np.size(img0, 1)):
        if (img0[x,y][0]==0):
            im1[x,y]=[0,255,0]

for x in range(np.size(img90, 0)):
```

```
        for y in range(np.size(img90, 1)):
            if (img90[x,y][0]==0):
                im2[x,y]=[255,0,0]

    for x in range(np.size(img135, 0)):
        for y in range(np.size(img135, 1)):
            if (img135[x,y][0]==0):
                im3[x,y]=[0,100,255]

    for x in range(np.size(img45, 0)):
        for y in range(np.size(img45, 1)):
            if (img45[x,y][0]==0):
                im4[x,y]=[120,255,0]




    #cv2.imshow("salidaoriginal.png",img135)
    angulos=[]
    angulos.append(img0)
    angulos.append(img45)
    angulos.append(img90)
    angulos.append(img135)
    cv2.imwrite("temp/temp135.png", img135)
    cv2.imwrite("temp/temp90.png", img90)
    cv2.imwrite("temp/temp45.png", img45)
    cv2.imwrite("temp/temp0.png", img0)
    return angulos
def procesarImg(imgEn):
    esqueletizar(imgEn)
    binarizar()
    return angulos()
```

```
def comparar():

    global imgEntrada
    ang=procesarImg(imgEntrada)
    imagesBD=load_images_from_folder('bd')
    angImg=load_images_from_folder('temp')
    mayor=100
    imgnum=0
    for imgAng in angImg:
        cont=0
        for img in imagesBD:
            value = compararDist(imgAng, img)
            if(value<10):
                mayor=value
                imgnum=cont
                imgR = ImageTk.PhotoImage(img)
                imgResult['image']=imgR

            cont+=1
    if(mayor<10):
        print(mayor)
        mayor=100-mayor
        lblPorcentaje['text']=str(mayor)+" %"
    else:
        lblPorcentaje['text']=str(0)+" %"
    ventana.mainloop()
    #print(imgnum)
def redim(img):
    img = img.resize((250,110))
imgEntrada=NONE
def compararDist(img1,img2):
```

```
    #Iniciar detector SIFT
    img1 = np.asarray(img1)
    img2 = np.asarray(img2)
    img1=cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
    img2=cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
    orb = cv2.ORB_create()
    # Encuentra los puntos clave y descriptores con SIFT
    kp1, des1 = orb.detectAndCompute(img1,None)
    kp2, des2 = orb.detectAndCompute(img2,None)
    # crea BFMatcher object
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    # Match descriptors.
    matches = bf.match(des1,des2)
    #print(matches)
    # Clasifícalos en el orden de su distancia.
    matches = sorted(matches, key = lambda x:x.distance)
    prom=0
    for match in matches:
        prom+=match.distance

    if(len(matches)!=0):
        prom=prom/len(matches)
    else:
        prom=100
    # Dibuja las primeras 30 coincidencias.
    return prom
def abrirImg():
    Tk().withdraw() # we don't want a full GUI, so keep the root window from appearing
    filename = askopenfilename() # show an "Open" dialog box and return the path to the
selected file
    #print(imgEntry)
    imgEnt = Image.open(filename)
```

```
    new_img = imgEnt.resize((250,110))

    filename=filename[0:len(filename)-4]
    print(filename)
    new_img.save(filename+'.png','png')
    imgEnt.close()
    img = PhotoImage(file=filename+'.png')
    global imgEntrada
    print(img)
    imgEntrada = Image.open(filename+'.png')

    #imgEntry2 = Label(frm, width=250, height=250,image=img).grid(row=3,column=0)

    imgEntry['image']=img
    ventana.mainloop()
def guardarImg():
    global imgEntrada
    print(imgEntrada)
    ang=procesarImg(imgEntrada)
    global contadorImg
    cv2.imwrite("bd/"+str(contadorImg+1)+"_135.png", ang[0])
    cv2.imwrite("bd/"+str(contadorImg+1)+"_90.png", ang[1])
    cv2.imwrite("bd/"+str(contadorImg+1)+"_45.png", ang[2])
    cv2.imwrite("bd/"+str(contadorImg+1)+"_0.png", ang[3])
    contadorImg+=1


lblIzquierda=Label(frm, text="Ingresar firma").grid(row=0,column=0,padx=5,pady=5)
lblMedio=Label(frm, text="Opciones").grid(row=0,column=1,padx=5,pady=5)
lblDerecha=Label(frm, text="Resultados").grid(row=0,column=2,padx=5,pady=5)

cargarImgButton=PhotoImage(file=imgEntradaDir)
```

```
imgEntry = Label(frm, width=250, height=110)
imgEntry.config(image=cargarImgButton)
imgEntry.grid(row=3,column=0,padx=5,pady=5)

cargarButton=Button(frm,text="Abrir
img",command=abrirImg).grid(row=1,column=0,padx=5,pady=5)
compButton=Button(frm,text="Comparar",command=comparar).grid(row=1,column=1,
padx=5,pady=5)
guardarButton=Button(frm,text="Guardar",command=guardarImg).grid(row=2,column
=1,padx=5,pady=5)
imgDefault=PhotoImage(file="img/result.png")
imgResult = Label(frm, width=250, height=110)
imgResult.config(image=imgDefault)
imgResult.grid(row=3,column=2,padx=5,pady=5)
lblResultado=Label(frm, text="Resultado:").grid(row=4,column=1,padx=5,pady=5)
lblPorcentaje=Label(frm, text="")
lblPorcentaje.grid(row=4,column=2,padx=5,pady=5)
ventana.mainloop()
```

Código Esqueletizado

```
import sys
from PIL import Image
from pprint import pprint as pp
from PIL import Image, ImageOps

C_BLACK = 0
C_WHITE = 1

def _isbw(col):
        c = 240
        if col[0] < c and col[1] < c and col[2] < c:
                col = C_BLACK
```

```
        else:
                col = C_WHITE

        return col

def _getcoord( size, pos ):
        x,y = pos
        w,h = size
        i = (y * w) + x
        return i

def _getbw( imgdata, size, pos ):
        return imgdata[ _getcoord(size,pos) ]

def _setbw( imgdata, size, pos, col ):
        imgdata[ _getcoord(size,pos) ] = col

def _getbwdata( img ):
        d = list(img.getdata())
        for i, c in enumerate(d):
                d[ i ] = _isbw( c )
                # print i, c, d[ i ]
        return d


#######################################################################
#####
#       Algorithm implementation
#######################################################################
#####

# step1_func = lambda parr: p2 + p4 + p6 > 0 and p4 + p6 + p8 > 0
```

```
# step2_func = lambda parr: p2 + p4 + p8 > 0 and p2 + p6 + p8 > 0
step1_func = lambda parr: parr[0] + parr[2] + parr[4] > 0 and parr[2] + parr[4] +
parr[6] > 0
step2_func = lambda parr: parr[0] + parr[2] + parr[6] > 0 and parr[0] + parr[4] +
parr[6] > 0

def do_step(imgdata, size, func):
        was_modified = False
        for j in range(1,h-1):
                for i in range(1,w-1):
                        p1 = _getbw( imgdata, size, ( i,  j   ) )
                        p2 = _getbw( imgdata, size, ( i,  j-1 ) )
                        p3 = _getbw( imgdata, size, ( i+1,j-1 ) )
                        p4 = _getbw( imgdata, size, ( i+1,j   ) )
                        p5 = _getbw( imgdata, size, ( i+1,j+1 ) )
                        p6 = _getbw( imgdata, size, ( i,  j+1 ) )
                        p7 = _getbw( imgdata, size, ( i,  j+1 ) )
                        p8 = _getbw( imgdata, size, ( i-1,j   ) )
                        p9 = _getbw( imgdata, size, ( i-1,j-1 ) )

                        # nimg.putpixel( (i,j),  )
                        # nimg.putpixel( (i,j), p1 )
                        A_Val  = (p2 == 0 and p3 == 1) + (p3 == 0 and p4 == 1)
                        A_Val += (p4 == 0 and p5 == 1) + (p5 == 0 and p6 == 1)
                        A_Val += (p6 == 0 and p7 == 1) + (p7 == 0 and p8 == 1)
                        A_Val += (p8 == 0 and p9 == 1) + (p9 == 0 and p2 == 1)

                        B_Val = sum([p2,p3,p4,p5,p6,p7,p8,p9])
                        parr = [p2,p3,p4,p5,p6,p7,p8,p9,p2]

                        if p1 == C_BLACK:
                                if 2 <= B_Val <= 6:
```

```
                                    if A_Val == 1:
                                        if func(parr):
                                            _setbw( imgdata, size, (i,j),
C_WHITE )
                                            was_modified = True
                                            # imgdata.putpixel( (i,j),
C_WHITE )
        return (imgdata, was_modified)


#####################################################################
#####
#       Work on image / main
#####################################################################
#####

if __name__ == '__main__':

        imgname = "entrada2.png"
        img = Image.open(imgname)
        w, h = img.size

        pixels = list(img.getdata())

        # Create black and white pixel bitmap image
        nimg = Image.new('1', img.size, -1 )

        # Convert source image to black and white pixels
        bwdata = _getbwdata( img )

        # Run the algorithm until no further modifications are required
        is_modified = True
```

```
        while is_modified:

                bwdata, modified1 = do_step(bwdata,img.size,step1_func)
                bwdata, modified2 = do_step(bwdata,img.size,step2_func)

                is_modified = modified1 | modified2

                print (is_modified, modified1, modified2)

        # Push the data to image
        nimg.putdata( bwdata )
        nimg.show()



        # And save
        #imagen = Image.open(nimg)
        #imagen.close()
       #fp = open('salida.jpg','w')
        nimg.save("salidaEn.png")
        nimg.close()
        #fp.close()
```

Codigo Binarizado

```
import numpy as np
import cv2

gray = cv2.imread('salidaEn.png', cv2.IMREAD_GRAYSCALE)

t, dst = cv2.threshold(gray, 170, 255, cv2.THRESH_BINARY)

#cv2.imshow('umbral', gray)
```

```
cv2.imwrite("binarizadaEn.png",dst)
cv2.imshow('result', dst)

cv2.waitKey(0)
```

Extraer caracteres

```
#####################################################################
##########
#  PROYECTO DE RECONOCIMIENTO DE FIRMAS A TRAVES DE VISION
ARTIFICIAL   #
#                                                   #
# VERSION       :1.0                                    #
# FECHA         :31-03-2019     HORA: 23:15:00                  #
# DEPENDENCIAS  :NUMPY, OPENCV                              #
#                                                   #
#####################################################################
##########
#importamos las librerias
import cv2
import numpy as np
from matplotlib import pyplot as plt
#cargamos la imgagen a analizar
ruta = "binarizadaEn.png"

im = cv2.imread(ruta)
#Generamos la matriz para 0 grados
matriz0 = [[1.0, 1.0],
          [0.0, 0.0]]
#Generamos la matriz para 90 grados
```

```
matriz90 = [[1.0, 0.0],
          [1.0, 0.0]]
#Generamos la matriz para 135 grados
matriz135 = [[1.0, 0.0],
           [0.0, 1.0]]
#Generamos la matriz para 45 grados
matriz45 = [[0.0, 1.0],
           [1.0, 0.0]]
#Asignamos los kernel a trabajar
kernel_0 = np.asarray(matriz0)
kernel_1 = np.asarray(matriz90)
kernel_2 = np.asarray(matriz135)
kernel_3 = np.asarray(matriz45)
#Creamos los filtros respectivos
img0 = cv2.filter2D(im, -1, kernel_0)
img90 = cv2.filter2D(im, -1, kernel_1)
img135 = cv2.filter2D(im, -1, kernel_2)
img45 = cv2.filter2D(im, -1, kernel_3)

im1 = cv2.copyMakeBorder(im,0,0,0,0,cv2.BORDER_REPLICATE)
im2 = cv2.copyMakeBorder(im,0,0,0,0,cv2.BORDER_REPLICATE)
im3 = cv2.copyMakeBorder(im,0,0,0,0,cv2.BORDER_REPLICATE)
im4 = cv2.copyMakeBorder(im,0,0,0,0,cv2.BORDER_REPLICATE)

for x in range(np.size(img0, 0)):
  for y in range(np.size(img0, 1)):
    if (img0[x,y][0]==0):
      im1[x,y]=[0,255,0]

for x in range(np.size(img90, 0)):
  for y in range(np.size(img90, 1)):
    if (img90[x,y][0]==0):
```

```
        im2[x,y]=[255,0,0]

for x in range(np.size(img135, 0)):
  for y in range(np.size(img135, 1)):
    if (img135[x,y][0]==0):
      im3[x,y]=[0,100,255]

for x in range(np.size(img45, 0)):
  for y in range(np.size(img45, 1)):
    if (img45[x,y][0]==0):
      im4[x,y]=[120,255,0]



#plt.subplot(231),plt.imshow(im,'gray'),plt.title('ORIGINAL')
#plt.subplot(232),plt.imshow(im1,'gray'),plt.title('0 GRADOS')
#plt.subplot(233),plt.imshow(im2,'gray'),plt.title('90 GRADOS')
#plt.subplot(234),plt.imshow(im3,'gray'),plt.title('135 GRADOS')
#plt.subplot(235),plt.imshow(im4,'gray'),plt.title('45 GRADOS')
#plt.show( )

#cv2.imshow("salidaoriginal.png",img135)
cv2.imwrite("salidaEn135.png", img135)
cv2.imwrite("salidaEn90.png", img90)
cv2.imwrite("salidaEn45.png", img45)
cv2.imwrite("salidaEn0.png", img0)
cv2.waitKey()
```