Reconocedor estadístico de dígitos escritos a mano con el uso de histogramas

Joshua Gamboa Calvo, *Estudiante, ITCR* Andrés Montero Gamboa, *Estudiante, ITCR,* Álvaro Moraría Villalobos, *Estudiante, ITCR*

***Resumen*—Este informe académico presenta la implementación de un modelo estadístico de reconocimiento de dígitos escritos a mano, seguido de le presentación de resultados y análisis detallado de los mismos; por último la investigación sobre dos sistemas de reconocimiento de caracteres escrito en la comunidad de *software* libre. Este análisis examina aspectos tales como la precisión en la categorización de dígitos y tiempo de ejecución del modelo.**

***Índice de Términos*—reconocimiento de dígitos, reconocimiento de patrones, visión por computadora, histogramas**

I. Introducción

Los algoritmos de detección de bordes son herramientas fundamentales en el mundo actual, facilitando una amplia gama de aplicaciones en diversas industrias y contribuyendo al avance tecnológico y la mejora de la eficiencia en numerosos campos. Su importancia radica en su capacidad para procesar y analizar información visual de manera rápida, precisa y automatizada, lo que impulsa la innovación y el desarrollo en la era de la tecnología digital.

Este reporte tiene como objetivo documentar el proceso de diseño, implementación y ejecución de pruebas de un modelo estadístico de reconocimiento de dígitos escritos a mano, el cual por medio del cálculo de histogramas y los momentos estadísticos de promedio y desviación estándar categorice las imágenes entrantes de acuerdo al modelo entrenado con base a un conjunto específico; el desarrollo de esta solución es posible gracias a la utilización del lenguaje de programación *Python* con apoyo con las bibliotecas *OpenCV* y *Numpy*.

Adicionalmente, el presente reporte abarca la revisión bibliográfica de dos modelos de reconocimiento de caracteres escritos a mano ya existentes con el fin de conocer el estado del arte en el área de investigación del presente reporte y extraer resultados comparativos respecto a la eficiencia y efectividad del modelo propietario.

Dada la naturaleza experimental este informe se obviará la definición teórica de conceptos relacionados y se asume que el lector cuenta con la base de conocimiento necesaria para poder seguir el hilo conductor planteado.

Como resultado se espera obtener un sistema básico de categorización de dígitos escritos a mano el cual sea comparable con modelos ya existentes en la misma área para evaluar su eficiencia y efectividad con respecto a las soluciones pertenecientes al estado del arte.

A. Tecnología empleada

Para implementar el categorizador de dígitos se hará uso de *Python*, el cual es un lenguaje de programación multiparadigma interpretado de alto nivel [1] La escogencia de este lenguaje se justifica en factores como: sintaxis sencilla; amplia comunidad orientada al reconocimiento de patrones, procesamiento de imágenes y visión por computadora; así como por la amplia variedad de bibliotecas formales disponibles.

En conjunto con el lenguaje de programación *Python* se hará uso de la biblioteca de software libre *OpenCV (Open Source Computer Vision Library)* la cual ofrece al rededor de 2500 funcionalidades relacionadas con la visión por computadora y el procesamiento digital de imágenes [2]. De esta herramienta se utilizará la interfaz especializada disponible para ejecutarse sobre el interprete de *Python*.

También se utilizará la biblioteca propia de *Python*: *NumPy*, la cual es un paquete matemático para computación científica. Esta resulta indispensable gracias a sus capacidades para operacionar matrices de manera sencilla y eficiente [3].

B. Descripción del documento

El presente reporte se encuentra organizado tal que: la primera y presente sección *I. Introducción* le brinda al lector la bienvenida y le ofrece una noción general de los contenidos sobre los cuales se profundiza más adelante; *II. Modelo de categorización de dígitos escritos a mano* expone el modelo propuesto por el equipo para la categorización de dígitos; la tercera sección *III. Conjunto de datos a utilizar* describe el conjunto de datos a utilizar en el entrenamiento y las pruebas del modelo caracterizado en la sección anterior; la cuarta sección *IV. Estado del arte en el reconocimiento de caracteres escritos a mano* describe los modelos de reconocimiento óptico de caracteres: *Tesseract* y *Keras*, ambos pertenecientes al el estado del arte; en la quinta sección *V. Resultados* expone los resultados obtenidos del proceso de validación de modelo entrenado en forma tabular; la sexta sección *VI. Análisis de resultados* extrae información de relevante producto de los resultados expuestos en la sección anterior mediante representación visual e inferencia; por último, la séptima sección *VII. Conclusión* ofrece un cierre general y reflexiona respecto a los hallazgos relevantes. Al final del documento se pueden encontrar las referencias a las fuentes consultadas.

II. Modelo de categorización de dígitos escritos a mano

En la presente sección se detalla el funcionamiento de cada módulo del **Reconocedor de dígitos escritos a mano**, en la cual se implementó una convolución y un clasificador propietario. Este se puede observar en la figura 1.

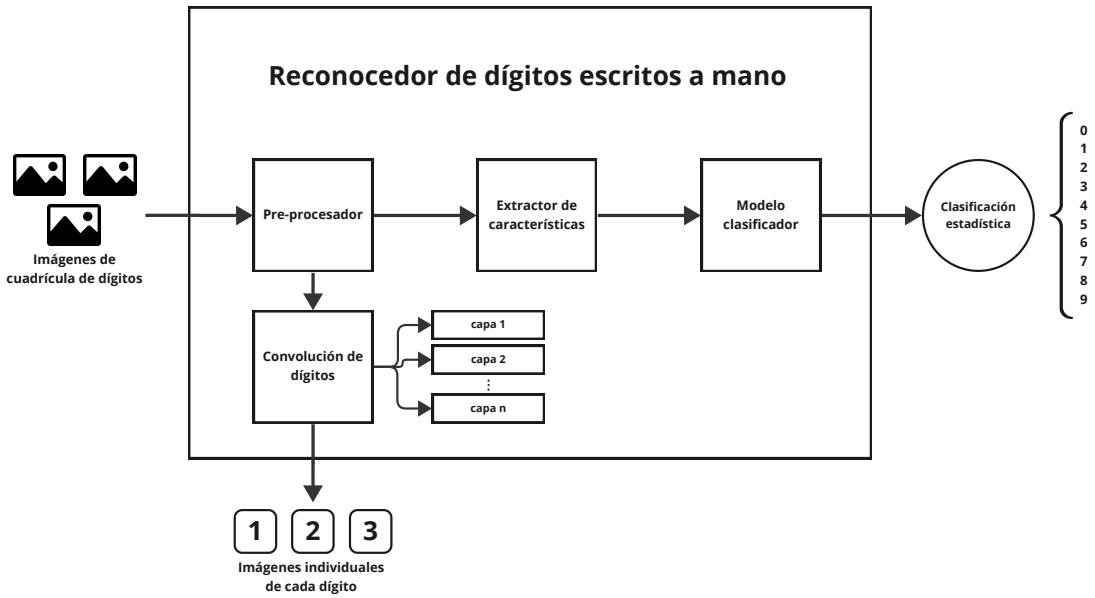
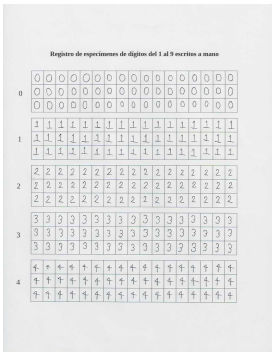


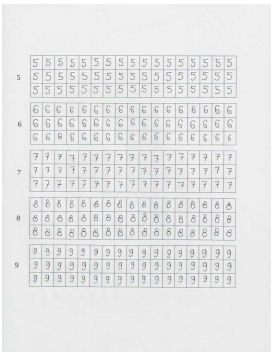
Fig. 1: Diagrama de clasificación de dígitos. Figura de elaboración propia.

1) **Pre-procesador**: En el primer módulo del reconocedor se busca recuperar las imágenes de los especímenes "crudos" para aplicarles transformaciones y seccionarlas para obtener como resultado un dígito por imagen.
   A la hora de cargar las imágenes se siguieron los tres siguientes pasos para procurar que los conjuntos de datos "crudos" puedan ser tratados de forma general: Primero se cargó la imagen en escala de grises para ignorar los colores de la imagen, luego se pasó la imagen a negativo con el objetivo de que la información de valor, en este caso los trazos que conforman los dígitos, fueran representados por valores diferentes a 0; Y por último se aplicó un umbral y binarización para trabajar únicamente con valores de 0 y 255.
   Luego se utilizó la clase propietaria de **Convolución de Dígitos** junto con capas personalizadas para recuperar y procesar todos los dígitos presentes en el conjunto de datos. Dentro de las capas creadas se tienen las siguientes: *capa de erosión y dilatación* para eliminar información no deseada, *capa de división uniforme* para subdividir una imagen en una cantidad de imágenes específica, *capa de recorte inteligente* para recortar los bordes de una imagen los cuales no contengan información, *capa de cambio de tamaño* para lograr uniformidad en el tamaño de cada dígito.
   Por último, se aplanó la lista de imágenes y se guardó en memoria colocando el nombre de la etiqueta, que en este caso corresponde a la clase a la que pertenece el dígito, en el nombre del archivo para poder identificarlo en el extractor de características.
2) **Extractor de características**: Para el extractor de características se utilizarán las imágenes generadas por el pre-procesador, para efectos de este proyecto las características a identificar en cada imagen van a ser los valores del histograma tanto de forma horizontal como vertical con un ancho de **4 píxeles**.
   Posteriormente de generar el vector de características de cada imagen del conjunto de datos pre-procesados, estos van a ser almacenados de forma binaria con la ayuda de la biblioteca *pickle* en memoria para su uso en el modelo clasificador.
3) **Modelo clasificador**: El último módulo que forma parte del clasificador de dígitos es el modelo en sí. Este fue implementado desde cero para cumplir con las especificaciones de la asignación.
   Este modelo contiene los siguientes métodos que permiten la clasificación de dígitos:

   - ***split***: Este método toma una lista de vectores de características junto a su etiqueta correspondiente y los ordena dentro de un diccionario de categorías, donde cada llave es una categoría y el valor es una lista de vectores de características.
     Luego hace un *shuffle* aleatorio y divide las observaciones de cada clase con la relación 75%/25% para entrenamiento y validación respectivamente.
   - ***fit***: Este método toma el conjunto de datos de entrenamiento y calcula el vector promedio y de varianza para cada clase y lo guarda en sus atributos para luego ser utilizado en el calculo de predicciones.
   - ***score***: Este método es el cual debe ser utilizado para calificar la precisión del modelo con respecto a un conjunto de datos de validación proporcionado. Este genera el porcentaje de precisión del modelo y la matriz de confusión generada por la validación.



(a) Hoja de ejemplo con cuadricula de números escritos a mano del 0 al 4



(b) Hoja de ejemplo con cuadricula de números escritos a mano del 5 al 9

Fig. 2: Hojas de ejemplo de conjuntos de datos a utilizar

III. Conjunto de datos a utilizar

Los datos a utilizar son números del 0 al 9 escritos a mano, cada dígito especifico se escribe a mano en una cuadricula de 3x17 especímenes, así generando 51 individuos del mismo número, esto en dos hojas de papel, la primera hoja abarca los dígitos del 0 al 4 2a y la segunda hoja abarca del numero 5 al 9 2b como se muestra en la en la figura 2. A su ves se utilizan por lo menos 15 ejemplares de esas hojas, con muestras de escritura a mano de diferentes personas.

IV. Estado del arte en el reconocimiento de caracteres escritos a mano

A. *Tesseract OCR*

Tesseract es un sistema de reconocimiento óptico de caracteres originalmente escrito en el lenguaje de programación C y desarrollado como un producto propietario de *Hewlett-Packard* Laboratories en Bristol, Reino Unido en colaboración con *Hewlett-Packard Co.*, Greeley Colorado USA entre 1985 y 1994. En el año 2005 *HP* liberó la propiedad intelectual de la pieza de *software* bajo un licenciamiento Apache 2.0 y a partir del 2006, hasta la actualidad *Google* ha tomado la dirección del proyecto para su mantenimiento y mejora [4].

De acuerdo con [4], *Tesseract* sigue un proceso de reconocimiento paso a paso tradicional, destacando la etapa de análisis de componentes conectados, que permite detectar fácilmente texto inverso. *Tesseract* fue pionero en reconocer texto blanco sobre negro de manera trivial. Luego, se organizan los componentes en "Blobs" y se analizan las líneas de texto para determinar si son de espaciado fijo o proporcional. La reconocimiento se realiza en dos pasos: primero, se intenta reconocer cada palabra, pasando las satisfactorias a un clasificador adaptativo como datos de entrenamiento. En el segundo paso, se vuelve a reconocer las palabras que no fueron identificadas adecuadamente. Finalmente, se resuelven los espacios difusos y se verifican hipótesis alternativas para localizar texto en mayúsculas pequeñas. El proceso anteriormente descrito se puede ver representado en el diagrama expuesto en la figura 3.

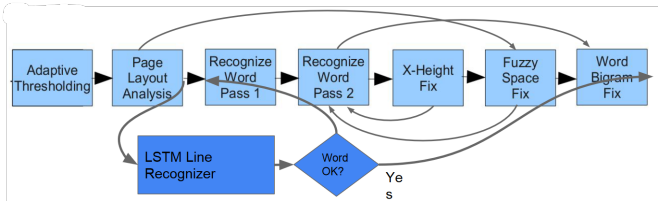


Fig. 3: Proceso de reconocimiento óptico de caracteres por *Tesseract OCR*. Figura tomada de [5]

En el año 1991 *Tesseract* fue sometido a una traducción masiva de su código fuente desde C hacia C++, esto con el objetivo de simplificar la representación de estructuras a través del paradigma orientado a objetos. Esta traducción dio como resultado el esquema de clases que se puede observar en la figura 4 utilizadas a lo largo del proceso de reconocimiento óptico de caracteres.

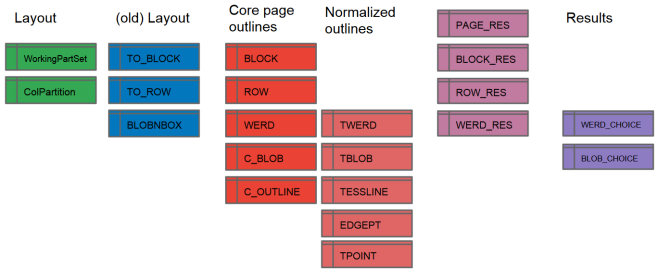


Fig. 4: Clases utilizadas para representar datos por *Tesseract OCR*. Figura tomada de [5]

A día de hoy *Tesseract* es una fina pieza de *software* con un elevando nivel de madurez y capacidad de identificar texto en más de 100 idiomas diferentes con un alto grado de efectividad. Actualmente el reconocedor óptico de caracteres se encuentra en su versión 5. Entre las mejoras significativas recientemente integradas destaca la implementación de redes neuronales *LSTM*, las cuales fueron introducidas en las versión 4.

B. *Keras*

Keras es una biblioteca de Redes Neuronales de Código abierto escrita en Python. Fue desarrollada por François Chollet en 2015 con el objetivo de simplificar la programación de algoritmos basados en aprendizaje profundo ofreciendo un conjunto de abstracciones más intuitivas y de alto nivel. *Keras* puede ejecutarse sobre TensorFlow, Microsoft Cognitive Toolkit o Theano. En 2017, el equipo de TensorFlow de Google decidió ofrecer soporte a Keras en la biblioteca de core de TensorFlow. [6]

Keras utiliza una variedad de estructuras de datos para representar y manipular modelos de aprendizaje profundo. De acuerdo con [7] algunas de las estructuras de datos más importantes incluyen:

1) **Modelos:** Los modelos en *Keras* se fabrican conectando bloques de construcción configurables entre sí.
2) **Capas:** Las capas son los bloques de construcción fundamentales de los modelos en *Keras*. Cada capa

recibe información de la capa anterior y pasa su salida a la siguiente capa.

3) **Funciones de pérdida y optimizadores:** Estas son funciones que definen cómo se actualizan los pesos del modelo durante el entrenamiento.
4) **Métricas:**Las métricas se utilizan para evaluar el rendimiento del modelo durante el entrenamiento y la prueba.

En la figura 5 se puede observar una representación general del flujo de la información sobre la red neuronal utilizada en su interior.

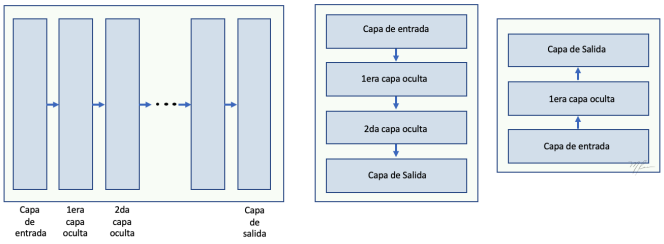


Fig. 5: Representación general de red neuronal utilziada por *Keras*. Figura tomada de [8]

Utilizando el conjunto de datos *MNIST* (Modified National Institute of Standards and Technology) [9] es un conjunto de datos de dígitos escritos a mano que se utiliza comúnmente para entrenar sistemas de procesamiento de imágenes. Cada imagen en MNIST es una imagen en escala de grises de 28x28 píxeles como se muestra en la figura 6.

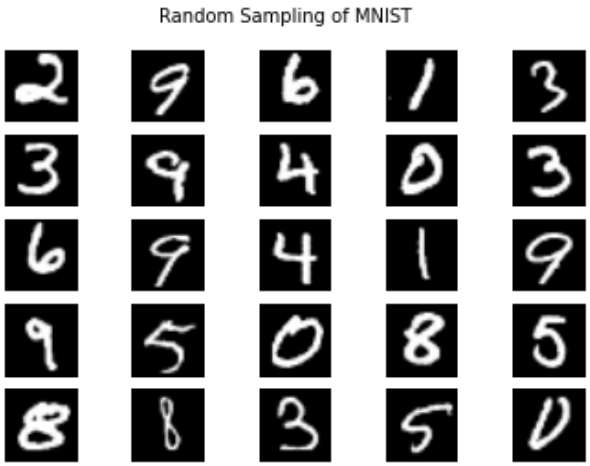


Fig. 6: Ejemplos de imágenes en MNIST. Figura tomada de [10]

De acuerdo con [11] los pasos clave en el uso de Keras para crear un modelo de aprendizaje profundo para reconocimiento dígitos escritos a mano son:

1) **Carga de datos**: Carga el conjunto de datos MNIST.
2) **Preprocesamiento de datos**: Escala las imágenes al rango [0, 1] y convierte las etiquetas en vectores de una sola codificación.
3) **Compilación del modelo**: Compila el modelo especificando la función de pérdida, el optimizador y las métricas.
4) **Entrenamiento del modelo**: Entrena el modelo con las imágenes y las etiquetas, especificando el número de épocas y el tamaño del lote.
5) **Evaluación del modelo**: Evalúa el rendimiento del modelo en un conjunto de datos de prueba.
6) **Predicción**: Usa el modelo entrenado para hacer predicciones en nuevas imágenes.

C. Hipótesis

1) El reconocedor óptico de caracteres desarrollado como parte de esta investigación será por lo menos un 5% más efectivo en el reconocimiento de escritura a mano con respecto a *Tesseract*. Esto debido a que si bien *Tesseract* es una herramienta de altísima fiabilidad esta no se encuentra optimizada para reconocimiento de trazos a mano alzada.
2) El reconocedor óptico de caracteres desarrollado como parte de esta investigación será por lo menos un 10% más rápido en el reconocimiento de escritura a mano con respecto a *Tesseract*. Esto debido a que el nivel de complejidad de sistema propietario es mucho inferior que al de la contraparte perteneciente al estado del arte.
3) El reconocedor óptico de caracteres desarrollado como parte de esta investigación será por lo menos un 10% menos efectivo en el reconocimiento de escritura a mano con respecto a *Keras*. Esto debido a que el enfoque estadístico implementado en la presente investigación no es comparable al alto rendimiento que entrega un modelo de aprendizaje profundo entrenado específicamente para reconocer dígitos escritos a mano, tal como el modelo de *Keras* con el conjunto de entrenamiento *MNIST*.
4) El reconocedor óptico de caracteres desarrollado como parte de esta investigación será por lo menos un 5% más rápido en el reconocimiento de escritura a mano con respecto a *Keras*. Esto debido a que el nivel de complejidad de sistema propietario es mucho inferior que al de la contraparte perteneciente al estado del arte.

V. Resultados

Como resultado del proceso de entrenamiento del modelo con el 70% de los especímenes descritos en el apartado *III. Conjunto de datos a utilizar*, se obtuvieron los vectores de características correspondientes a los dígitos del 0 al 9. Dichos valores se encuentran representados en forma de histogramas en la figura 10.

Producto del proceso de validación del modelo con el 30% restante de los especímenes descritos en el apartado anteriormente citado, se obtuvo que el reconocedor identifica correctamente dígitos un 62% de las ocasiones. En la tabla I se puede observar el nivel de exactitud del modelo identificando para cada una de las clases, dichas clases se encuentran ordenadas de manera ascendente según el % de aproximado de aciertos asociado.

Dígito	% aproximado de aciertos
3	39%
5	55%
0	59%
6	59%
7	59%
9	59%
8	66%
2	68%
4	77%
1	93%

CUADRO I: Cuantificación relativa a la cantidad aproximada de aciertos al identificar cada uno de los dígitos del 0 al 9. Tabla de elaboración propia.

En la figura 7 se puede apreciar una representación visual mediante un gráfico de barras de la información expuesta de manera tabular en I.

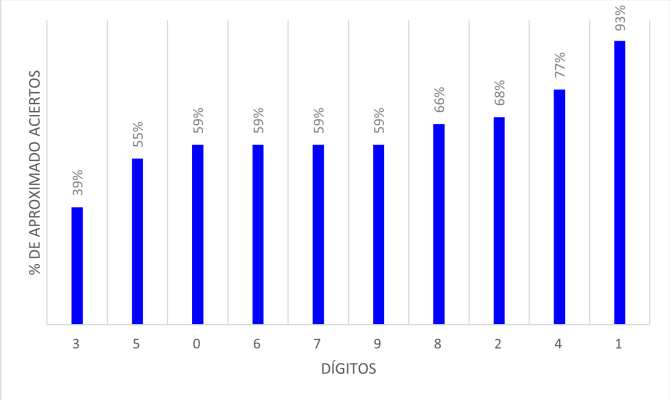


Fig. 7: Cuantificación relativa a la cantidad aproximada de aciertos al identificar cada uno de los dígitos del 0 al 9. Figura de elaboración propia.

VI. Análisis de resultados

En términos generales caben destacar las siguientes observaciones: el dígito 1 fue el que modelo logró reconocer con un mayor grado de efectividad (aproximadamente un 93% de aciertos); el dígito 3 fue el que logró reconocer con menor grado de efectividad (aproximadamente un 39% de aciertos); los dígitos 0, 6, 7 y 9 presentan un valle en cuanto a la capacidad de reconocerles correctamente (aproximadamente un 59% de aciertos); entre el dígito con peor grado de efectividad y el segundo peor grado de efectividad (dígitos 3 y 5 correspondientemente) existe una mejora en la capacidad de reconocimiento acertado de un 16%, mientras que entre el dígito con mejor grado de efectividad y el segundo mejor grado de efectividad (dígitos 1 y 4 correspondientemente) existe una mejora en la capacidad de reconocimiento acertado de la misma magnitud; la diferencia en el grado de efectividad de reconocimiento entre el dígito con peor grado de efectividad y el dígito con el mejor es un 54%.

Tomando en consideración la siguiente clasificación: se considera como dígitos con ángulos rectos: 1, 2, 4, 5 y 7; y se considera como dígitos sin ángulos rectos: 0, 3, 6, 8 y 9. Dicho lo anterior se puede asegurar que el modelo clasifica con mayor facilidad aquellos dígitos que presentan ángulos rectos, esto ya que el 30% de las clases con mayor cantidad de predicciones correctas pertenecen a este subconjunto. Ahora, respecto al 30% de las clases con menor cantidad de predicciones correctas el 66% de las clases están conformados por dígitos sin ángulos rectos, mientras que el el 33% restante lo conforman dígitos de ángulos rectos.

Profundizando en cuanto al 30% de las clases con menor cantidad de predicciones correctas anteriormente mencionado (conformado por los dígitos 3, 5 y 0). En la figura 8 se puede observar una comparativa referencial de la estructura de los glifos contenidos en esta subclase, donde se evidencia una alta similitud en su construcción. Si bien dicha figura referencia estructura de dígitos tipográficos estos guardan una relación cercana con sus potenciales versiones manuscritas. Dicho análisis evidencia que el enfoque estadístico en el cual se basa el modelo de predicción cuenta con fuertes limitaciones al discriminar dígitos de construcción similar.

(a) Dígitos pertenecientes al 30% de las clases con menor cantidad de predicciones correctas.

(b) Superposición de dígitos pertenecientes al 30% de las clases con menor cantidad de predicciones correctas.

Fig. 8: Dígitos pertenecientes al 30% de las clases con menor cantidad de predicciones correctas y referencia de superposición. Ambas figuras de elaboración propia.

Con base en los resultados obtenidos se construye la matriz de confusión que se puede observar en la figura 9. Dicha herramienta coteja los valores objetivo supervisados con respecto a las predicciones efectuadas por el modelo estadístico. Con ayuda de la codificación de color como mapa de calor, esta representación permite correlacionar visualmente posibles puntos de mejora donde el modelo potencialmente “confunde” dígitos.

De la representación matricial de los resultados expuesta en la figura 9 se logra extraer las 3 concentraciones de principales de clasificación errónea por parte del modelo, siendo estas: 12 ocasiones donde el modelo identificó un dígito 3 como un dígito 5, 11 ocasiones donde el modelo identificó un dígito 6 como un dígito 2 y 10 ocasiones donde el modelo identificó

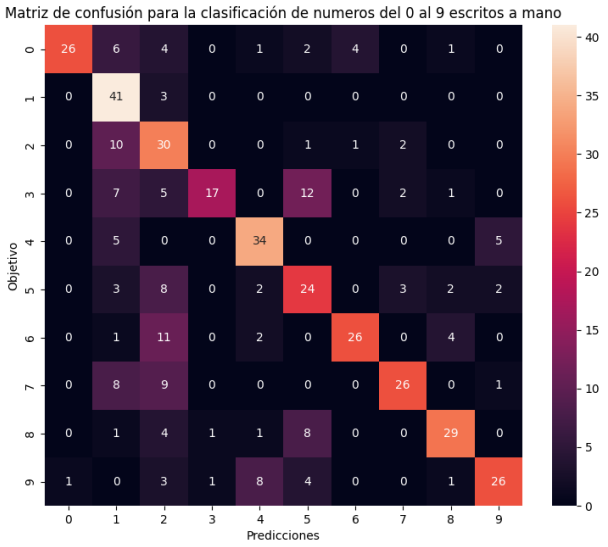


Fig. 9: Matriz de confusión producto de la cuantificación absoluta de las clasificaciones efectuadas por el modelo contra los valores objetivo en dígitos del 0 al 9. Figura de elaboración propia.

un dígito 2 como un dígito 1. Esta clase de análisis facilita la asignación de esfuerzos para la mejora iterativa del modelo.

Como parte del proceso de entendimiento de los resultados, se debe de considerar la presencia de un sesgo en la caracterización de los dígitos por parte del modelo. Dicha inclinación se debe a que en la etapa de entrenamiento el equipo de desarrollo se vio forzado a la eliminación de especímenes pertenecientes a dígitos particulares por dificultades de normalización. El conjunto reducido de entrenamiento se compone de 440 especímenes distribuidos equitativamente entre las diferentes clases.

VII. Conclusiones

1) El uso de histogramas como características a tomar en cuenta en el modelo de reconocimiento de caracteres escritos a mano, resulta un acercamiento propenso a dar un nivel de precisión bajo si los caracteres no son escalados ni centrados correctamente, esto debido a que es sensible a cambios de tamaño y posición.
2) La caligrafía de las personas que generaron el conjunto de datos es un factor que afecta directamente a la precisión del modelo, debido a que este escala y centra todos los caracteres de forma automática. En casos como el numero uno “1”, si la persona lo escribe como una linea recta vertical semejante a una ele “l”, el histograma que este genera posee grandes diferencias con respecto al que que generaría si se escribiese de la forma convencional con una linea horizontal como base, una linea vertical y una línea inclinada adyacente al extremo superior de la línea vertical.
3) El sistema de reconocimiento de dígitos escritos a mano que implementa un enfoque de tres etapas que incluye preprocesamiento, extracción de características y clasificación. Este enfoque permite transformar las imágenes crudas en un formato más manejable, identificar características clave en cada imagen y clasificar eficazmente los dígitos.
4) El sistema de reconocimiento óptico de caracteres desarrollado en esta investigación se distingue de otros modelos existentes. Este modelo, las características clave se derivan de los valores del histograma, tanto horizontal como verticalmente, con un ancho de 4 píxeles. en contraste con la del Tesseract OCR, que organiza los componentes en estructuras conocidas como “Blobs” y realiza un análisis de las líneas de texto para determinar si su espaciado es fijo o proporcional.

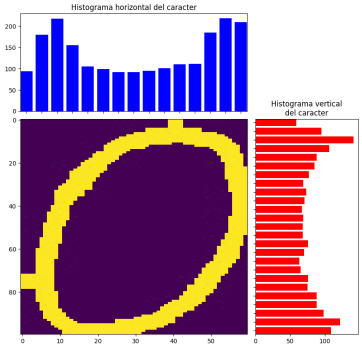
Referencias

[1] P. S. Foundation, “What Is Python? Executive Summary,” Python, 2023. [Online]. Available: https://www.python.org/doc/essays/blurb/
[2] OpenCV, “About OpenCV,” OpenCV, 2018. [Online]. Available: https://opencv.org/about/
[3] NumPy Team, “Numpy,” numpy.org, 02 2024. [Online]. Available: https://numpy.org/about/
[4] R. Smith, “An overview of the tesseract ocr engine,” Tesseract Documentation, 2007. [Online]. Available: https://tesseract-ocr.github.io/docs/tesseracticdar2007.pdf
[5] ——, “2. architecture and data structures: A quick tour of the tesseract code,” 2007.
[6] S. Navarro, “¿qué es keras en deep learning?” keepcoding, 2024. [Online]. Available: https://keepcoding.io/blog/keras-en-deep-learning/
[7] TensorFlow, “Keras,” TensorFlow, 2018. [Online]. Available: https://www.tensorflow.org/guide/keras?hl=es-419
[8] M. Rivera, “Arquitecturas de rede neuronales con múltiples ramas,” unipython. [Online]. Available: http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/arquitecturas/arquitecturas_Keras.html
[9] dbpedia., “About: Mnist database,” dbpedia, 2024. [Online]. Available: https://dbpedia.org/page/MNIST_database
[10] A. Gibiansky, “K nearest neighbors: Simplest machine learning,” 2024. [Online]. Available: https://andrew.gibiansky.com/blog/machine-learning/k-nearest-neighbors-simplest-machine-learning/
[11] afael Fernandez, “Desarrolla tu primera red neural en python con keras paso a paso,” unipython, 2018. [Online]. Available: https://unipython.com/desarrolla-primera-red-neural-python-keras-paso-paso/

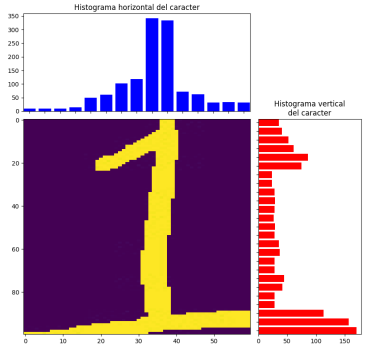
Joshua Gamboa Calvo Estudiante de la carrera de Ingeniería en Computación en el Instituto Tecnológico de Costa Rica. Carné 2020037148. Correo electrónico joshgc.19@estudiantec.cr.

Andrés Montero Gamboa Estudiante de la carrera de Ingeniería en Computación en el Instituto Tecnológico de Costa Rica. Carné 2020048696. Correo electrónico amonterog@estudiantec.cr.

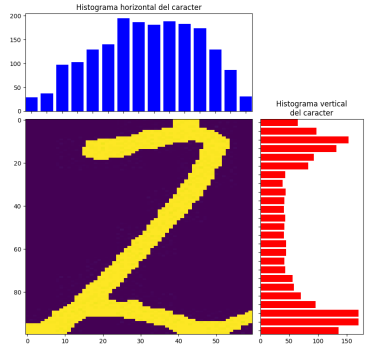
Álvaro Moraría Villalobos Estudiante de la carrera de Ingeniería en Computación en el Instituto Tecnológico de Costa Rica. Carné 2018319327. Correo electrónico newconker@estudiatec.cr.



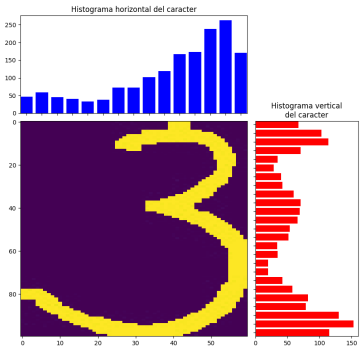
(a) Histograma del vector de características para el dígito 0.



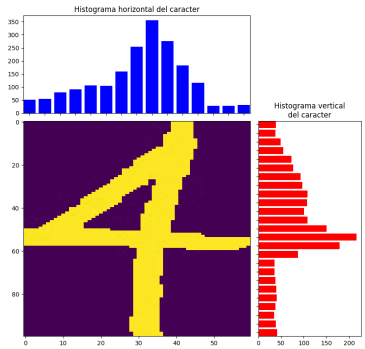
(b) Histograma del vector de características para el dígito 1.



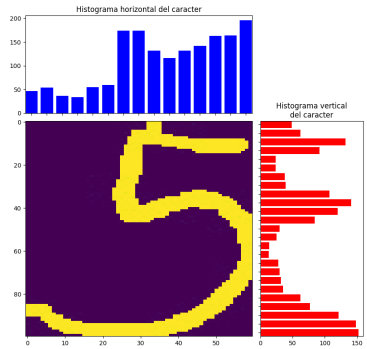
(c) Histograma del vector de características para el dígito 2.



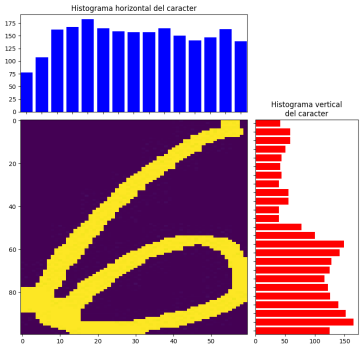
(d) Histograma del vector de características para el dígito 3.



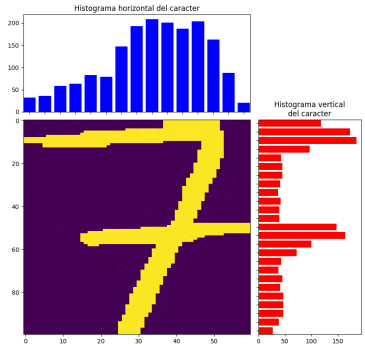
(e) Histograma del vector de características para el dígito 4.



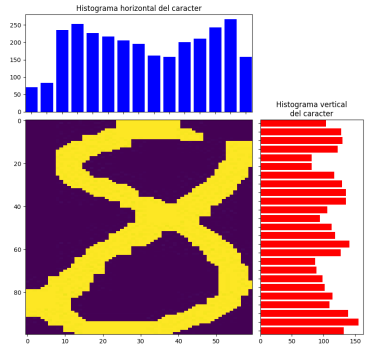
(f) Histograma del vector de características para el dígito 5.



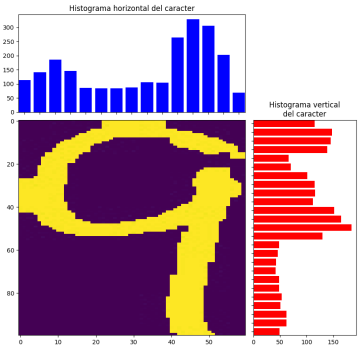
(g) Histograma del vector de características para el dígito 6.



(h) Histograma del vector de características para el dígito 7.



(i) Histograma del vector de características para el dígito 8.



(j) Histograma del vector de características para el dígito 9.

Fig. 10: Recopilación de histogramas de vectores de características para los dígitos del 0 al 9 en escritura a mano. Todas figuras de elaboración propia.