UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

Reconocimiento y resolución de expresiones matemáticas mediante el uso de Reconocimiento de Imagen

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Monrabal Adán, Francisco José

Tutor/a: Linares Pellicer, Jordi Joan

CURSO ACADÉMICO: 2023/2024

Resumen

El uso de aplicaciones y herramientas relacionadas con el cálculo proporciona una gran ayuda en la vida diaria de muchas personas, siendo estas esenciales en contextos académicos. Este documento detalla el desarrollo de un resolutor de operaciones matemáticas. Para ello se ha empleado el uso de técnicas de visión por computador. Además del desarrollo de una red neuronal para la identificación y predicción de caracteres. Todo esto detrás de una API REST para facilitar el acceso a la lógica desarrollada.

Palabras clave: Inteligencia Artificial; Visión por computador; API REST; Reconocimiento de imágenes; Redes Convoluciones; Reconocimiento Óptico de Caracteres.

Abstract

The use of applications and tools related to calculus lends a great help in the everyday life of everyone, these tools being essential in academic contexts. This document details the construction of an equation solver. For this purpose we used computer vision techniques. Furthermore the development of a neural network with the purpose of identifying and predicting characters. All this logic being behind an API REST for the ease of use of that same logic.

***Keywords**: Artificial Intelligence; Computer Vision; API REST; Image Recognition; Convolutional Networks; Optical Character Recognition.*

Índice general

Índice de figuras

Índice de tablas

1 Introducción

Cada vez es más común y prevalente, el uso de IA, *Inteligencia Artificial*, y *Redes neuronales* profundas también conocidas como *Deep Neural Network*, DNNs. Con fines como, generación de imágenes, detección de patrones, tratamiento y generación de audio, tratamiento y generación de textos, y una lista más de usos.

Todas estas nuevas funcionalidades nos facilitan muchos trabajos que antes dependían de un gran número de horas, y de conocimientos en el susodicho tema. Actualmente con estas nuevas facilidades, no es necesario gastar tiempo en tareas como la generación de un código simple para un determinado lenguaje. Podemos delegar ese trabajo a una IA, y centrarnos en otras partes del proyecto, como pueda ser la arquitectura del código, seguridad, etc. Por las razones mencionadas anteriormente, los tiempos de entrega en estas tareas, pueden ser reducidos.

Un tema que también se ha abordado en los últimos tiempos es el reconocimiento de texto. Existen distintos softwares como, *Google Document AI* [1], el cual es capaz de reconocer páginas de texto en distintos formatos, como pueda ser imágenes, PDFs y documentos escritos a mano. Extrayendo el texto de dichos documentos y exportándolo a un documento de *Google docs* [1].

En este documento se expone el desarrollo de un proyecto cuyo objetivo es: dada una imagen de una ecuación reconocer esa misma ecuación y resolverla. A diferencia de otros OCRs, *Optical Character Recognition*, intentaremos resolver dicha ecuación.

1.1 Motivación

Debido a la falta de soluciones comerciales que aborden el reconocimiento y la resolución de operaciones en un mismo sistema. Surge la motivación de crear una solución capaz de abordar ambas acciones, reconocimiento y resolución, en un mismo software.

También es interesante hacer una comparación de la solución desarrollada, con otros documentos académicos. Los cuales sí presentan soluciones donde ambas acciones están presentes.

1.2 Objetivos

Los objetivos de este proyecto son los siguientes:

- Implementar una solución capaz de reconocer operaciones matemáticas presentes en una imagen y resolver la operación reconocida.
- La solución ha de ser capaz de como mínimo resolver ecuaciones de primer grado.
- Exposición de conocimientos necesarios como puedan ser, API REST, Redes neuronales y técnicas de Visión por computador.
- Dar una breve descripción de la situación actual o estado del arte, respecto a los temas relacionados con la solución, mostrando los puntos más interesantes.
- Detallar el proceso realizado para la construcción y desarrollo de la solución, así como las herramientas empleadas.
- Evaluación del resultado, y comparación frente al estado del arte.

1.3 Estructura del documento

El documento está estructurado de manera que, empezaremos por una breve explicación del *estado del arte* 2. Continuando con los *fundamentos* 3 necesarios para la comprensión de la aplicación. Seguidamente mencionar las *herramientas* 4 utilizadas. Prosiguiendo con la explicación de la *experimentación* 5, para más tarde mostrar los *resultados* 6. Finalmente terminaremos con los puntos de *conclusiones* 7 y *trabajos futuros* 8.

2 Estado del arte

En este capítulo, revisaremos trabajos relacionados tanto en el ámbito de OCRs 2.1, como resolutores de ecuaciones 2.2. Concretamente, software que no necesariamente se va a dedicar únicamente a la resolución de dichas ecuaciones, sino a la resolución de problemas matemáticos con una mira más amplia. Acabaremos hablando sobre las opciones disponibles que contemplan ambos campos como solución.

2.1 OCRs

OCR, *Optical Character Recognition*, es un campo de la *Inteligencia Artificial*, el cual su principal objetivo es el de analizar un texto, reconocerlo, y digitalizarlo. Esto no es exclusivo de textos únicamente, puesto que también puede ser reconocido cualquier tipo de carácter que el modelo del OCR soporte, como es en nuestro caso con una nomenclatura matemática.

Esta sección está dividida en: OCRs comerciales que pueden ser interesantes destacar, y la librería *Tesseract*. La librería *Tesseract*, puede ser utilizada junto a un lenguaje de programación, con el fin de poder interactuar con el texto el cual ha sido extraído por el modelo. En concreto usamos *Tesseract*, porque es una librería reconocida, y está disponible en el lenguaje *Python*, lenguaje el cual cobra una importancia considerable en el proyecto.

2.1.1 OCRs Comerciales

Por lo general, muchos de estos OCRs comerciales, pueden ser agrupados en su mayoría como intérpretes de PDFs, imágenes, u otro tipo de formatos donde encontrar estos textos. Estos OCRs reconocen el texto de dichos documentos, y lo digitalizan a un formato normalmente a elección del consumidor, o a elección del creador del software.

Muchas veces, estos softwares no realizan un trabajo adicional al mencionado. Por lo que comercialmente las opciones de analizar una operación matemática, y aportar un resultado en el proceso, son en su mayoría escasas.

Casos de OCRs comerciales, podrían ser *Google Document AI* [1], *IBM Datacap* [2] o *Adobe Acrobat OCR* [3], entre muchos otros.

Uno de los casos más interesantes es el caso de *Mathpix OCR* [4]. Este ofrece un OCR enfocado al reconocimiento de casos como ecuaciones, diagramas químicos, tablas, etc., ofreciendo distintos outputs, para distintos tipos de documentos, como pueden ser LaTeX, *PDF*, *HTML*, y otros más.

2.1.2 Tesseract [5]

Desarrollado por Google, es de código abierto y uno de los OCRs más precisos que hay en el mercado. Además es uno de los más populares, que puede utilizarse junto al lenguaje de programación Python para poder tratar el texto.

Es una de las opciones más utilizadas cuando se trata de *Visión por computador* 3.4.

2.2 Resolutores de operaciones matemáticas

Definiremos como resolutores de operaciones matemáticas a, las soluciones que posean funcionalidades directamente dedicadas al tratamiento, resolución, cálculo o análisis de materias de índole matemático.

La principal razón por la que se van a mencionar estos resolutores, es la de añadir riqueza y otros puntos de vista al documento. Haciendo ver así que la potencia de estos, junto a un OCR, podría formar una solución más completa que un simple OCR.

2.2.1 Wolframalpha [6]

Las prestaciones de este software son muy amplias, desde álgebra lineal, pasando por cálculo y análisis matemático, representaciones gráficas, etc. Este resolutor, también posee una API para poder atacar, y hacer peticiones. Es uno de los softwares más competentes actualmente.

2.3 OCRs orientados a la resolución de operaciones

La inmensa mayoría de OCRs comerciales, no pasan del proceso de digitalización del texto. Parece que la tendencia es separar estas funcionalidades y dar un producto de una de las dos partes, o bien un OCR o un resolutor.

Combinaciones de ambos, pueden ser encontrados en documentos con carácter más académico, como por ejemplo la aproximación que proponen *Shinde, Rajwardhan and Dherange, Onkar and Gavhane, Rahul and Koul, Hemant and Patil, Nilam* en *Handwritten mathematical equation solver* [7]. En este documento, detallan el como resolver dos tipos de ecuaciones, una que ellos denominan como simple, y otra compleja. Residiendo la prin-

cipal diferencia, en que la compleja, es una ecuación polinómica, y la otra una simple operación. Para ello han desarrollado tres modelos: dos para operaciones simples, y otro para las complejas.

Para el caso de la operación simple, se emplea una combinación de dos modelos. Un modelo DNN, *Deep Neural Network* el cual está encargado de extraer características de la imagen, y un RNN, *Recursive Neural Network*, encargado de identificar la operación y resolverla.

Para el segundo tipo de ecuación, la ecuación polinómica, usan una CNN para la aproximación al problema. Este recibe una serie de ROIs, *Regions Of Interest*, que representan cada carácter, y luego son alimentados al modelo para identificar.

Aunque las comparaciones de resultados entre los modelos no se pueden realizar, por el hecho de que los problemas son distintos. Enseñar los porcentajes de acierto puede ser interesante, para saber sobre que rango de acierto nos estamos moviendo.

Sr.no	Model	Accuracy
1.	Dense Model (Simple Equation)	98 %
2.	RNN Model (Simple Equation)	96 %
3.	CNN Model (Complex Equation)	85 %

Tabla 2.1: Resultados de precisión de los 3 modelos, como se puede observar la mayor precisión la ostenta el resolutor de ecuaciones simples. Pero, dado que ambos no abordan el mismo tipo de problema, la comparación de modelos no es justa.

Otro documento de interesante mención es *HANDWRITTEN LINEAR EQUATION SOLVER* [8]. Este documento es mucho más extenso y detallado que el anterior. En este caso se parece al nuestro, con la diferencia de que en este han decidido que las imágenes finales a procesar, son el resultado de un dibujo. Este dibujo se traza y envía desde un HTML. Después la imagen recibida es segmentada en sub-imágenes representativas de cada carácter, y alimentadas a una CNN, donde son reconocidas.

Train acc	Val acc	Train loss	Val loss
96.28 %	98.88 %	14.7 %	4.26 %

Tabla 2.2: Métricas del modelo propuesto

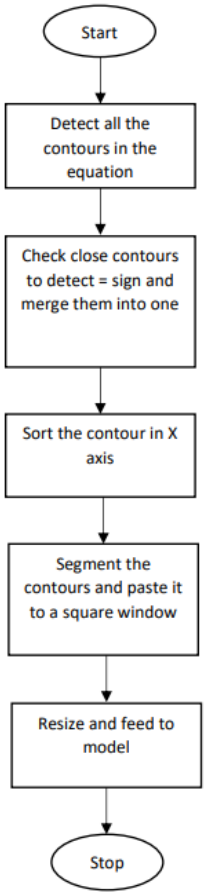

Figura 2.1: Flujo expuesto del procesado de la imagen

2.4 Puntos a destacar

Después de todos los datos expuestos con anterioridad, es de interés destacar los siguientes puntos:

- En lo que se refiere a soluciones comerciales, hay una clara separación entre soluciones de reconocimiento de caracteres y resoluciones de operaciones matemáticas.
- El uso de DNNs para la identificación de caracteres, da resultados precisos. Aunque la mayoría de aproximaciones se hacen mediante CNNs.
- La aproximación mediante CNNs no es la única aproximación posible.
- Ambos documentos académicos, tratan con operaciones cuya complejidad gráfica es muy baja. Refiriéndonos como complejidad gráfica a, *la cantidad de trazados no lineales en la ecuación, como puedan ser, exponentes, fracciones, decimales, integrales, etc.* Distinguir estos trazados conlleva un esfuerzo adicional de preprocesado de imagen.

- *Shakya* [8] habla de como la detección de patrones como fracciones no es posible. Pero en el caso de la solución de *Mathpix OCR* [4], se puede observar como sí que es capaz de detectar estos caracteres, lo que da a entender que es un punto posible pero complicado de abarcar.

3 Fundamentos

La finalidad de este capítulo, es la proporción de contexto y conocimientos básicos relacionados con la parte experimental detallada en capítulos posteriores. Aun así, cierta información no va a ser suministrada en este documento, dado que se sale del propósito del mismo.

Trataremos cuatro temas principales, en primer lugar, la explicación de una API REST, así como los principios teóricos de la misma 3.1. A continuación, el paradigma de programación MVC 3.2. Continuaremos con conceptos básicos sobre redes neuronales, así como distintas capas y activaciones 3.3. Y por último Visión por computador y algunas de las técnicas a destacar utilizadas durante el desarrollo de la aplicación 3.4.

3.1 API REST

Para comprender el funcionamiento de una *Application Programing Interface*, API de ahora en adelante, la cual cumple con los principios REST, *Representational State Transfer*, debemos primero responder a las preguntas: ¿Qué es un API? y ¿Qué es una arquitectura REST?.

Más adelante, también necesitaremos ver el protocolo HTTP 3.1.2, para poder terminar de explicar las arquitecturas REST, y cerraremos viendo cómo todas estas piezas juegan entre sí, para poder explicar el funcionamiento de una API REST.

3.1.1 API

Conjunto de código que funciona como intermediario entre dos sistemas, actuando como fachada de una lógica, la cual es atacada por otro sistema para la explotación de la lógica que hay detrás de esta misma fachada. Esta fachada sirve de supuesto contrato, estipulando al sistema que la explota, cuáles son sus capacidades y sus limitaciones.

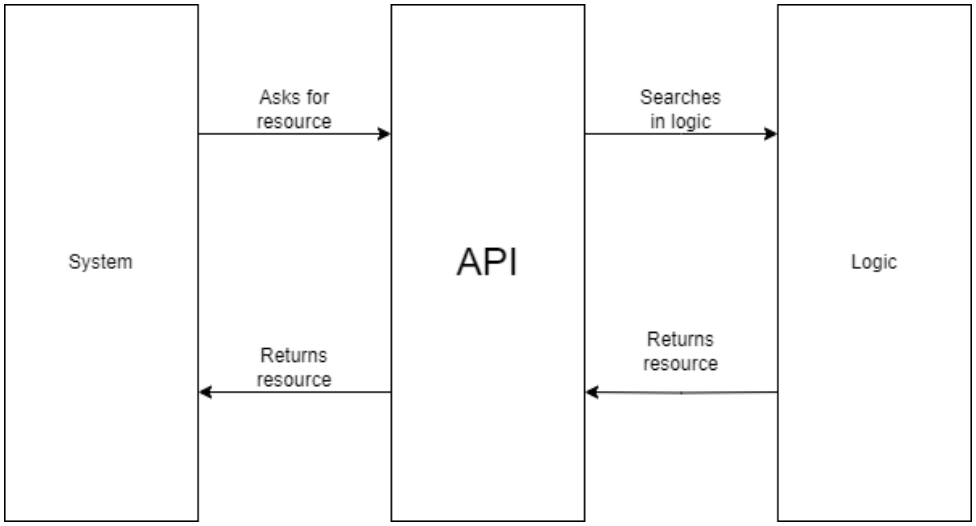

Figura 3.1: Sistema a la izquierda atacando a la API, esta consulta en la lógica y devuelve el resultado

Una de las principales características de esta estructuración del código, es la reutilización del mismo, dado que los dos sistemas no son fuertemente dependientes, por lo que varios sistemas pueden aprovecharse de una misma API.

3.1.2 HTTP

Hyper text transfer protocol, protocolo de red definido como protocolo de aplicación por el modelo OSI. Está encargado de la transferencia de información en un modelo de cliente/-servidor. Este posee una serie de operaciones para conseguir dicha funcionalidad. Dado que el protocolo tiene muchas operaciones, nosotros solo vamos a ver las cuatro operaciones que nos interesan.

- **GET**: Busca acceder y recuperar recursos disponibles en el servidor.
- **POST**: Busca la creación de nuevos recursos en el servidor.
- **PUT**: Busca la actualización de unos recursos ya disponibles por parte del servidor.
- **DELETE**: Busca borrar un recurso disponible en el servidor.

3.1.3 REST

REpresentational State Transfer, REST, es una arquitectura de software orientada a cliente/ servidor y sistemas distribuidos. Siendo su principal medio de comunicación el protocolo de red HTTP, que como se ha mencionado en el punto anterior, se basa en cuatro operaciones básicas: POST, GET, PUT, DELETE, para realizar la comunicación entre los equipos.

Cada una de estas operaciones, es lo que se denomina stateless, no guarda información sobre el cliente una vez la transacción ha terminado. Estas operaciones, usualmente están muy relacionadas con operaciones CRUD, *Create, Read, Update, Delete*, estando cada una de ellas directamente relacionadas con uno de los comandos.

HTTP	CRUD
POST	Create
GET	Read
PUT	Update
DELETE	Delete

Tabla 3.1: Operaciones REST, relacionadas con su respectiva operación equivalente CRUD

Normalmente, todas estas operaciones están relacionadas con una serie de *Endpoints*, los cuales están definidos mediante una URI, *Uniform Resource Identifier*. Estos *Endpoints* son atacados mediante las operaciones HTTP mencionadas antes, con el fin de acceder a un recurso expuesto por REST.

3.1.4 API REST

Habiendo expuesto todos los conocimientos anteriores, una *API REST*, podría definirse como una API con una arquitectura cliente/servidor. Esta, es explotada mediante la red a través del uso del protocolo HTTP, para realizar las operaciones a los distintos *Endpoints*, suministrados por el servidor.

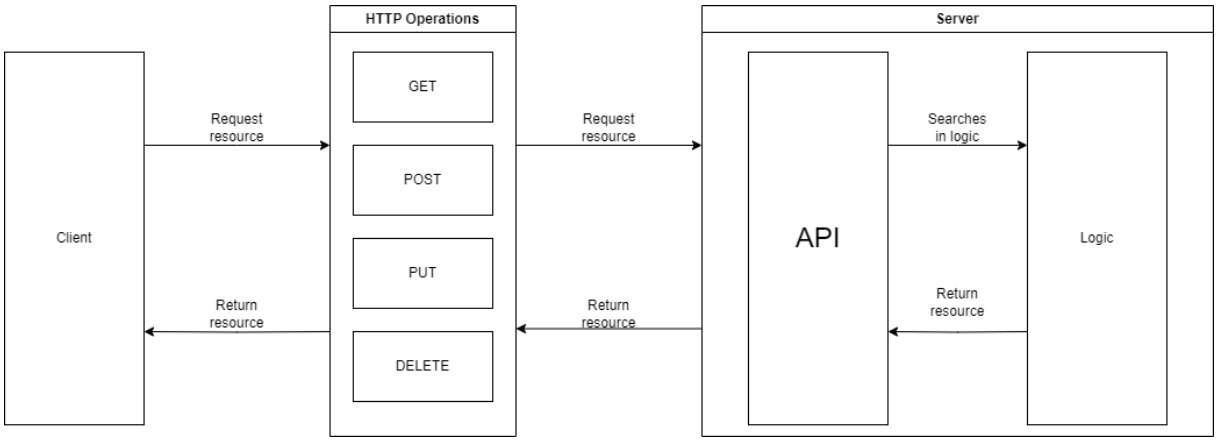

Figura 3.2: A la izquierda el cliente hace una petición mediante HTTP a la API, la cual procesa la petición y devuelve el recurso.

3.2 MVC

Model View Controller, MVC, es un patrón de arquitectura de software, cuyo fin es ordenar el código de una aplicación y delegar responsabilidades específicas a las distintas partes, creando una comunicación específica entre estas, para facilitar la escalabilidad del código.

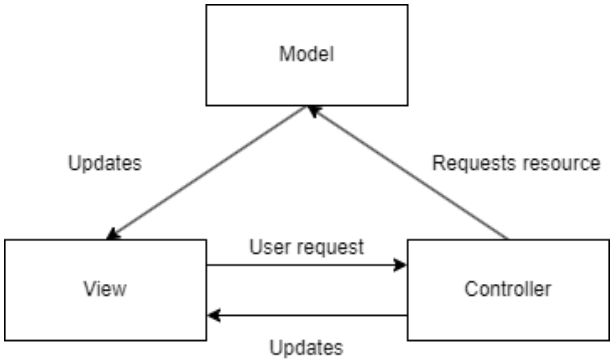

Figura 3.3: MVC, como podemos observar el modelo es capaz de comunicarse con la vista

La estructura usual que suele presentar el patrón MVC, está reflejada en el diagrama 3.3. Este patrón permite al modelo comunicarse con la vista después de que el controlador haya realizado una petición en nombre de la vista. En el caso de este proyecto, la vista y el modelo nunca pueden comunicarse, todas las interacciones entre modelo y vista pasan por el controlador, desacoplando así dependencias entre el modelo y la vista, resultando en un código más escalable. En algunos casos este patrón es llamado *Model View Presenter* MVP, donde la vista, únicamente habla con el controlador, ahora presentador, y delega la poca lógica que tenía antes al presentador.

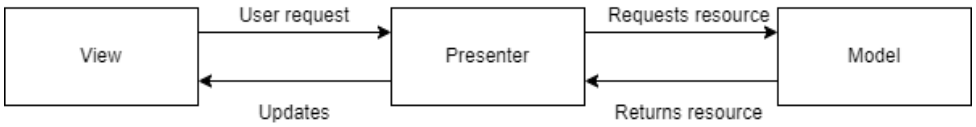

Figura 3.4: MVP, la vista ha de pasar estrictamente por el presentador para hablar con el modelo y viceversa

3.2.1 Modelo

Componente responsable del modelo de datos y la lógica de negocio de la aplicación. Normalmente recibe una petición del controlador, esta es procesada y devuelta al controlador, donde este se encargará de redirigirla a la vista.

3.2.2 Vista

Componente responsable de presentar la información que recibe del controlador de manera adecuada.

3.2.3 Controlador

Componente responsable de la comunicación y flujo de la aplicación entera. Está encargado de que la vista y el modelo hablen entre ellos haciendo de intermediario. También está encargado de manejar los eventos y saber que partes del modelo y la vista ha de lanzar en cada caso, controlando el flujo de la aplicación y su comportamiento.

3.3 Redes neuronales

Una red neuronal es un conjunto de estructuras capaces de procesar información. Normalmente este conjunto de estructuras están agrupadas en una serie de formaciones de neuronas, las cuales denominamos capas. Estas capas a su vez son agrupadas entre ellas, superpuestas y conectadas, dando lugar a una estructura final compuesta de capas, que a su vez están compuestas por neuronas.

Dada la naturaleza de la agrupación de estas capas, podemos distinguir varios tipos en función de su posición en la agrupación.

- **Capas de entrada**: Capa inicial donde se produce la recepción de los datos.
- **Capas de salida**: Capa final, la cual recibe los cálculos de las capas intermedias y devuelve el resultado final.
- **Capas ocultas**: Capa o capas intermedias entre la capa de entrada y la capa final, normalmente en estas se producen la mayoría de cálculos.

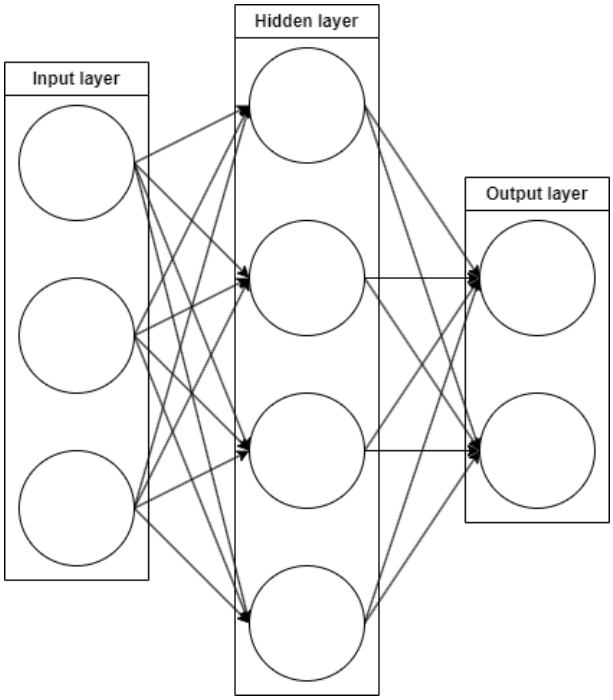

Figura 3.5: Red Neuronal Profunda, DNN, podemos ver como todas las neuronas de cada capa se conectan entre sí, además de la distinción entre los 3 tipos de capas en función de su posición dentro de la red.

Normalmente, en todas las subsecuentes capas después de la inicial, cada una de estas neuronas almacena un valor, el cual normalmente es indicativo de una de las características del objeto que se está procesando. El valor contenido por estas neuronas se expresa en la siguiente ecuación.

$$y(x, w) = f(\sum_{j=1}^{n} x_j w_j)$$

El resultado depende de x_j siendo este el valor de las anteriores neuronas, w_j el valor de los pesos que conectan a la neurona actual con la anteriores, f la función de activación seleccionada para la capa actual, y n el número total de neuronas de la capa a la que pertenece la neurona que se procesa.

Existen muchas funciones de activación, pero vamos a ver únicamente las dos que vamos a utilizar:

- **ReLu**: *Rectified Linear Unit* es una de las activaciones más simples y es la que más vamos a usar. $f(x) = max(0, x)$.
- **Softmax**: Función cuyo uso principal es convertir los valores de salida de la última capa en una distribución con valores de probabilidad, para así poder distinguir más fácilmente el output a que clase se refiere. $f(x) = \frac{e^{xi}}{\sum_{J}^{j=1} e^{xj}}$

A continuación vamos a describir los dos tipos de capas a utilizar.

3.3.1 Capas densas

Son el tipo de capas más utilizadas y comunes, donde las neuronas están conectadas con todas las neuronas de la capa anterior en el caso de que no sea la inicial ni la posterior. Dado que nosotros vamos a procesar imágenes, normalmente las capas densas en sí, no son utilizadas para estas tareas. En este caso, las utilizaremos como capas de input y output.

El funcionamiento de estas capas lo hemos explicado en el punto anterior. Todas las neuronas de la capa anterior se conectan cada una de ellas a todas las neuronas de la siguiente capa. Cada una de esas conexiones tiene una serie de pesos, que se utilizan para calcular el valor de la neurona.

3.3.2 Capas convoluciones

Este tipo de capas son muy frecuentemente usadas en el tratamiento y clasificación de imágenes. El funcionamiento de estas, está basado en un kernel de unas prefijadas dimensiones, el cual va recorriendo la imagen y extrayendo características de la misma, y realizando operaciones con cada valor de los píxeles de la imagen. Resultando así en una abstracción de dichas características, y reduciendo la imagen a una matriz más pequeña.

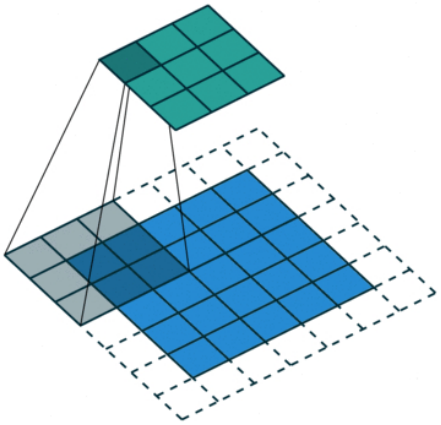

Figura 3.6: En la parte de arriba la matriz resultante de la imagen, abajo la imagen, también podemos observar la proyección del kernel en la imagen. Fuente [9]

Además, estas capas están fuertemente vinculadas con otro tipo de capas, llamadas *Pooling layers*, las cuales deciden cómo se extraen los datos de la imagen y por ende, el valor de la consecuente matriz que simplifica la imagen anterior.

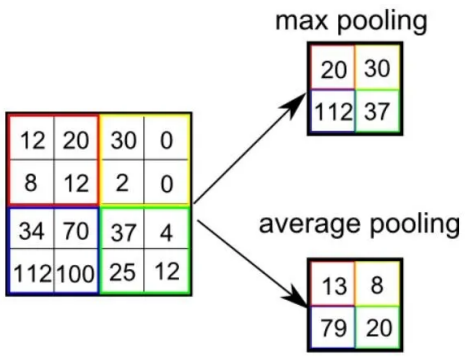

Figura 3.7: Matrices resultantes de distintos poolings. Fuente [9]

3.4 Visión por computador

La visión por computador es uno de los campos que está cogiendo más relevancia, dado el hecho de su fuerte relación con la inteligencia artificial y su aumento en los últimos años.

Podemos definir Visión por computador como, un conjunto de técnicas a través de las cuales podemos adquirir, procesar, analizar, y entender imágenes, mediante el uso de un computador. Esta serie de técnicas son muy útiles a la hora de tratar con IA, dado el hecho de poder hacer un preprocesado de las imágenes a tratar, antes de analizarlas con un modelo, y así facilitar tanto procesamientos, como complejidades de desarrollo por parte de los modelos.

La mayoría de imágenes, están representadas por una o más matrices superpuestas, las cuales están formadas por lo que denominamos píxeles, siendo estos la representación mínima de la imagen.

Según la cantidad de capas, y la representación de las mismas, estas dan lugar a distintas codificaciones. Desde imágenes en escala de gris, las cuales son una única matriz y cuyos píxeles representan la intensidad del gris. Pasando por imágenes RGB, *Red Green Blue*, que están compuestas por tres capas, cada una de ellas identificando la intensidad del color que representan, dando lugar así a imágenes en color. Hasta imágenes multiespectrales, con una gran cantidad de capas, las cuales abarcan desde luz visible, hasta infrarrojos y ultravioleta.

También un concepto que ya se ha nombrado y volverá a ser nombrado a lo largo del documento son las Regiones de Interés o *Region of Interest*, ROI. Estas no son más que subregiones de la imagen delimitadas por dos puntos, y que representan un área de interés de la imagen. Normalmente estos ROIs suelen representar una subregión a analizar.

Únicamente vamos a mostrar dos técnicas, las cuales están implicadas en el desarrollo de la experimentación.

3.4.1 Umbralización

Técnica dentro del campo de la segmentación, siendo una de las más simples y antiguas. Teniendo una imagen de una única matriz, ya pueda ser en escala de grises o uno de los colores RGB que compone una imagen, asignamos un valor T el cual va a valer de punto de referencia para todos los demás píxeles. En caso de que el valor de cada uno de los píxeles sea superior o inferior a T se le asignará el valor máximo de píxel o el valor mínimo.

$$dst(x,y) = \begin{cases} maxValue & if src(x,y) > T(x,y) \\ 0 & otherwise \end{cases}$$

La función se aplica a cada píxel de la matriz que representa a la imagen, siendo $dst(x,y)$ la función aplicada al píxel destino, $maxValue$ el valor máximo posible del píxel, $src(x,y)$ el valor del píxel de la matriz original, 0 el valor mínimo, y $T(x,y)$ el valor del umbral por el cual juzgaremos el píxel.

Este no es el único tipo de umbralización que existe, hay más casos y tipos, pero este es el más simple y el que más vamos a utilizar.

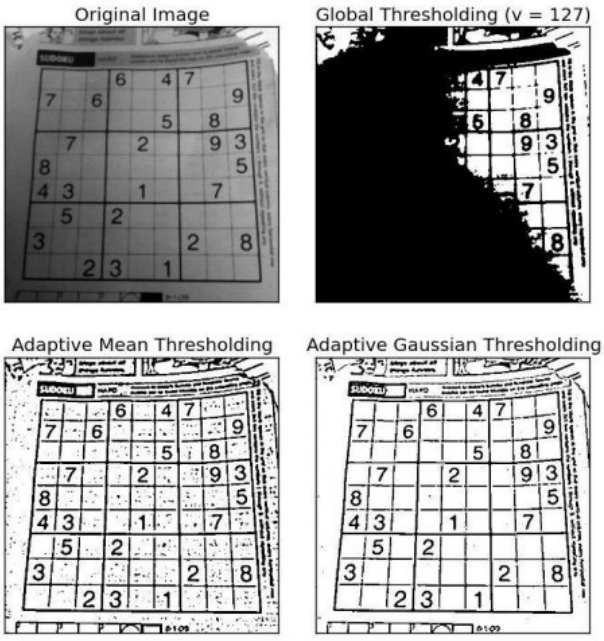

Figura 3.8: Aplicación de diferentes tipos de umbralización

Esta técnica puede resultar útil a la hora de eliminar ruido de la imagen, o querer extraer características de la misma, pudiendo después segmentar las partes que nos interesan de la imagen.

3.4.2 Detección de bordes

La detección de bordes, trata de identificar y mostrar todos los posibles bordes de una imagen, mediante el uso de una serie de funciones matemáticas. Aunque hay muchas técnicas de detección de bordes, únicamente se va a acometer la detección Canny desarrollada por John F. Canny.

El proceso de detección Canny se puede dividir en cuatro pasos bien diferenciados.

1. Aplicar un *filtro Gausiano* [10], para reducir la cantidad de ruido de la imagen, evitando así falsos positivos.

2. Aplicamos un kernel *Sobel* [11] sobre la imagen, tanto en el eje x como en el eje y, dándonos el gradiente de intensidad para cada eje en esa región del kernel. Por lo que con ambos gradientes podemos calcular el gradiente del borde y el ángulo del mismo.

$$gradBorde(G) = \sqrt{G_x^2 + G_y^2}$$

$$Angulo(\theta) = tan^{-1}(\frac{G_y}{G_x})$$

3. Se realiza un recorrido entero de la imagen y sabiendo los gradientes y sus ángulos, eliminamos todos los píxeles que no sean el máximo respecto a sus vecinos, con el fin de eliminar falsos bordes.

4. Para finalizar, se eliminan todos aquellos bordes cuyos gradientes no estén comprendidos entre los valores máximos y mínimos preestablecidos antes de iniciar el algoritmo.

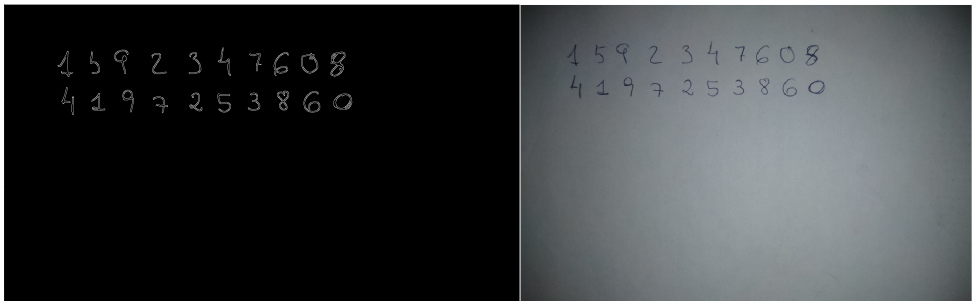

Figura 3.9: A la derecha imagen original, a la izquierda el resultado tras la detección de bordes mediante el método Canny.

4 Herramientas

Vamos a describir brevemente las herramientas utilizadas, muchas de las que vamos a mencionar a continuación, son librerías del lenguaje de programación Python, el cual mencionaremos también.

4.1 Hardware

El entrenamiento y ejecución de la aplicación se ha desarrollado en una máquina cuya tarjeta de vídeo era una Nvida GEFORCE RTX 4080. Una de las grandes ventajas que presta este hardware es el uso de tecnologías como CUDA, acortando tiempos de entrenamiento y permitiéndonos realizar una mayor cantidad de pruebas en menos tiempo.

Otras especificaciones de la máquina son un procesador Intel Core i7 13700K y 32 GB Memoria RAM DDR4.

4.2 Python [12]

Lenguaje de programación que ha predominado durante todo el proyecto. Es uno de los lenguajes más utilizados en proyectos de reconocimiento de imágenes, inteligencia artificial y otro tipo de proyectos que tengan que ver con un enfoque de ciencia de datos. Esto se debe a la amplia cantidad de librerías disponibles para el lenguaje y su facilidad de uso e importación.

En nuestro caso hemos utilizado la versión de python 3.10.

4.3 Flask [13]

Librería sencilla y ligera para el desarrollo de aplicaciones servidor como pueda ser una API REST, siendo este el uso el cual le hemos dado. La elección de esta librería frente a otras se debe a la sencillez de programar, y el uso que le íbamos a dar al backend, con lo cual, librerías más complejas como Django se salían del alcance del proyecto que queríamos desarrollar, ya que lo único que buscamos es un par de endpoints a los que poder atacar con cliente sencillo.

4.4 OpenCV [14]

Una de las principales librerías usadas en el desarrollo del proyecto. OpenCV es una librería destinada al desarrollo de aplicaciones de visión por computador, es una de las más utilizadas a día de hoy por varias razones.

Es de código abierto. Viene con una gran cantidad de funcionalidades para el trato de imágenes 2D y 3D, modelos de reconocimiento pre-entrenados y disponible en una variedad de lenguajes. Además en nuestro caso al usar versión de python, esta es un wrapper que aprovecha la facilidad del lenguaje python con la velocidad de CUDA y lenguajes de más bajo nivel.

4.5 Imutils [15]

Librería que aporta unas pequeñas nuevas funcionalidades a la librería de OpenCV con el fin de facilitar ciertas transformaciones de las imágenes a tratar.

4.6 Sympy [16]

Librería ligera cuyo propósito es la resolución de operaciones matemáticas. Nos permite la simplificación de muchos cálculos, que de otra manera serían más complejos de implementar. En nuestro caso es la encargada de resolver la interpretación de la ecuación recibida desde el modelo.

4.7 Keras [17]

Una de las librerías de redes neuronales más populares y de alto nivel disponibles, su objetivo es simplificar el desarrollo de modelos de redes neuronales, para una implementación más rápida y sencilla de los mismos. En nuestro caso utilizaremos Tensor Flow como soporte de la implementación de Keras.

4.8 Numpy [18]

Librería de Python muy conocida y altamente usada. Se especializa en cálculo de vectores y matrices de grandes dimensiones, proporcionándonos una comodidad y velocidad de procesado a la hora de tratar datos del conjunto de datos.

5 Experimentación

En este capítulo se presentan, las dificultades, desarrollos y soluciones a las dificultades encontradas.

Comenzaremos por hacer un análisis del conjunto de datos empleados 5.1, así como del proceso de diseño 5.2 y entrenamiento del modelo 5.3. Seguiremos con la preparación de la imagen para la ejecución de inferencia 5.4. Después continuaremos con la interpretación del resultado de la inferencia 5.5. Continuaremos con el cálculo de la ecuación interpretada 5.6, y acabaremos con una revisión de la arquitectura de la aplicación desde un punto de vista de cliente/servidor 5.7, y la estructura de la aplicación 5.8.

5.1 Conjunto de datos

El conjunto de datos procede de la página Kaggle [19]. Es un conjunto de números y caracteres matemáticos previamente etiquetados. Los caracteres están dibujados mediante un lápiz electrónico, los cuales son almacenados como ficheros .inkml, para después ser digitalizados en .png por un script, disponible en la descripción del conjunto de datos. Pero aunque el título sea *Handwritten*, realmente no son imágenes RGB tomadas de muestras físicas, sino que son una recreación de trazados tomados y después digitalizados.

5.1.1 Análisis

Contiene una cantidad total de 375974 imágenes en escala de grises, con un tamaño de 45 x 45 píxeles, divididas entre 82 clases. Las imágenes ya vienen binarizadas con valores de 0 o 255.

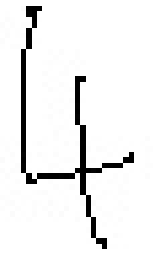

Figura 5.1: Muestra de la clase *4*, como podemos observar el trazado es de un grosor de un único píxel.

Para el desarrollo de la experimentación no vamos a necesitar todas las clases. Para nuestro problema en específico, vamos a utilizar únicamente 20 clases, que después de una serie de problemas encontrados, los cuales detallamos en 5.1.2, aplicamos varias transformaciones de *Data augmentation* y nos quedamos con el resultado final de 18 clases.

Clase	Num. Muestras
-	33,997
(	14,294
)	14,355
+	25,112
0	6,914
1	26,520
2	26,141
3	10,909
4	7,396
5	3,545
6	3,118
7	2,909
8	3,068
9	3,737
División	199
Puntos suspensivos	601
x	26,594
y	9,340
z	5,870
=	13,104

Tabla 5.1: Clases y muestras iniciales, como se puede observar el conjunto de datos está bastante desbalanceado

Como podemos observar hemos utilizado los caracteres más básicos, dado que nuestra mira inicial es el desarrollo de un resolutor de ecuaciones de primer grado. Otra gran observación, es el hecho de que el conjunto de datos está considerablemente desbalanceado.

Además, uno de los problemas presentados inicialmente es el hecho de que, el símbolo de *producto* proporcionado por el conjunto, es muy similar al carácter x, por lo que no era posible utilizarlo. Así que se ha tomado la decisión de utilizar la otra nomenclatura que está designada para señalar un producto 5.2. El mayor problema, es el hecho de que el conjunto no presenta ninguna clase similar, por lo que más adelante 5.1.2, veremos como hemos sorteado este problema.

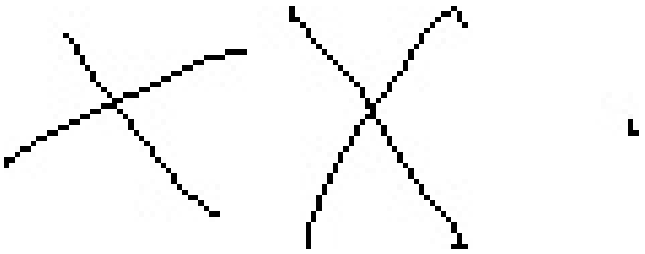

Figura 5.2: A la izquierda muestra de la clase *x*, en medio una muestra de la clase *Producto*, y a la derecha el carácter de *Producto* deseado, como se puede observar las muestras son prácticamente idénticas, por lo que el modelo le va a resultar altamente complicado distinguirlas.

Como última observación, que no se dio hasta el entrenamiento del modelo, los trazados de los caracteres son delgados, y más concretamente de una anchura de un único píxel. Estos trazados, aunque muy similares en forma, distan mucho de la realidad en cuanto a grosor 5.3.

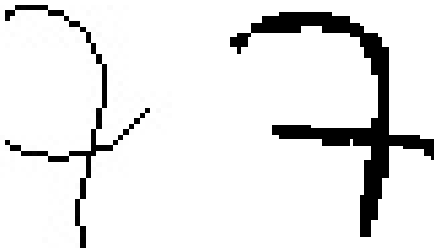

Figura 5.3: A la izquierda muestra de la clase *7* a la derecha caso real de carácter *7*, se puede apreciar la diferencia en grosor.

5.1.2 Data augmentation

Después de ver los problemas planteados durante el análisis del conjunto de datos en el análisis 5.1.1, en esta parte veremos las soluciones adoptadas, explicando las transformaciones realizadas, y mostrando el conjunto de datos final usado para entrenar el modelo.

Antes ha sido mencionado el hecho de que el conjunto de datos del que partimos no posee los caracteres de *Producto* necesarios. Por lo que para extraerlos, el conjunto no tiene una clase parecida a un punto en medio, pero sí que tiene una clase de puntos suspensivos, siendo esta la razón por la que ha sido incluida esta clase en la tabla 5.1.

Con el fin de abordar el problema del *Producto*, los puntos suspensivos contienen un punto en mitad de la imagen, el cual puede ser utilizado como carácter para *Producto*. Por lo que la única transformación que deberíamos de realizar es, cambiar los valores de los píxeles de alrededor del punto, por ende cambiando los píxeles del contorno de la imagen en un rango de 15 píxeles al valor de 255, siendo este el valor blanco de la imagen, y así resultando en un punto central que podemos utilizar como *Producto*.

Figura 5.4: A la izquierda puntos suspensivos, a la derecha resultado de la transformación, dando lugar a una muestra del carácter *Producto*.

Para resolver el problema del grosor de trazado de los caracteres, y el hecho de que algunas clases como *División* tenían un problema de falta de muestras. Se ha aumentado toda la cantidad de muestras por tres, aplicando una transformación de aumento de grosor, dando resultado a tres tipos de grosor: el *inicial*, *medio* y *grueso*.

Esto ha sido gracias a la función de *OpenCV erode()* [20]. En la mayoría de los casos la función para hacer las lineas más gruesas es *dilate()*, pero en nuestro caso que es negro sobre blanco e*rode()* es la función contraria. Aplicando la función sobre el conjunto con un grosor adicional de 3 píxeles y 5 píxeles, manteniendo las imágenes iniciales, resolvemos el problema del grosor de trazado, triplicando la cantidad de muestras.

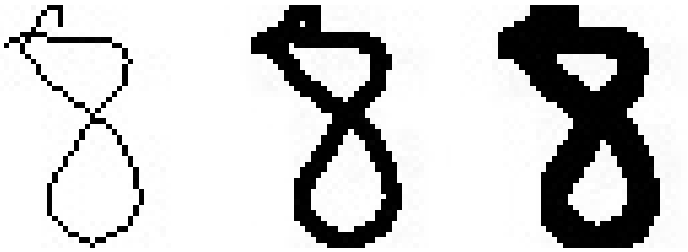

Figura 5.5: Las 3 muestras con distinto grosor

Por último, durante el entrenamiento, que veremos después algo más en profundidad, surgió el problema de la confusión constante por parte del modelo, entre las clases *0* y *Producto*. Entre este hecho, y el caso de que en la mayoría de ocasiones, las ecuaciones no plasman la operación *Producto* mediante su símbolo, pero se sugiere de manera implícita, como pueda ser en el caso de una x seguida de un *(*, se eliminó el carácter del conjunto de datos.

Otro caso similar es el caso de =, el cual durante el proceso de segmentación, al no estar unidos los trazados, no se segmentaban en un único carácter =, sino en dos caracteres de la clase -. Por lo que desde entonces se ha eliminado el carácter = y se hace una detección del mismo mediante contexto.

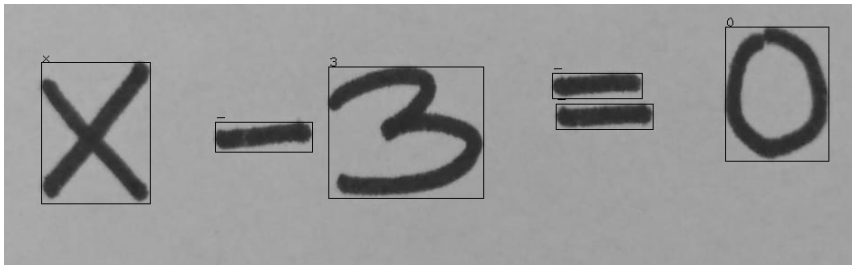

Figura 5.6: Ecuación reconocida, como se puede observar la segmentación separa el carácter = en dos -

Por lo que el conjunto resultante después de todas las transformaciones es el siguiente 5.2:

Clase	Num. Muestras
-	101,991
(	42,882
)	43,065
+	75,336
0	20,742
1	79,560
2	78,423
3	32,727
4	22,188
5	10,635
6	9,354
7	8,727
8	9,204
9	11,211
División	597
x	79,782
y	28,020
z	17,610

Tabla 5.2: Clases y muestras finales, se han eliminado las clases = y *Puntos suspensivos*

Como apunte final, se sopesó el uso de capas de *Data augmentation* en la definición del modelo, pero se descartaron por la razón de que la mayoría de ellas eran cambios de color o contraste, las cuales no nos interesa dado que es una binarización, y cambio de orientaciones, las cuales tampoco nos interesa, porque un cambio de orientación en ciertas clases puede dar lugar a confusiones con otras clases.

5.2 Diseño del modelo

A continuación vamos a exponer el diseño del modelo propuesto.

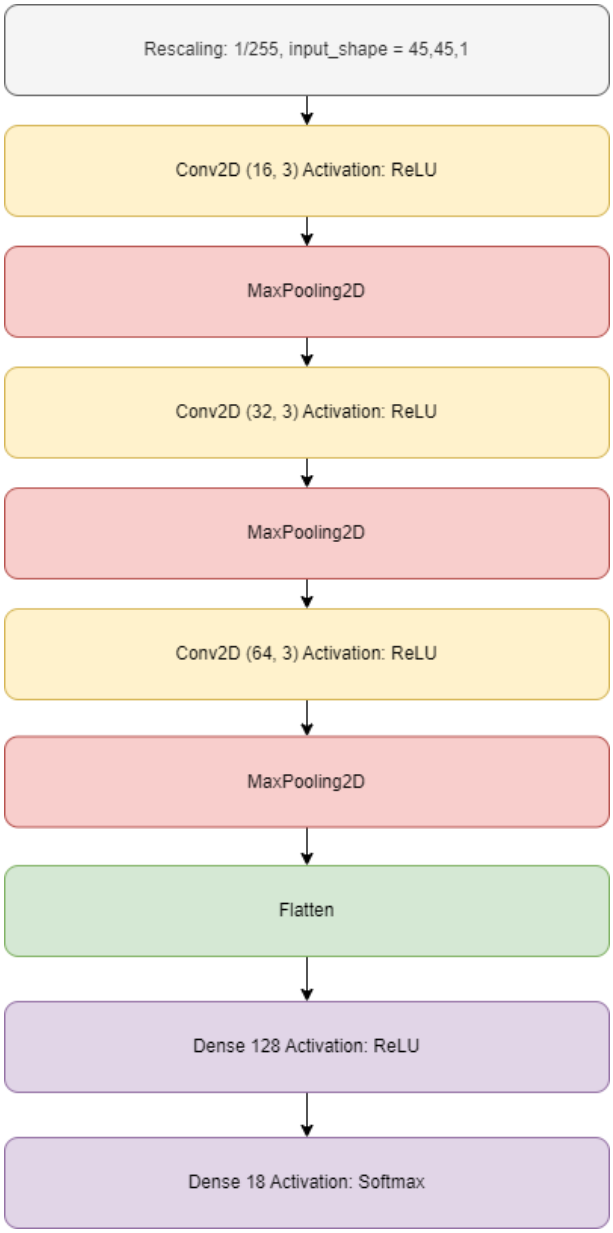

Figura 5.7: Definición de las capas del modelo propuesto.

Como se puede observar el modelo está constituido por una capa de input de reescalado, la cual reduce los valores de la imagen. Recordemos que están binarizados entre 0 y 255, dividiéndolos entre 255, para resultar en valores del 0 al 1 y poder agilizar las operaciones con números más pequeños.

Seguidamente podemos ver 3 bloques constituidos por dos capas. Una convolucional con activación por *ReLU*, y una *Max Pooling*. Después de estos 3 bloques convolucionales, tenemos una capa Flatten, para transformar las matrices de las capas convolucionales en un vector, el cual puede ser interpretado por las capas densas que vienen a continuación.

La primera, es una capa densa de 128 neuronas, con función de activación por *ReLU*. La última capa del modelo es una capa densa con activación por *Softmax*, para que el output de la red sea una distribución de probabilidad, y poder determinar la predicción de clase del modelo.

El diseño del modelo resultante es algo clásico, sin ninguna complejidad más allá de lo que podríamos esperar de un modelo CNN normal. Esto es así debido a los resultados obtenidos del mismo como ya veremos en el *entrenamiento* 5.3 que, dada su precisión, no surgió la necesidad de experimentar adicionalmente con el modelo.

Se inició con un modelo clásico y sencillo convolucional, y de ahí la intención era ampliar en función de las necesidades y los resultados, pero dada la alta precisión del modelo no se presentó la necesidad de ampliar el modelo.

Otro punto a destacar, es el uso de una CNN desde cero para abordar el problema, cuando hay disponibles modelos ya pre entrenados, a los cuales mediante *Transfer learning* [21] podríamos entrenar más rápido y con resultados excelentes. Realmente esta aproximación, aunque enriquecedora, nos parecía menos interesante que empezar con un modelo CNN desde cero, y afrontar las complejidades que este pudiera presentar.

El optimizador empleado es *Adam*, dado que es uno de los más famosos y utilizados para tratar con CNNs. Como función de pérdida *Sparse Categorical Crossentropy* ha sido utilizada, por su popularidad en casos de clasificación múltiple como es el nuestro.

5.3 Entrenamiento

Para el entrenamiento, se ha dividido el conjunto de datos en un 80 % para entrenamiento y el 20 % para validación. Con una cantidad de epochs totales de 5, y un tamaño de batch de 256. En la tabla 5.3 podemos ver la precisión y pérdida del modelo, y en la gráfica se puede apreciar, como no se produce *overfitting*, por lo cual, el modelo, no está sobre entrenado para la tarea, pudiéndose reentrenar con nuevas clases de caracteres y nuevas muestras de las clases ya existentes.

Train Acc	Val Acc	Train Loss	Val Loss
0.9915	0.9917	0.0245	0.0245

Tabla 5.3: Tasas de pérdida y precisión del modelo.

5.4 Adaptación de imagen para la ejecución de la inferencia

Esta sección abarca todas las técnicas y funciones que son empleadas para el tratamiento previo de la imagen, como puedan ser la detección de bordes, eliminación de ruido, y segmentación. Todo con el fin de extraer los caracteres, en una serie de ROIs con un tamaño de 45 x 45 píxeles, preparados para alimentarlos al modelo y extraer las predicciones.

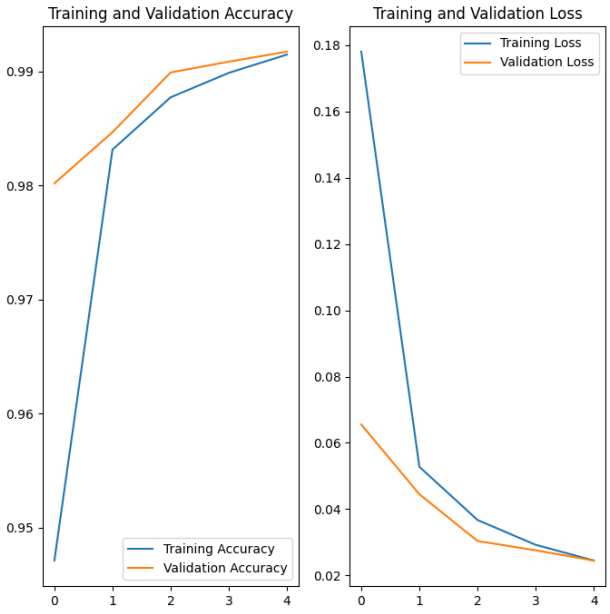

Figura 5.8: Gráficas de pérdida y precisión del modelo, aunque con pocas epochs es menos visible, ambas gráficas convergen.

5.4.1 Detección de bordes

El mayor problema inicial, consistía en el hecho de que alimentar las ecuaciones directamente al modelo no era algo plausible. Primero, no podemos hacer clases de cada ecuación y resultado, y segundo, detectar la ecuación mediante únicamente el modelo, delegando la extracción de características al mismo, lo dotaría de una complejidad innecesaria, sin asegurarnos buenos resultados.

Por lo que surge la necesidad de detectar en la imagen, lo que puede ser o no, un carácter, y la manera de suplir esta necesidad es mediante detección de bordes. De esta manera sabemos que en un papel en blanco, nuestro entorno de imagen ideal donde apenas podemos encontrar ruido, todo borde que detecte puede ser un carácter, el cual después será filtrado.

Para la detección y extracción de bordes hemos utilizado la función *Canny()*, disponible en la librería de *OpenCV*. Mediante esta función podemos detectar todos los bordes presentes en la imagen 5.9.

Después, mediante la función *findContours()*, podemos extraer las posiciones de todos esos contornos en la imagen, los cuales ordenaremos, y tomando los puntos más extremos de estos contornos, trazaremos un ROI que contenga los contornos representando un posible carácter. De esta manera aislamos todos los posibles caracteres dentro de una serie de ROIs ordenados de izquierda-arriba a derecha-abajo de la imagen.

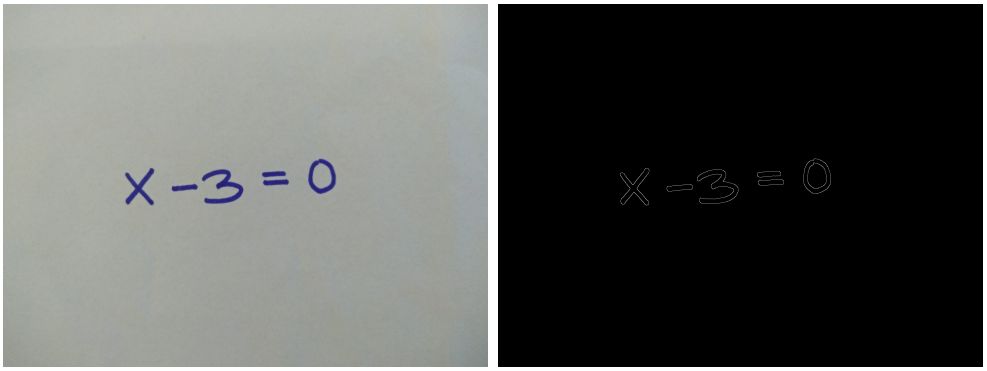

Figura 5.9: A la izquierda la imagen original, a la derecha después de aplicar el algoritmo de detección Canny.

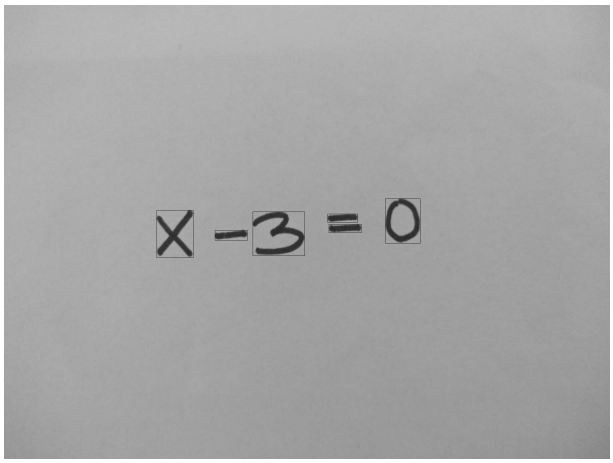

Figura 5.10: ROIs de posibles localizaciones de caracteres

5.4.2 Eliminación de ruido

Dependiendo de la calidad de la luz en el momento de tomar la imagen, nos podemos encontrar con situaciones donde se produce una gran cantidad de ruido, ya pueda ser por diferencia en sombras, calidad del papel, etc.

Durante el proceso de detección mediante el uso de *desenfoques Gausianos*, un gran número de veces no es necesario aplicar más algoritmos de eliminación de ruido. Pero por ejemplo en el caso del papel reciclado puede haber una serie de puntos propios del papel, que pueden producir una inmensa cantidad de ruido, y alterar la detección.

Recordemos que extraer caracteres falsos puede resultar fatal, dado que ya no estamos interpretando la ecuación original. Es por ello, que para el caso de elementos pequeños se ha desarrollado un pequeño algoritmo, el cual compara el tamaño del ROI extraído, y en caso de que este sea más que pequeño que el umbral proporcionado a la función, es eliminado de la lista de ROIs extraídos.

```
def remove_small_boxes(b_list, min_w=15, min_h=15):

    ret = []

    for box in b_list:
        if box[2] >= min_w or box[3] >= min_h:
            ret.append(box)

    return ret
```

Listing 5.1: Función para eliminar los ROIs a partir de un umbral determinado

El problema que presenta este algoritmo es el hecho de que, en el caso de que las características que nos interese extraer de la imagen sean realmente pequeñas, este algoritmo las eliminaría de la lista. Dado este problema, también se pensó en trazar una media del tamaño de todos los ROIs, y eliminar aquellos que se alejasen de la media, arriba o abajo, por exceso de un umbral. Pero el problema que plantea es que, si la imagen presenta un alto número de ROIs pequeños por una gran cantidad de ruido, la media priorizaría el ruido al poseer una mayor número de ROIs, eliminando la ecuación.

5.4.3 Agrupación de detecciones superpuestas

En los casos de caracteres con trazados algo difuminados y una mala calidad de luz, puede darse que el algoritmo de detección de borde saque muchos bordes de un único carácter real 5.11. Estas *piezas de carácter* no pueden administrarse al modelo, pero podemos saber las dimensiones de este carácter y su posición, si de alguna manera, logramos saber de toda la agrupación de *piezas de carácter*, cual ostenta el valor mayor de arriba-izquierda y cual el de abajo-derecha, pudiendo así crear un nuevo ROI que contenga todas estas piezas.

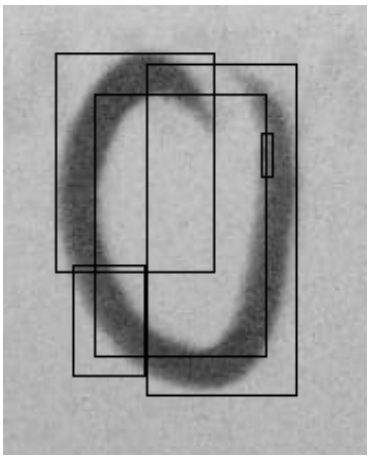

Figura 5.11: Carácter difuminado, se puede ver como el algoritmo de detección de bordes ha separado el carácter en muchos bordes independientes, cuyos ROIs están superpuestos entre sí.

Para ello hemos de detectar todos aquellos ROIs los cuales estén superpuestos unos de otros y, además detectar de todos ellos cuales son los más extremos.

Antes debemos definir que es exactamente un ROI, *Region Of Interest*, siendo este:

$$ROI = ((x_{tl}, y_{tl}), (x_{br}, y_{br}))$$

Un ROI es definido por un vector, el cual contiene dos puntos, siendo la representación de un punto, (x, y). Un punto superior izquierdo, *top-left*, el cual está representado mediante la nomenclatura tl, y un punto bajo derecho, *bottom-right*, el cual está representado por la nomenclatura br.

Además de esto debemos también recordar, que en la librería de OpenCV, la representación del eje de coordenadas, no es la misma que se presupone de normal, incrementado los valores de los ejes de coordenadas cuanto más derecha-abajo avancemos, y decrementando en el sentido opuesto.

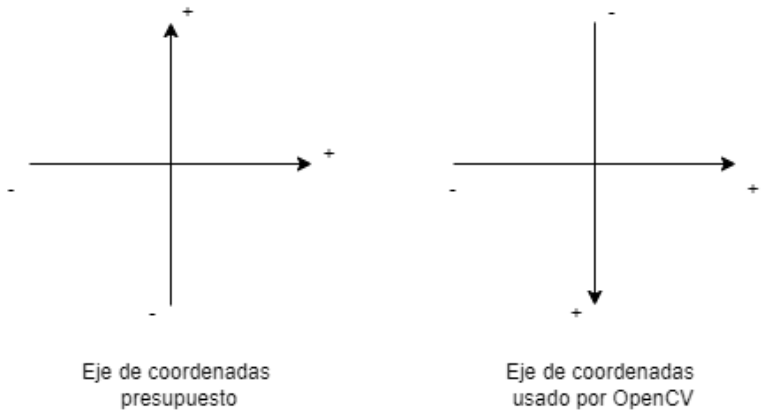

Figura 5.12: Diferencia entre los ejes de coordenadas usados comúnmente y el que usa *OpenCV*.

Expuestos estos hechos, la función para contener ambos ROIs:

$$mergedROI(u, v) = ((min(x_{tlu}, x_{tlv}), min(y_{tlu}, y_{tlv})), (max(x_{bru}, x_{brv}), max(y_{bru}, y_{brv})))$$

Función para juntar dos ROIs en un ROI contenedor de ambos, donde u y v son los dos ROIs que deseamos contener.

Definida la función, está es aplicada a todo el conjunto de ROIs, presentes en la imagen.

```
def group_overlapped_boxes(boxes):

    ret = []
    prev_box = [[0, 0], [0, 0]]

    for box in boxes:
        if overlap(prev_box, box):
            prev_box = merge_boxes(prev_box, box)

            if box == boxes[-1]:
                ret.append(prev_box)
        else:
            ret.append(prev_box)
            prev_box = box

            # If we are at the end and the two boxes
            # are not overlapped
            if box == boxes[-1]:
                ret.append(box)

    # Remove the first element that we added for the comparison
    ret.remove([[0, 0], [0, 0]])

    return ret
```

Listing 5.2: Función compara todas las posibles agrupaciones, y en el caso de que estén superpuestas agruparlas. Esta función presupone que la lista de ROIs está ordenada de izquierda-arriba a derecha-abajo.

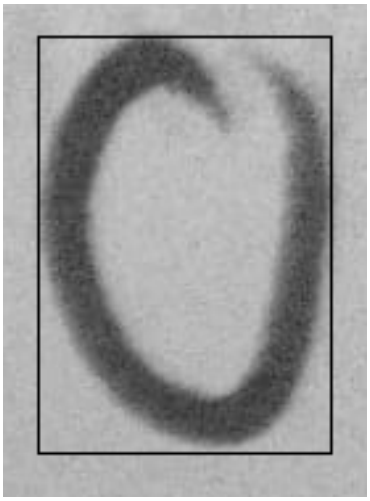

Figura 5.13: Carácter presentado anteriormente después de utilizar el algoritmo de agrupación, tenemos un único ROI preciso del carácter, preparado para la inferencia del modelo.

Por último destacar un problema originado del algoritmo, en caso donde un trazado de dos caracteres invada el ROI de otro carácter, aunque estos no estén en contacto en la imagen, dado el algoritmo de agrupación, y el hecho de que el área de los dos ROIs coinciden, el algoritmo los agrupara. Lo cual es un problema, y se recomienda que durante el proceso de escritura de la operación, se intente separar lo mejor posible los caracteres.

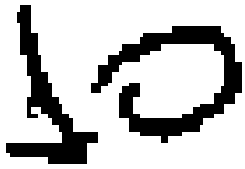

Figura 5.14: Fallo durante la agrupación, como se puede ver el el final de la base de *1* y el principio del *0* están dentro del ROI del carácter adyacente.

5.4.4 Segmentación de caracteres

Después de los algoritmos y procedimientos de preprocesado de imagen descritos anteriormente, nos quedamos con una lista de ROIs, los cuales definen los caracteres que han sido encontrados en la imagen, pero todavía no sabemos exactamente que caracteres son.

Estos ROIs son utilizados para crear sub-imágenes de la original, de un tamaño de 45 x 45 píxeles, binarizados. Estos son alimentados al modelo, para que este realice predicciones sobre las susodichas sub-imágenes.

Una vez terminada la inferencia o predicciones, queda la interpretación de las mismas, y la resolución de esa interpretación.

5.5 Interpretación del resultado de la inferencia

Una vez que la inferencia del modelo acaba, y recibimos las predicciones del mismo, debemos de realizar la interpretación de estas predicciones. Por ejemplo en el apartado, 5.1.2 y en la figura 5.6, se ha mencionado el hecho de porque se ha eliminado el carácter = del modelo, las razones siendo el hecho de su ausente detección, y el hecho de ser detectado como dos caracteres -.

Realmente no debemos entrenar el OCR para que reconozca caracteres completos, dado que algunos de estos están separados por espacios, los cuales la detección de bordes no es capaz de concebir que son parte del mismo carácter, véase el caso de =.

Así pues, aquí entra la interpretación del resultado de la inferencia. En este caso, si en la lista de caracteres se da la fortuna de recibir dos caracteres seguidos como -, resultando en *(... , -, -, ...)*, podemos presuponer que, dado que la lista de caracteres viene ordenada de izquierda a derecha, esos dos - no son dos caracteres -, sino un carácter =. Dado que la otra posibilidad es que sí que sean dos caracteres - independientes, no teniendo sentido la operación por parte del usuario, por tanto delegaremos el error al usuario.

Otra interpretación que también debemos hacer, es el caso de los *Productos*, los cuales fueron eliminados del modelo por una razón parecida a el caso de =. En este caso nosotros debemos predecir, por el contexto de la ecuación, donde ha de haber un producto. Para ello se dan las siguientes reglas las cuales aplicaremos a la función.

Carácter (	
Caso	Aplicar *Producto*
Número	Pre
x	Pre
)	Pre

Tabla 5.4: Caso del carácter (, Pre indica colocar un *Producto* antes del carácter y Post después del carácter.

Carácter x	
Caso	Aplicar *Producto*
Número	Pre & Post

Tabla 5.5: Caso del carácter x, Pre indica colocar un *Producto* antes del carácter y Post después del carácter.

Carácter)	
Caso	Aplicar *Producto*
Número	Post
x	Post

Tabla 5.6: Caso del carácter), Pre indica colocar un *Producto* antes del carácter y Post después del carácter.

Iterando a través de cada carácter de la función, y aplicado las normas estipuladas anteriormente, podríamos transformar una ecuación que recibimos de la siguiente manera:

$$2x(x+5) - -3$$

En esta ecuación lista para resolver:

$$2 * x * (x+5) = 3$$

5.6 Cálculo de la ecuación interpretada

Una vez ya se ha interpretado la ecuación, únicamente queda acomodar la ecuación al formato adecuado para resolver la ecuación mediante la librería *Sympy* 4.6. El proceso consiste en separar la ecuación en dos partes, las dos partes a cada lado del igual, y alimentarlas a la clase *Eq* para después ejecutar el método *solve()*, el cual recibe la ecuación y devuelve una lista con los resultados.

Estos resultados luego son dibujados en la imagen junto a todas los ROIs y devueltos al usuario.

5.7 Cliente / Servidor

La solución que hemos implementado, hace uso de una estructura de cliente/servidor. La razón, un servidor es una solución mucho más explotable y escalable para un uso de gran cantidad de usuarios. Además, nos permite poder actualizar la solución sin necesidad de intervención del usuario, como pudieran ser actualizaciones.

En nuestro caso hemos desarrollado un pequeño cliente en *Python*, el cual ataca a una API REST que actúa como contrato al servidor. El cliente ha de conocer la IP y el puerto en el cual está escuchando el servidor, si no, no será capaz de hablar con él.

Además tanto el cliente como el servidor poseen unos archivos de configuración, *client_ con -figuration.py* y *config.py*, en los cuales se detallan constantes de la naturaleza de: rutas donde guardar imágenes, ruta donde está localizado el conjunto de datos, ruta donde está localizado el modelo, etc.

- **Cliente**: Es una aplicación muy sencilla, que como se ha expuesto está escrita en *Python*, y únicamente ataca al servidor, subiendo una imagen guardada previamente en el equipo donde se está ejecutando el cliente. Después de subir la imagen espera la respuesta del servidor.

- **Servidor**: El servidor es una abstracción del equipo donde se está ejecutando la aplicación REST, el cual recibe la petición mediante la misma aplicación y devuelve al cliente el recurso o un mensaje de error.

5.7.1 API REST

La API REST está definida en el fichero *REST_ server.py*, desde el cual se ejecuta la API REST. Para el desarrollo de la misma se ha aprovechado la librería Flask 4.3, únicamente dispone de un *Endpoint*, el cual está definido bajo la URL de *https://(IP Servidor)/equation*.

Este *Endpoint* en cuanto recibe una petición, procesa la imagen recibida, la guarda y llama al proceso de inferencia, una vez recibe resultado, envía una respuesta al cliente, con un mensaje o la propia imagen ya procesada, en función del éxito de la lógica.

5.8 Estructura de la aplicación

Vamos a describir la estructura de la aplicación desde dos puntos de vista distintos, uno desde el punto de vista de flujo de ejecución y otro desde la separación MVP.

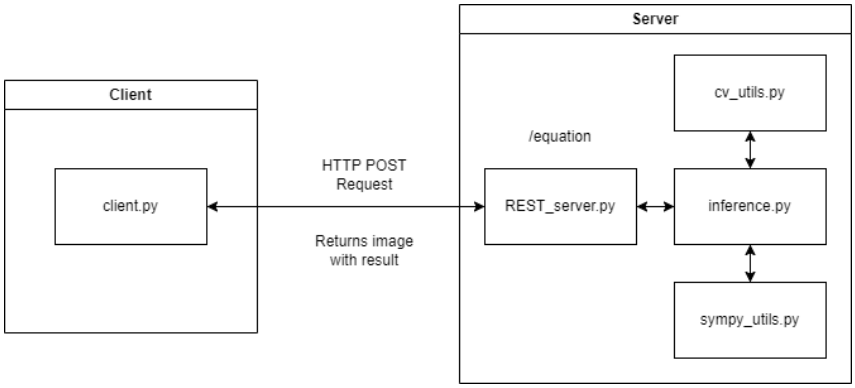

Figura 5.15: Flujo de la aplicación. Se puede ver como el cliente hace una petición POST al servidor, solicitando el cálculo de la imagen que va dentro de la petición, esta, llama al proceso de inferencia y una vez procesado, devuelve una imagen con el resultado.

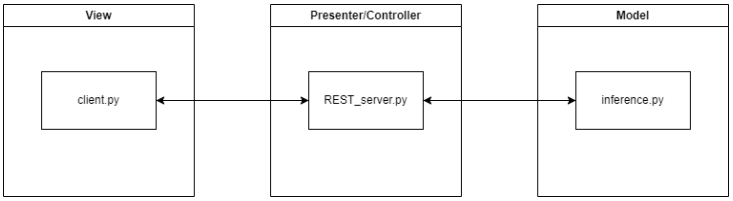

Figura 5.16: Implementación del patrón MVP, como podemos ver, el cliente únicamente habla con el *REST_ server.py*, el cual contacta con el modelo, nunca se produce una comunicación entre la vista y el modelo.

La utilización de estas estructuras y patrones se debe principalmente, al hecho de que son de los más usados a día de hoy para aplicaciones web, dada su escalabilidad, desacople de dependencias y facilidad de implementación. Además, dado que la aplicación está detrás de una API REST, cualquier cliente que ataque a ese *Endpoint* con una imagen va a ser capaz de procesarla, sin necesidad de la utilización de nuestro cliente.

6 Resultados

En este capítulo vamos a analizar los datos previamente expuestos, como la tasa de acierto del modelo, y nuevos datos, con el fin de valorar el resultado final de la aplicación. Esta valoración va a ser expuesta no solo con los datos que se van a aportar, además también se va a realizar una comparativa con los resultados presentados en el estado del arte.

6.1 Modelo

Como hemos mencionado antes en 5.3, el modelo presenta una precisión del 0.99, y una tasa de pérdida de 0.0245. Aunque no podamos hacer comparaciones justas con el primer modelo, dado que el modelo propuesto por *Shinde* [7] no segmenta previamente la ecuación en sí mediante el uso de visión por computador, el resultado final si que es comparable, y será abordado en la siguiente sección. Sin embargo sí que podemos hacer la comparación con el documento publicado por *Shakya* [8], en cuyo caso nuestras métricas son más precisas que los resultados presentados por ellos, los cuales siguen un procedimiento de segmentación previa mediante Visión por computador igual que nuestro caso.

6.2 Precisión del resultado final

Con todos los pasos descritos anteriormente, vamos a mostrar distintas imágenes con resultados interesantes.

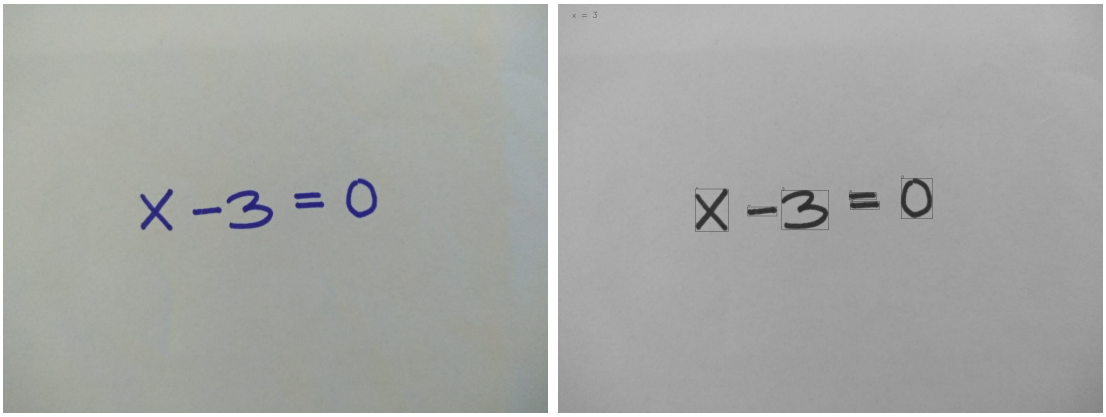

Figura 6.1: A la izquierda la imagen original, a la derecha imagen procesada por la solución, este caso es el que denominamos una ecuación sencilla.

Este caso es uno de los casos más básicos, el cual hemos ido mostrando varias veces durante el documento. Es una simple ecuación de primer grado, como todas las que vamos a mostrar, donde se puede observar las detecciones que hemos realizado mediante los algoritmos de detección de bordes. La clasificación de estos caracteres, según el modelo que hemos entrenado, se puede apreciar arriba a la izquierda de cada carácter. El resultado final de la ecuación está arriba a la izquierda en la imagen.

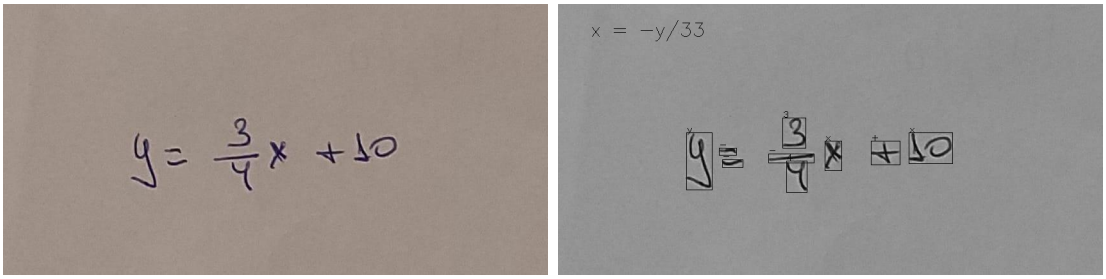

Figura 6.2: A la izquierda la imagen original, a la derecha imagen procesada por la solución, este caso tiene unas características que nuestra solución no es capaz de abordar, como es el caso de la fracción.

Este caso es interesante por tres razones. Primera aunque solo analiza ecuaciones con incógnita x, es capaz de procesar las ecuaciones que presentan una variable y, simplificándolas en el resultado sin resolver la incógnita y. La segunda otra razón, es el caso del 10 que hemos mostrado previamente en la figura 5.14, donde el algoritmo de agrupación ha juntado ambos caracteres, y la ecuación ha dado un resultado erróneo debido a ello. La tercera razón es el caso de la fracción, como podemos observar, con nuestro resolutor, no somos capaces de identificar fracciones, por lo que el sistema lo interpreta como un -. Para evitar esto, habría que añadir más trabajo a la solución, incorporando una rutina que en caso de detectar el carácter -, ver cuales son las longitudes del mismo y su posición relativa respecto a otros caracteres, viendo así, si estos están en posiciones superiores o inferiores al carácter -.

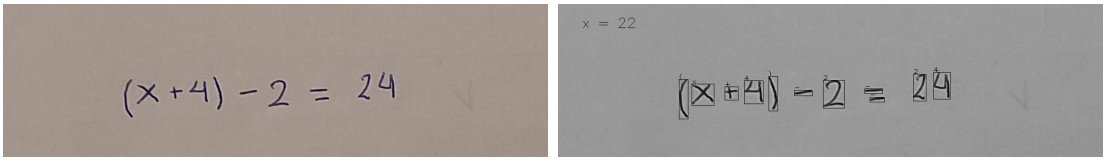

Figura 6.3: A la izquierda la imagen original, a la derecha imagen procesada por la solución, este caso de ecuación es algo más compleja.

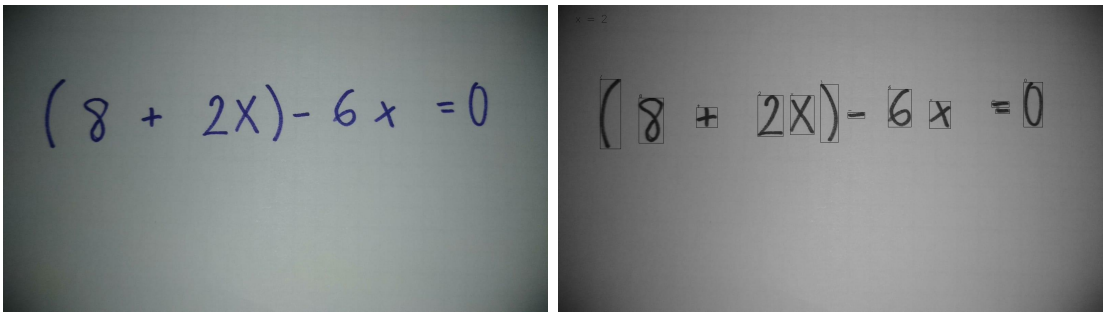

Figura 6.4: A la izquierda la imagen original, a la derecha imagen procesada por la solución, la ecuación es algo más compleja que la anterior.

Las figuras 6.3 y 6.4, únicamente muestran casos de ecuaciones algo más complejas que la figura 6.1, donde el resultado es correcto.

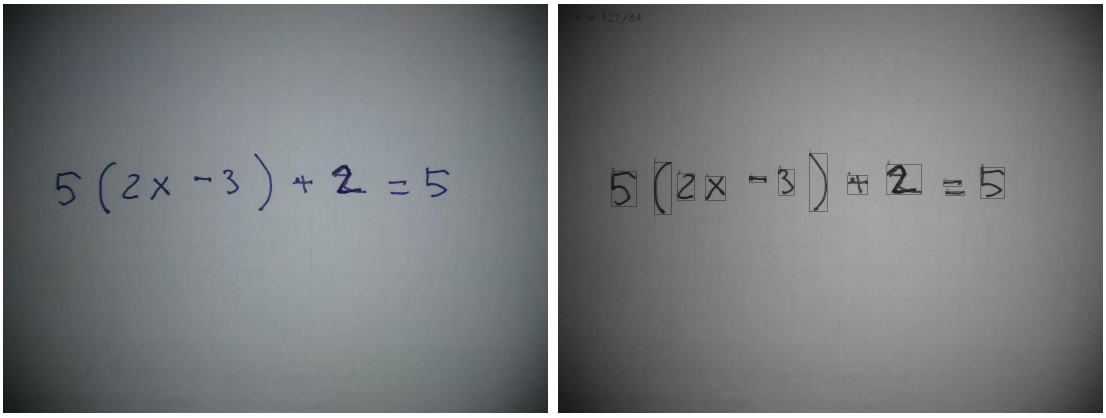

Figura 6.5: A la izquierda la imagen original, a la derecha imagen procesada por la solución, este caso se puede ver un error de clasificación en el carácter *+*.

Este caso muestra el fallo de resolución, por una mala escritura por parte del usuario, es el caso del carácter *+* entre los caracteres *)* y *2*, este carácter tiene un punto encima el cual lo hace parecer un *4*. En este caso, la culpa no es tanto del modelo por no reconocerlo correctamente, más que del usuario por la forma inadecuada de plasmar el carácter.

Como último punto a destacar, la precisión de la solución respecto a una ecuación dada, no es algo que podamos medir de manera constante por la cantidad de variables que se escapan, como pueda ser, desenfoque de la imagen, irregularidad del papel o fondo, errores por parte del usuario durante el trazado, etc. Pero, si que es de interés indicar que, la precisión de la solución en condiciones perfectas de imagen no es la misma que la

precisión del modelo a la hora de resolver caracteres individuales, dado que el fallo en la clasificación de un único carácter, en la mayoría de casos, supone una resolución errónea. Así que nos gustaría presentar lo que puede ser una aproximación a la precisión de la solución, siendo esta:

$$accuracy(n) = accmodel^n$$

Donde *accmodel* es la precisión del modelo, *accuracy* la precisión de la solución, y n el número de trazados a clasificar.

7 Conclusiones

Tras analizar los resultados de la solución propuesta podemos determinar los siguientes puntos:

- Respecto a los documentos académicos, los resultados de precisión obtenidos en el modelo es superior a ambos casos. Aunque en uno de ellos no es comparable. En el caso de *Shakya* [8], los valores de precisión y pérdidas son superiores. Por ello los resultados obtenidos van a la par con los documentos aportados en el *estado del arte* 2.

- En cuanto al reconocimiento, uno de los productos comerciales que hemos apuntado anteriormente, *Mathpix OCR* 2, a diferencia de nuestra solución, sí que es capaz de reconocer patrones, como integrales, fracciones, etc. También cabe destacar que, nuestra solución no es capaz de detectar esos patrones, al igual que el resto de soluciones presentadas de ámbito académico. La posibilidad del reconocimiento de estos patrones es un objetivo plausible, pero aumentaría la complejidad de este documento y los tiempos de desarrollo, por lo que no se ha abordado.

- El uso de preprocesado de imágenes para preparar una serie de ROIs listos para ser alimentados al modelo, parece ser la aproximación correcta, dada la mejora en la tasa de precisión respecto a *Shinde* [7].

- La aproximación propuesta es muy susceptible a ruido y fallos, dado que uno de estos compromete en gran medida el resultado. En la mayoría de casos donde se produce un error o ruido sin tratar, el resultado es erróneo, dado que no tiene manera de rescatar esos fallos.

- Las soluciones para la interpretación de caracteres matemáticos, basadas en la detección de bordes para extraer los posibles caracteres, deberían de enfocarse, no en la detección de caracteres, si no en la detección de trazados. Esto se debe a que mediante el análisis de los trazados y su contexto, es una manera más precisa de determinar caracteres.

- Las CNN parecen ser la mejor solución para la predicción de caracteres. Se puede ver reflejado también en el *estado del arte* 2 dado que esta está mencionado en ambos documentos.

- La incorporación de un OCR para reconocer operaciones matemáticas, a una aplicación como es *Wolframalpha* [6], podría ser una herramienta muy potente a manos de un resolutor de tantas capacidades.

8 Trabajos futuros

Los trabajos futuros se pueden dividir en los siguientes puntos de mejora:

- El desarrollo para más tipos de operaciones matemáticas, como puedan ser ecuaciones de más grados, operaciones trigonométricas, integrales, etc.
- Desarrollar una aplicación móvil, para poder tomar directamente fotos y atacar a la API sin necesidad de una consola.
- Añadir un mayor número de clases para reconocer nuevos caracteres
- Mejorar el preprocesado de imagen, con el fin de conseguir resultados más precisos.

Bibliografía

[1] Google. "Google Document AI." (), dirección: https://cloud.google.com/document-ai (vid. págs. 1, 4).

[2] IBM. "IBM Datacap." (), dirección: https://www.ibm.com/es-es/products/data-capture-and-imaging (vid. pág. 4).

[3] Adobe. "Adobe Acrobat." (), dirección: https://acrobat.adobe.com (vid. pág. 4).

[4] "Mathpix." (), dirección: https://mathpix.com/ocr (vid. págs. 4, 7).

[5] "Tesseract." (), dirección: https://github.com/tesseract-ocr/tesseract (vid. pág. 4).

[6] W. Research. "WolframAlpha." (), dirección: https://www.wolframalpha.com/ (vid. págs. 4, 42).

[7] R. Shinde, O. Dherange, R. Gavhane, H. Koul y N. Patil, "Handwritten mathematical equation solver," *International Journal of Engineering Applied Sciences and Technology (IJEAST)*, vol. 6, n.º 10, págs. 146-149, 2022 (vid. págs. 4, 37, 41).

[8] D. Shakya, G. K. Shrestha, S. Juwal, S. R. Karmacharya, E. A. Karn y E. B. Chawal, "HANDWRITTEN LINEAR EQUATION SOLVER," inf. téc., 2022 (vid. págs. 5, 7, 37, 41).

[9] S. Saha. "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way." (2018), dirección: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (vid. págs. 14, 15).

[10] OpenCV. "Filtros de suavizado." (), dirección: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html (vid. pág. 17).

[11] F. P. of Computer Vision. "Edge Detection Using Gradients | Edge Detection." (2021), dirección: https://www.youtube.com/watch?v=lOEBsQodtEQ (vid. pág. 17).

[12] Google. "Python." (), dirección: https://docs.python.org/3.10/ (vid. pág. 18).

[13] "Flask." (), dirección: https://flask.palletsprojects.com/en/3.0.x/ (vid. pág. 19).

[14] I. Corporation. "OpenCV." (), dirección: https://docs.opencv.org/4.x/index.html (vid. pág. 19).

[15] "Imutils." (), dirección: https://github.com/PyImageSearch/imutils (vid. pág. 19).

[16] "Sympy." (), dirección: https://docs.sympy.org/latest/index.html (vid. pág. 19).

[17] "Keras." (), dirección: https://keras.io/ (vid. pág. 20).

[18] "NumPy Documentation." (), dirección: https://numpy.org/doc/ (vid. pág. 20).

[19] "Handwritten math symbols dataset." (2017), dirección: https://www.kaggle.com/datasets/xainano/handwrittenmathsymbols (vid. pág. 21).

[20] OpenCV. "Eroding and Dilating." (), dirección: https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html (vid. pág. 24).

[21] D. Scientest. "¿Qué es el Transfer Learning?" (), dirección: https://datascientest.com/es/que-es-el-transfer-learning (vid. pág. 27).