Diseño e implementación de un generador de código fuente a partir de diagramas de flujo manuscritos

Nombre Estudiante
Juan Antonio Lagos Carrera

Nombre del Programa
Sidekick-S

Área de trabajo final
Aplicaciones multimedia de nueva generación

Nombre Tutor/a de TF
David García Solórzano

Fecha Entrega
22/01/2023

Universitat Oberta de Catalunya

Quiero agradecer a todos los profesores que me han llevado hasta este punto de mi vida. En especial a Meritxell y Dani por ser mis guías espirituales, a Bene por no tirar nunca la toalla conmigo (aunque te lo puse difícil en alguna ocasión) y a David por aceptarme como tu padawan y mostrarme el camino hacia la meta.

También te agradezco a ti, Carlos, mi pollito, que nos hayamos mantenido juntos en esta aventura. Lo hemos conseguido.

Pero a quien más tengo que agradecer es a mi familia ya que sin ellos nunca lo habría conseguido. Víctor, Mónica, gracias por estar siempre cuando os he necesitado. Papá, mamá, espero que estéis orgullosos de mí, un poco mayor, pero lo he conseguido. Julio, siempre serás mi canijo, gracias por ser cómplice de mis nuevas aventuras, me esforzaré por seguir siendo un buen ejemplo para ti. Antonio, Carlos, sois los motores de papá, espero poder motivaros tanto como lo hacéis vosotros conmigo. Andrea, mi alma gemela, siempre me has apoyado y nunca has dejado de creer en mí, aunque yo sí dejase de hacerlo, te amo.

A todos, gracias.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Diseño e implementación de un generador de código fuente a partir de diagramas de flujo manuscritos*
Nombre del autor:	*Juan Antonio Lagos Carrera*
Nombre del consultor/a:	*David García Solórzano*
Fecha de entrega:	*01/2023*
Titulación o programa:	*Grado de Ingeniería de Tecnologías y Servicios de Telecomunicación*
Área del Trabajo Final:	*Aplicaciones multimedia de nueva generación*
Idioma del trabajo:	*Castellano*
Palabras clave	*Program synthesis, OCR, computer vision, ANTLR4, CS1 courses, higher education*

Resumen del Trabajo

En este trabajo se ha desarrollado una aplicación que tiene como finalidad auxiliar a los estudiantes de estudios STEM en sus inicios en la programación. El programa recibe un diagrama de flujo manuscrito para devolver el código fuente en el lenguaje de programación que el estudiante seleccione de entre los disponibles.

La aplicación funciona en cuatro pasos:

1. Inicialmente el usuario debe escribir un diagrama de flujo en un papel en blanco. Posteriormente debe escanearlo o sacarle una fotografía y hacer que sea el input del programa.
2. En una primera iteración, el programa identifica todos los elementos dibujados en la imagen que no son texto a través de un proceso de detección de objetos (1), los clasifica y ordena para detectar el flujo dentro del diagrama. En la misma iteración, el programa reconoce los textos escritos mediante un mecanismo OCR (2) y los ubica en las figuras del diagrama para luego ser tratados.
3. Después de la detección, clasificación y ordenación de todos los elementos escritos, el programa genera un pseudocódigo que emplea posteriormente.
4. Finalmente, con el pseudocódigo generado el programa devuelve el código fuente del diagrama en el lenguaje que previamente ha seleccionado el estudiante.

Esto ayudará al estudiante aportándole el código fuente para que pueda comparar con el código que él mismo realice, permitiendo autoevaluarse.

Abstract

The main purpose of this bachelor's thesis, is to develop an application which main goal is to help STEM students in their begginings of their programming studies. The program takes a handwritten flowchart in order to return the source code on the desired language.

The program works in four steps:

1. Firstly, the user must draw a flowchart in a blank paper, take a picture or scan it and use it as the input for the program.
2. Then, the program identifies all the drawing elements in the picture relatives to the flowchart, through an object detection process (1) and sort them in order to see the flow inside the program. Next, the program gets all the handwritten text through an OCR (2) system and put it into flowchart figures which has been detected previously to be processed.
3. After that, the application generates a pseudocode with all those elements to be used in the next step.
4. Finally, it returns the source code in the language that the user has been chosen before using the pseudocode generated before.

This will help student giving them the source code in order to make an auto evaluation.

Índice

Lista de figuras

1. Introducción

El software desarrollado en este TFG recibe un diagrama de flujo y devuelve el código fuente en algún lenguaje de programación que el usuario pueda seleccionar de entre los disponibles (Python y Java).

La aplicación funcionará en cuatro pasos:

1. Inicialmente el usuario dibuja un diagrama de flujo en un papel en blanco. Posteriormente lo escanea o le saca una fotografía y la emplea como input para el programa.
2. En una primera iteración, el programa identifica todos los elementos dibujados en la imagen que no sean texto a través de un proceso de detección de objetos (1), los clasifica y ordena para detectar el flujo dentro del diagrama. En la misma iteración, el programa reconoce los textos escritos mediante un mecanismo OCR (2) y los ubica en sus respectivos lugares para luego ser tratados.
3. Después de la detección, clasificación y ordenación de todos los elementos escritos, el programa genera un pseudocódigo para ser evaluado por ANTLR4 (3).
4. Finalmente, con el pseudocódigo generado el programa devuelve el código fuente del diagrama.

1.1. Contexto y justificación del Trabajo

El software desarrollado en este TFG puede ser de ayuda en el ámbito académico para los estudiantes que inician sus estudios de programación, ya que esta primera aproximación es el momento más complejo para un estudiante. Este software puede ayudarlos a generar ellos mismos sus propios problemas, solucionarlos y realizar una autoevaluación, con lo que podrán mejorar su rendimiento académico.

Hay muchos estudios que se centran en los primeros cursos de programación ya que en este momento es cuando existe un mayor número de abandono y se centran en cómo solucionarlo (4) o detectarlo precozmente (5) para evitarlo. Entre sus causas están el no entender la lógica de un ordenador y como los procesadores ejecutan las instrucciones y como es algo abstracto, genera frustración en el alumno. Este problema se puede abordar por multitud de frentes como realizar una tutorización del estudiante (6). Personalmente, considero que es la mejor opción puesto que en lo personal es como he logrado “romper” ese primer muro, pero también entiendo que no es posible realizarlo a gran escala y de ahí el objetivo de este TFG. El potencial que ofrece este TFG no solamente se ciñe al programa que se entregue con la

memoria, considero que lo más importante son las implementaciones futuras que se pueden realizar a través de él para ayudar a estudiantes de diferentes niveles de la forma en que el profesorado precise.

1.2. Objetivos del Trabajo

Para que el programa consiga su finalidad, ha sido necesario abordar los siguientes objetivos:

- Generar un sistema de detección de objetos en imágenes.
- Desarrollar e implementar un sistema OCR de diseño propio, así como implementar sistemas OCR de terceros y en la nube.
- Plantear un algoritmo de ordenación para los bloques (tipo de objeto en el diagrama y texto escrito a mano) detectados por los dos sistemas anteriores.
- En base a los bloques detectados, generar un pseudocódigo del diagrama de flujo.
- Empleando un parser generator (7), generar código fuente a partir del pseudocódigo.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

Uno de los Objetivos de Desarrollo Sostenible (8) de la Agenda 2030 (9) de las Naciones Unidas es la educación de calidad. Este TFG puede ayudar a promover la educación en materia de programación sin la necesidad de la tutorización directa de un profesor. De esta manera, puede ser empleado para fines autodidactas o en cursos MOOC, ayudando a los estudiantes en sus estudios STEM.

1.4. Enfoque y método seguido

Durante el proceso de recopilación del estado del arte se ha podido observar cómo hay diversas opciones para cada uno de los diferentes bloques del software, por lo que se han adoptado estrategias diferentes para cada uno de ellos.

Para el bloque de detección de figuras se ha empleado un modelo de reconocimiento de objetos de PyTorch (10) llamado YOLOv5 (11) al que hay que entrenar. Asimismo, se ha decidido usar este modelo porque con las demás opciones era preciso generar los modelos para entrenarlos

posteriormente. De esta forma, únicamente se ha tenido que buscar una dataset, incorporar una serie de objetos nuevos y con ella entrenar el modelo.

El bloque de clasificación ha sido desarrollado íntegramente desde cero ya que es algo muy concreto de este software. Por otra parte, en el bloque OCR se ha desarrollado un sistema propio con OpenCV (12) empleando la detección de contornos. Pero el software también da la posibilidad al usuario de emplear dos sistemas OCR para Python ya entrenados, así como una solución en la nube mediante el sistema OCR de Azure (13), ya que este último es con el que mejores resultados se obtiene.

Finalmente, para generar finalmente el código fuente se emplea un parser generator ya que mejora la escalabilidad del software facilitando la incorporación de nuevos lenguajes de programación en la salida del mismo.

1.5. Planificación del Trabajo

Inicialmente ha sido necesario un equipo con una tarjeta gráfica que permitiese entrenar el modelo de detección de objetos, siendo empleada una RTX 3060. También se ha necesitado acceso al portal de Azure para emplear su servicio de visión por computador en el reconocimiento OCR. En el resto del proyecto no ha habido ninguna necesidad especial más, con la salvedad de no poder emplear un equipo Apple con arquitectura Apple Silicon por la incompatibilidad del mismo con Tensorflow (14).

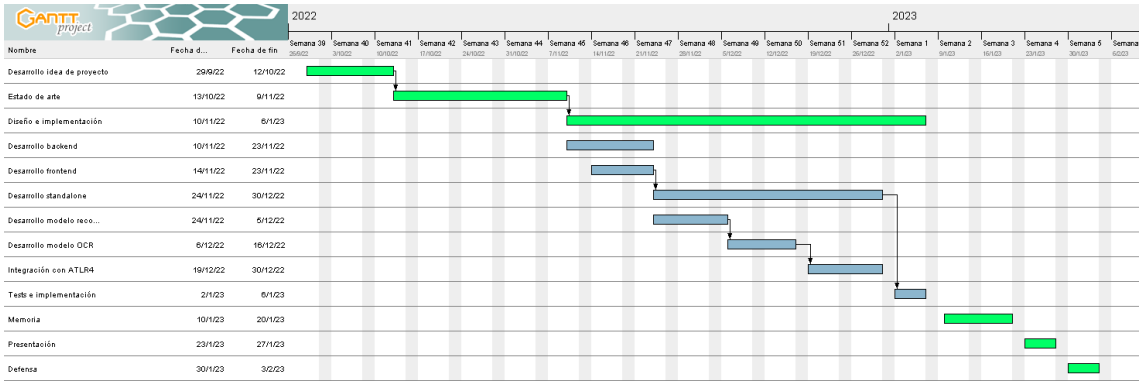


Figura 1: Diagrama Gantt para la planificación del trabajo

1.6. Breve sumario de productos obtenidos

Tras finalizar el proyecto se ha obtenido un programa capaz de obtener el código fuente a partir de un diagrama de flujo manuscrito.

Además de adjuntarse junto a esta memoria el código fuente, se puede encontrar en el siguiente repositorio:

https://github.com/alochimpasplum/Sidekick-S-TFG

1.7. Breve descripción de los otros capítulos de la memoria

- **Capítulo 2**: En este capítulo se detalla el proceso que se ha seguido para obtener un modelo de reconocimiento de los objetos de los diagramas de flujo manuscritos.
- **Capítulo 3**: En este capítulo se procede a explicar la implementación de varios sistemas OCR, tanto bibliotecas ya existentes, como el empleo de una API de Azure o el desarrollo de un sistema propio.
- **Capítulo 4**: En este capítulo se explica el algoritmo ideado para la ordenación de los elementos del diagrama de flujo.
- **Capítulo 5**: En este capítulo se expone el procedimiento para preparar los datos extraídos del diagrama de flujo para su interpretación por ANTLR4, así como el proceso que este realiza para la generación del código fuente.
- **Capítulo 6**: En este capítulo se indica el flujo general del programa, así como elementos de la arquitectura de software empleados.
- **Capítulo 7**: En este capítulo se expone el resultado final del programa, así como su interfaz.
- **Capítulo 8**: En este capítulo se expresan las conclusiones extraídas durante el desarrollo del TFG, así como las intenciones futuras sobre el software generado.
- **Capítulo 9**: Glosario de acrónimos y tecnicismos.
- **Capítulo 10**: Bibliografía.
- **Capítulo 11**: Anexos incluidos para ampliar información sobre el desarrollo del TFG.

2. Reconocimiento de objetos en imágenes

La primera acción que realiza el programa es la detección de los objetos dibujados en el diagrama de flujo. Para ello, se ha buscado una dataset que tuviese una cantidad de elementos suficiente para adiestrar un modelo de detección de objetos en imágenes.

La que se ha empleado ha sido una modificación de una dataset realizada por los estudiantes del UPIIZ (15) que se puede encontrar en Kaggle (16). Durante el proceso de entreno de la máquina de detección de objetos se ha notado que los objetos “Arrow line right” y “Arrow line left” se confundían a la hora de realizar la detección de los mismos, por lo que se ha tenido que hacer una modificación de esta dataset. Esta modificación ha consistido en agregar un nuevo objeto dentro de la dataset. Este objeto es el “pointer”, el puntero de la flecha, que posteriormente será empleado por el algoritmo de ordenación para decidir la orientación de las flechas, eliminando así el problema de confusión de las flechas derecha e izquierda.

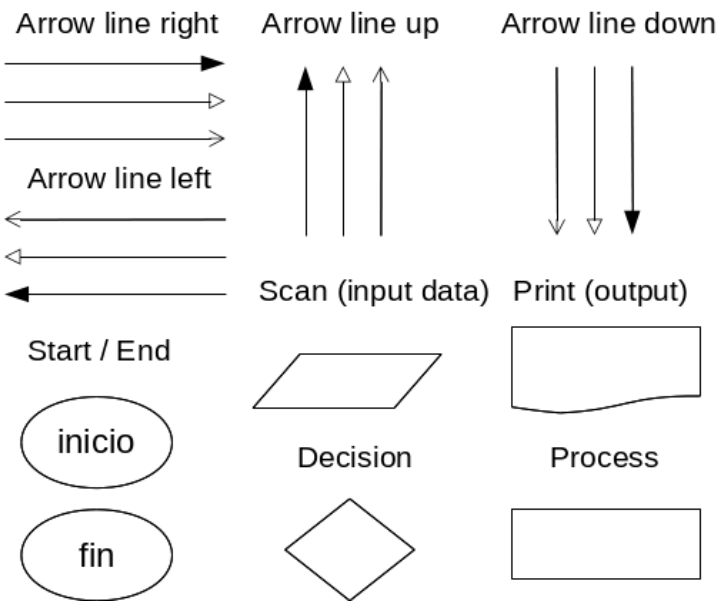


Figura 2: Figuras aceptadas por el modelo de detección de objetos

Para realizar la citada modificación, se han valorado varias opciones para ralizar el reetiquetado de las imágenes de la dataset. La primera opción que a valorar ha sido la de emplear el software labelImg (17), aunque finalmente se ha descartado porque el proceso de etiquetado resultaba tedioso. Finalmente, la opción que ha sido empleada ha sido el portal Roboflow (18) ya que permite realizar un etiquetado más ágil, así como su exportación a diferentes sistemas de detección de objetos.

Finalmente, la dataset empleada se puede encontrar en Roboflow (19) y contiene los objetos descritos en la Figura 2 con las cantidades indicadas en la Figura 3.

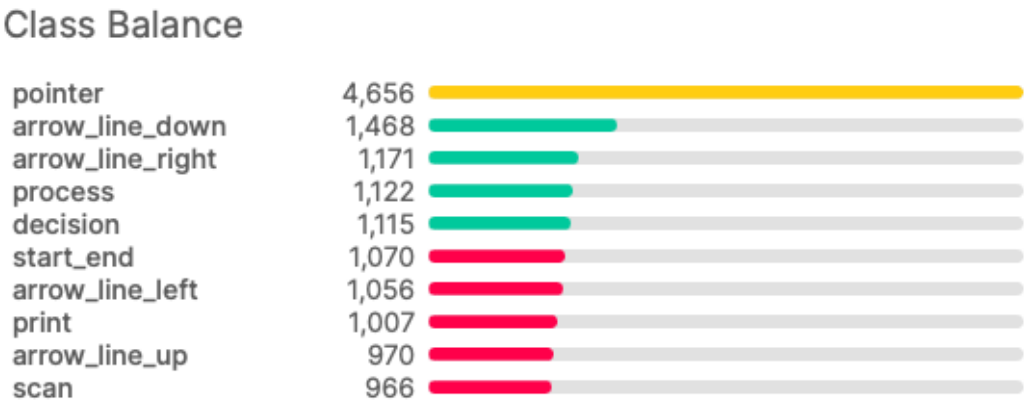


Figura 3: Número de figuras en la dataset de reconocimiento de objetos

2.1. TensorFlow Lite

Inicialmente se planteó la posibilidad de emplear la librería de Tensorflow Lite (20) por su bajo consumo de recursos, lo que abriría la capacidad del programa para ser usado en dispositivos móviles y que estos mismos fuesen capaces de realizar los cómputos necesarios para la inferencia de los objetos.
Para este fin, se ha empleado la computación en la nube de Google Colab (21) porque permite entrenar modelos de machine learning sin necesidad de un hardware específico para ese fin.

Una vez se ha decidido comenzar con este sistema, se ha preparado un notebook dentro de Google Colab (22) que permite obtener los cinco modelos disponibles. En la Figura 4 se observa una comparación entre los modelos:

Arquitectura	Tamaño (Mb)*	Latencia (ms)**	Precision media***
EfficientDet-Lite0	4.4	37	25.69%
EfficientDet-Lite1	5.8	49	30.55%
EfficientDet-Lite2	7.2	69	33.97%
EfficientDet-Lite3	11.4	116	37.7%
EfficientDet-Lite4	19.9	260	41.96%

** Tamaño de los modelos enteros cuantificados.*

*** Latencia medida en Pixel 4 con 4 subprocesos en la CPU.*

**** La precisión media es el mAP (precisión media media) del conjunto de datos de validación de COCO 2017.*

Figura 4: Comparativa modelos TensorFlow Lite

Tal y como podemos observar, se ve que los modelos con una latencia menor obtienen una precisión media menor, mientras que los que mayor latencia tienen, aumentan más su precisión.

Una vez finalizado el entrenamiento con los cinco modelos disponibles se extraen los siguientes datos:

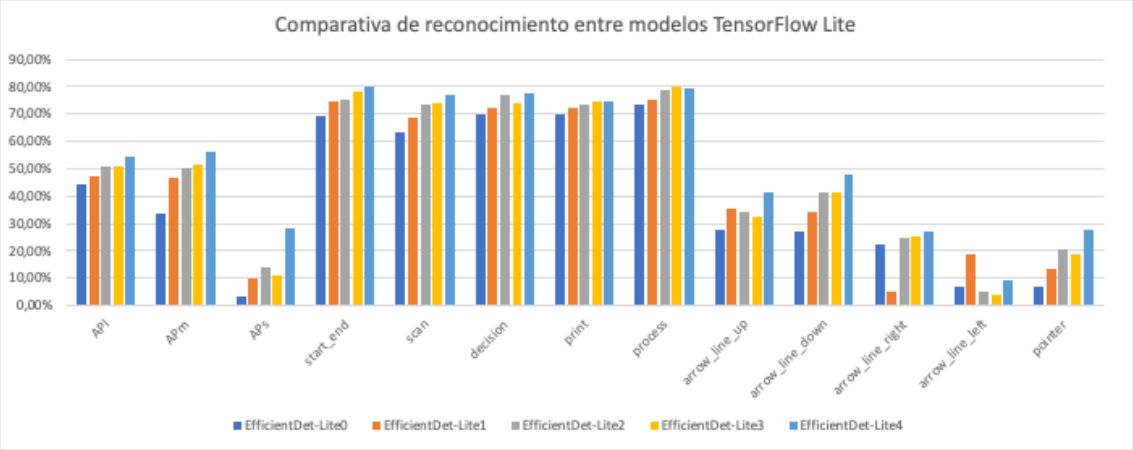


Figura 5: Comparativa de los modelos ya entrenados de TensorFlow Lite

Se puede observar como se corresponde esta gráfica con lo esperado en la Figura 4. Aquí se comprueba que los modelos con un tiempo de latencia menor obtienen un porcentaje de detección menor que los que tienen un tiempo de latencia mayor, sin embargo, no existen unas diferencias tan notables entre los modelos 4 y 5 en las figuras que no son flechas.

No obstante, la detección ronda el 80% de acierto para figuras que no son flechas, y el 50-30% entre las flechas y el puntero de estas. Este factor ha sido tenido en cuenta puesto que estos porcentajes son muy bajos y pueden llevar a erratas en la generación del código. Si la meta del programa es ayudar a estudiantes, no se puede permitir que el programa les devuelva un resultado erróneo.

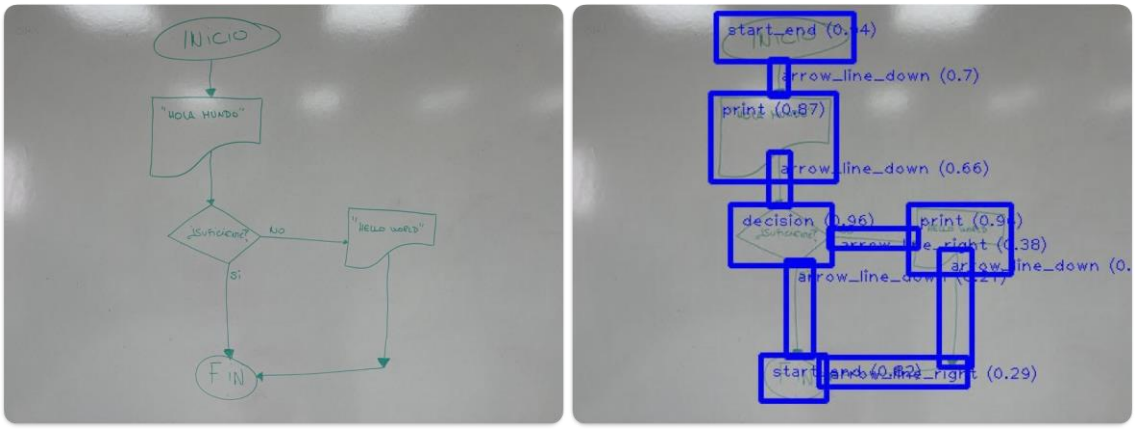


Figura 6: Primera detección de objetos con TensorFlow Lite

2.2. YOLOv5

Una vez observada la limitación que supone emplear TensorFlow Lite, se han explorado otras opciones, siendo YOLOv5 la seleccionada debido a la facilidad que da para su entrenamiento y a los buenos resultados que se obtienen con su modelo.

Para el adiestramiento de este modelo, en un inicio se ha intentado realizar en con Google Colab, pero los altos tiempos de procesamiento han impedido la realización a través de esta plataforma. Por ello ha sido necesario emplear un equipo con una tarjeta gráfica RTX 3060 para el adiestramiento de los modelos. Como Google Colab cierra las instancias a las 24h de comenzar el adiestramiento, no ha sido posible adiestrar ningún modelo. Sin embargo, empleando el equipo con la RTX 3060 los adiestramientos finalizaron entorno a las 3h de ejecución.

Para ello se ha realizado la descarga e instalación de los requisitos necesarios para el adiestramiento de YOLOv5 que vienen marcados en su repositorio (23). Estos requisitos son una serie de bibliotecas que deben estar instaladas en el entorno de ejecución de Python para que el algoritmo de adiestramiento de YOLO pueda funcionar correctamente.

Una vez satisfechos estos requisitos, hay que decidir qué modelo de entre los que dispone YOLO se va a emplear para el adiestramiento.

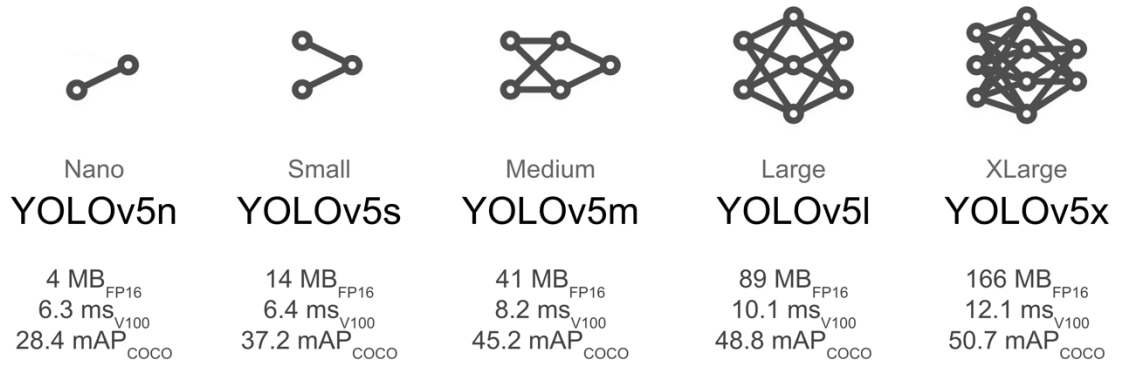


Figura 7: Modelos de reconocimiento YOLOv5

Se puede apreciar como igual que sucede con los modelos de TensorFlow Lite, cuanto más compleja es la red neuronal del modelo de YOLO, mayor es el tiempo de latencia, así como mayor es el tamaño del modelo entrenado, pero aumenta en consecuencia la probabilidad de éxito durante el proceso de detección de objetos. Cabe reseñar que estos no son los únicos modelos de los que dispone YOLOv5, pero si son los que se han considerado de mayor interés.

Como no hay mucha diferencia ente los modelos “m”, “l” y “x” de YOLO, se han evaluado todos los modelos con el fin de realizar unas estadísticas y decidir cuál es la mejor opción para la dataset empleada. Para el adiestramiento, se procede a ejecutar el siguiente comando desde consola.

python train.py --img 1280 --batch 5 --epochs 500 --data D:/DatasetFlowChart/data.yaml --weights yolov5s.pt --device 0

Es interesante indicar que significan algunos de estos parámetros:

- **--batch**: Este parámetro le indica a la red neuronal cuantas muestras deben emplearse antes de actualizarse los valores del modelo, calculando los valores de acierto. En sí, es en cuantos trozos debe dividirse la dataset en cada propagación dentro de la red neuronal. Caben reseñar los siguientes tipos de algoritmos de adiestramiento en función del tamaño seleccionado:
    - Batch Gradient Descent: Batch Size = Training Set Size
    - Stochastic Gradient Descent: Batch Size = 1
    - Mini-Batch Gradient Descent: 1 < Batch Size < Training Set Size
- **--epochs**: Este parámetro indica cuantas veces se va a recorrer la dataset de entreno.
- **--wieghts**: Este parámetro indica con que modelo de YOLO se va a realizar el adiestramiento.

Estos valores deben ser tenidos en cuenta puesto que influyen directamente en el adiestramiento del modelo. Esto es debido a que no por aumentar estos valores se van a obtener unos resultados mejores, se puede caer muy fácilmente en el overfit o underfit del modelo (24).

En mi caso he empleado un batch-size de 5 puesto que, como la dataset consta de 9 elementos, los partirá a la mitad a la hora de hacer el adiestramiento. Además, unos 500 epochs. Con estos valores es como he obtenido unos mejores resultados y un menor tiempo de adiestramiento del modelo.

Una vez adiestrados los diferentes modelos se han obtenido las métricas indicadas en la Figura 8.

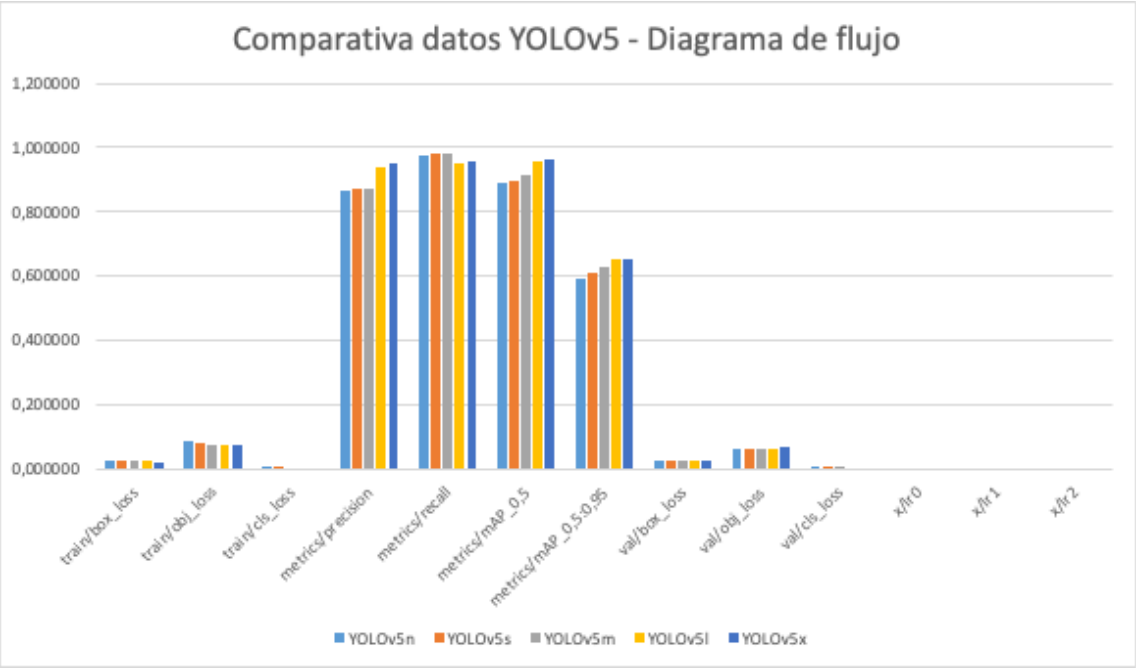


Figura 8: Precisión de los modelos YOLOv5 ya entrenados

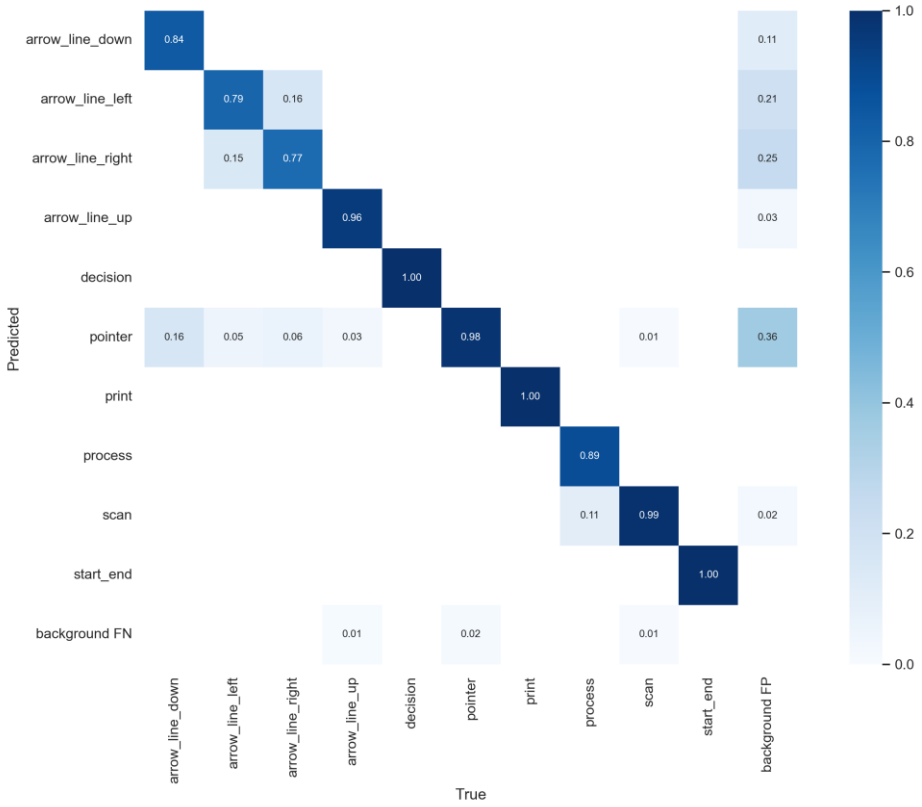


Figura 9: Cuadro de confusión en las detecciones del modelo "l"

De los datos que se observan en la Figura 8 se extrae que en los modelos “l” y “x” la precisión está muy próxima al 100% habiendo muy poca diferencia de precisión entre ambos. Otra prueba que se ha realizado ha sido una prueba para calcular el tiempo de inferencia por parte de ambos modelos ya entrenados, siendo el del modelo “x” (Figura 10) el doble que el del modelo “l” (Figura 11). Por estos motivos se empleará el modelo “l”.

```
YOLOv5  v6.1-263-g0537e8d Python-3.9.13 torch-1.11.0+cu113 CUDA:0 (NVIDIA GeForce RTX 3060, 12288MiB)
Setup complete  (16 CPUs, 31.8 GB RAM, 124.5/930.9 GB disk)

Benchmarks complete (13.12s)
                   Format  mAP@0.5:0.95  Inference time (ms)
0                 PyTorch        0.0257               23.44
1             TorchScript        0.0257               23.55
2                    ONNX        0.0256               26.03
3                OpenVINO           NaN                 NaN
4                TensorRT           NaN                 NaN
5                  CoreML           NaN                 NaN
6   TensorFlow SavedModel           NaN                 NaN
7     TensorFlow GraphDef           NaN                 NaN
8         TensorFlow Lite           NaN                 NaN
9     TensorFlow Edge TPU           NaN                 NaN
10          TensorFlow.js           NaN                 NaN
```

Figura 10: Test de tiempo de inferencia del modelo YOLO "l"

```
YOLOv5  v6.1-263-g0537e8d Python-3.9.13 torch-1.11.0+cu113 CUDA:0 (NVIDIA GeForce RTX 3060, 12288MiB)
Setup complete  (16 CPUs, 31.8 GB RAM, 125.2/930.9 GB disk)

Benchmarks complete (17.51s)
                   Format  mAP@0.5:0.95  Inference time (ms)
0                 PyTorch        0.0292               40.48
1             TorchScript        0.0292               45.56
2                    ONNX        0.0292               45.56
3                OpenVINO           NaN                 NaN
4                TensorRT           NaN                 NaN
5                  CoreML           NaN                 NaN
6   TensorFlow SavedModel           NaN                 NaN
7     TensorFlow GraphDef           NaN                 NaN
8         TensorFlow Lite           NaN                 NaN
9     TensorFlow Edge TPU           NaN                 NaN
10          TensorFlow.js           NaN                 NaN
```

Figura 11: Test de tiempo de inferencia del modelo YOLO "x"

Finalmente, se puede observar en la Figura 9 como las flechas izquierda y derecha se pueden confundir entre ellas de una forma analítica. Mientras que en la Figura 12 se observa de forma gráfica.

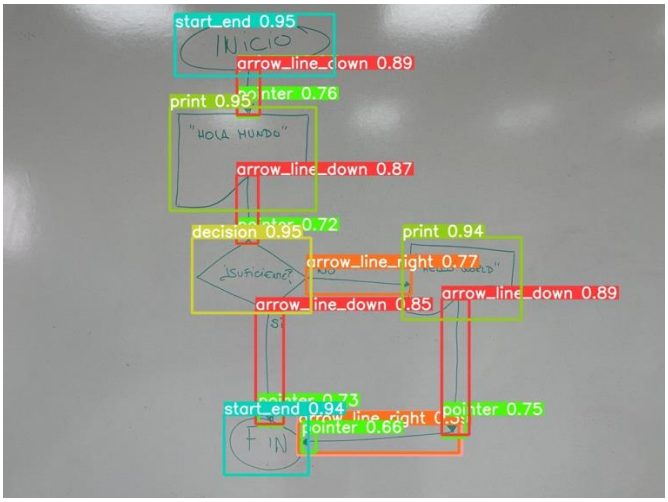


Figura 12: Primera detección de objetos con YOLOv5

2.3. Modificaciones en la imagen

Pese a los buenos resultados obtenidos con el modelo entrenado, se ha decidido realizar una serie de modificaciones en las imágenes de entrada antes de ser inferenciadas por el modelo de detección de objetos. Estas modificaciones son habituales en sistemas de visión por ordenador ya que ayudan al sistema a realizar las detecciones. Las que se han habilitado para ser empleadas son las siguientes:

- Paso a escala de grises
- Ecualización
- Desenfoque gaussiano
- Desenfoque mediano

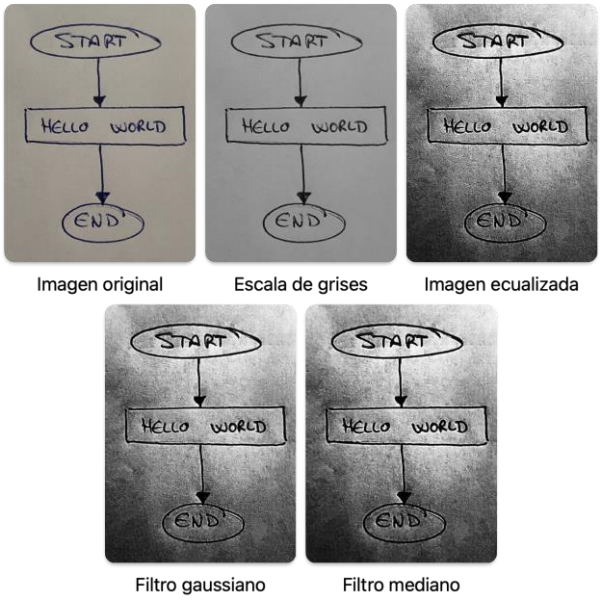


Figura 13: Modificaciones en las imágenes

También se ha probado la posibilidad de realizar dilataciones y aperturas en la imagen, pero como los resultados no han sido buenos, no se han incluido dentro de las posibilidades del software.

Tal y como se puede observar en la figura 13, no existe una gran diferencia entre emplear el filtro gaussiano y el filtro medio. También se puede observar como la ecualización no ofrece grandes ventajas, pero, el paso a escala de grises si mejora la imagen.

Por todo ello, las pruebas que se explicarán en el Capítulo 3 y los anexos 1 y 2, el programa realiza el paso a escala de grises, pero no las demás operaciones salvo que el usuario lo indique en el código del programa. Esto se ha configurado de esta forma debido a que no aporta nada habilitarlas.

2.4. Generación de los objetos "Block" (Bloques)

Figura 14: Diagrama UML de la clase "Block"

Una vez se han detectado los objetos dentro del diagrama se procede a generar un objeto de tipo "Block" (de ahora en adelante bloque) por cada uno de los objetos y almacenarlos dentro de un listado para facilitar las operaciones del programa, su diagrama UML se puede observar en la Figura 14.

Estos bloques tienen los siguientes atributos:

- **id**: es un identificador único para cada bloque.
- **label**: identifica el tipo de bloque según la enumeración “LABEL” (Figura 15).
- **x_min, x_max, y_min, y_max**: estos valores indican los límites de la figura dentro del dibujo en coordenadas cartesianas.
- **confidence**: indica la precisión de la detección de la figura.
- **Texts**: es un listado de objetos de tipo Text (se explicará en el Capítulo 3) que alberga el texto escrito dentro del dibujo.
- **Next_Blocks**: guarda el id de los siguientes bloques en orden de ejecución del diagrama.
- **Previous_Blocks**: guarda el id de los anteriores bloques en orden de ejecución del diagrama.
- **Next_Blocks_Conditionals**: en caso de ser un bloque de tipo condicional (se explicará en el Capítulo 4), las claves de este diccionario indican los siguientes bloques, y los valores asociados la condición que se debe de cumplir.

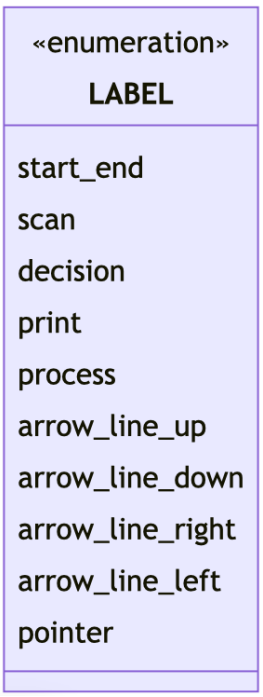


Figura 15: Diagrama UML de la enumeración "LABEL"

También tienen los siguientes métodos:

- **sort_text()**: este método ordena y une los textos dentro del bloque (se explicará en el Capítulo 3).
- **to_string()**: este método se ha implementado con fines de testeo del programa, devuelve por consola todos los atributos del objeto. Se ha implementado ya que en ocasiones ha resultado ser más útil que el depurador del IDE.

En las Secciones 3 y 4 se explicará en detalle el uso de estos objetos.

3. Sistema OCR

El sistema OCR es el siguiente paso en la detección del diagrama de flujo. Inicialmente se han probado los sistemas EasyOCR y Tesseract.

EasyOCR es una biblioteca (25) para su empleo en Python que soporta el reconocimiento de texto en más de 80 idiomas. Cabe reseñar que no precisa de ningún tipo de instalación, funciona únicamente instalando el paquete desde la consola de Python. Por este motivo fue la primera opción a valorar ya que se buscaba realizar un programa que fuese sencillo de ejecutar. Sin embargo, los resultados no fueron los esperados. Tal y como se puede comprobar en el Anexo 1, esta biblioteca funciona muy bien para realizar detección de texto escrito a máquina, pero no funciona bien a la hora de detectar texto manuscrito (ya sea en mayúsculas o minúsculas).

Tesseract (26) es otra biblioteca OCR que basa su funcionamiento en un reconocimiento de líneas de caracteres, buscándoles un sentido mediante un diccionario. Pero al contrario que EasyOCR, para su funcionamiento necesita realizar la instalación de su kernel. En este caso se puede observar tanto en el Anexo 2 como en la Figura 16 que no ha obtenido buenos resultados en ningún caso, por lo que ha habido que continuar buscando alternativas.

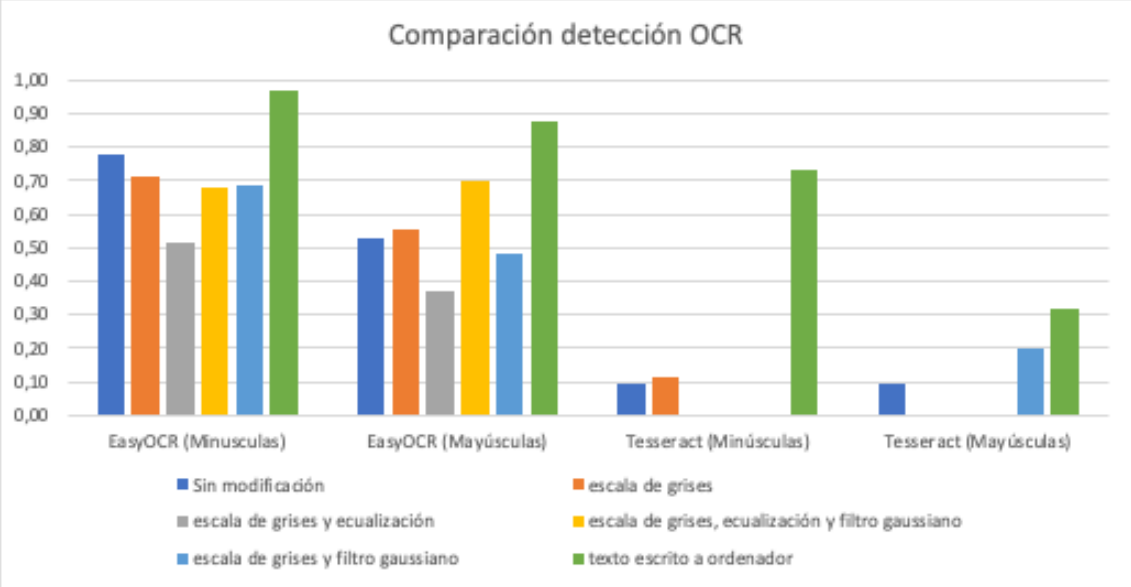


Figura 16: Comparación EasyOCR y Tesseract

3.1. Sistema OCR basado en TensorFlow, Keras y OpenCV

Tras probar las dos opciones anteriores, el siguiente paso en el desarrollo fue desarrollar un sistema OCR aprovechando los conocimientos adquiridos con las transformaciones de imágenes realizadas con OpenCV. Para el diseño e implementación de este sistema se ha tomado como base un artículo publicado por el investigador Davide Spalla (27).

Durante su funcionamiento realiza dos transformaciones a la imagen, la primera es un paso a escala de grises de la imagen y la segunda es una binarización de la imagen. Con la binarización se introduce un nivel límite y los pixeles que tengan un gris superior se vuelven negros, mientras que los que tienen un límite inferior se vuelven blancos. De este modo se facilita la operación de detección de texto.

Para la confección de la dataset con la que se entrenará el modelo de detección se han empleado tres datasets: MNIST (28), A-Z Handwritten Alphabets (29) y Handwritten math symbols (30).

Las dos primeras datasets vienen dadas en formato CSV mientras que la última no, por lo que se ha convertido a formato CSV para facilitar la mezcla de las tres datasets. El código fuente de esta conversión está disponible en el Anexo 4, siendo empleado el fichero “ImagesToCSV.py”. La dataset proporcionada una vez descomprimida viene dividida en subcarpetas por cada uno de los símbolos matemáticos, por ello se genera un listado llamado “symbols” en el que están los nombres de las carpetas a incluir dentro de la dataset ya que no se van a incluir todos los que tiene la dataset.

La dataset en formato CSV tiene como primera columna el id del objeto que luego vendrá representado, como las dataset MNIST y A-Z tiene en total 35 símbolos, los símbolos matemáticos comenzarán con el id 36. Por ello se ha generado un diccionario “symbols_index” que su función es tener como clave el valor del símbolo matemático (“-“, “=”, “”, etc.) y como valor el id que tendrá dentro del CSV. Cabe destacar que los valores de estos símbolos son los nombres de las subcarpetas de la dataset descomprimida.

A continuación, lo hace es iterar por todas las subcarpetas de la dataset y si el nombre de la carpeta en cuestión (el valor del símbolo matemático) está incluido en la lista “symbols” de los símbolos que se van a incluir en la dataset, este pasa a iterar por todas las imágenes de la subcarpeta. Lo primero que hace con cada imagen es pasarla a escala de grises, así por cada píxel solamente tiene que almacenar un valor (0-255) para el tono de gris y no los tres valores de los tonos RGB. A continuación, como las imágenes contenidas dentro de la dataset en formato CSV tienen un tamaño de 28x28 pixeles, redimensiona la imagen a ese tamaño y hace una operación bitwise_not, que

consiste en cambiar el valor de los bits (los que son 0 pasan a ser 1 y viceversa). Finalmente convierte el array multidimensional en un array unidimensional con la operación flatten (Figura 17). Finalmente, se inserta en la dataset que se está generando una fila nueva, siendo el primer elemento el id del símbolo matemático y los siguientes el array unidimensional de la imagen.

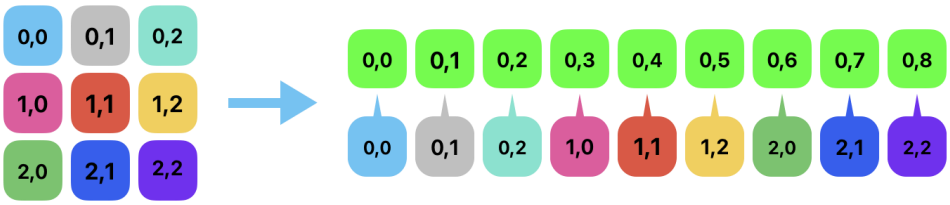


Figura 17: Operación flatten

Una vez listas las tres datasets en formato CSV, se pasa a realizar el adiestramiento del modelo OCR. Este adiestramiento puede ser observado en el Anexo 4, en el fichero “Train.py”.

En la Figura 18 se puede observar el diagrama de flujo de este fichero, que ahora se detallará. Inicialmente se seleccionará el hardware con el que se realizará el adiestramiento de la máquina. El programa intenta emplear la GPU en caso de que CUDA (31) se encuentre instalado y configurado, en caso contrario empleará la CPU para el adiestramiento del modelo.

A continuación, descarga la dataset MNIST, transforma las imágenes contenidas a un tamaño de 28x28 pixeles y guarda el listado con los arrays unidimensionales de las imágenes de las datasets de entrenamiento y test. Lo siguiente que hace es leer la datasets A-Z y símbolos matemáticos en formato CSV y almacenarlas en dos listados para cada dataset, uno con las etiquetas y otro con las imágenes.

La siguiente fase en el proceso de adiestramiento se focaliza en mezclar las datasets en una única dataset que será dividida en una porción de entreno y otra de pruebas, con una proporción 80%-20%.

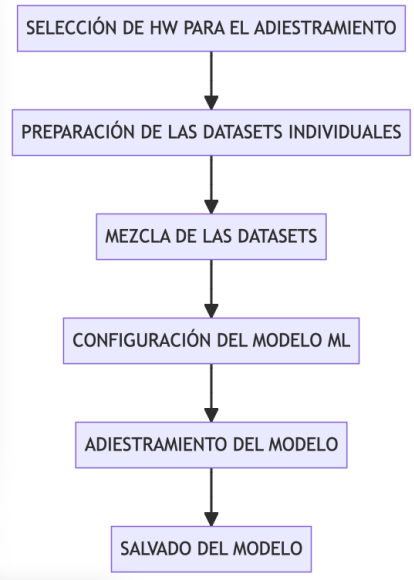


Figura 18: Diagrama de flujo del entrenamiento del modelo OCR

Antes de comenzar la fase de adiestramiento del modelo, es preciso configurar la red neuronal. Para ello primero se genera una capa convolucional de dos dimensiones (Conv2D), otra para indicar el tamaño de las muestras que tomará (MaxPooling2D) ya que esta hace un remuestreo de la entrada, a continuación, se agrega otra capa convolucional de dos dimensiones y se vuelve a remuestrear la entrada. Las dos últimas capas serán de neuronas ocultas (Dense), ya que serán las encargadas de predecir los resultados de los inputs.

Finalmente, se realiza el adiestramiento del modelo con 250 epochs (aumentando este número no se ha conseguido mejorar el resultado de las predicciones del modelo) y una vez entrenado se guarda.

Una vez realizado el adiestramiento del modelo, el programa ya está listo para implementar el sistema OCR. Este se puede encontrar en el programa, en la ruta “OCR/HandwrittenOCR/OCR.py”.

Para su funcionamiento, primero debe pasar un reconocimiento OCR por parte del sistema EasyOCR con la finalidad de obtener las coordenadas de los textos dentro de los objetos. Hay que reseñar que debido a los resultados que, aun siendo buenos, no llegan a ser perfectos, por lo que este sistema no se aconseja para su uso en producción. Por este motivo este sistema no reconoce los textos de las condiciones, quedan reconocidos en el sistema EasyOCR.

A continuación, hay que llamar a la función “OCR” y pasarle por parámetros el dibujo del diagrama de flujo, el listado de bloques y, si se quiere, una modificación del umbral para la binarización (por defecto tiene un valor de 128) que se va a emplear. Una vez se llama a esta función, lo primero que hace es pasar la imagen a un array cv2, a continuación la transforma a escala de grises y finalmente hace una binarización de la misma con el umbral definido en el parámetro de entrada de la función.

El siguiente paso que hace el sistema es iterar por todos los bloques y recortar la imagen en las coordenadas donde están incluidos esos textos. Una vez obtenidos esos recortes comienza el proceso de reconocimiento del texto dentro de ellos. De igual forma que durante el proceso de entreno del modelo, lo primero que realiza es comprobar si existe una GPU con CUDA configurado e instalado para decidir si realizará los cálculos con la CPU o la GPU. Una vez realizado esto, se hacen unas modificaciones en el recorte de la imagen, primero se pasa a escala de grises y luego se pasa por un filtro de distorsión gaussiana. Ahora es el momento en el que se realizan las detecciones, primero se hace una operación sobre la imagen que busca los bordes de las figuras dentro del recorte (Canny, Figura 19) para luego realizar a su vez otros recortes sobre esos bordes de forma que se separa letra a letra el texto.

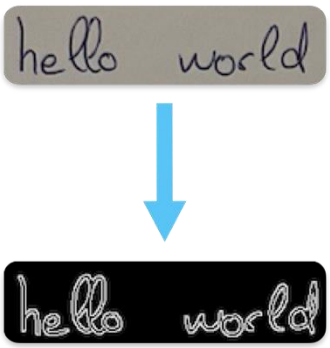


Figura 19: Operación Canny

Una vez obtenidos los contornos de cada letra o símbolo matemático (de ahora en adelante se tratarán como símbolos) de forma independiente se realiza una binarización de cada una, así como una redimensión a un tamaño de 28x28 pixeles. Finalmente, el modelo hace detección de las letras.

```
labelNames = {
    0: "A", 1: "B", 2: "C", 3: "D", 4: "E", 5: "F", 6: "G", 7: "H", 8: "I", 9: "J",
    10: "K", 11: "L", 12: "M", 13: "N", 14: "O", 15: "P", 16: "Q", 17: "R", 18: "S", 19: "T",
    20: "U", 21: "V", 22: "W", 23: "X", 24: "Y", 25: "Z", 26: "0", 27: "1", 28: "2", 29: "3",
    30: "4", 31: "5", 32: "6", 33: "7", 34: "8", 35: "9", 36: "-", 37: "(", 38: ")", 39: "+",
    40: "=", 41: "div", 42: "geq", 43: "gt", 44: "lt", 45: "leq", 46: "neq"}
```

Figura 20: Diccionario “labelNames”

Como la dataset con la que se ha adiestrado el modelo indicaba un id (del 0 al 46) a cada uno de los símbolos empleados, las detecciones realizadas por el modelo devuelven una probabilidad para cada uno de esos id en cada detección. Esta asociación entre el id y el valor del símbolo viene indicada en el diccionario “labelNames” que se puede observar en la Figura 20. Ahora el programa lo que hace es tomar esas predicciones y ordenarlas para obtener el resultado final de la detección.

Figura 21: Diagrama UML de la clase "Letter"

Una vez ordenadas las predicciones se genera un objeto de tipo “Letter” (Figura 21) con los siguientes atributos:

- **x_min, x_max, y_min, y_max**: estos valores indican los límites de la figura dentro del dibujo en coordenadas cartesianas.
- **Confidence**: se trata de la probabilidad de acierto en el proceso de detección por parte del modelo.
- **Value**: guarda el valor del símbolo.
- **Index**: es el índice de la letra en la dataset.
- **Replace_chars**: este diccionario se emplea para reemplazar los valores del diccionario labelNames que se corresponden con ellos por los valores reales a los que apunta (Figura 22). Este proceso se realiza de forma automática en el momento en el que se instancia el objeto.

```
replace_chars = {
    "div": "/",
    "geq": ">=",
    "gt": ">",
    "lt": "<",
    "leq": "<=",
    "neq": "!="
}
```

Figura 22: Diccionario “replace_chars”

Cabe reseñar que en este punto el programa genera una de sus limitaciones en caso de emplear este método OCR. Al basar su funcionamiento en la búsqueda de bordes, se debe dibujar el diagrama de flujo en un papel en blanco, no puede tener marcas si no, pasa lo que se muestra en la Figura 23, las detecta y las interpreta como si fuesen símbolos llevando a una detección errónea.

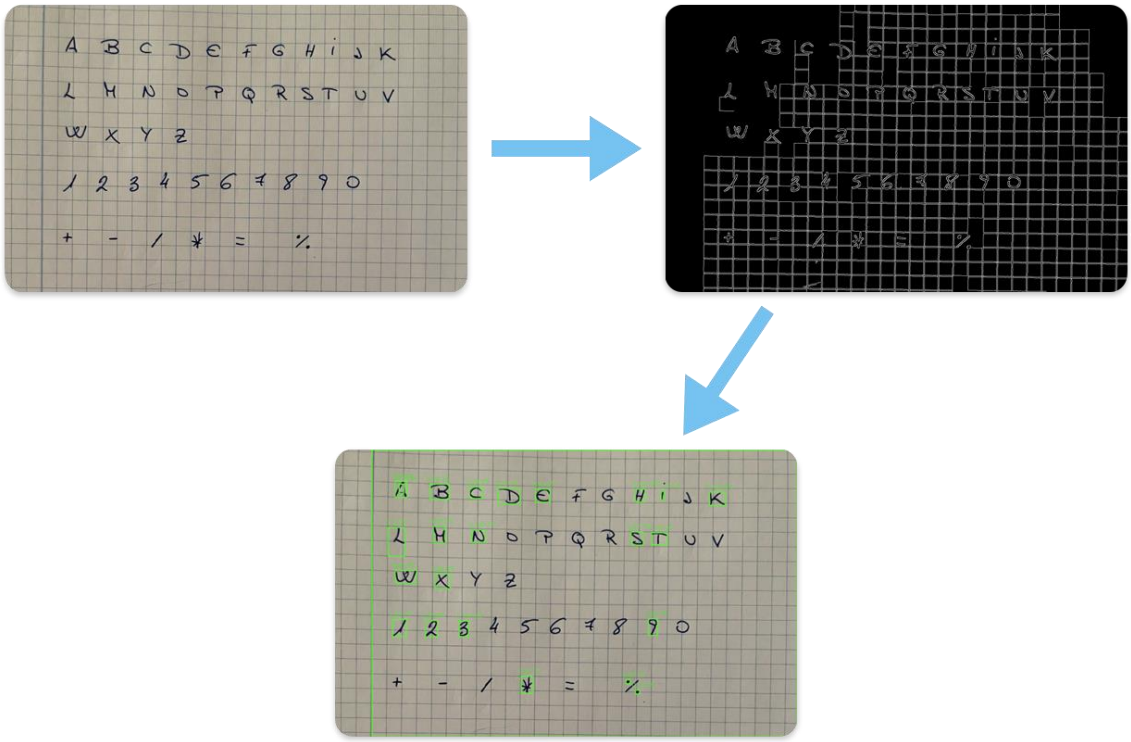


Figura 23: Prueba de detección con un folio a cuadros

Sin embargo, con la misma prueba, pero con el fondo del folio liso se obtienen mejores resultados como se puede observar en la Figura 24.

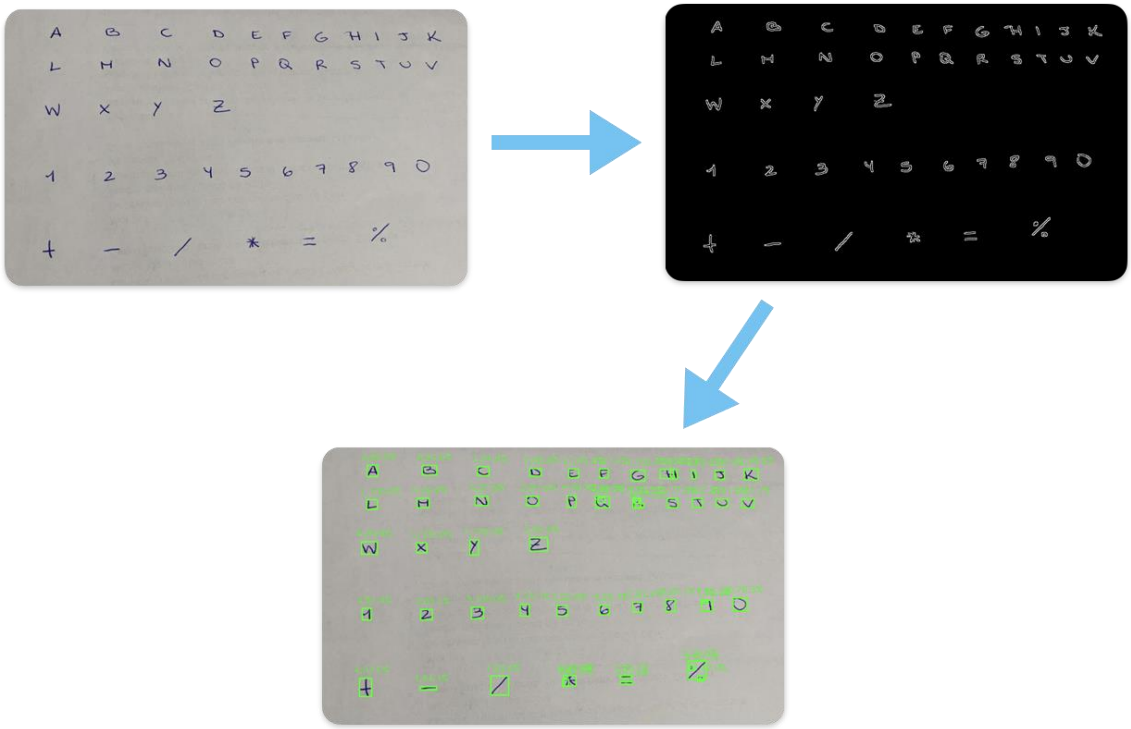


Figura 24: Prueba de detección con un folio liso

En este punto se procede a evaluar la calidad de las detecciones realizadas sobre todos los símbolos que acepta el modelo empleando páginas de prueba con 10 símbolos manuscritos de cada uno. Los resultados gráficos de las pruebas se pueden observar en el Anexo 5, mientras que en la Figura 25 se muestra el cuadro de confusión del modelo. En este cuadro se representa para cada símbolo a detectar la posibilidad de error con los demás símbolos.

En el siguiente subapartado se tratará el proceso para la corrección de estas confusiones por parte del modelo.

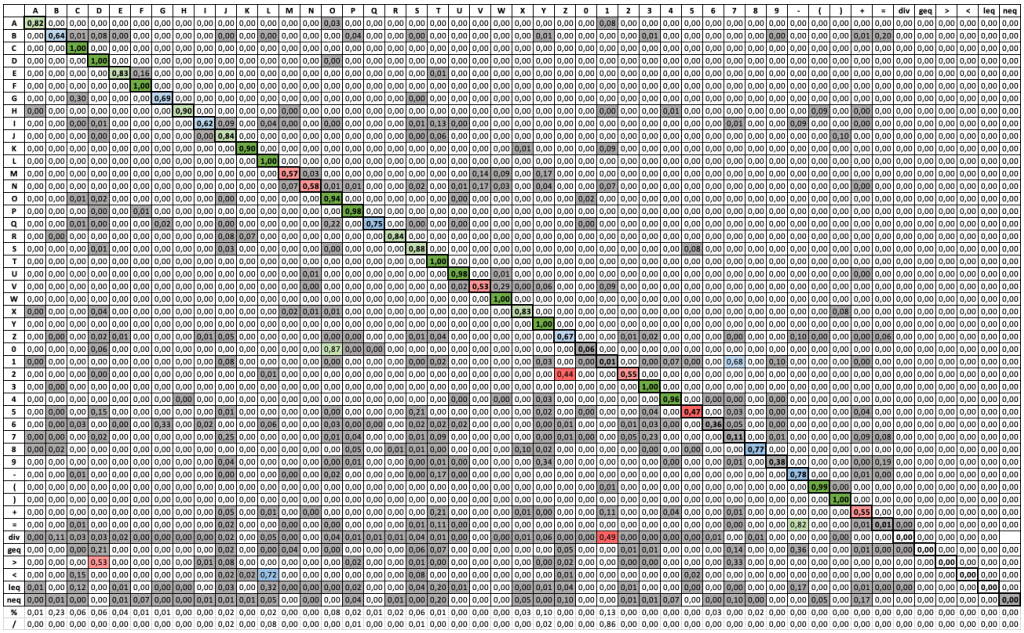


Figura 25: Cuadro de confusión del modelo generado con errores

Para finalizar con la detección de textos, se devuelven todos los símbolos contenidos dentro del bloque como un listado de objetos “Letter” para ser procesado y obtener así el texto del bloque. En un primer lugar se obtiene el ancho medio de los símbolos para emplearlo como umbral para la detección de espacios entre palabras. A continuación, se itera entre los objetos “Letter” de izquierda a derecha en el eje x, de forma que se van uniendo al texto contenido dentro del bloque. Durante esta iteración se evalúa la distancia con el símbolo anterior, de forma que si está por encima del umbral de espacios se inserta uno en el texto. Con el texto generado, el siguiente paso consiste en instanciar un objeto de tipo “Text” para cada bloque con los textos generados a partir de las detecciones. Para las coordenadas de estos objetos se toman las coordenadas máximas y mínimas que ocupan los símbolos de forma que el objeto instanciado ocupe todos los símbolos. En cuanto al atributo “confidence” realiza una media aritmética de todos los símbolos que componen el texto generado.

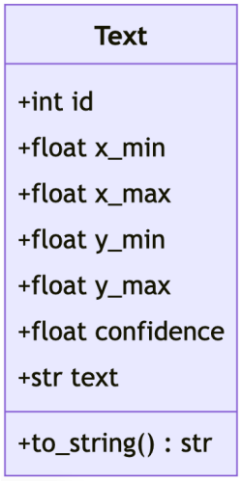


Figura 26: Diagrama UML de la clase “Text”

3.2. Corrección del modelo de detección OCR desarrollado

Una vez obtenida la tabla de confusión y las detecciones erróneas de forma gráfica se puede observar que los fallos vienen dados por la confusión entre uno o dos símbolos diferentes, por lo que lo óptimo ha sido solucionarlo mediante código siempre que ha sido posible.

Estas operaciones han sido las siguientes:

- El primer fallo por corregir es uno recurrente, y es que en muchos símbolos se realizan detecciones múltiples, así que lo que se ha realizado es eliminar esas detecciones internas empleando los atributos de los objetos “Letter” que indican su posición en los ejes cartesianos.
- La letra “B” presenta un error de detección con la letra “P”. En la imagen de muestra se observa que tiene una mancha y eliminándola se corrige.
- En la letra “E” se ha observado que tiene confusiones con la letra “F”, así que el procedimiento a seguir será eliminar las letras “F” que ocupen una posición similar a la de una letra “E”.
- La letra “G” tiene fallos de detección con la letra “C” y por similitud no es posible corregirlo mediante código.
- La letra “H” tiene fallos de detección con el símbolo “(“, observando los datos de prueba suministrados, se observa que ese fallo viene dado por la forma de la “H”, entonces ese error lo puede tener también con el símbolo “)”. Para solucionarlo se comprueban en los extremos izquierdo y derecho de la letra H (dentro de los límites del símbolo) si existen los símbolos “(“ o “)”, y si es así, se eliminan.
- La letra “I” tiene varios problemas que solucionar:
    - El primero es que en algunos casos realiza una doble detección con la letra “T”, así que se van a eliminar las letras “T” que ocupen una posición similar a las letras “I”. Esto se puede realizar porque la detección de las letras “F” no tiene fallos, por lo que se conoce a ciencia cierta que una detección en esta posición será un error.
    - Otro problema que tiene es que realiza detecciones dobles de la letra “I” con un desplazamiento, con lo que el método inicial de eliminación de duplicados no funciona. Se eliminarán estos duplicados.
    - El último problema que tiene es que realiza la detección como una “J” y un “-“. Esto se puede resolver ya que en la detección de esos símbolos no existe ningún problema. Lo que se hará será buscar los símbolos “J” que tengan un símbolo “-“ en su interior.
- La letra “J” presenta alguna confusión con el símbolo “)” y no es posible reparar esta detección mediante código.

- La letra “M” presenta algunos problemas que no se pueden solventar. Únicamente se pueden eliminar las detecciones múltiples que hacen que coincida la letra “Y” con la letra “V”.
- La letra “N” presenta varios problemas:
    - Dentro de la letra “N” realiza detecciones de otras letras, así que se eliminarán.
    - Se presentan casos de una detección múltiple, dando resultado a una letra “V” dentro de una letra “M”, por lo que, para solventar este problema cuando se dé, se eliminará la “V” y se transformará la “M” por una “N”.
    - En la imagen de prueba hay una detección de una “W” con un “1” pegado. Por similitud, se buscarán las “W” que tengan un “1”, “(“ o “)” pegados a la izquierda o derecha, se eliminará y se transformará la “W” por una “N”.
- La letra “Q” presenta problemas de detección y tiene confusiones con la letra “O” pero no se puede solucionar mediante código.
- La letra “R” detecta internamente a la letra “K”. Como la letra “K” no tiene problemas en la detección, se repararán las letras “R” eliminando las letras “K” en su interior.
- La letra “S” se confunde con el número “5”. Esto se solucionará posteriormente cuando se formen las palabras según el contexto.
- Los problemas de detección de la letra “V” no tienen solución mediante código.
- La letra “Y” se detecta correctamente, pero en ocasiones realiza detecciones dobles desplazadas, por lo que se eliminará una de esas letras desplazadas.
- El número “0” se confunde con la letra “O”, este problema se resolverá posteriormente cuando se formen las palabras según su contexto.
- El número “1” tiene dos problemas:
    - Se confunde con la letra “J”, este problema se solucionará en el momento de formar palabras según el contexto.
    - Se confunde con el número “7”, este problema no tiene solución mediante código.
- El número “2” se confunde con la letra “Z”, este problema se solucionará cuando se formen las palabras según su contexto.
- El número “5” presenta dos problemas:
    - Se confunde con las letras “S y D”, este problema se solucionará cuando se formen las palabras según el contexto.
    - Se confunde con el símbolo “+”, este problema se solucionará cuando se formen las palabras en caso de que el numero 5 no vaya solo. Se explicará la solución más adelante.
- El número “6” se confunde con las letras “G” y “L”, este problema se solucionará cuando se formen las palabras según su contexto.

- El número “7” presenta los siguientes problemas:
    - El único problema que se puede solventar mediante código es la duplicidad en las detecciones, este detecta un símbolo “+” dentro de otro, por lo que se eliminará el otro y el símbolo “+” cambiará a ser un “7”.
    - El siguiente problema es la confusión con las letras “T” y “J”, que se resolverá cuando se formen las palabras según su contexto.
    - El último problema es la confusión con el número “3” que no tiene solución mediante código.
- El número “8” se confunde con la letra “X”, este problema se solucionará cuando se formen las palabras según su contexto.
- El número “9” se confunde con la letra “Y”, este problema se solucionará cuando se formen las palabras según su contexto.
- El símbolo “-“ se confunde con la letra “T”, este problema se solucionará cuando se formen las palabras solamente si esa detección no va dentro de una palabra, si la detección de la “T” se hace estando sola, se cambiará por un símbolo “-“.
- El símbolo “+” presenta varios problemas:
    - Presenta detecciones de una letra “J” con otro símbolo dentro, esto se solucionará eliminando el símbolo interior y cambiando la “J” por un “+”.
    - Presenta confusiones con la letra “T” y el número “5”, esto se resolverá en el momento de formar palabras según su contexto, y en caso de estar fuera de una palabra, se pondrá un símbolo “+”.
- El símbolo “=“ no se detecta por el sistema de detección empleado. Al reconocer formas, detecta dos símbolos “-“ por lo que se soluciona buscando adyacencias de los símbolos “-“.
- El operador de división se puede intuir por dos símbolos y cada uno presenta sus problemas:
    - El símbolo “/” lo detecta como un 1, esto se puede solucionar cuando se generen las palabras interpretando si el operador se encuentra separado a la izquierda y derecha por otras palabras, si es así se interpretará como una división.
    - El símbolo “%” no sigue un patrón que se pueda interpretar, por lo que se desaconseja su empleo.
- Al símbolo “≥” le ocurre algo similar que al símbolo “=”, consta de dos símbolos y el modelo detecta a los dos por lo que hay que interpretarlo. El símbolo “-“ lo puede detectar como tal o como una “T” o una “M”, y el símbolo “>” lo detecta como una de las siguientes letras: “D”, “S”, “Z” o el número “7”. Por ello, se soluciona buscando adyacencias.
- Al símbolo “≤” le ocurre lo mismo que al símbolo “≥” pero cambiando los símbolos con los que confunde el “<”. Este es detectado como una de

las siguientes letras: “C” o “L”. Por ello, se soluciona buscando adyacencias.

- El símbolo “>” se confunde como: “D”, “S”, “Z” o el número “7”, por lo que se definirá lo que es en el momento en el que se generen las palabras si queda uno de esos símbolos solo entre dos palabras.
- El símbolo “<” se confunde como “C” o “L”, por lo que se definirá lo que es en el momento en el que se generen las palabras si queda uno de esos símbolos solo entre dos palabras.

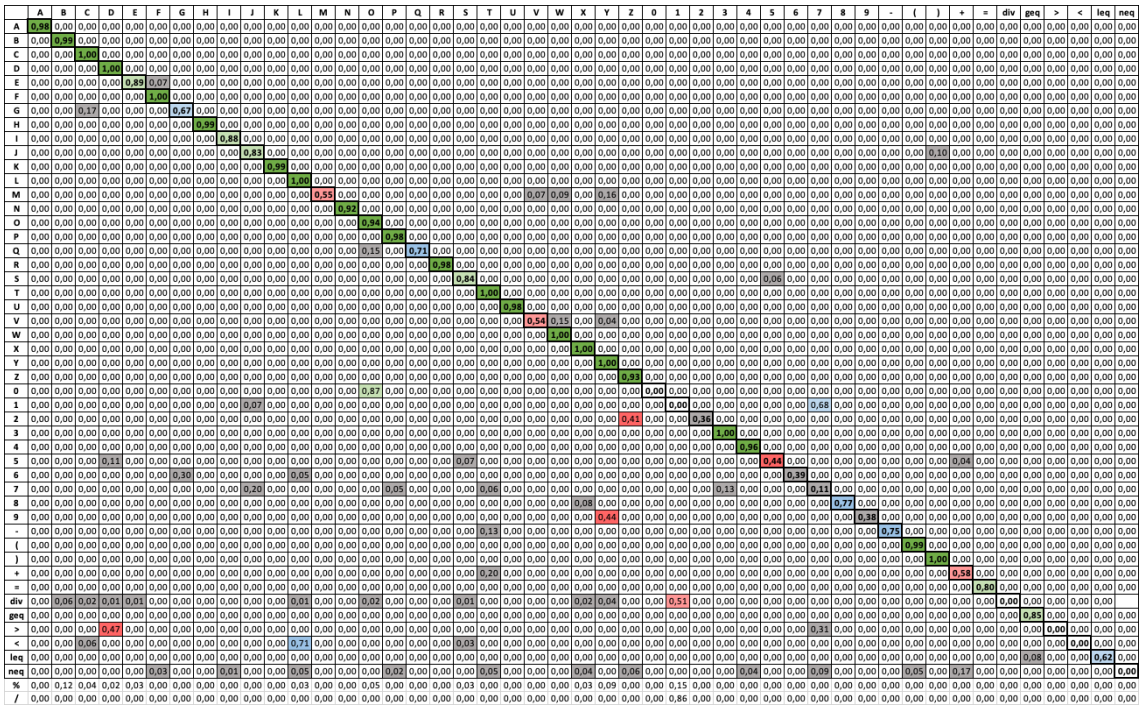


Figura 27: Cuadro de confusión del modelo generado con reparaciones

Tras haber realizado las modificaciones indicadas anteriormente, se vuelven a hacer las pruebas de detección y se obtiene el cuadro de confusión (Figura 27). En él se puede observar como ha mejorado la detección y son menos los símbolos que darán problemas.

En la Figura 28 se observa una prueba de detección de texto y se puede comprobar que el resultado obtenido no es satisfactorio, así que este sistema OCR tampoco es óptimo.

Pese a los malos resultados también se da la opción al usuario de seleccionar este sistema OCR aunque se recomienda emplear el descrito en el subapartado 3.3.

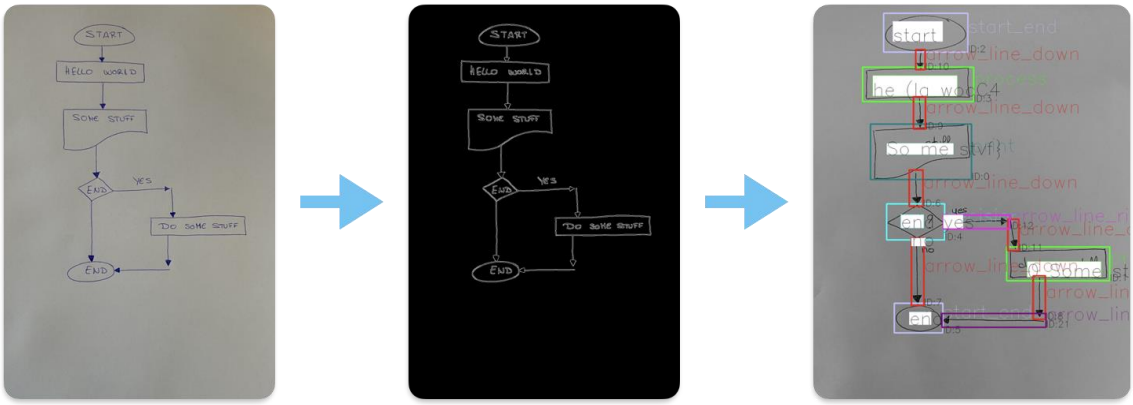


Figura 28: Prueba OCR con el modelo generado

3.3. Azure Computer Vision OCR

Tras evaluar los sistemas OCR previos, el siguiente paso que se ha realizado es la detección OCR mediante un sistema en la nube, más concretamente el ofrecido por Azure.

Para ello, lo primero que habrá que hacer será dar de alta un nuevo grupo de recursos en el que se incluirá el sistema de Computer Vision de Azure. A continuación, dentro de ese grupo de recursos se creará un recurso de Computer Vision. Finalmente se han copiado URL del extemo y una de las claves de acceso para realizar las consultas a la API.

Se comenzará llamando al método “OCR” del fichero “AzureOCR.py”, al cual hay que pasarle por parámetros la imagen y el listado con los bloques detectados. Pero todavía no es posible trabajar con la API de Azure. Para trabajar con ella, en su documentación (32) se indica que se debe pasar una URL de una imagen para realizar la detección de textos. Por este motivo el siguiente paso será crear una cuenta en imgBB (33) y apuntar la URL del extremo y la clave de la API. Estos están especificados en el apartado de la web imgBB API (34) y se muestran una vez autenticado en ella.

Pese a que es una mala práctica, estas URL junto con las claves se han definido en el fichero “Constants.py” y agregado al repositorio para que sea posible probar el software. Sin embargo, en un entorno real de producción no se aconseja exponer las claves y para ello se debe generar un fichero en el que se guarden (al menos) las claves de acceso a las API y posteriormente incluir dentro del fichero “.gitignore” el fichero donde están guardadas estas claves. De esta forma las claves no entrarán dentro del control de versiones del repositorio y no serán expuestas.

Para realizar la subida de la imagen imgBB se ha generado el fichero “ImgbbUploadFile.py” con un único método que devuelve la URL de la imagen

subida al servidor. Lo primero que hace es generar la URL para hacer la petición POST según marca la documentación. A continuación, el dibujo del diagrama se convierte a formato JPG para que la solicitud sea de un tamaño menor y se convierte a un string base64. Finalmente se genera la petición POST con la URL y como datos, el dibujo codificado a base64 y se hace la petición al servidor y el método devuelve la URL de la imagen. Un ejemplo de la respuesta del servidor se encuentra en el Anexo 6 nombrado como “imgbb.json”.

La petición POST es un tipo de petición empleada por el protocolo HTTP que le indica al extremo que se va a insertar algún dato, en el caso de este software, el dibujo codificado en base64. Otras peticiones comunes (35) son GET (indica una solicitud de datos), PUT (indica una solicitud de actualización de datos) y DELETE (indica una solicitud de eliminación de datos). Estas peticiones tienen un acrónimo que es CRUD (Create, Read, Update, Delete) para referirse a ellas cuando una API permite esas operaciones.

Una vez se ha obtenido la URL de la imagen subida a imgBB, se envía la solicitud a la API de Azure de forma que se obtiene un listado de todos los textos detectados dentro de la imagen subida. Estos textos se devuelven por parte de Azure con las coordenadas que ocupan junto al texto reconocido. Por cada una de estas detecciones se genera un objeto de tipo Text (Figura 26) y estas se incluyen dentro de un listado para poder ser iteradas. Cuando todas las detecciones están dentro del listado, se buscan los textos que están dentro de los bloques y los textos que están fuera de ellos y se asocian con una condición en los bloques de tipo “decision”.

Finalmente se devuelven los bloques con los textos reconocidos.

4. Algoritmo de ordenación de los bloques

Este algoritmo (Figura 29) es básico para el funcionamiento de la aplicación. Cuenta con cinco pasos, los dos primeros se han explicado en los Capítulos 2 y 3, en este Capítulo se explicarán los siguientes pasos hasta llegar al punto de obtener un flujo de ejecución de principio a fin del diagrama de flujo.

Figura 29: Diagrama de flujo del algoritmo de ordenación de bloques

Una vez se han reconocido los bloques y el texto escrito, así como se han colocado estos textos en sus respectivos bloques, el siguiente paso en el algoritmo es el de ordenar las flechas. Durante este proceso el programa es cuando elimina los errores en la detección de los bloques flecha derecha e izquierda. Para realizar esta tarea, se genera un listado de bloques al que se irán incluyendo los bloques ya revisados, otro para almacenar los bloques de tipo “arrow” y otro para almacenar los bloques de tipo “pointer”.

A continuación, se itera sobre todos los bloques del conjunto, los bloques de tipo “pointer” se insertan en el listado para ellos, los bloques de tipo “arrow” se insertan en su listado y todos los demás se insertan en el listado de bloques revisados. Una vez realizado esto se itera sobre los bloques del listado que incluye los bloques de tipo “arrow” y por cada uno de ellos se itera sobre el listado de bloques de tipo “pointer”. Por cada uno de los bloques de tipo “pointer” se busca el objeto de tipo “arrow” en el que está contenido. Una vez se han encontrado, se procede a buscar el lado del rectángulo del que más cerca está el objeto “pointer” del objeto “arrow”, en función del lado que sea se deduce la orientación de la flecha del bloque.

El siguiente paso será la de buscar los caminos de las flechas. Este paso es importante ya que deja listos los bloques para ser ordenados en el próximo paso. Inicialmente itera sobre todos los bloques y efectúa las operaciones propias del paso sobre los bloques de tipo “arrow”. A continuación, por cada uno de los bloques se buscan las proximidades a los demás bloques. Esto se realiza haciendo uso del teorema del coseno, implementado sobre el fichero “Math_Calcs.py”. Por cada uno de los bloques “arrow” se vuelve a iterar por todos los bloques que componen el diagrama y, en función del tipo de bloque se toman unos puntos diferentes. Estos puntos serán llamados “a” y b” para las coordenadas del bloque vecino y “p” para el bloque “arrow”. En función del tipo de bloque a evaluar estos puntos se establecen unas coordenadas u otras en función del cálculo que se va a realizar.

Para el cálculo del bloque siguiente:

- **Bloque "arrow_down":** los puntos "a" y "b" serán los puntos superiores del bloque vecino y el punto "p" el punto medio inferior del bloque "arrow".
- **Bloque "arrow_up":** los puntos "a" y "b" serán los puntos inferiores del bloque vecino y el punto "p" el punto medio superior del bloque "arrow".
- **Bloque "arrow_left":** los puntos "a" y "b" serán los puntos más a la derecha del bloque vecino y el punto "p" el punto medio más a la izquierda del bloque "arrow".
- **Bloque "arrow_right":** los puntos "a" y "b" serán los puntos más a la izquierda del bloque vecino y el punto "p" el punto medio más a la derecha del bloque "arrow".

Para el cálculo del bloque previo:

- **Bloque "arrow_down":** los puntos "a" y "b" serán los puntos inferiores del bloque vecino y el punto "p" el punto medio superior del bloque "arrow".
- **Bloque "arrow_up":** los puntos "a" y "b" serán los puntos superiores del bloque vecino y el punto "p" el punto medio inferior del bloque "arrow".
- **Bloque "arrow_left":** los puntos "a" y "b" serán los puntos más a la izquierda del bloque vecino y el punto "p" el punto medio más a la derecha del bloque "arrow".
- **Bloque "arrow_right":** los puntos "a" y "b" serán los puntos más a la derecha del bloque vecino y el punto "p" el punto medio más a la izquierda del bloque "arrow".

Como ya se ha indicado, se emplea el teorema del coseno para el cálculo de estas distancias. Para ello se genera un triángulo con las coordenadas de los puntos que se han indicado antes (Figura 30).

Para realizar el cálculo de esta distancia, se realizará sobre los segmentos A y B, siendo la distancia el menor de los segmentos.

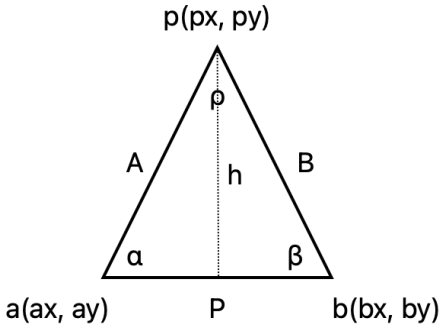


Figura 30: Coordenadas del triángulo empleado

Antes de comenzar a realizar los cálculos, se simplifica la fórmula hasta obtener la que interesa implementar en el código del programa:

$$A^2 = B^2 + P^2 - 2BP*cos(\alpha) \rightarrow A^2 - B^2 - P^2 = -2BPcos(\alpha)$$
$$cos(\alpha) = (-A^2 + B^2 + P^2) / 2BP$$
$$cos(\beta) = (A^2 - B^2 + P^2) / 2AP$$

Interesa obtener el coseno porque con él se obtendrá el valor de los ángulos α y β, y con estos valores es con lo que se decidirá cuál de los segmentos A y B es el más corto.

En caso de que α sea mayor de 90º querrá decir que el segmento A es más corto, mientras que, si β es mayor de 90º, este será el más corto y querrá decir que los bloques no están enfrentados. En caso de que ambos tengan un valor menor de 90º se empleará la definición del seno para el cálculo de la altura. En este último caso, la distancia será la altura ya que querrá decir que los bloques están enfrentados. Estos casos se entienden mejor observando las figuras 31, 32, 33 y 34.

$$sen(\alpha) = A / h \rightarrow h = A * sen(\alpha)$$

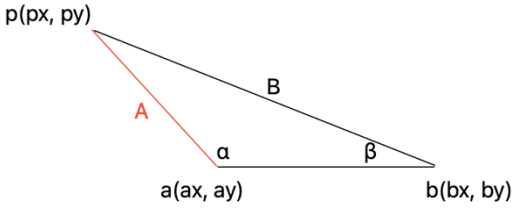


Figura 33: Ángulo α > 90º

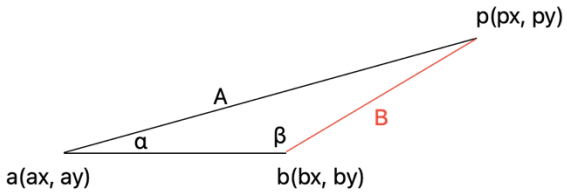


Figura 31: Ángulo β > 90º

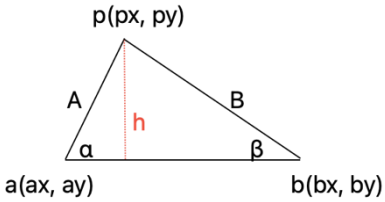


Figura 32: Ambos ángulos < 90º

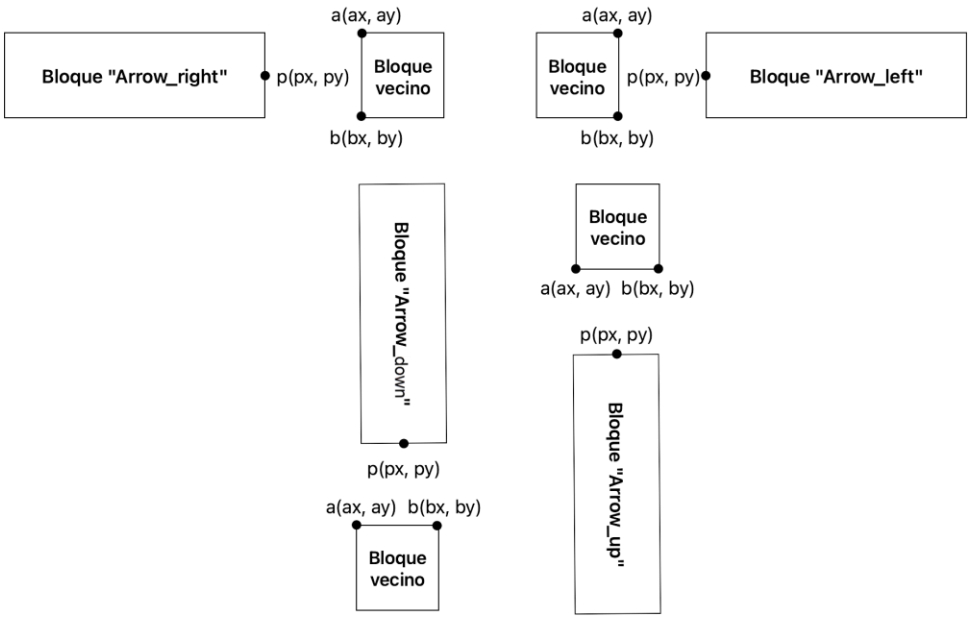


Figura 34: Ubicación de los puntos en la búsqueda del próximo bloque

Con estos puntos se rellenan dos diccionarios, uno empleado para los bloques vecinos precedentes y otro para los bloques vecinos antecedentes. En estos diccionarios se emplea como clave el id del bloque evaluado y como valor la distancia con respecto al bloque de tipo “arrow”. Una vez están rellenos se devuelven los id con una distancia menor y de esta forma se obtienen los bloques previo y posterior de cada flecha.

El siguiente paso del algoritmo es la ordenación de los bloques que no son de tipo “arrow” en función de estos últimos. Para ello lo primero que hace es iterar por todos los bloques de tipo “arrow” cuyo bloque predecesor no sea otro bloque de tipo “arrow”. A continuación, se van buscando los bloques que van a continuación uno detrás de otro hasta que se llega a un bloque que no sea de tipo “arrow”. Para prevenir un bucle infinito, esta búsqueda se realiza como máximo 10 veces. Una vez se encuentra el siguiente bloque se almacena su id en el bloque previo y, si el bloque “arrow” tiene un texto (será un condicional), se almacena en el diccionario de bloques condicionales el id con la condición.

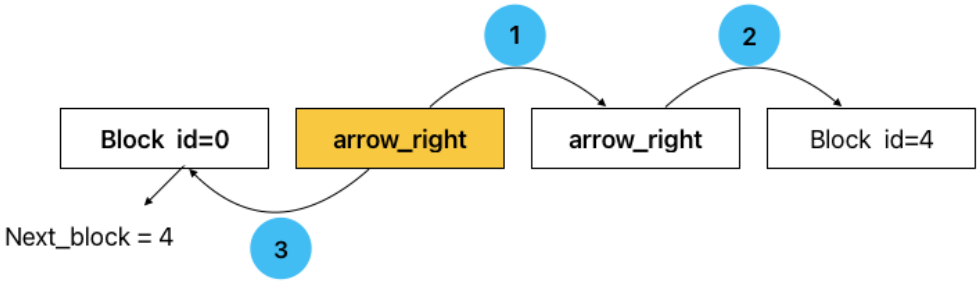


Figura 35: Algoritmo de búsqueda de bloques previos

Para la búsqueda del bloque previo ya no se emplean los bloques de tipo “arrow”. Lo que se hace es iterar entre todos los demás bloques y para cada uno de ellos, buscar el bloque que le sigue, y a este, indicarle que el bloque anterior es el que le precede.

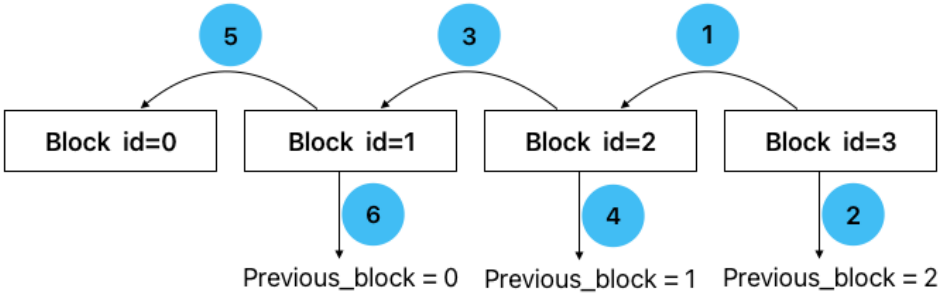


Figura 36: Algoritmo de búsqueda de bloques posteriores

El último paso del algoritmo consiste en eliminar todos los bloques “arrow”, por lo que en este paso el programa maneja todos los bloques ya ordenados de inicio a fin. Con la operación de eliminar los bloques “arrow” se consigue eliminar instancias de objetos en memoria que ya no tienen ningún valor mejorando la velocidad del programa.

5. Parser generator (ANTLR4)

Antes de comenzar conviene entender el funcionamiento de ANTLR4, para ello consta de tres pasos:

1. **Análisis léxico**: lo realiza el lexer, tomando la gramática regular y el flujo de texto entrante, dando como resultado al flujo de tokens, que son las unidades mínimas con un significado dentro del programa, siguiendo el mismo orden en el que se han encontrado dentro del flujo de entrada.
2. **Análisis sintáctico**: lo realiza el parser, tomando como entrada el flujo de tokens generado por el analizador léxico y la gramática libre de contexto. En este paso se ordenan los tokens para que tenga un sentido sintáctico correcto, dando como resultado un árbol de sintaxis concreta.
3. **Análisis semántico**: toma como entrada el árbol de sintaxis concreta generado por el analizador sintáctico y en conjunto con la gramática de atributos genera el árbol de sintaxis abstracta, permitiendo generar el código fuente final.

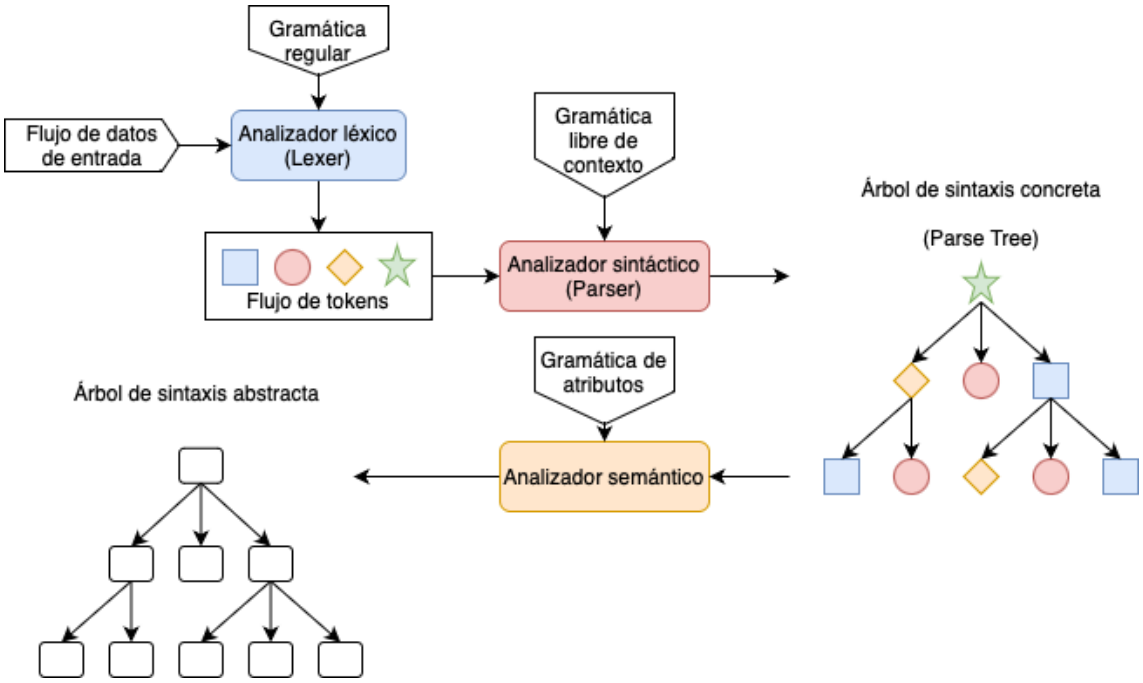


Figura 37: Algoritmo ANTLR4

5.1. Paso de bloques a pseudocódigo

Para realizar esta operación se emplea un objeto de tipo “Pseudocode” para almacenar las líneas de pseudocódigo, las variables junto con su tipo y las posibles funciones a las que apunta el diagrama de flujo.

Este objeto permitirá también realizar una exportación de los atributos a un pseudolenguaje legible por un humano, así como a un pseudolenguaje que entienda ANTLR4.

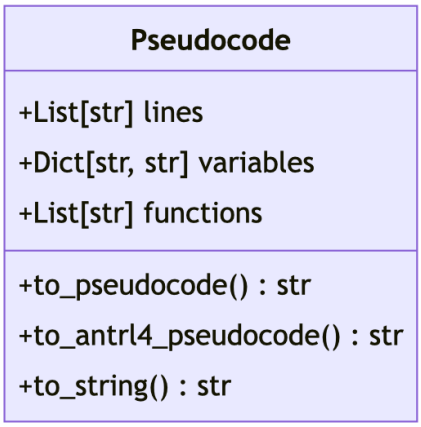


Figura 38: Diagrama UML de la clase "Pseudocode"

Para generar este objeto hay que realizar unas operaciones sobre cada uno de los bloques en orden y en función de su tipo de bloque, así como de otras características se genera la línea de pseudocódigo del bloque. Como el funcionamiento de ANTLR4 se basa en buscar tokens dentro de un flujo de datos entrante, se prepara un pseudocódigo que ayude a ANTLR4 a interpretar el pseudocódigo. Estas etiquetas son las siguientes:

- **<VAR_DECLARATION>** - Le indicará a ANTLR4 que se va a realizar la declaración de una variable. Esto se usará en el proceso de conversión a pseudocódigo de ANTLR4.
- **<PRINT>** - Indica que se va a efectuar una instrucción de impresión por pantalla.
- **<SCAN>** - Indica que se va a efectuar una operación de lectura de datos por consola.
- **<FUNC>** - Indica la llamada a una función o que la función raíz no es la principal (main).
- **<BASE_FUNC>** - Indica que la función raíz se trata de la principal.
- **<CONDITIONAL> </CONDITIONAL>** - Indica el inicio y fin de una instrucción de tipo condicional.
- **<CONDITION> </CONDITION>** - Indica el inicio y fin de una rama de una instrucción condicional.
- **<END>** - Indica el final del código
- **<INT>** - Indica que una variable es de tipo “int”.
- **<STRING>** - Indica que una variable es de tipo “string”.

Además, en el Anexo 8 se indican unas limitaciones que deben seguir los usuarios en el momento en el que realizan el dibujo para que este pueda ser interpretado correctamente. Para generar el pseudocódigo es necesario rellenar los atributos del objeto “Pseudocode” (Figura 38) para luego ser empleado por sus métodos cuando se genere el pseudocódigo. En caso de que

no se indique lo contrario, las instrucciones se irán agregando al listado “Lines”. Lo primero a evaluar es el primer bloque del diagrama, si esta incluye el texto “START” o “INICIO” se tomará como una función principal (main), en otro caso se tomará como que es el nombre de una función. Por ello, se incluirá la etiqueta “<BASE_FUNC>” en caso de ser la función principal o la etiqueta “<FUNC>” en el caso contrario.

A continuación, busca el siguiente bloque y evalúa cuantos bloques le siguen y pueden pasar tres cosas:

1. Solamente tiene un bloque a continuación, con lo que se pasa a evaluar el bloque.
2. Tiene más de un bloque a continuación, esto quiere decir que se trata de un bloque de tipo “decisión”, lo que indica que se trata de un bloque condicional. En este caso, se agrega la etiqueta “<CONDITIONAL”> y a continuación el texto dentro del bloque (con él ANTLR4 decidirá de que tipo de condicional se trata). Además, por cada una de las ramas de condicional se incluirá la etiqueta “<CONDITION>” junto con el texto de la condición del bloque incluida en el diccional “Next_Block_Conditionals” (Figura 14) y se irán evaluando los bloques de la rama. Al finalizar cada rama se incluirá la etiqueta “</CONDITION>”, y una vez finalizadas todas se incluirá la etiqueta “</CONDITIONAL>” para indicar el final del condicional.
3. No tiene más bloques a continuación, por lo que, si todo ha ido bien, también será de tipo “start_end” y significará que se ha llegado al final del diagrama de flujo. En este caso se agrega la etiqueta “<END>” para indicar el final del programa.

Para la evaluación de los bloques, lo primero que se hace es ver de que tipo de bloque se trata, para a continuación realizar las operaciones pertinentes:

- **Bloque de tipo “scan”:** incluye una etiqueta de tipo “<SCAN>” junto con el nombre de la variable. En caso de que la variable no esté incluida en el diccionario de variables se incluye y se establece de tipo “string”.
- **Bloque de tipo “print”:** incluye una etiqueta de tipo “<PRINT>” y a continuación el nombre de la variable. En caso de que la variable no esté incluida en el diccionario de variables se incluye y se establece de tipo “string”.
- **Bloque de tipo “process”:** en este tipo de bloques lo primero que se hace es dividir el texto del bloque en un list con una posición por cada palabra o símbolo matemático o asignación. Lo que hace a continuación es iterar sobre ese listado e ir incorporando una línea con todos los elementos, pero si se trata de una variable (se hace la comprobación con una búsqueda con una expresión regular), si está incluida en el

diccionario de variables se cambia su valor a “int” y si no lo está se agrega con valor “int”.

Una vez generado el objeto “Pseudocode” ya solo queda generar el pseudocódigo para que ANTLR4 pueda interpretarlo. Para realizar este paso lo que hace el programa es unir en una cadena de texto todas las líneas del array de pseudocódigo del paso anterior. Cabe reseñar que el sistema agrega un salto de línea tras cada elemento del array para que un humano lo pueda interpretar correctamente, pero si se dejase en una única línea funcionaría igual ya que ANTRL4 se ha configurado (se explicará en el subapartado 5.2) para no contar ese carácter como un token válido, por lo que no lo interpretará. Además, después de la primera línea itera por el diccionario de variables agregándolas de forma que tengan la etiqueta “<VAR_DECLARATION>” seguida del id y finalmente el tipo de la misma.

En este punto ya está listo el pseudocódigo para ser interpretado por ANTLR4.

5.2. Interpretación del pseudocódigo y generación del árbol de sintaxis abstracta

Para realizar este proceso, lo primero que hay que generar es un fichero en el que se incluya la gramática que evaluará ANTLR4. Este fichero tiene extensión “g4” y no se puede emplear de forma directa, si no que cuando esté listo habrá que compilarlo.

El fichero de gramática consta de dos partes, una primera que especifica la gramática regular y una segunda en la que se especifica la gramática libre de contexto. En el momento en el que se programa este fichero es importante el orden ya que cuando ANTLR4 realice la interpretación lo realiza de forma secuencial.

La parte de la gramática regular se especifica de la siguiente forma: primero se coloca el nombre del token que va a generar ANTLR4 y después que valores dentro del string serán los que tomen ese token. Estos valores pueden ser fijos (como las etiquetas que se han incorporado al pseudocódigo), pueden ser una elección entre uno u otro o un patrón de una expresión regular (regex). Además, también se pueden especificar tokens que una vez reconocidos, estos sean desechados (como los saltos de línea que se han comentado en el subapartado 5.1) enviándolos a una tubería “skip”. Un ejemplo de esta gramática se puede observar en la Figura 39.

```
VARIABLE_DECLARATIONS : '<VAR_DECLARATION>';
VARIABLE_TYPE : '<INT>' | '<STRING>';
ID : [a-zA-Z_][a-zA-Z0-9_]*;
WS : [ \t\n]+ -> skip;
ANY: .;
```

Figura 39: Ejemplo de gramática regular ANTLR4

La siguiente parte es la de la gramática libre de contexto. Esta se especifica indicando una raíz para a continuación indicar los tokens que la componen para, finalmente, indicar el id que tendrá en el árbol de sintaxis concreta. Atendiendo a la Figura 40, se puede ver como en la línea 24 se especifica la raíz “var”. Esta a su vez apunta hacia dos raíces: “var_decl” y “var_assign”, a las que se les ha asignado los id “Var_A” y “Var_B”. En la línea 28 se puede comprobar como la raíz “var_decl” (id “Var_A”) es un nodo final del árbol. Este se forma con el token “VARIABLE_DECLARATIONS” + expr (otra raíz) + el token “VARIABLE_TYPE”, que como podemos observar en la Figura 37 se forma con la cada de caracteres “<INT>” o “<STRING>”.

```
24  var : var_decl      # Var_A
25      | var_assign    # Var_B
26      ;
27
28  var_decl: VARIABLE_DECLARATIONS expr VARIABLE_TYPE;
29  var_assign: expr ASSIGN expr;
30
31  math_op: expr ASSIGN expr (PLUS | MINUS | MULT | DIV) expr      # Math_Operation
32         | expr (PLUS | MINUS | MULT | DIV) ASSIGN expr           # Math_Operation_Simplified
33         ;
```

Figura 40: Ejemplo de gramática libre de contexto ANTLR4

Una vez codificado el fichero de gramática, el siguiente paso es realizar su compilación para su posterior uso en el programa. Esto es tan sencillo como usar el fichero jar que se puede descargar desde su web (36). Cabe destacar que, pese a que esté desarrollado para su uso con java, el equipo de desarrollo ha desarrollado una conversión que lo permite emplear con los siguientes lenguajes:

- Java
- C#
- Python 2 y 3
- JavasScript
- Go
- C++
- Swift
- PHP
- Dart

Tras la compilación del fichero de gramática, se generan una serie de ficheros necesarios para su implementación en el programa. De todos ellos se emplearán directamente tres: lexer, parser y visitor. Estos son los pasos que necesita ANTLR4 para su funcionamiento. Con el lexer se genera el flujo de tokens, el parser lo analiza y genera el árbol de sintaxis concreta para finalmente recorrerlo con el visitor.

5.3. Análisis del pseudocódigo por parte de ANTLR4

Tal y como se ha indicado en el subapartado 5.2, inicialmente se interpreta el pseudocódigo por parte del lexer generando este el flujo de tokens. A continuación, este flujo de tokens se hace pasar por el parser, que genera el árbol de sintaxis concreta (en el Capítulo 7 hay dos ejemplos).

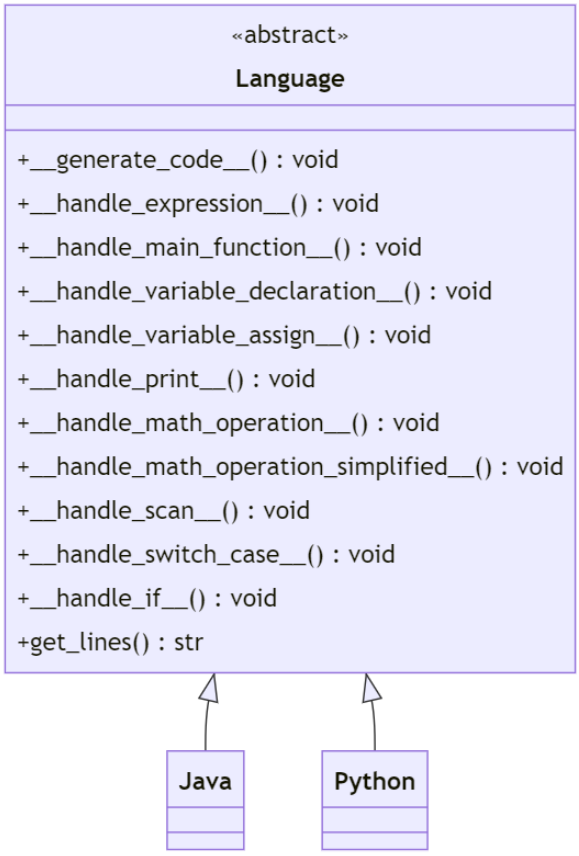


Figura 41: Implementación de la clase abstracta "Language"

En este punto del programa es cuando hay que codificar el analizador semántico empleando la clase “Visitor“, y para ello hay que implementar el patrón de diseño Visitor (se explicará en el subapartado 6.3). Para realizar esta implementación se ha generado una clase abstracta llamada “Lenguaje”, la cual debe ser implementada por parte de cada lenguaje de destino final (en el caso de este TFG, Python y Java) de modo que compartan la nomenclatura en los métodos para la generación del código fuente pero que esta implementación sea diferente para cada uno de los lenguajes (cada uno tiene su sintaxis).

Por otra parte se generarán dos clases que implementarán la clase “Visitor“ previamente compilada con ANTLR4, una de ellas será la encargada de interpretar la raíz del árbol de sintaxis concreta (AntlrToProgram) para posteriormente hacer que vaya recorriendo todo el árbol. E interpretándolo con la otra clase, que interpreta las expresiones (AntlrToExpression). Esta interpretación se realizará devolviendo instancias de las clases generadas para albergar la información del árbol de sintaxis concreta, generando así el árbol de sintaxis abstracta. Asimismo, todas estas clases implementan una clase abstracta llamada “Expression”, de forma que se pueden llamar por su superclase y luego diferenciarlas según su tipo de instancia.

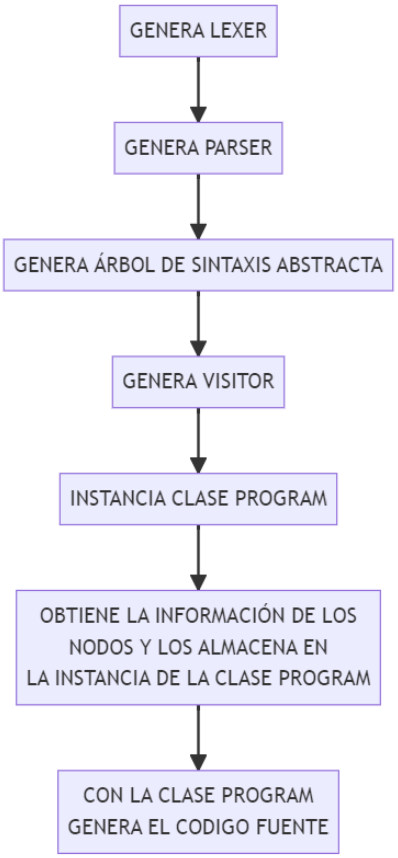


Figura 42: Diagrama de flujo de la generaicón del codigo fuente

Para realizar el proceso en el que se obtiene la información de los nodos, el programa realiza una instancia de la clase que guardará la información del nodo base del árbol de sintaxis concreta. A continuación, va recorriendo los nodos del árbol e irá instanciando las clases heredadas de “Expression” mientras las agrega a un listado para luego acceder a ellas. En este punto ya está completamente implementado el patrón Visitor, generando un acceso ágil y sencillo a la información sin haber tocado el código que las alberga. Para una mejor comprensión se incluye una muestra de esta implementación en la Figura 43.

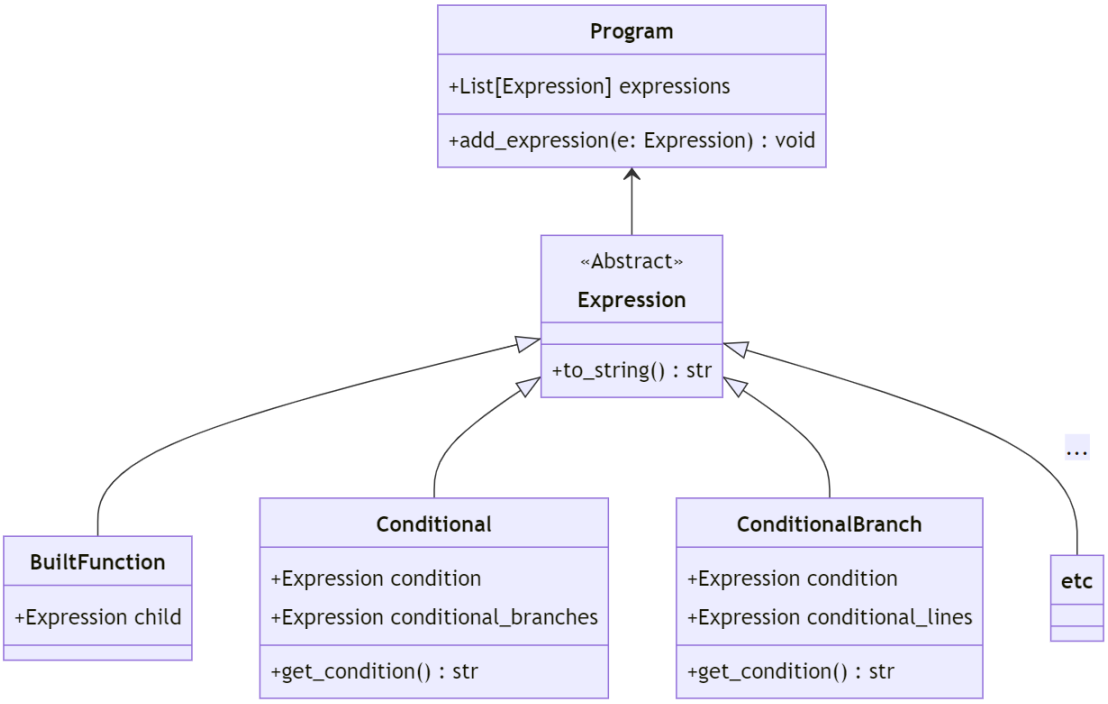


Figura 43: Implementación UML de los datos del patrón Visitor

Una vez se ha extraído la información del árbol de sintaxis concreta generando así el árbol de sintaxis abstracta, el programa ya está en disposición de generar el código fuente. Para ello realiza una instancia del lenguaje seleccionado (en este caso, Java o Python) y a continuación comienza a iterar por todas las líneas de expresiones que contiene la instancia de la clase “Program”. Por cada una de estas expresiones evalúa que tipo de instancia es (se puede observar en la Figura 43) y en función del tipo, realiza las operaciones oportunas para agregar al código fuente la sintaxis correcta en función del lenguaje (Figura 44).

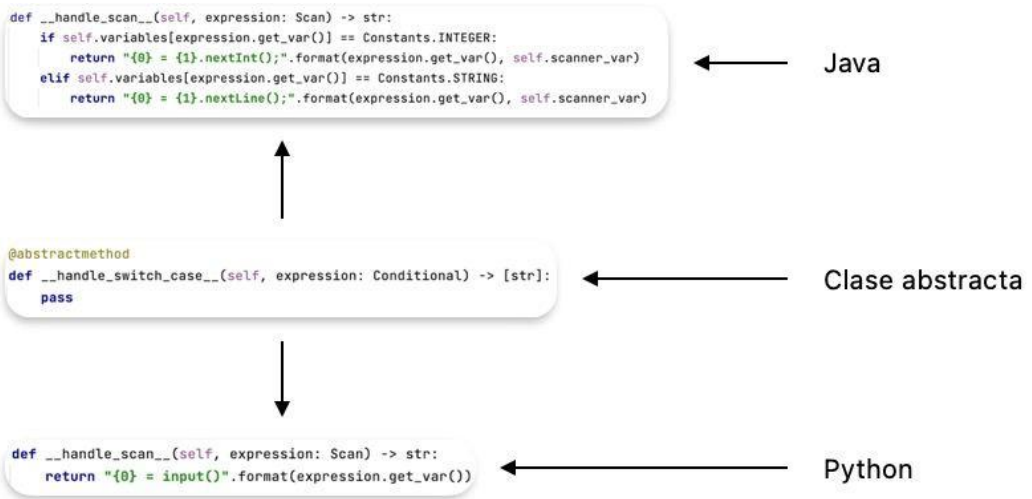


Figura 44: Interpretación de la expresión en función del lenguaje de destino

Finalmente se exporta el código fuente y se almacena en la carpeta “./User Files/Output Files” con el nombre de la imagen de entrada y la extensión correcta para el lenguaje seleccionado.

6. Diseño y desarrollo

6.1. Flujo general del programa

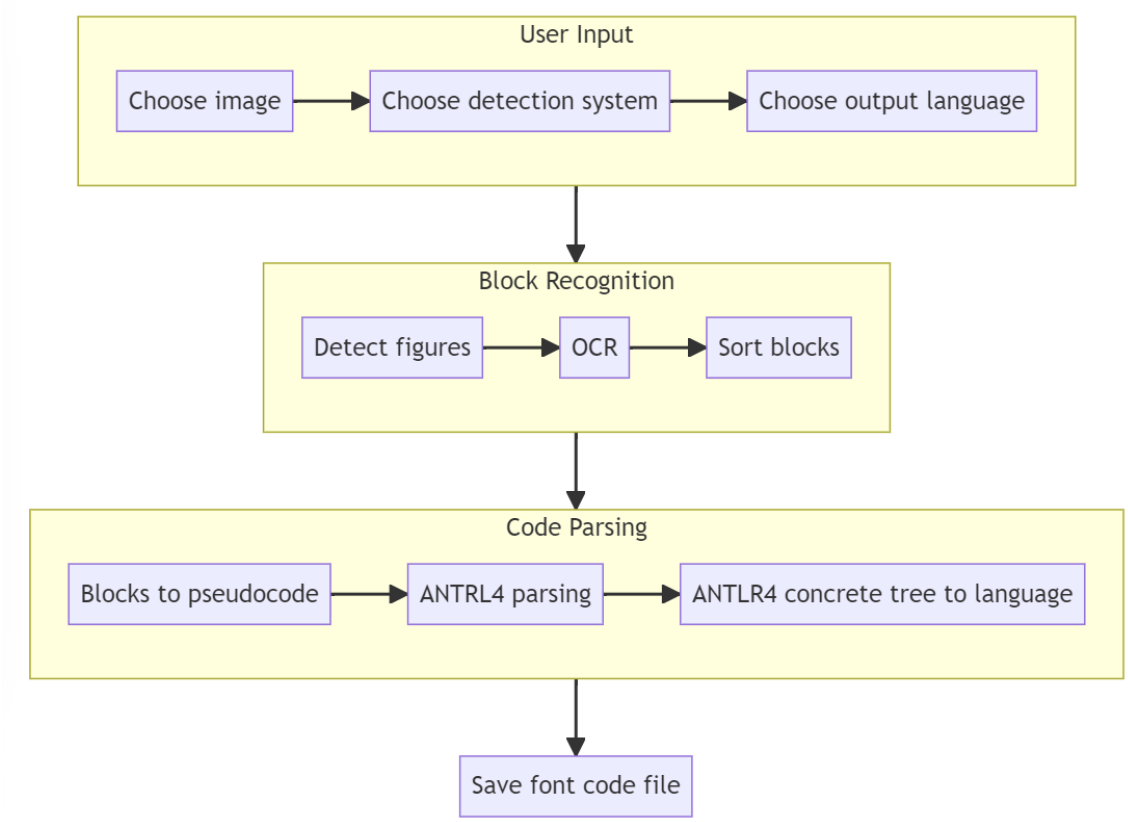


Figura 45: Flujo general del programa

Una vez explicados los pasos en detalle se puede explicar el flujo general del programa según la Figura 45.

Antes de comenzar la ejecución del programa, el usuario debe poner las imágenes a evaluar en la carpta *"./User Files/Input Images"* para a continuación, ejecutar el script "main.py" desde la consola de Python. Una vez realizado este paso, el usuario deberá seleccionar la imagen que quiere evaluar seguido del sistema de detección OCR que va a emplear y finalmente el lenguaje de programación al que busca convertir el diagrama de flujo.

Una vez realizadas estas elecciones, el programa continuará con el siguiente paso en el que se engloban el reconocimiento de las figuras (Capítulo 2), el reconocimiento del texto manuscrito (Capítulo 3) y finamente las operaciones de ordenación de todos estos elementos (Capítulo 4). A continuación, realizará las operaciones necesarias para realizar la conversión de los bloques reconocidos a código fuente (Capítulo 5) y finalmente guardará el fichero con el código fuente en la carpeta "./User Files/Output Files".

6.2. Principios SOLID

Durante el desarrollo se han intentado seguir los principios de desarrollo SOLID (37), definidos por Robert C. Martin (38) ya que estos aseguran un código más sencillo de mantener y escalar.

Constan de los siguientes elementos:

- **S** – *Principio de responsabilidad única*: Un objeto solamente debe realizar un único cometido.
    - Un ejemplo de este principio se puede observar en el fichero “Image_Correction.py”, cuya función es la de realizar modificaciones en las imágenes.
- **O** – *Principio de Abierto/Cerrado*: Una clase debe poder ser extendida pero no modificada.
    - No hay ejemplos en el proyecto de este principio.
- **L** – *Principio de sustitución de Liskov*: Si una clase es heredada por otra, la clase hija no debe alterar el comportamiento de la clase padre en ningún momento.
    - No hay ejemplos en el proyecto de este principio.
- **I** – *Principio de segregación de interfaces*: Ninguna clase debe implementar métodos que no usa, por ello se deben emplear interfaces.
    - Un ejemplo se puede observar en los ficheros “Language.py”, “Java.py” y “Python.py”. En ellos observamos como las clases empleadas para realizar las operaciones de conversión de pseudocódigo a código fuente heredan de la clase abstracta “Language”. Esta clase abstracta incluye los métodos que deben de emplear las clases que la heredan, de forma que estas tienen la utilidad que necesitan.
- **D** – *Principio de inversión de dependencias*: El código del núcleo de la aplicación no debe depender de las implementaciones de los módulos que componen las funcionalidades.
    - Un ejemplo es el del fichero “Main.py”. Cuando este llama al método “detect_blocks” del fichero “BlockDetection.py” lo que hace es esperar que le devuelva el listado de bloques junto con sus textos sin importarle el cómo.

6.3. Patrón visitor

En ingeniería de software es muy común emplear patrones de diseño para solucionar problemas recurrentes. Durante el desarrollo de este TFG ha sido necesaria la implementación del patrón visitor (39) para obtener la información generada por ANTLR4.

El patrón visitor basa su funcionamiento en la división entre los datos y la lógica del programa. Por una parte, existen una serie de clases que albergan la información (los nodos del árbol de sintaxis concreta) pero no tienen implementados métodos que permitan realizar operaciones. Y por otra parte están las clases que acceden a las clases que contienen la información, así como los métodos necesarios para manipularla.

En el caso del TFG se emplea por las clases que heredan de la clase “Expression” tal y como se puede observar en la Figura 43.

7. Resultados

El resultado de final del programa es una sucesión como la que sigue en la Figura 46, en la cual se muestran las pantallas por las que pasa el usuario hasta llegar a la generación del código fuente.

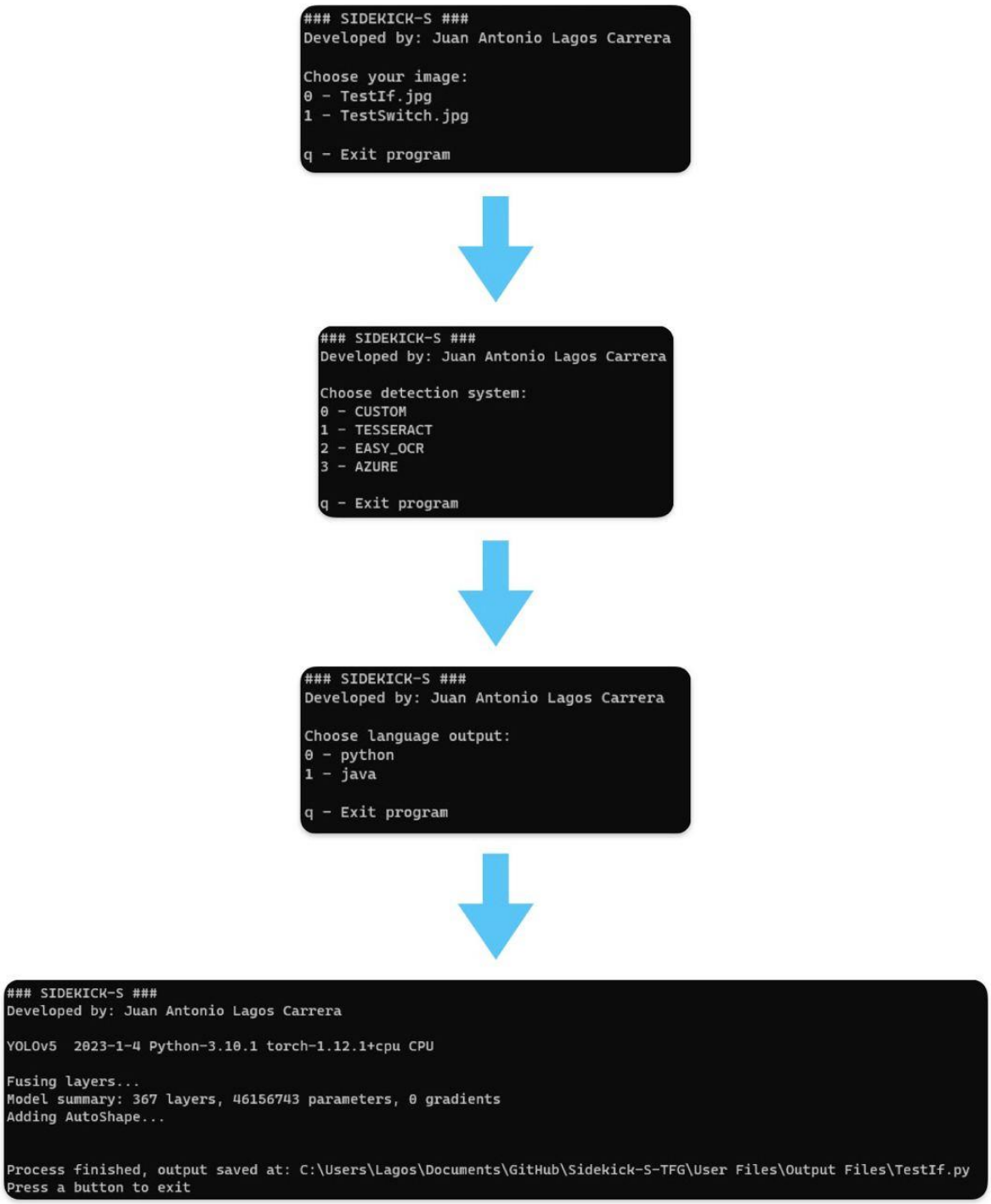


Figura 46: Pantallas del programa

A continuación se mostrarán dos ejemplos con las resultados del programa.

7.1. Ejemplo 1: (Test If)

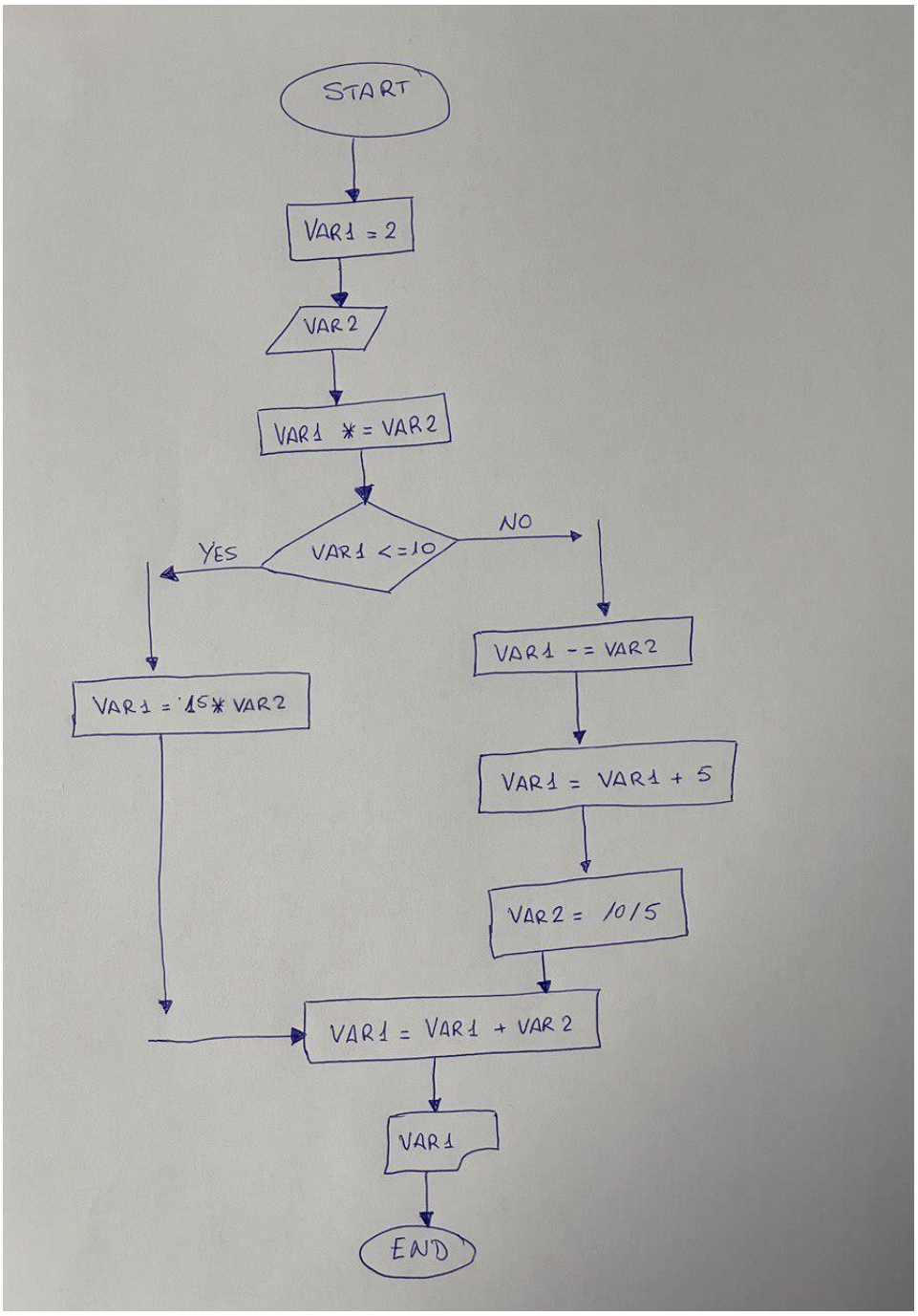


Figura 47: Test If

El sistema genera el siguiente pseudocódigo:

```
<BASE_FUNC>
        <VAR_DECLARATION>VAR1<INT>
        <VAR_DECLARATION>VAR2<INT>

        VAR1=2
        <SCAN>VAR2
        VAR1*=VAR2
        <CONDITIONAL> VAR1 <= 10
                <CONDITION> YES
                        VAR1=15*VAR2
                </CONDITION>
                <CONDITION> NO
                        VAR1-=VAR2
                        VAR1=VAR1+5
                        VAR2=10/5
                </CONDITION>
        </CONDITIONAL>
        VAR1=VAR1+VAR2
        <PRINT>VAR1
<END>
```

Con ese pseudocódigo, ANTLR4 genera el siguiente árbol de sintaxis concreta:

```
(prog
        (function (main_function <BASE_FUNC>))
        (var (var_decl <VAR_DECLARATION> (expr VAR1) <INT>))
        (var (var_decl <VAR_DECLARATION> (expr VAR2) <INT>))
        (var (var_assign (expr VAR1) = (expr 2)))
        (function (built_function (scan <SCAN> (expr VAR2))))
        (math_op (expr VAR1) * = (expr VAR2))
        (conditional <CONDITIONAL>
                (condition (expr VAR1) <= (expr 10))
                (conditional_branches
                        (conditional_branch <CONDITION> (expr YES)
                                (conditional_lines
                                        (math_op (expr VAR1) = (expr 15) * (expr VAR2)))
                                                                </CONDITION>)
                        (conditional_branch <CONDITION> (expr NO)
                                (conditional_lines
                                        (math_op (expr VAR1) - = (expr VAR2))
                                        (math_op (expr VAR1) = (expr VAR1) + (expr 5))
                                        (math_op (expr VAR2) = (expr 10) / (expr 5)))
                                                                </CONDITION>))
        </CONDITIONAL>)
        (math_op (expr VAR1) = (expr VAR1) + (expr VAR2))
        (function (built_function (print <PRINT> (expr VAR1))))
<EOF>)
```

Y finalmente, se obtiene el código fuente para los lenguajes Python y Java:

- **Python:**

```
def main():
	VAR1: int = None
	VAR2: int = None
	VAR1 = 2
	VAR2 = input()
	VAR1 *= VAR2
	if VAR1 <= 10:
		VAR1 = 15 * VAR2
	else:
		VAR1 -= VAR2
		VAR1 = VAR1 + 5
		VAR2 = 10 / 5
	VAR1 = VAR1 + VAR2
	print(VAR1)
```

- **Java:**

```
import java.util.Scanner;

public static void main(String[] args) {
	Scanner scan = new Scanner(System.in);
	int VAR1 = null;
	int VAR2 = null;
	VAR1 = 2;
	VAR2 = scan.nextInt();
	VAR1 *= VAR2;
	if (VAR1 <= 10) {
		VAR1 = 15 * VAR2;
	} else {
		VAR1 -= VAR2;
		VAR1 = VAR1 + 5;
		VAR2 = 10 / 5;
	}
	VAR1 = VAR1 + VAR2;
	System.out.println(VAR1);
}
```

7.2. Ejemplo 2: (Test Switch)

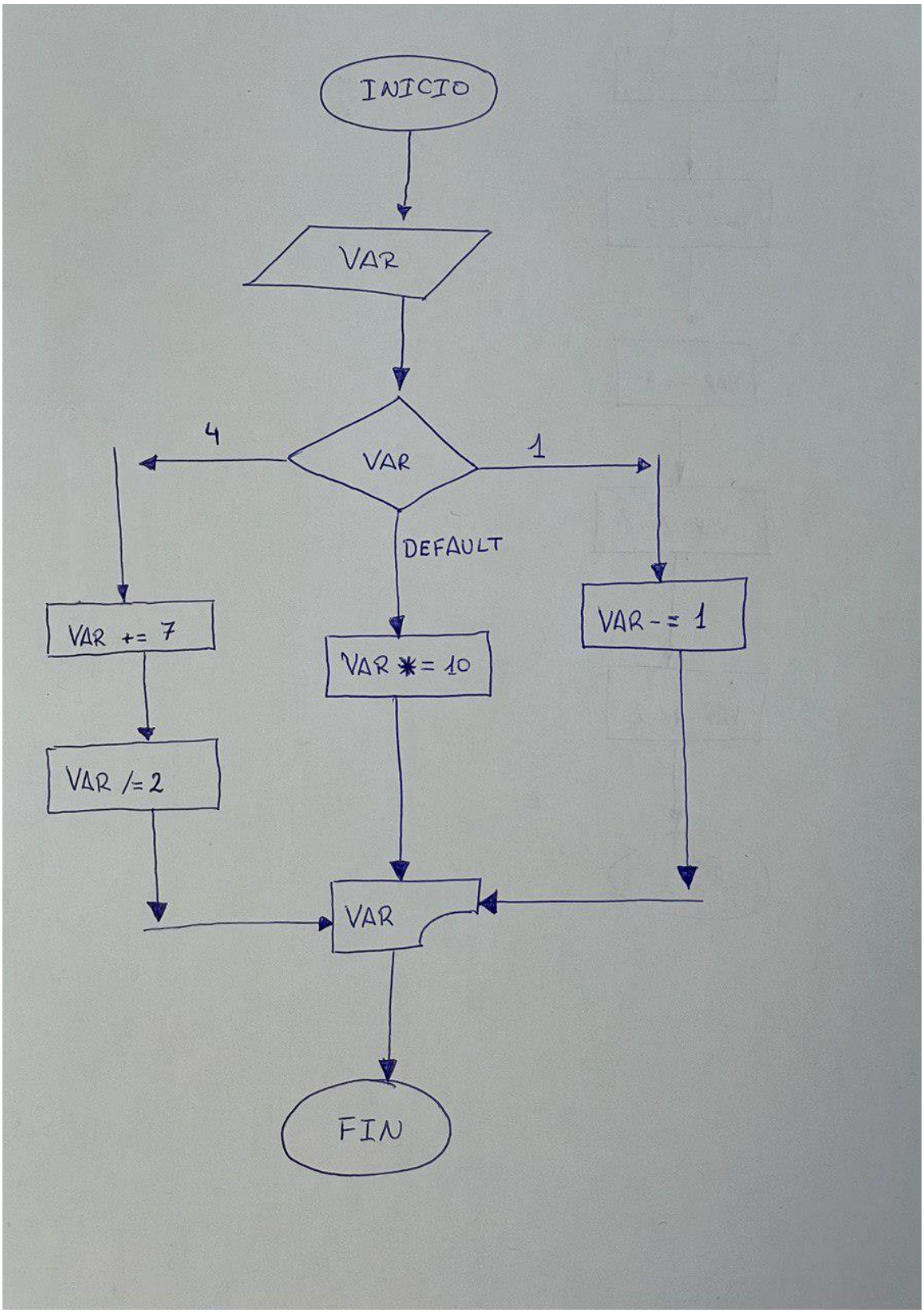


Figura 48: Test Switch

El sistema genera el siguiente pseudocódigo:

```
<BASE_FUNC>
        <VAR_DECLARATION>VAR<INT>

        <SCAN>VAR
        <CONDITIONAL> VAR
                <CONDITION> DEFAULT
                        VAR*=10
                </CONDITION>
                <CONDITION> 1
                        VAR-=1
                </CONDITION>
                <CONDITION> 4
                        VAR+=7
                        VAR/=2
                </CONDITION>
        </CONDITIONAL>
        <PRINT>VAR
<END>
```

Con ese pseudocódigo, ANTLR4 genera el siguiente árbol de sintaxis concreta:

```
(prog
        (function (main_function <BASE_FUNC>))
        (var (var_decl <VAR_DECLARATION> (expr VAR) <INT>))
        (function (built_function (scan <SCAN> (expr VAR))))
        (conditional <CONDITIONAL>
                (condition (expr VAR))
                (conditional_branches
                        (conditional_branch <CONDITION> (expr DEFAULT)
                                (conditional_lines
                                        (math_op (expr VAR) * = (expr 10)))
                                                                </CONDITION>)
                                        (conditional_branch <CONDITION> (expr 1)
                                                (conditional_lines
                                                        (math_op (expr VAR) - = (expr 1)))
                                                                </CONDITION>)
                                        (conditional_branch <CONDITION> (expr 4)
                                                (conditional_lines
                                                        (math_op (expr VAR) + = (expr 7))
                                                        (math_op (expr VAR) / = (expr 2)))
                                                                </CONDITION>))
        </CONDITIONAL>)
        (function (built_function (print <PRINT> (expr VAR))))
<EOF>)
```

Y finalmente, se obtiene el código fuente para los lenguajes Python y Java.

- **Python:**

```python
def main():
	VAR: int = None
	VAR = input()
	if VAR == 1:
		VAR -= 1
	elif VAR == 4:
		VAR += 7
		VAR /= 2
	else:
		VAR *= 10
	print(VAR)
```

- **Java:**

```java
import java.util.Scanner;

public static void main(String[] args) {
	Scanner scan = new Scanner(System.in);
	int VAR = null;
	VAR = scan.nextInt();
	switch (VAR) {
		case 1:
			VAR -= 1;
			break;
		case 4:
			VAR += 7;
			VAR /= 2;
			break;
		default:
			VAR *= 10;
	}
	System.out.println(VAR);
}
```

8. Conclusiones y trabajos futuros

Una vez finalizado el desarrollo del software, se extraen varias conclusiones:

- La primera y más importante en un software de este tipo es sobre el procesado de la imagen previo a ser inferenciada, tanto para la búsqueda de objetos como para detección OCR. Si la imagen es procesada correctamente se mejora notablemente la probabilidad de acierto en las predicciones de los modelos. Un punto muy a su favor que he notado es el paso a escala de grises y posterior binarización de la imagen. Eliminando los canales RGB y quedando solamente el canal de escala de grises, el modelo de ML solamente tiene que evaluar 1/3 de los datos originales. Además, con la binarización se exageran las diferencias y la imagen pasa a tener únicamente la información necesaria.

- En caso de ser necesario para un proyecto realizar una identificación de objetos o clasificación de imágenes no es necesario desarrollar una red neuronal, existen multitud de redes neuronales ya configuradas listas para ser adiestradas (como YOLOv5). Cuando se comenzó la realización de este TFG, YOLO se encontraba en su versión 5, y en su conclusión, ya se encontraba en la versión 9, por lo que es muy posible que en la actualidad estas predicciones estén todavía más mejoradas.

- Otro punto para destacar es que los mecanismos de detección de texto manuscrito no funcionan correctamente, pero por contraposición, funcionan muy bien interpretando texto escrito a máquina en las imágenes.

- En cuanto al parser, se ha observado que ANTLR tiene una potencia muy grande no solo para realizar conversión e interpretación de texto. ANTLR4 funciona interpretando un flujo de datos entrante, si este flujo es de otro tipo y está correctamente configurado el fichero de gramática, también es posible realizar esas interpretaciones. Entre los usos que se intuyen interesantes están la conversión de ficheros CSV y hojas de cálculo.

Los resultados del TFG han sido mejores que los esperados. La potencia de ANTLR ha permitido que el programa pueda escalar con la generación de nuevos lenguajes de forma relativamente rápida. Esto se pudo comprobar cuando se generó el fichero de conversión a Java, el tiempo fue de unas 3 horas, mientras que un sistema empleado antes de ANTLR costaba más de dos días la codificación de un nuevo lenguaje.

Pero no todo ha sido lo esperado. En el desarrollo del sistema OCR se ha empleado más de un mes cuando lo proyectado fue de dos semanas. Además de este tiempo extra empleado, los resultados no fueron satisfactorios en el estado actual del sistema, por lo que si se busca que sea de utilidad en un futuro, habrá que realizar modificaciones en él. Derivado de este tiempo extra y ya que el sistema OCR es crítico para el funcionamiento del programa, se ha decidido emplear un sistema OCR en la nube pese a que no era lo esperado. Cabe decir que los resultados de este sistema son muy buenos y mejora mucho el software, pero no era lo que se buscaba en un inicio ya que se buscaba un sistema standalone que no dependiese de la conexión a internet.

Con los conocimientos adquiridos y las lecciones aprendidas durante la realización de este software, se plantearán las siguientes modificaciones en un futuro:

- Mejorar el sistema OCR propio a través de mecanismos de heurística como se explicó en el subapartado 3.2. Otro sistema de mejora sería aplicar un diccionario de forma que las palabras se interpreten por similitud.

- En cuanto a la arquitectura de software, mejorar la herencia en las clases ya que ha habido algunas que podrían haberse implementado mediante clases abstractas como pueden ser las clases “Letter” y “Text”.

- Otra modificación es la generación de una API para su empleo por plataformas web. Pero no solamente con ese fin, debido a los requisitos de bibliotecas, se planteará su distribución en contenedores como Docker. De esta forma se eliminan los requisitos de software preinstalado en las máquinas cliente.

- Trabajar en las limitaciones actuales del programa, intentando minimizarlas o eliminarlas.

Para concluir, otra línea de trabajo futura dentro de este software es la interpretación de diagramas UML manuscritos. Durante las pruebas de viabilidad de este TFG se valoró la posibilidad, llegando incluso a realizar una dataset (40) desde cero. El problema que hubo fue la baja precisión del modelo obtenido (Anexo 10), por lo que la solución a ello será ampliarla hasta llegar al punto de que tenga las muestras suficientes para adiestrar un buen modelo de detección de objetos.

9. Glosario

- **API**: Application Programming interface. Proporciona una interfaz de comunicación entre programas.
- **Base64**: es un esquema de codificación de binario a texto ASCII.
- **Bloque**: objeto de tipo “Block”.
- **IDE**: Integrated Development Environment, es un programa que proporciona servicios al desarrollador para facilitar su labor programado.
- **Inferencia**: es el proceso de predicción de un modelo de ML.
- **ML**: Machine Learning
- **OCR**: Optical Character Recognition, es un proceso mediante el cual se reconoce texto manuscrito.
- **Regex**: Es una expresión de texto regular que sirve para localizar trozos de texto dentro de un texto mayor.
- **URL**: Uniform Resource Locator. Es una dirección a un recurso web.

10. Bibliografía

1. What Is Object Detection? [Internet]. [citado 5 de octubre de 2022]. Disponible en: https://es.mathworks.com/discovery/object-detection.html
2. What Is Optical Character Recognition (OCR)? [Internet]. 2022 [citado 5 de octubre de 2022]. Disponible en: https://www.ibm.com/cloud/blog/optical-character-recognition
3. ANTLR [Internet]. [citado 6 de octubre de 2022]. Disponible en: https://www.antlr.org/
4. Medeiros RP, Ramalho GL, Falcão TP. A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education. IEEE Trans Educ. mayo de 2019;62(2):77-90.
5. Niyogisubizo J, Liao L, Nziyumva E, Murwanashyaka E, Nshimyumukiza PC. Predicting student's dropout in university classes using two-layer ensemble machine learning approach: A novel stacked generalization. Comput Educ Artif Intell. 1 de enero de 2022;3:100066.
6. Vanegas E, Salazar Y, Lerma P, Plaza I. Impact of the tutoring program on engineering student dropout. En: 2022 Congreso de Tecnología, Aprendizaje y Enseñanza de la Electrónica (XV Technologies Applied to Electronics Teaching Conference). 2022. p. 1-5.
7. Reading 18: Parser Generators [Internet]. [citado 19 de octubre de 2022]. Disponible en: https://web.mit.edu/6.005/www/fa15/classes/18-parser-generators/
8. Gamez MJ. Objetivos y metas de desarrollo sostenible [Internet]. Desarrollo Sostenible. [citado 11 de enero de 2023]. Disponible en: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/
9. Moran M. La Agenda para el Desarrollo Sostenible [Internet]. Desarrollo Sostenible. [citado 11 de enero de 2023]. Disponible en: https://www.un.org/sustainabledevelopment/es/development-agenda/
10. PyTorch [Internet]. [citado 12 de octubre de 2022]. Disponible en: https://www.pytorch.org
11. ultralytics/yolov5 [Internet]. Ultralytics; 2022 [citado 12 de octubre de 2022]. Disponible en: https://github.com/ultralytics/yolov5
12. Home [Internet]. OpenCV. [citado 12 de octubre de 2022]. Disponible en: https://opencv.org/
13. Servicios de informática en la nube | Microsoft Azure [Internet]. [citado 6 de octubre de 2022]. Disponible en: https://azure.microsoft.com/es-es/
14. TensorFlow [Internet]. [citado 12 de octubre de 2022]. Disponible en: https://www.tensorflow.org/
15. Bienvenido al Politécnico en Zacatecas - ZACATECAS [Internet]. [citado 14 de enero de 2023]. Disponible en: https://www.zacatecas.ipn.mx/
16. Flowchart 3b [Internet]. [citado 6 de octubre de 2022]. Disponible en: https://www.kaggle.com/datasets/davbetm/flowchart-3b
17. heartexlabs/labelImg [Internet]. Heartex; 2023 [citado 14 de enero de 2023]. Disponible en: https://github.com/heartexlabs/labelImg
18. Roboflow: Give your software the power to see objects in images and video [Internet]. [citado 14 de enero de 2023]. Disponible en: https://roboflow.com/

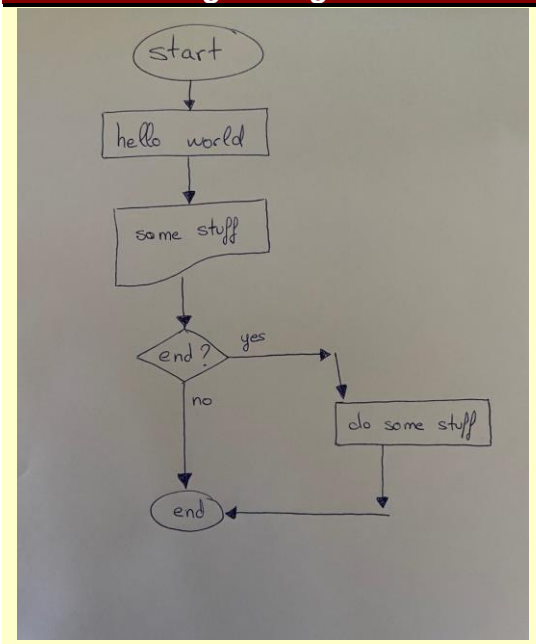
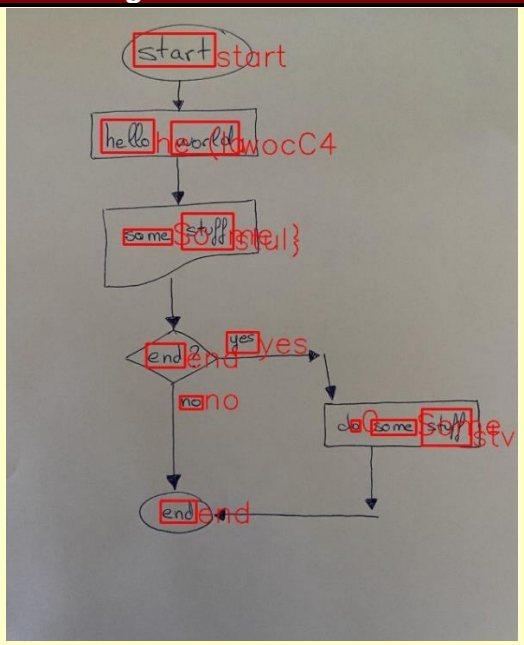
19. Lagos Carrera JA. Flowchart Dataset [Internet]. [citado 6 de octubre de 2022]. Disponible en: https://app.roboflow.com/yolo-umkl5/flowchart-etfvh/2
20. TensorFlow Lite | AA para dispositivos móviles y perimetrales [Internet]. TensorFlow. [citado 12 de octubre de 2022]. Disponible en: https://www.tensorflow.org/lite?hl=es-419
21. Google Colaboratory [Internet]. [citado 14 de enero de 2023]. Disponible en: https://colab.research.google.com/?hl=es
22. Lagos Carrera JA. Google Colaboratory - Flowchart recognition.ipynb [Internet]. [citado 14 de enero de 2023]. Disponible en: https://colab.research.google.com/drive/1PbsMMWdpfuK8LiKuL4XdT-TktOMdKi0s?hl=es#scrollTo=efDpXqhM8f1S
23. Train Custom Data [Internet]. Disponible en: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data
24. Overfit y underfit | TensorFlow Core [Internet]. TensorFlow. [citado 14 de enero de 2023]. Disponible en: https://www.tensorflow.org/tutorials/keras/overfit_and_underfit?hl=es-419
25. EasyOCR [Internet]. Jaided AI; 2022 [citado 12 de octubre de 2022]. Disponible en: https://github.com/JaidedAI/EasyOCR
26. Tesseract OCR [Internet]. tesseract-ocr; 2022 [citado 12 de octubre de 2022]. Disponible en: https://github.com/tesseract-ocr/tesseract
27. Spalla D. Recognizing handwriting with Tensorflow and OpenCV [Internet]. Deepnote. [citado 20 de enero de 2023]. Disponible en: https://deepnote.com/@davidespalla/Recognizing-handwriting-with-Tensorflow-and-OpenCV-cfc4acf5-188e-4d3b-bdb5-a13aa463d2b0
28. samuel100. Base de datos MNIST de dígitos manuscritos - Azure Open Datasets [Internet]. [citado 5 de octubre de 2022]. Disponible en: https://learn.microsoft.com/es-es/azure/open-datasets/dataset-mnist
29. A-Z Handwritten Alphabets in .csv format [Internet]. [citado 5 de octubre de 2022]. Disponible en: https://www.kaggle.com/datasets/sachinpatel21/az-handwritten-alphabets-in-csv-format
30. Handwritten math symbols dataset [Internet]. [citado 14 de enero de 2023]. Disponible en: https://www.kaggle.com/datasets/xainano/handwrittenmathsymbols
31. CUDA Zone - Library of Resources [Internet]. NVIDIA Developer. 2017 [citado 15 de enero de 2023]. Disponible en: https://developer.nvidia.com/cuda-zone
32. PatrickFarley. Inicio rápido: Reconocimiento óptico de caracteres (OCR) - Azure Cognitive Services [Internet]. [citado 15 de enero de 2023]. Disponible en: https://learn.microsoft.com/es-es/azure/cognitive-services/computer-vision/quickstarts-sdk/client-library
33. Subir Imágenes — Alojamiento De Imágenes [Internet]. ImgBB. [citado 15 de enero de 2023]. Disponible en: https://imgbb.com
34. Upload Image — Free Image Hosting [Internet]. ImgBB. [citado 15 de enero de 2023]. Disponible en: https://api.imgbb.com
35. HTTP request methods - HTTP | MDN [Internet]. [citado 15 de enero de 2023]. Disponible en: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
36. Download ANTLR [Internet]. [citado 18 de enero de 2023]. Disponible en: https://www.antlr.org/download.html

37. SOLID: The First 5 Principles of Object Oriented Design | DigitalOcean [Internet]. [citado 6 de octubre de 2022]. Disponible en: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design
38. UBC [Internet]. [citado 19 de enero de 2023]. Disponible en: http://cleancoder.com/products
39. Visitor [Internet]. [citado 19 de enero de 2023]. Disponible en: https://refactoring.guru/es/design-patterns/visitor
40. Lagos Carrera JA. UML Dataset [Internet]. [citado 6 de octubre de 2022]. Disponible en: https://app.roboflow.com/yolo-umkl5/uml/overview

11. Anexos

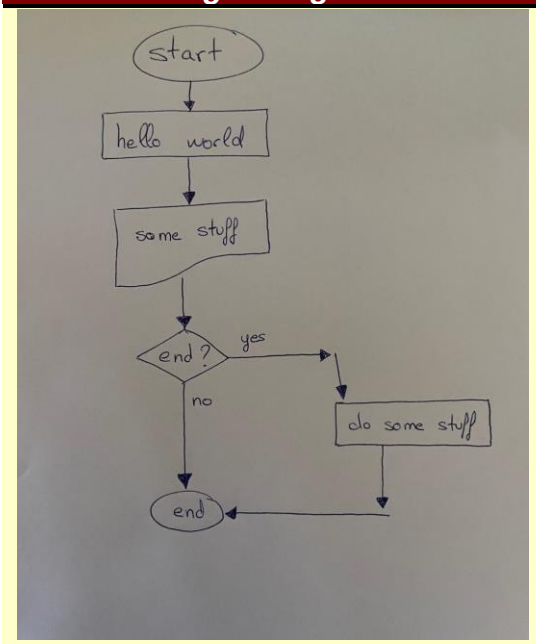
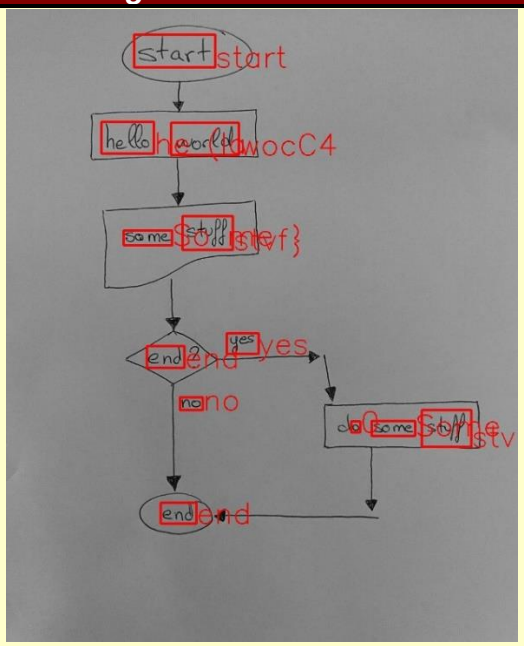
ANEXO 1: Pruebas de reconocimiento con EasyOCR

Prueba inicial sin ningún tipo de modificación en la imagen (letras minúsculas):

Imagen original	*Imagen con detecciones*
	

Textos originales	Textos detectados	%
start	start	0.95
hello world	he (la wocc4	0.57
some stuff	so me stul}	0.58
end?	end	0,83
do some stuff	0 some stvpp	0.66
end	end	0.98
yes	yes	0.65
no	no	0.99

Imagen con paso a escala de grises (letras minúsculas):

Imagen original	*Imagen con detecciones*
	

Textos originales	Textos detectados	%
start	start	0.95
hello world	he (la wocc4	0.51
some stuff	so me stvf}	0.57
end?	end	0.63
do some stuff	0 some stvpp	0.68
end	end	0.98
yes	yes	0.4
no	no	0.99

Imagen con escala de grises y ecualización (letras minúsculas):

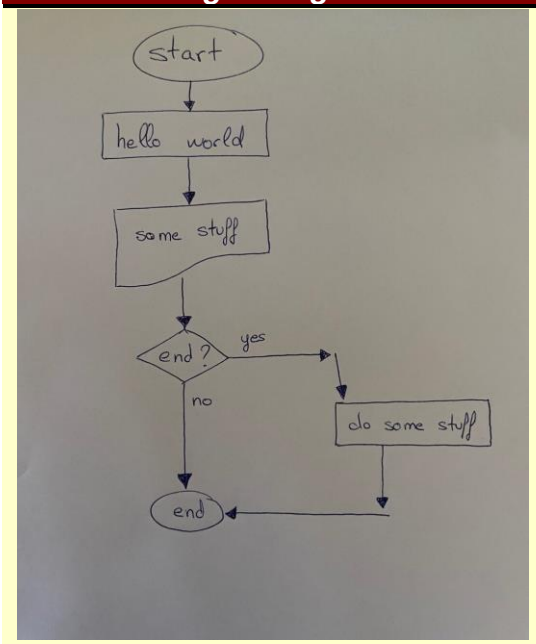
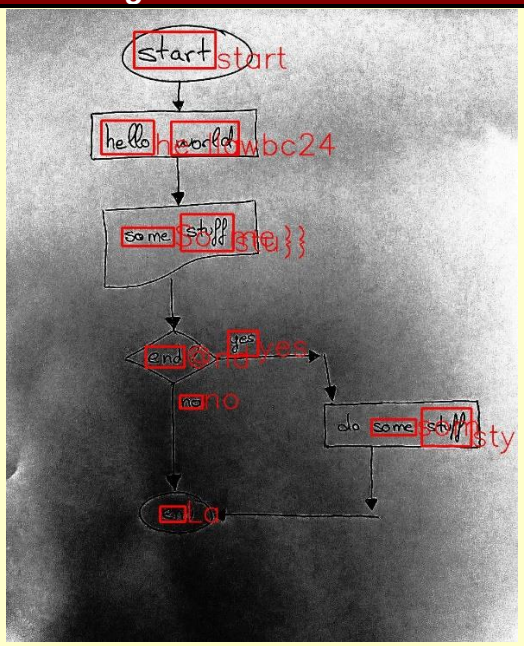
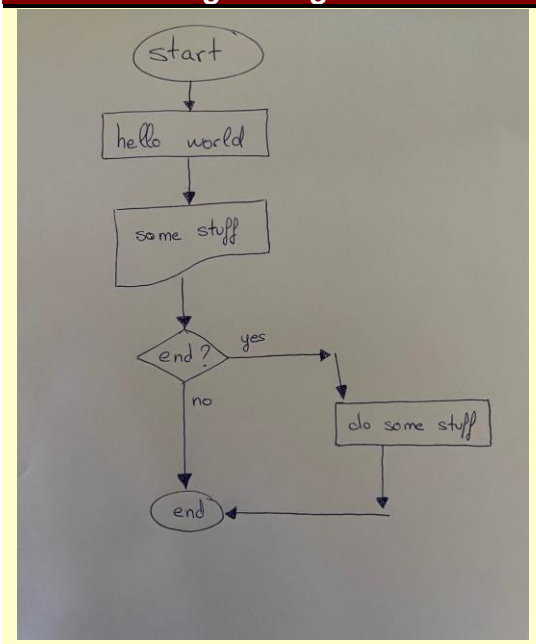
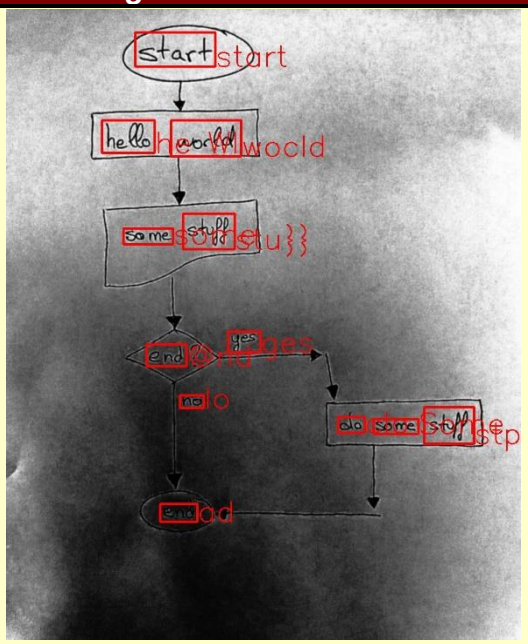
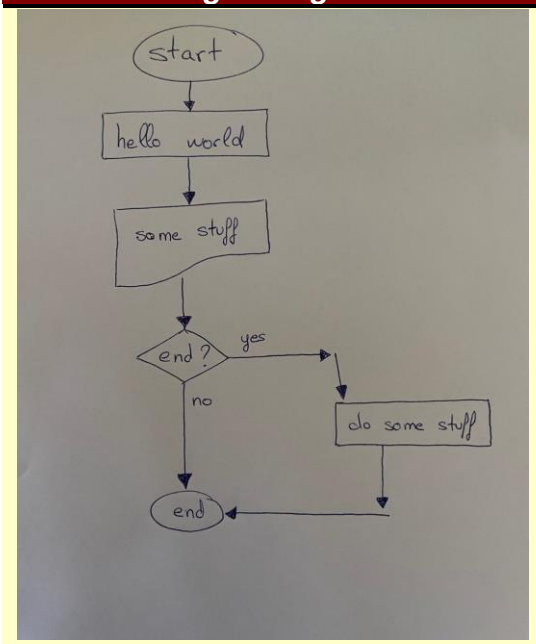
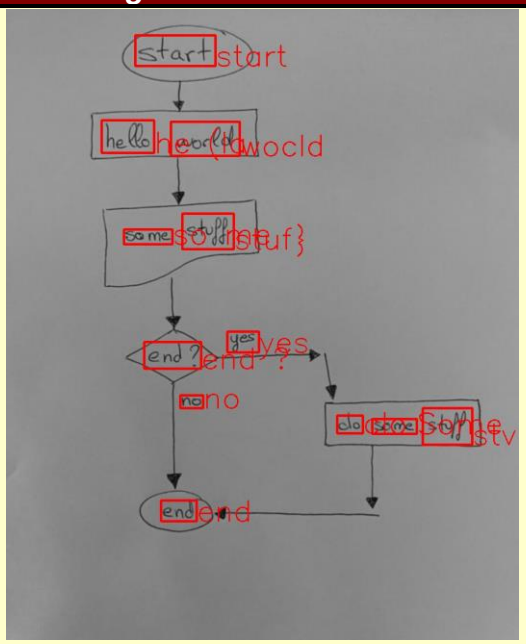
Imagen original	*Imagen con detecciones*	
		
Textos originales	**Textos detectados**	**%**
start	start	0.53
hello world	he lla wbc24	0.25
some stuff	so me stu}}	0.44
end?	@nd	0.97
do some stuff	som stylp	0.62
end	la	0.01
yes	yes	0.79
no	no	0.53

Imagen en escala de grises, ecualización y filtro gaussiano (letras minúsculas):

Imagen original	*Imagen con detecciones*
	

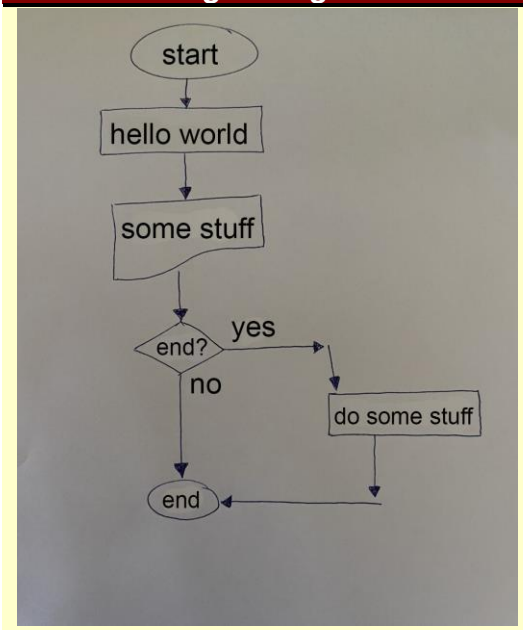
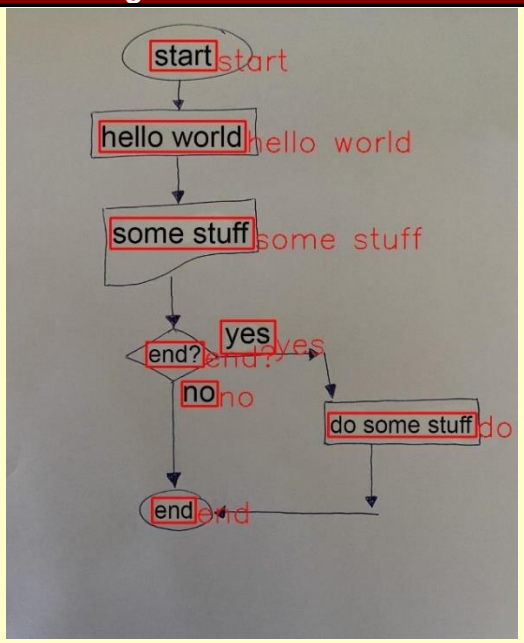
Textos originales	Textos detectados	%
start	start	0.92
hello world	he Wl wocld	0.66
some stuff	some stu}}	0.34
end?	@nd	0.76
do some stuff	clo Some stpp	0.7
end	ad	0.56
yes	ges	0.76
no	lo	0.75

Imagen en escala de grises y filtro gaussiano (letras minúsculas)**:**

Imagen original	***Imagen con detecciones***
	

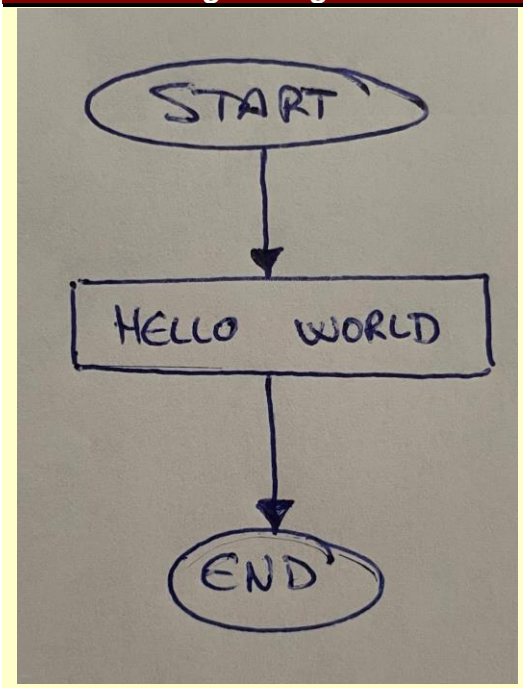
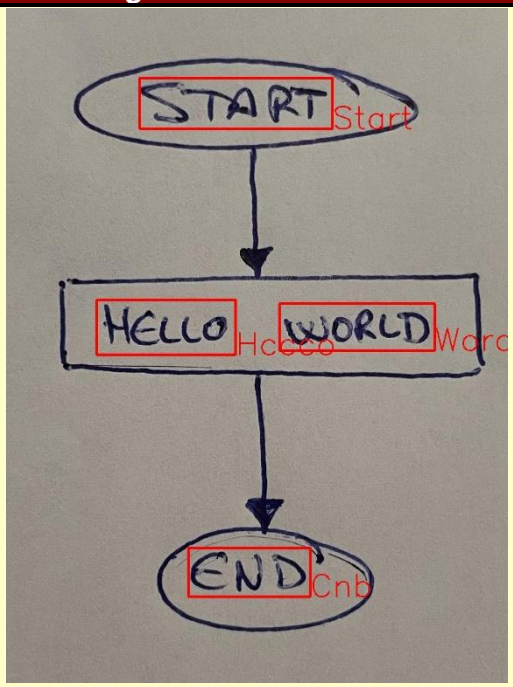
Textos originales	**Textos detectados**	**%**
start	start	0.89
hello world	he (la wocld	0.49
some stuff	so me stuf}	0.42
end?	end ?	0.72
do some stuff	clo Some stvpp	0.62
end	end	0.75
yes	Yes	0.59
no	no	0.99

Prueba sustituyendo el texto manuscrito por texto escrito a ordenador (letras minúsculas):

Imagen original	*Imagen con detecciones*
	

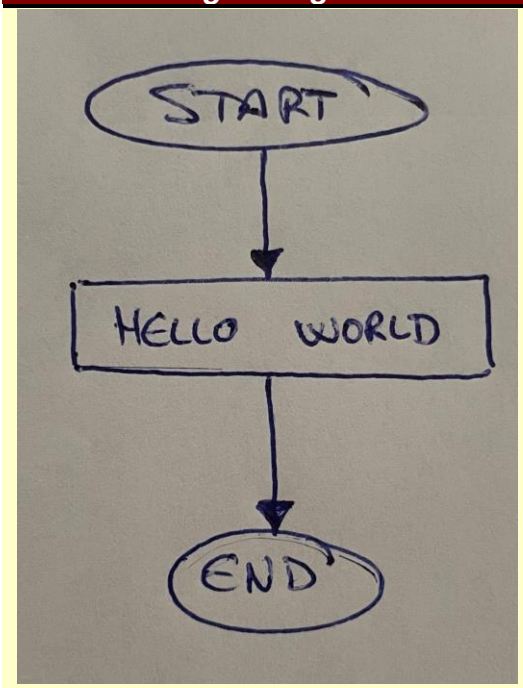
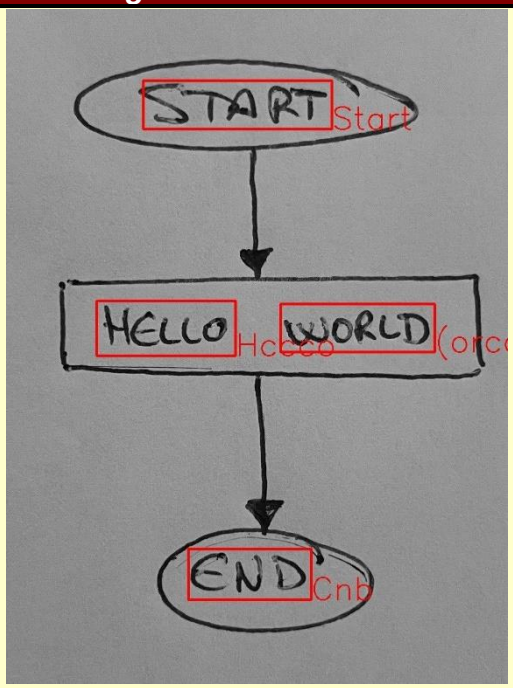
Textos originales	Textos detectados	%
start	start	0.99
hello world	hello world	0.99
some stuff	some stuff	0.87
end?	end?	0.99
do some stuff	do some stuff	0.94
end	end	0.99
yes	yes	0.99
no	no	0.99

Prueba inicial sin ningún tipo de modificación en la imagen (letras en mayúsculas):

Imagen original	*Imagen con detecciones*
	

Textos originales	**Textos detectados**	**%**
START	Start	0.88
HELLO WORLD	Hccco Worcd	0.35
END	Cnb	0.35

Imagen con paso a escala de grises (letras en mayúsculas):

Imagen original	*Imagen con detecciones*
	

Textos originales	**Textos detectados**	**%**
START	Start	0.92
HELLO WORLD	Hccco (orcd	0.44
END	Cnb	0.30

Imagen con escala de grises y ecualización (letras en mayúsculas):

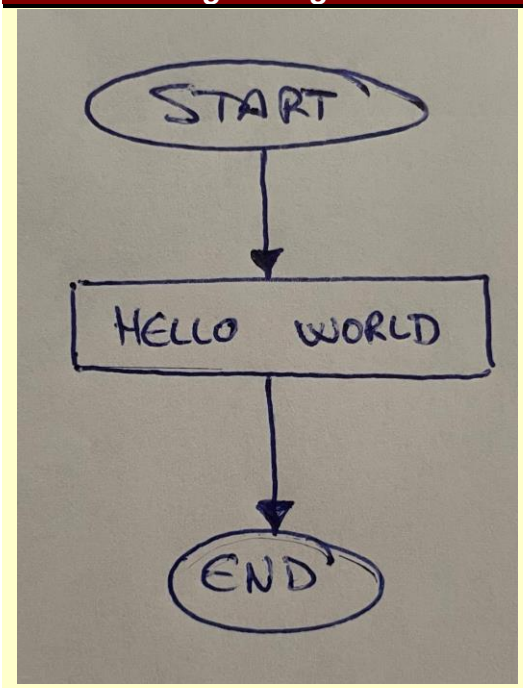
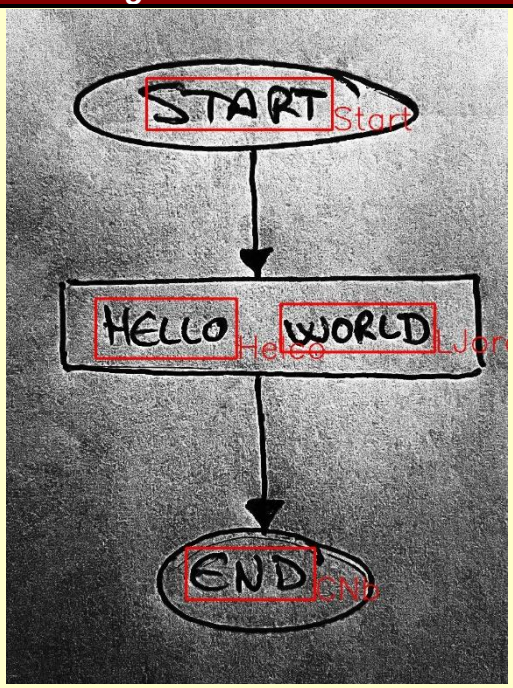
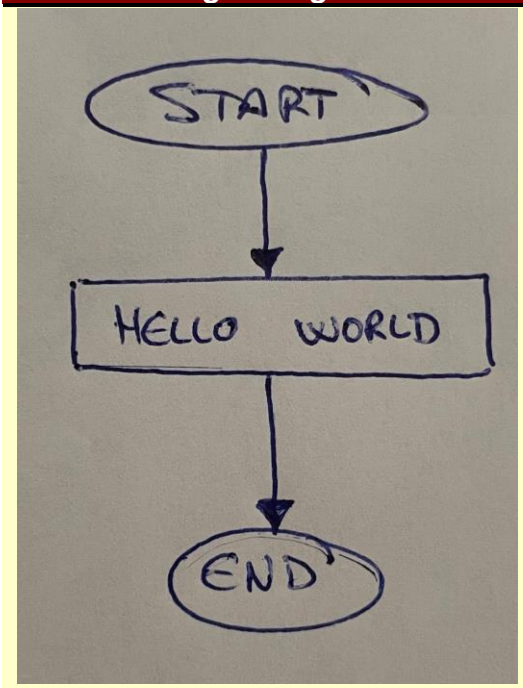
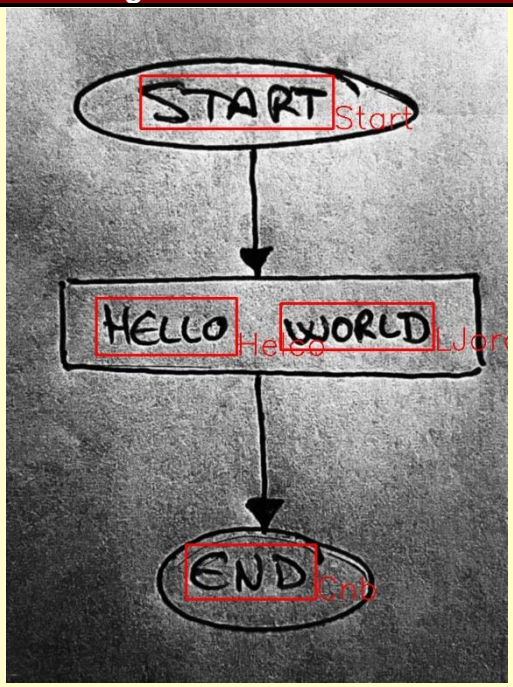
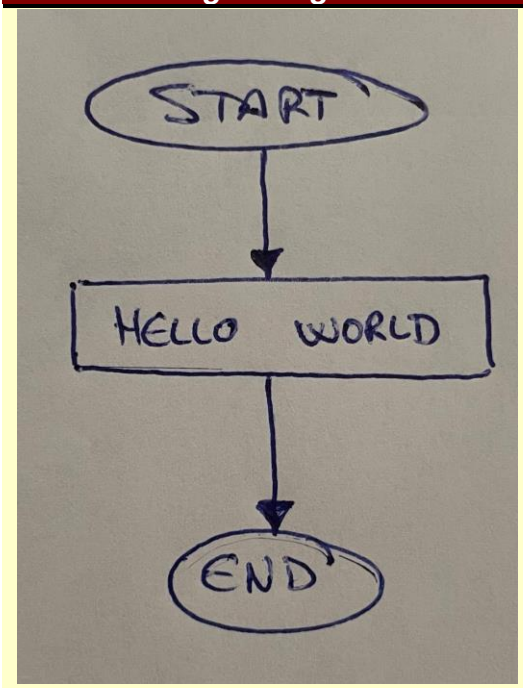
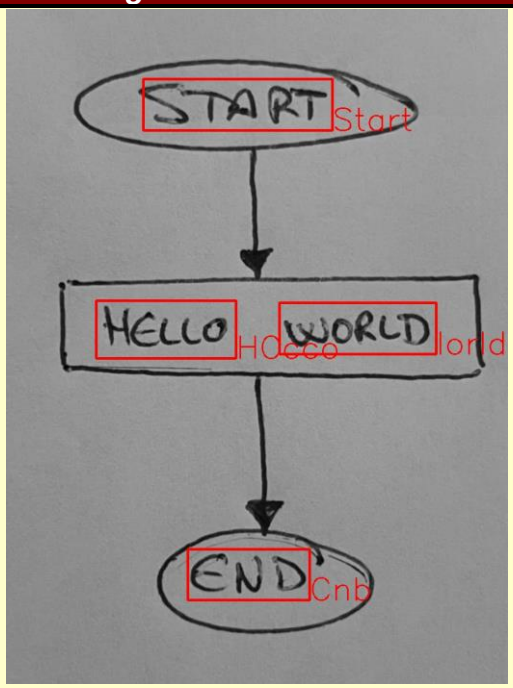
Imagen original	*Imagen con detecciones*	
		
Textos originales	**Textos detectados**	**%**
START	Start	0.45
HELLO WORLD	Helco LJorcd	0.3
END	CNb	0.37

Imagen en escala de grises, ecualización y filtro gaussiano (letras en mayúsculas):

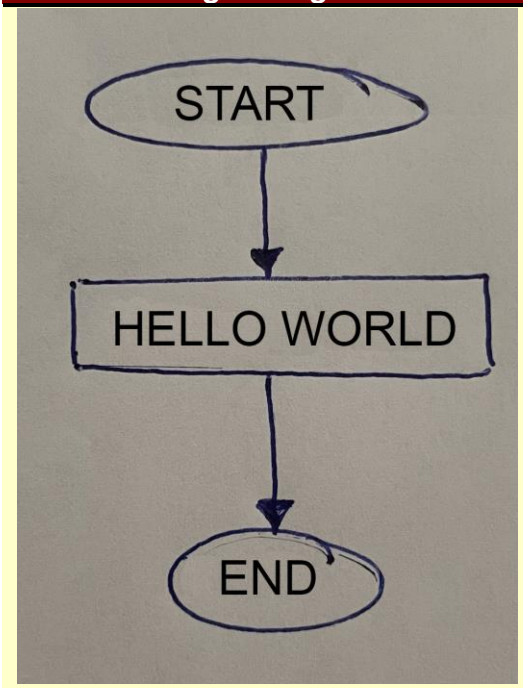
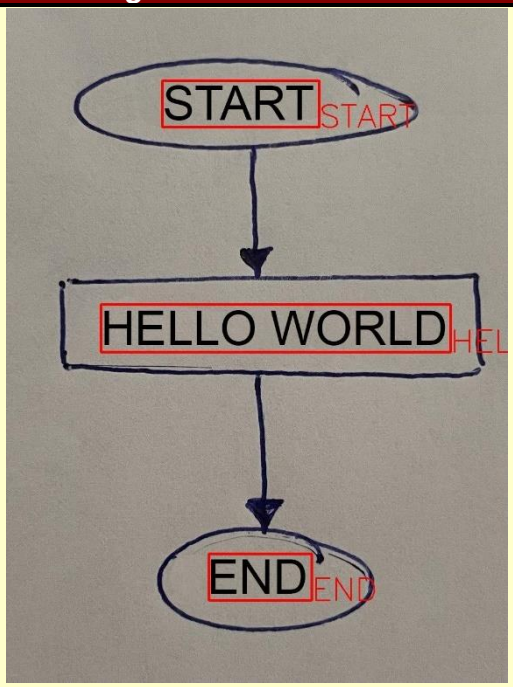
Imagen original	*Imagen con detecciones*
	

Textos originales	**Textos detectados**	**%**
START	Start	0.92
HELLO WORLD	Helco LJorcd	0.47
END	Cnb	0.71

Imagen en escala de grises y filtro gaussiano (letras en mayúsculas):

Imagen original	*Imagen con detecciones*	
		
Textos originales	**Textos detectados**	**%**
START	Start	0.94
HELLO WORLD	HCcco lorld	0.26
END	Cnb	0.25

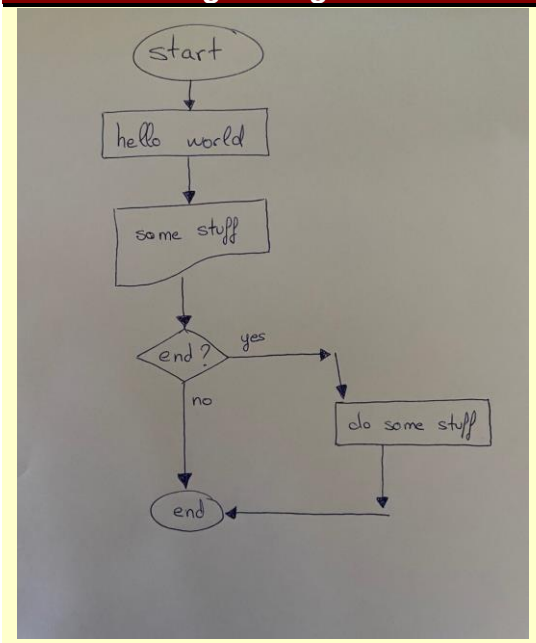
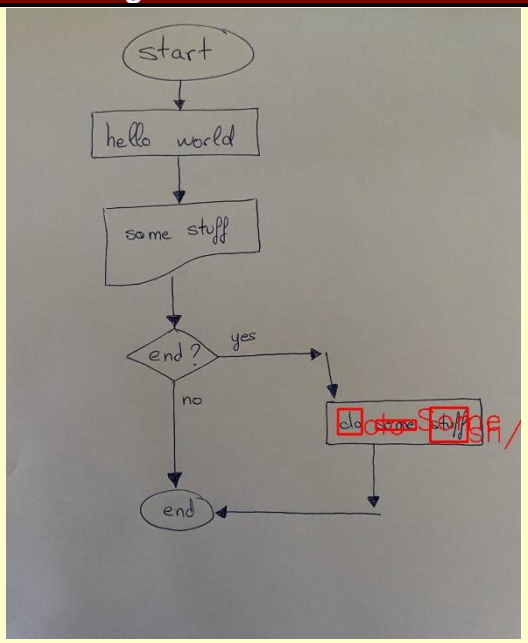
Prueba sustituyendo el texto manuscrito por texto escrito a ordenador (letras en mayúsculas):

Imagen original	*Imagen con detecciones*
	

Textos originales	**Textos detectados**	**%**
START	START	0.99
HELLO WORLD	HELLO WORLD	0.66
END	END	0.99

ANEXO 2: Pruebas de reconocimiento con Tesseract

Prueba inicial sin ningún tipo de modificación en la imagen (minúsculas):

Imagen original	*Imagen con detecciones*
	

Textos originales	Textos detectados	%
start	-	-
hello world	-	-
some stuff	-	-
end?	-	-
do some stuff	clo Some sh/p	0.76
end	-	-
yes	-	-
no	-	-

Imagen con paso a escala de grises (minúsculas):

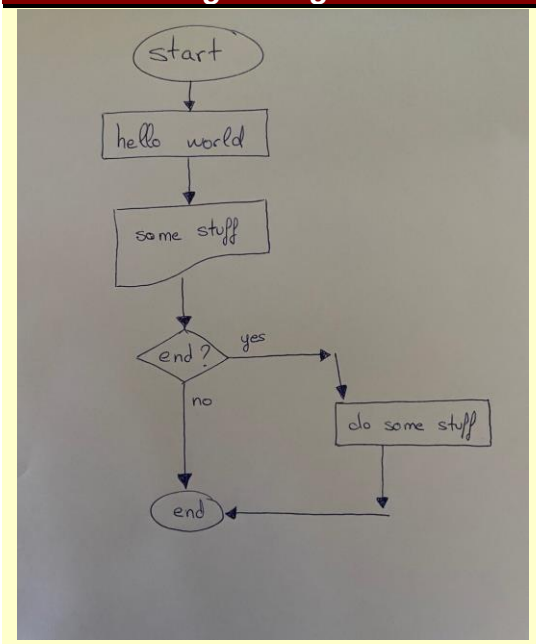
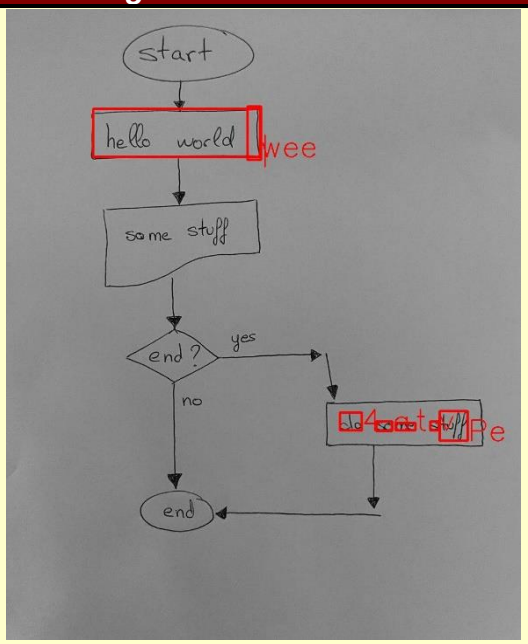
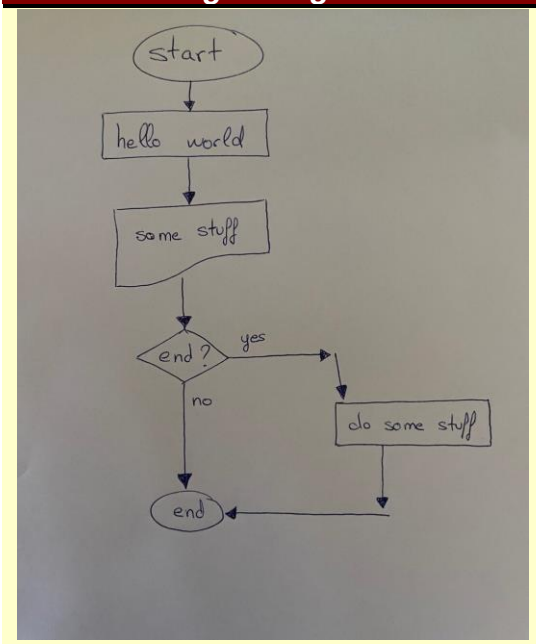
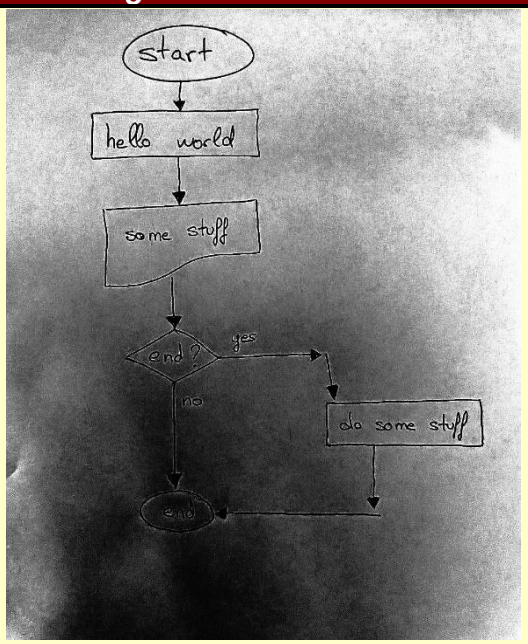
Imagen original	*Imagen con detecciones*	
		
Textos originales	**Textos detectados**	**%**
start	-	-
hello world	wee |	0.47
some stuff	-	-
end?	-	-
do some stuff	4 a t v Pe	0.44
end	-	-
yes	-	-
no	-	-

Imagen con escala de grises y ecualización (minúsculas):

Imagen original	*Imagen con detecciones*
	

Textos originales	Textos detectados	%
start	-	-
hello world	-	-
some stuff	-	-
end?	-	-
do some stuff	-	-
end	-	-
yes	-	-
no	-	-

Imagen en escala de grises, ecualización y filtro gaussiano (minúsculas):

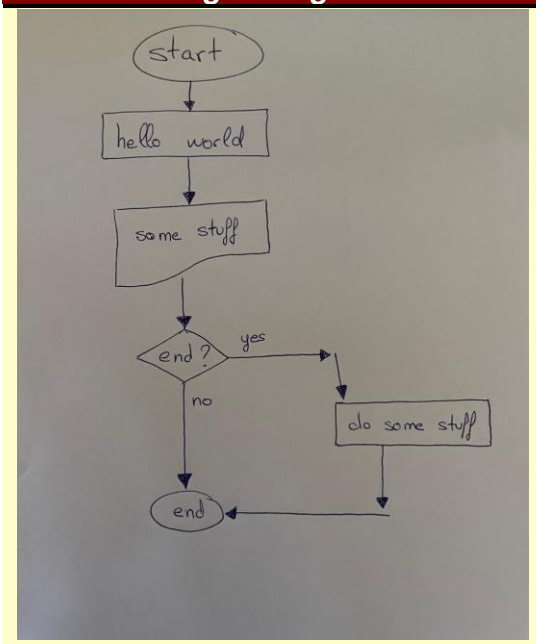
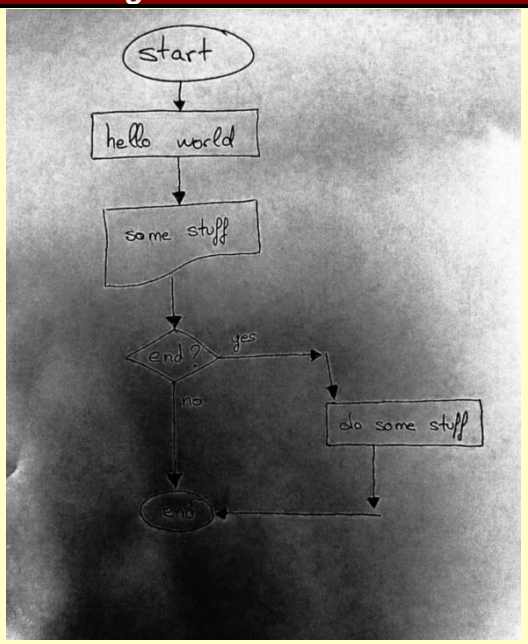
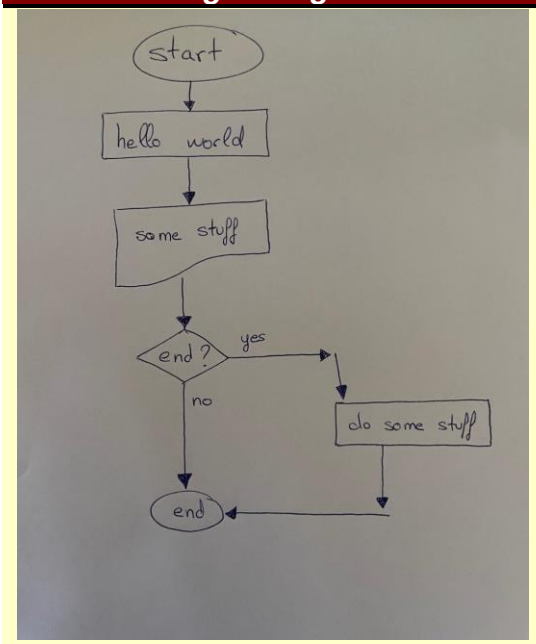
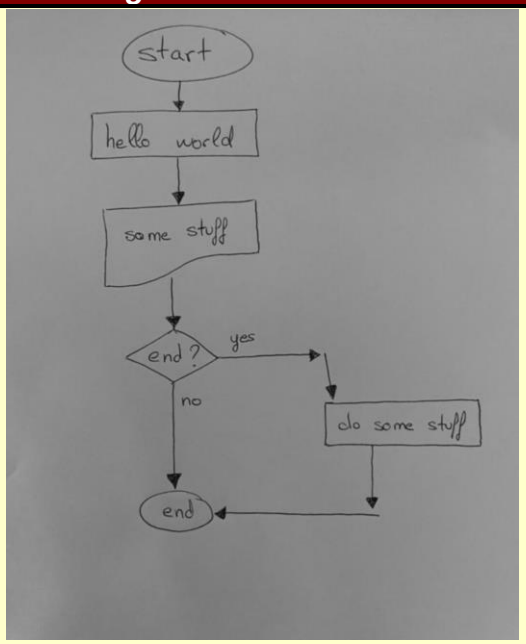
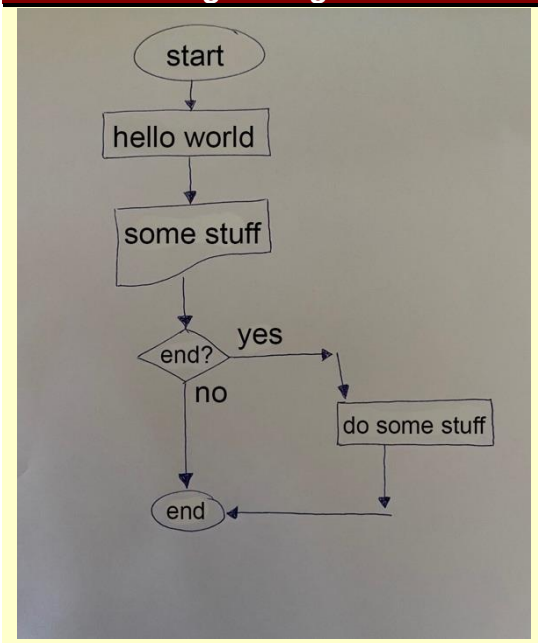
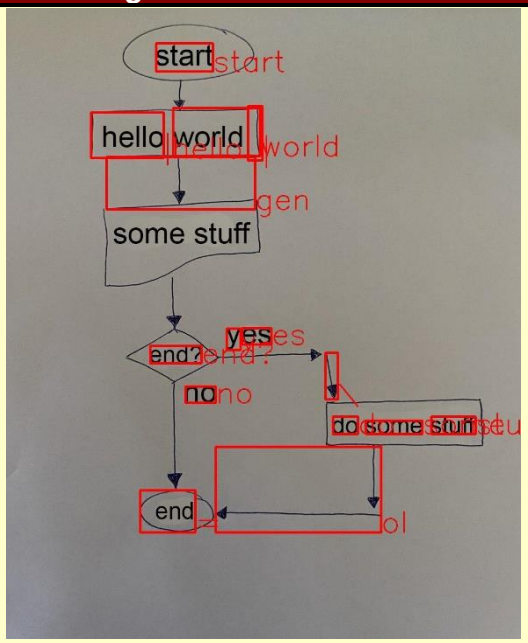
Imagen original	*Imagen con detecciones*	
		
Textos originales	**Textos detectados**	**%**
start	-	-
hello world	-	-
some stuff	-	-
end?	-	-
do some stuff	-	-
end	-	-
yes	-	-
no	-	-

Imagen en escala de grises y filtro gaussiano (minúsculas):

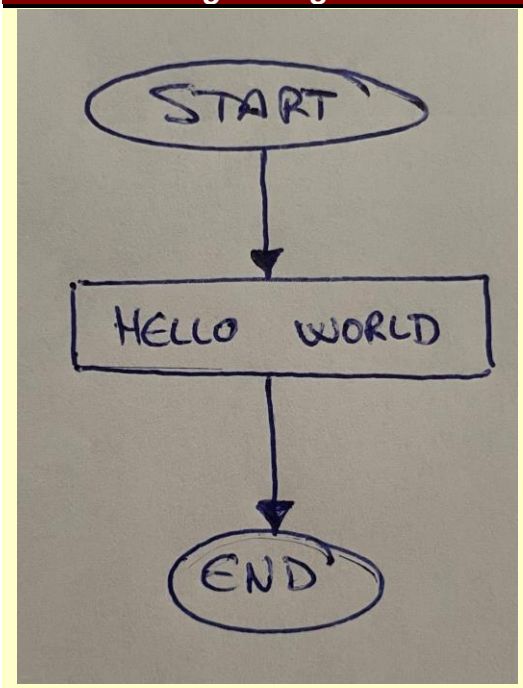
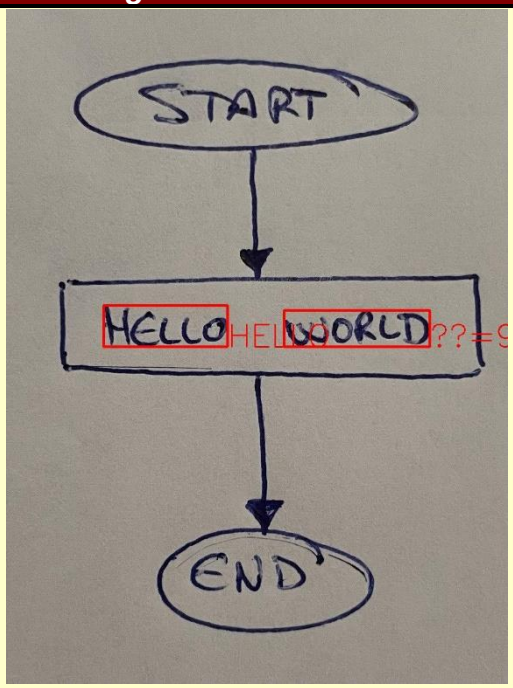
Imagen original	*Imagen con detecciones*	
		
Textos originales	**Textos detectados**	**%**
start	-	-
hello world	-	-
some stuff	-	-
end?	-	-
do some stuff	-	-
end	-	-
yes	-	-
no	-	-

Prueba sustituyendo el texto manuscrito por texto escrito a ordenador (minúsculas):

Imagen original	*Imagen con detecciones*
	

Textos originales	**Textos detectados**	**%**
start	start	0.96
hello world	|hello world |	0.76
some stuff	-	-
end?	end?	0.95
do some stuff	do some stuff	0.96
end	=	0.36
yes	es	0.96
no	no	0.91

Prueba inicial sin ningún tipo de modificación en la imagen (mayúsculas):

Imagen original	*Imagen con detecciones*
	

Textos originales	**Textos detectados**	**%**
START	-	-
HELLO WORLD	HELLO §=9woRLD	0.29
END	-	-

Imagen con paso a escala de grises (mayúsculas):

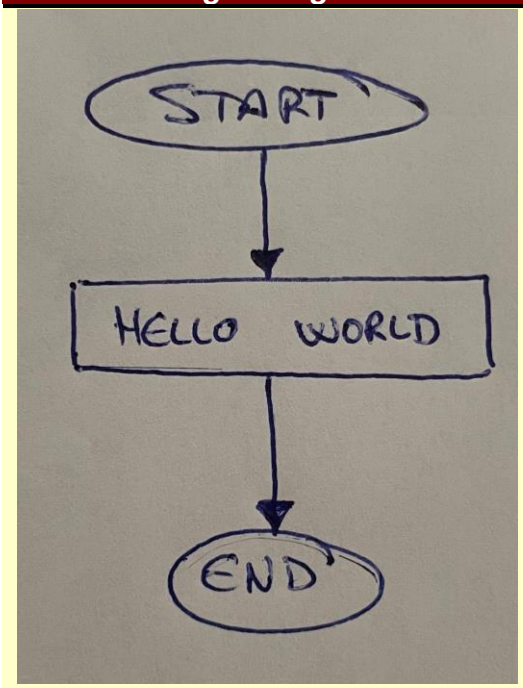
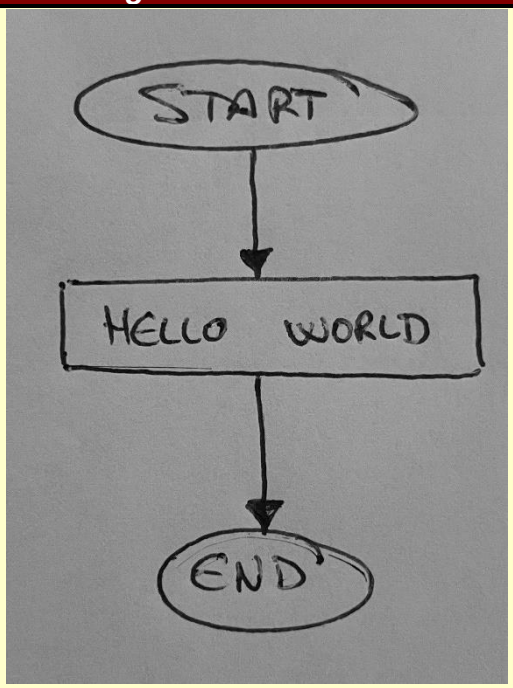
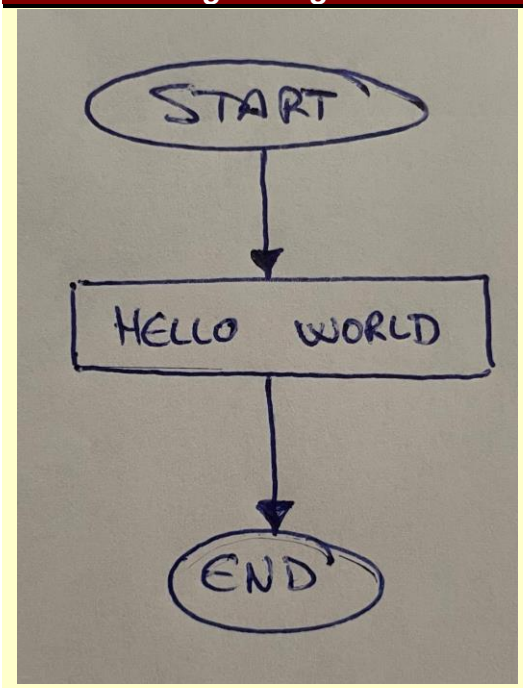
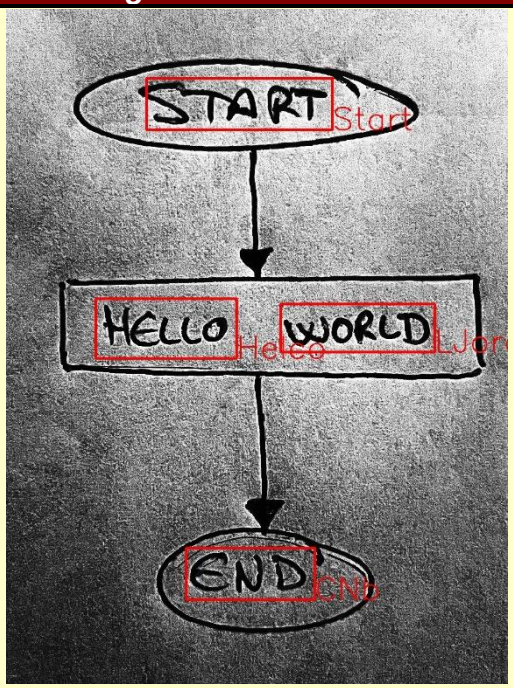
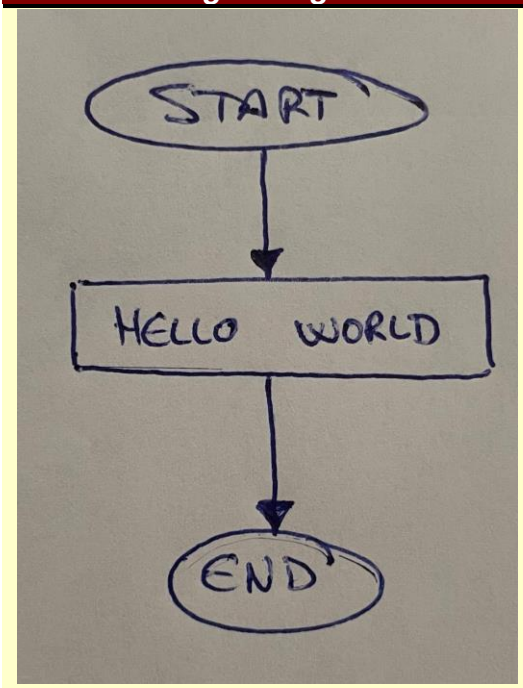
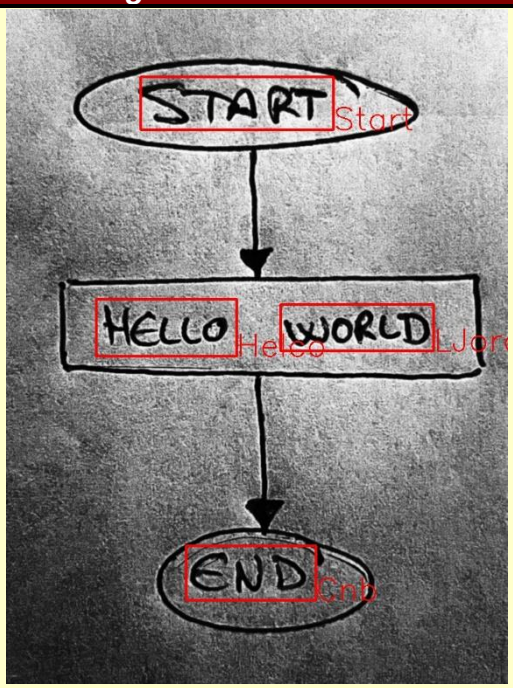
Imagen original	*Imagen con detecciones*	
		
Textos originales	**Textos detectados**	**%**
START	-	-
HELLO WORLD	-	-
END	-	-

Imagen con escala de grises y ecualización (mayúsculas):

Imagen original	*Imagen con detecciones*
	

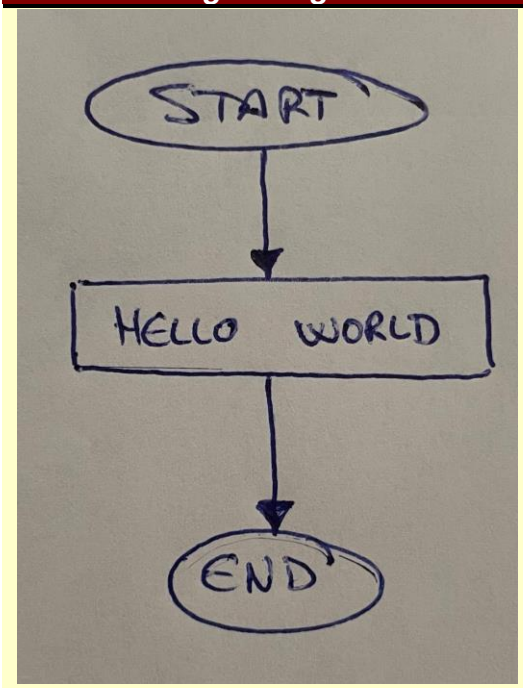
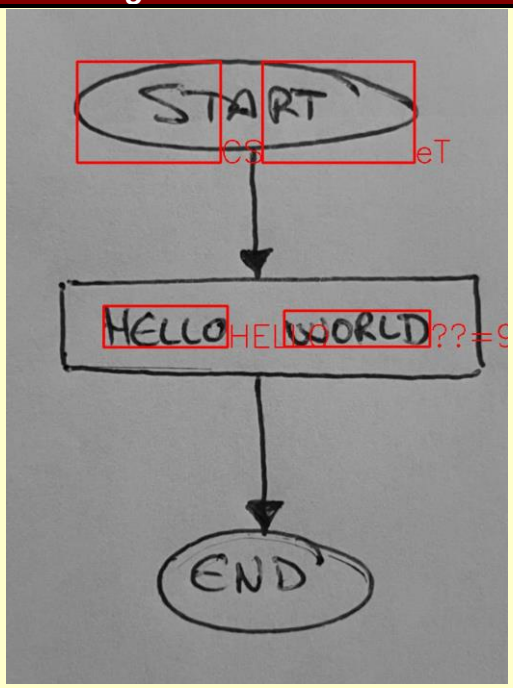
Textos originales	Textos detectados	%
START	-	-
HELLO WORLD	-	-
END	-	-

Imagen en escala de grises, ecualización y filtro gaussiano (mayúsculas):

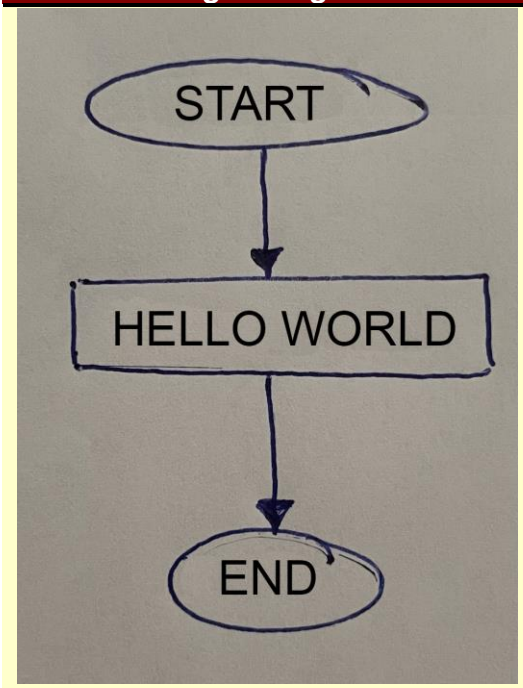
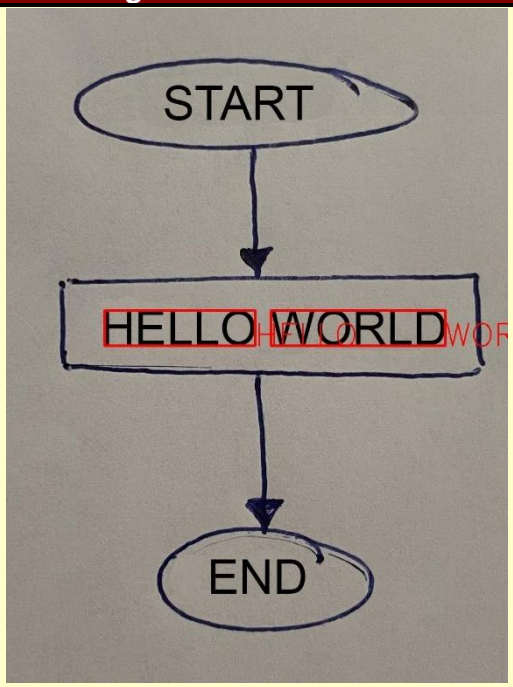
Imagen original	*Imagen con detecciones*
	

Textos originales	Textos detectados	%
START	-	-
HELLO WORLD	-	-
END	-	-

Imagen en escala de grises y filtro gaussiano (mayúsculas):

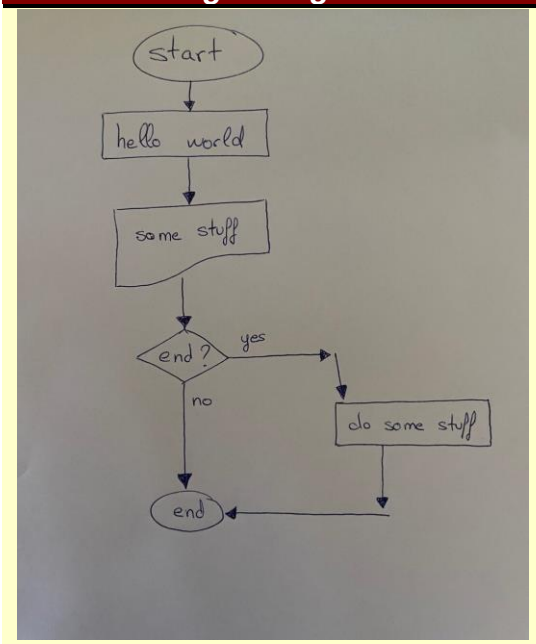
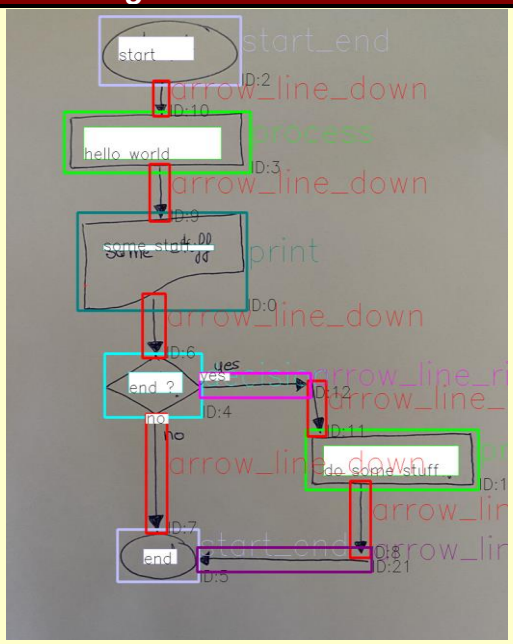
Imagen original	*Imagen con detecciones*	
		
Textos originales	**Textos detectados**	**%**
START	CS eT	0.05
HELLO WORLD	HELLO §=9woRLD	0.55
END	-	-

Prueba sustituyendo el texto manuscrito por texto escrito a ordenador (mayúsculas):

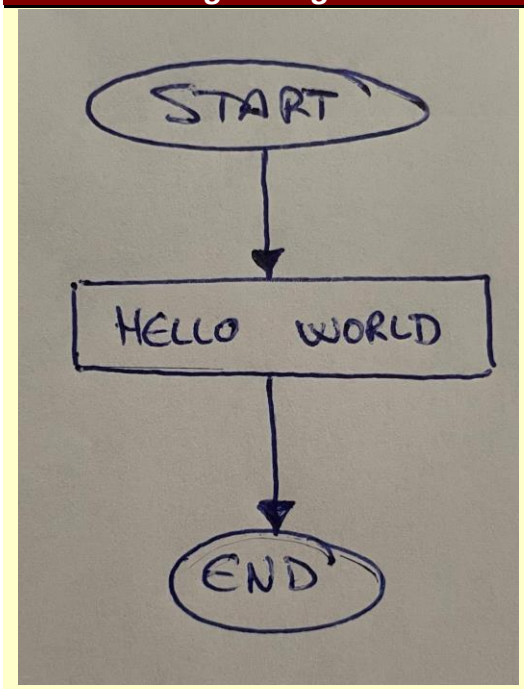
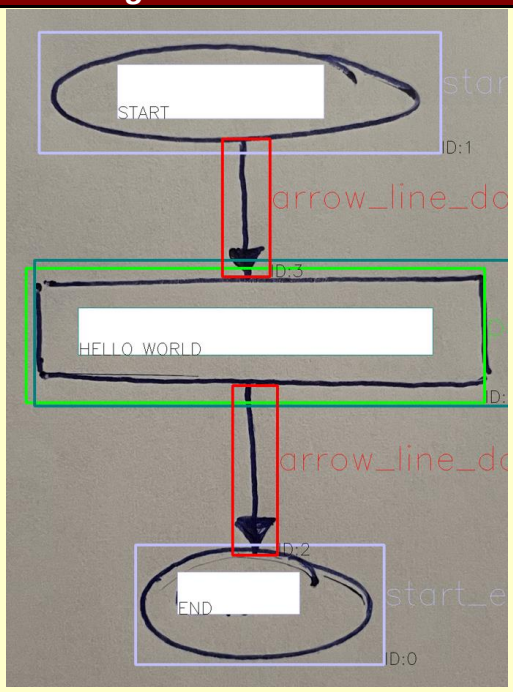
Imagen original	*Imagen con detecciones*
	

Textos originales	Textos detectados	%
START	-	-
HELLO WORLD	HELLO WORLD	0.96
END	-	-

ANEXO 3: Pruebas de reconocimiento con Azure Computer Vision

Imagen original	*Imagen con detecciones*
	

Textos originales	Textos detectados	%
start	start	0.99
hello world	hello world	0.96
some stuff	some stuff	0.95
end?	end ?	0.99
do some stuff	do some stuff	0.95
end	end	0.99
yes	yes	0.99
no	no	0.99

Imagen original	*Imagen con detecciones*	
		
Textos originales	**Textos detectados**	**%**
START	START	0.99
HELLO WORLD	HELLO WORLD	0.98
END	END	0.99

ANEXO 4: Código fuente empleado para el adiestramiento del modelo de reconocimiento OCR desarrollado

ImagesToCSV.py

```
import os
import cv2
import csv
import numpy

directory: str = "./Datasets/handwritten_math_symbols"

symbols: [str] = ["-", "(", ")", "+", "=", "div", "geq", "gt", "lt", "leq", "neq"]
symbols_index: {} = {
    "-": 36,
    "(": 37,
    ")": 38,
    "+": 39,
    "=": 40,
    "div": 41,
    "geq": 42,
    "gt": 43,
    "lt": 44,
    "leq": 45,
    "neq": 46
}

directories = [f for f in os.scandir(directory) if f.name in symbols]
file = open('./Datasets/math_symbols.csv', 'w', newline='')
writer = csv.writer(file)

for d in directories:
    print("Starting {}...".format(d.name))
    index: int = symbols_index[d.name]
    files: [str] = [x for x in os.listdir(d.path) if os.path.isfile(os.path.join(d, x))]

    for f in files:
        original_file: str = "{}/{}".format(d.path, f)
        original_img = cv2.imread(original_file, cv2.IMREAD_GRAYSCALE)
        img_resized = cv2.resize(original_img, (28, 28),
                                 interpolation=cv2.INTER_LINEAR)
        img_resized = cv2.bitwise_not(img_resized)

        img_array = img_resized.flatten()
        img_array = numpy.insert(img_array, 0, index)

        writer.writerow(img_array)

file.close()
```

DatasetSplitter.py

```
import os
import shutil

train_percent: float = 0.8
validation_percent: float = 1 - train_percent

datasets_folder: str = "./Datasets"
input_dataset_folder: str = datasets_folder + "/handwritten_math_symbols"
output_dataset_folder: str = datasets_folder +
"/handwritten_math_symbols_ready"

if not os.path.exists(output_dataset_folder):
    os.makedirs(output_dataset_folder)

symbols: [str] = ["-", "(", ")", "+", "=", "div", "geq", "gt", "lt", "leq", "neq"]

directories = [f for f in os.scandir(input_dataset_folder) if f.name in symbols]

for d in directories:
    print("copying {}...".format(d.name))

    files: [str] = [x for x in os.listdir(d.path) if os.path.isfile(os.path.join(d, x))]

    length: int = int(len(files))
    train: int = int(length * train_percent)
    validation: int = int(length * validation_percent)

    if not os.path.exists("{}/train/{}".format(output_dataset_folder, d.name)):
        os.makedirs("{}/train/{}".format(output_dataset_folder, d.name))
    if not os.path.exists("{}/validation/{}".format(output_dataset_folder, d.name)):
        os.makedirs("{}/validation/{}".format(output_dataset_folder, d.name))

    for i, f in enumerate(files):
        original_file: str = "{}/{}".format(d.path, f)

        if i < train:
            shutil.copyfile(original_file, "{}/train/{}/{}".format(output_dataset_folder,
                                                                   d.name, f))
        else:
            shutil.copyfile(original_file,
                            "{}/validation/{}/{}".format(output_dataset_folder, d.name, f))
```

Train.py

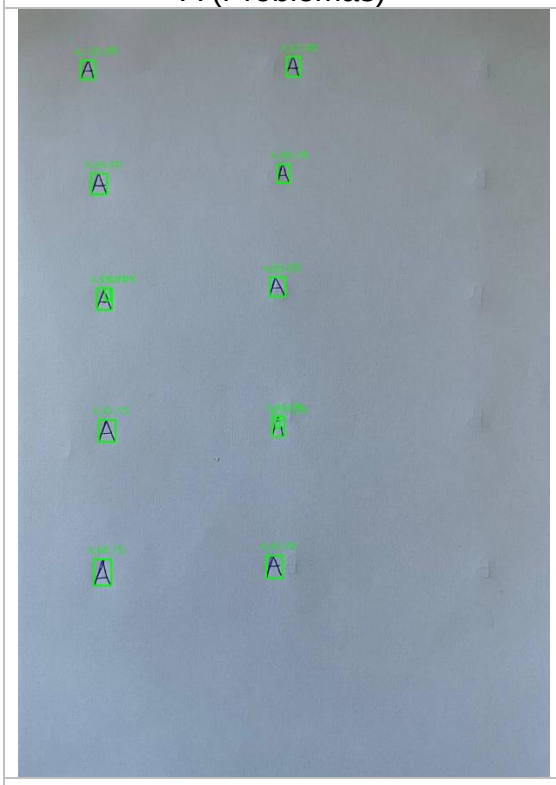
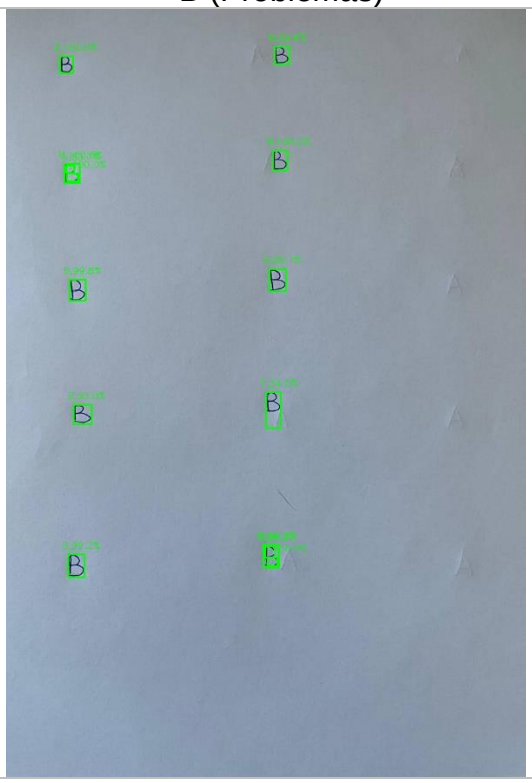
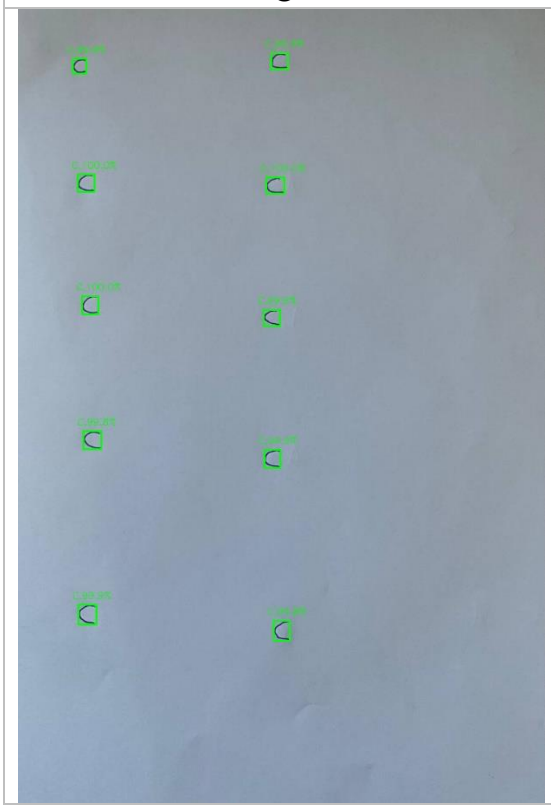
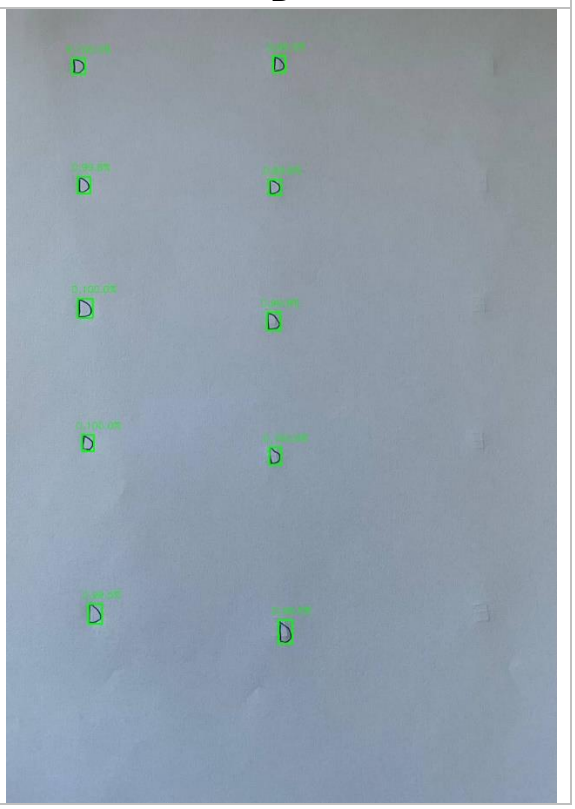
```
import tensorflow as tf
import tensorflow_datasets as tfds
import cv2
import numpy as np
import pandas as pd
import keras
import os

os.environ['CUDA_VISIBLE_DEVICES'] = '-1'

if tf.test.gpu_device_name():
    print('GPU found')
else:
    print("No GPU found")

mnist = keras.datasets.mnist
(train_images_mnist,train_labels_mnist),(test_images_mnist,test_labels_mnist)
= mnist.load_data()

train_images_mnist =
np.reshape(train_images_mnist,(train_images_mnist.shape[0],28,28,1))
test_images_mnist =
np.reshape(test_images_mnist,(test_images_mnist.shape[0],28,28,1))

# download from https://www.kaggle.com/datasets/sachinpatel21/az-
handwritten-alphabets-in-csv-format
# download from https://www.kaggle.com/datasets/sagyamthapa/handwritten-
math-symbols
az_data_path = r'C:\Users\Lagos\OneDrive\TFG\Datasets'

AZ_data = pd.read_csv(az_data_path +'/A_Z Handwritten Data.csv',header =
None)
# the first column contains label values, while the remaining are the flattened
array of 28 x 28 image pixels
AZ_labels = AZ_data.values[:,0]
AZ_images = AZ_data.values[:,1:]
# images are reshaped to be used by the flow method of a keras
ImageGenerator
AZ_images = np.reshape(AZ_images,(AZ_images.shape[0], 28, 28, 1))

math_symbols_path = az_data_path + '/math_symbols.csv'
math_symbols_data = pd.read_csv(math_symbols_path, header=None)
math_symbols_labels = math_symbols_data.values[:, 0]
math_symbols_images = math_symbols_data.values[:, 1:]
math_symbols_images = np.reshape(math_symbols_images,
(math_symbols_images.shape[0], 28, 28, 1))
```

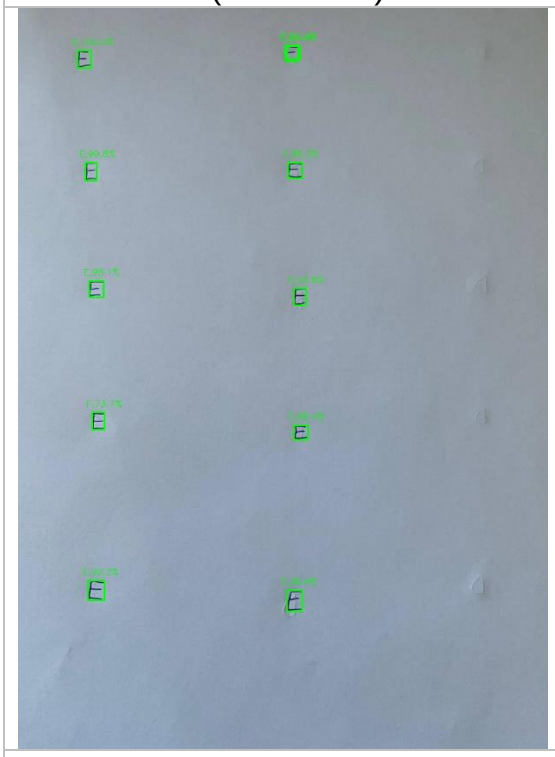
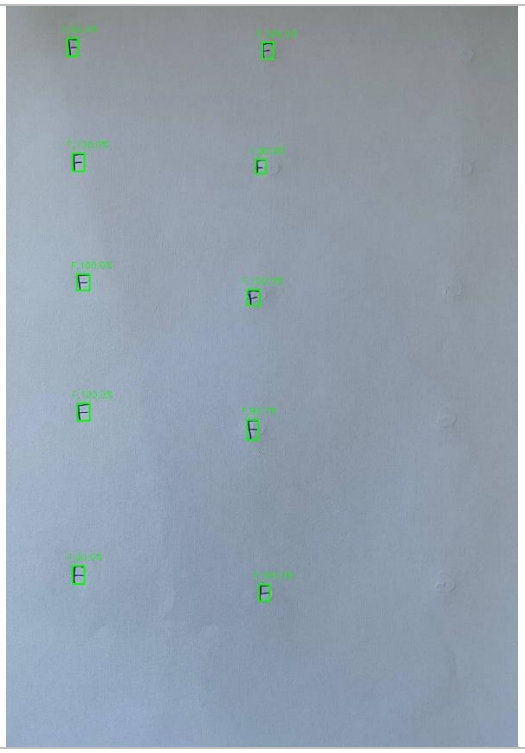
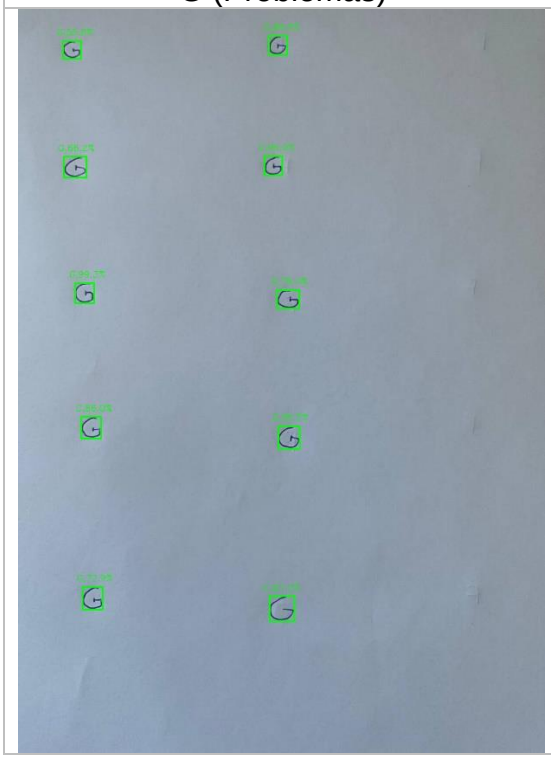
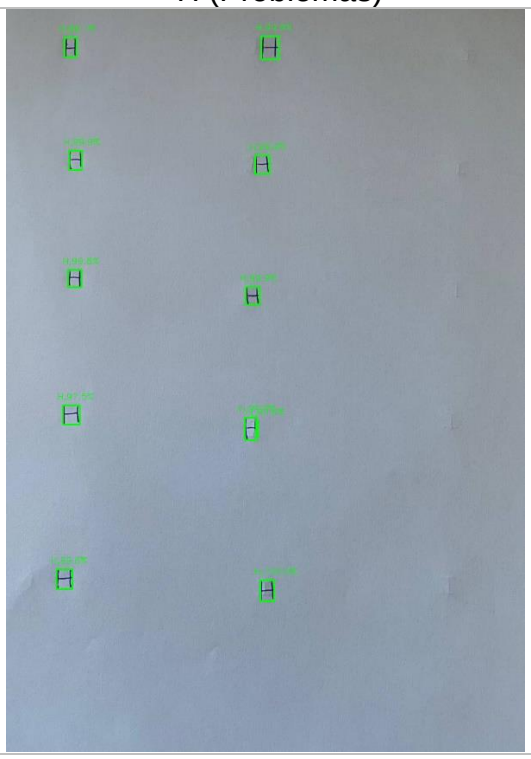
```
# join datasets
# split AZ data in train and test
from sklearn.model_selection import train_test_split

test_size_AZ = float(len(test_labels_mnist))/len(train_labels_mnist)
print(f'AZ test set size: {test_size_AZ}')
train_images_AZ, test_images_AZ, train_labels_AZ, test_labels_AZ =
train_test_split(AZ_images,AZ_labels, test_size=test_size_AZ)
#shift mnist labels
train_labels_mnist = train_labels_mnist + max(AZ_labels)+1
test_labels_mnist = test_labels_mnist + max(AZ_labels)+1

test_size_math = float(len(test_labels_mnist))/len(math_symbols_labels)
print(f'Math test set size: {test_size_math}')
train_images_math, test_images_math, train_labels_math, test_labels_math =
train_test_split(math_symbols_images,math_symbols_labels,
test_size=test_size_math)


# concatenate datasets
train_images = np.concatenate((train_images_AZ,train_images_mnist,
train_images_math),axis=0)
train_labels = np.concatenate((train_labels_AZ,train_labels_mnist,
train_labels_math))
test_images = np.concatenate((test_images_AZ,test_images_mnist,
test_images_math),axis=0)
test_labels = np.concatenate((test_labels_AZ,test_labels_mnist,
test_labels_math))

print('Data ready')

from keras.optimizers import RMSprop
model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 150x150 with 3 bytes
color
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    # Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class
('cats') and 1 for the other ('dogs')
    tf.keras.layers.Dense(len(np.unique(train_labels)), activation='softmax')
])

model.compile(optimizer=RMSprop(learning_rate=1e-4),
              loss='sparse_categorical_crossentropy',
```

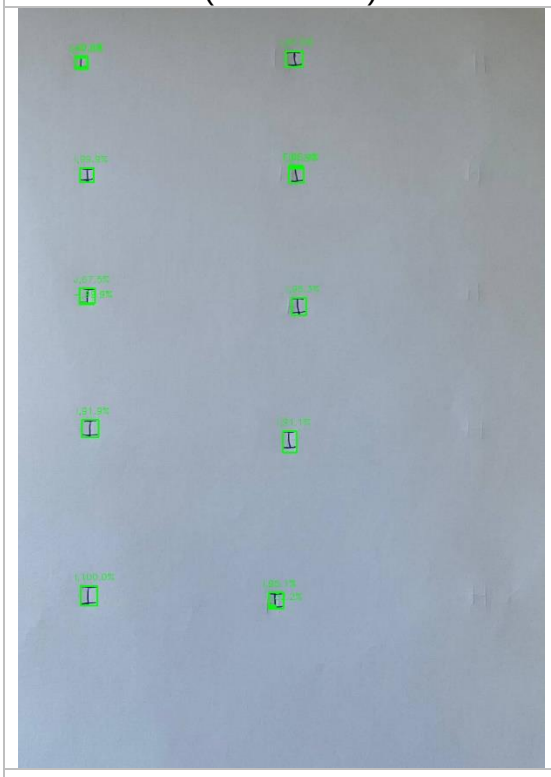
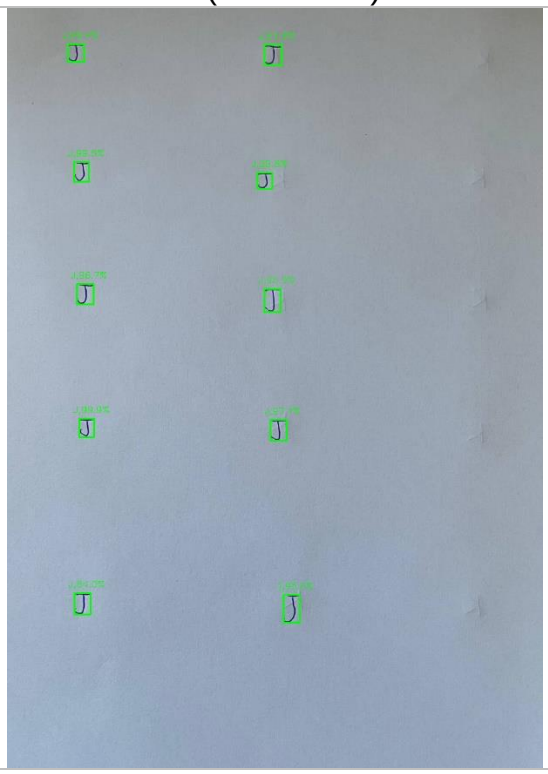
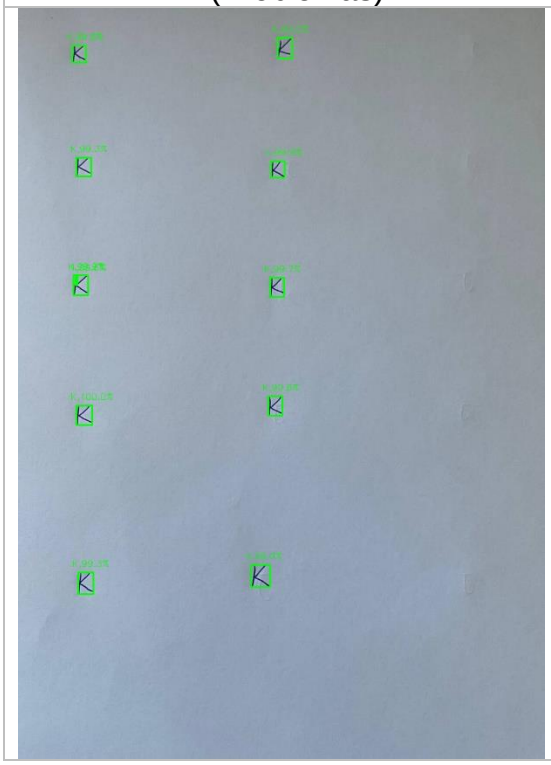
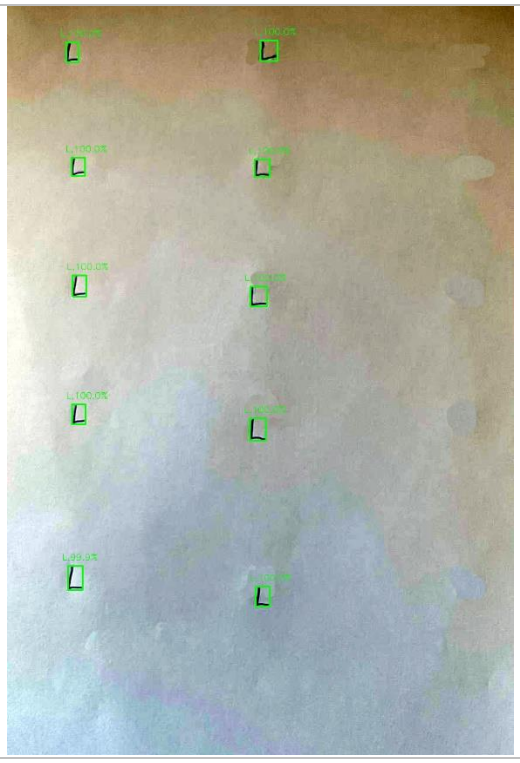
```
            metrics = ['accuracy'])

model.summary()

from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=15,
      width_shift_range=0.1,
      height_shift_range=0.1,
      shear_range=0.1,
      zoom_range=0.2,
      horizontal_flip=False,
      fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches using generator
train_generator = train_datagen.flow(train_images, train_labels,
batch_size=50, shuffle=True)
validation_generator = test_datagen.flow(test_images, test_labels,
batch_size=50, shuffle=True)

history = model.fit(
      train_generator,
      steps_per_epoch=500,
      epochs=250,
      validation_data=validation_generator,
      validation_steps=50,
      verbose=2)
model.save('model_full.h5')

(ds_train, ds_test), ds_info = tfds.load(
    'mnist',
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True,
)


def normalize_img(image, label):
  """Normalizes images: `uint8` -> `float32`."""
  return tf.cast(image, tf.float32) / 255., label


ds_train = ds_train.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_train = ds_train.cache()
ds_train = ds_train.shuffle(ds_info.splits['train'].num_examples)
```

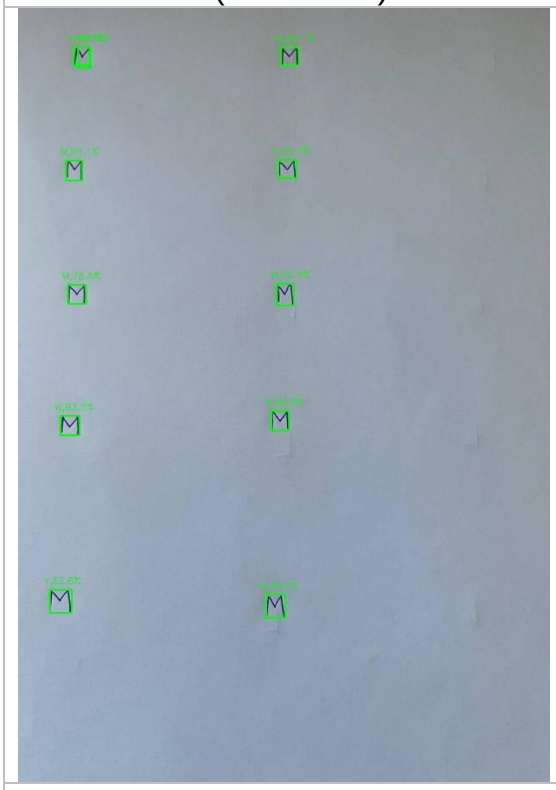
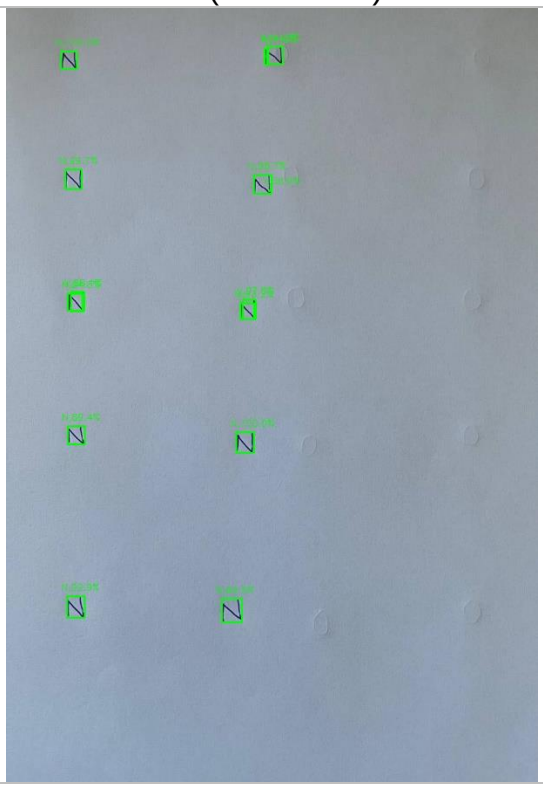
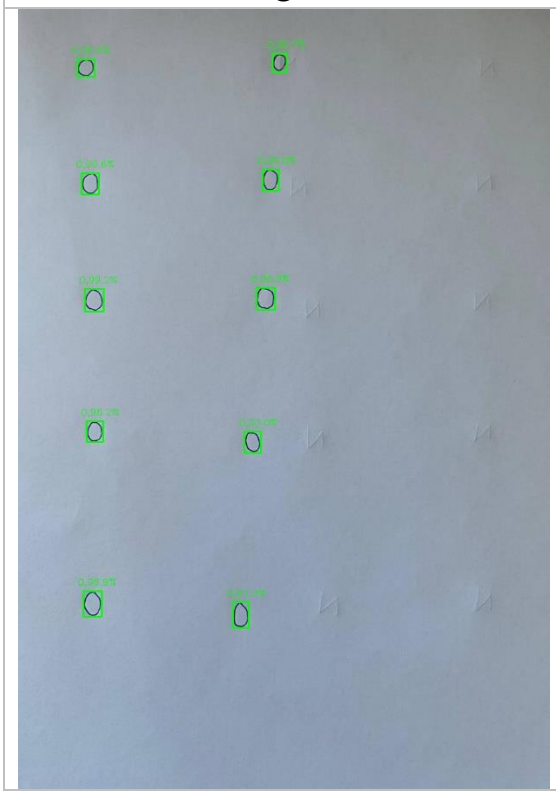
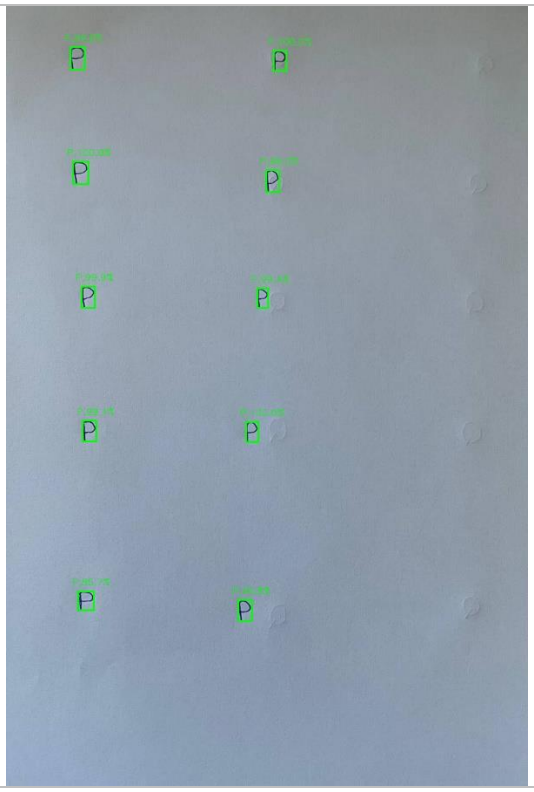
```
ds_train = ds_train.batch(128)
ds_train = ds_train.prefetch(tf.data.AUTOTUNE)

ds_test = ds_test.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_test = ds_test.batch(128)
ds_test = ds_test.cache()
ds_test = ds_test.prefetch(tf.data.AUTOTUNE)

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10)
])
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)

model.fit(
    ds_train,
    epochs=6,
    validation_data=ds_test,
)

img_path: str = 'HandwrittenOCR/tests/HelloWorld.jpg'
```

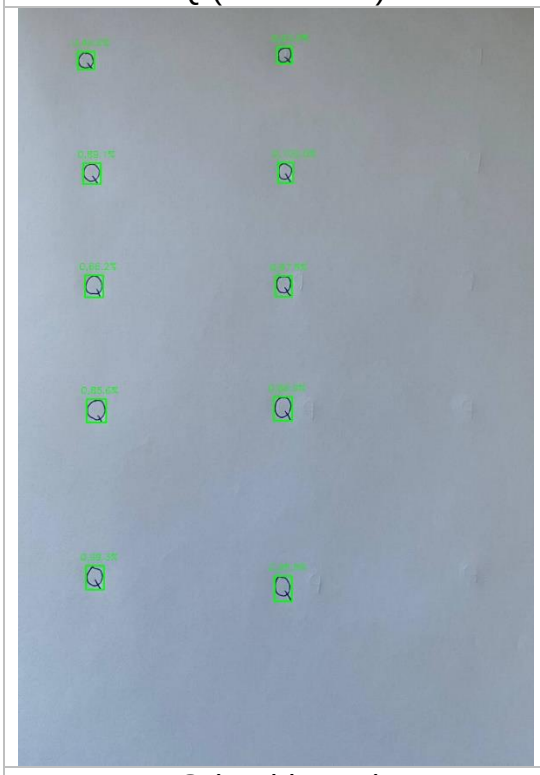
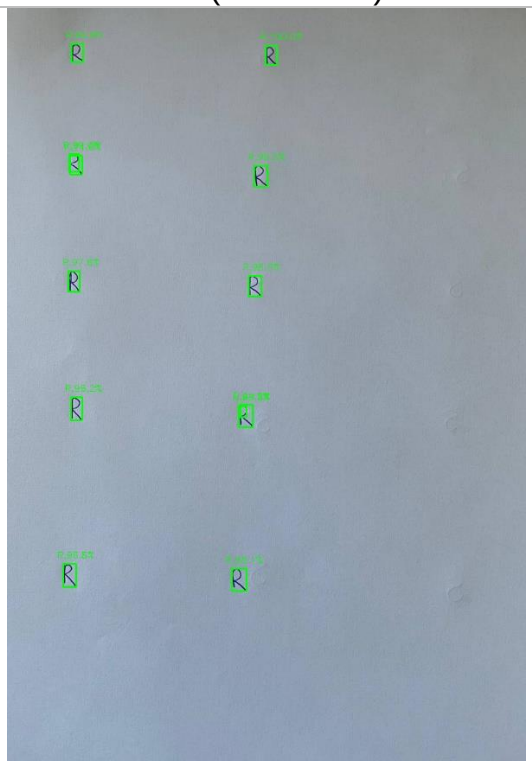
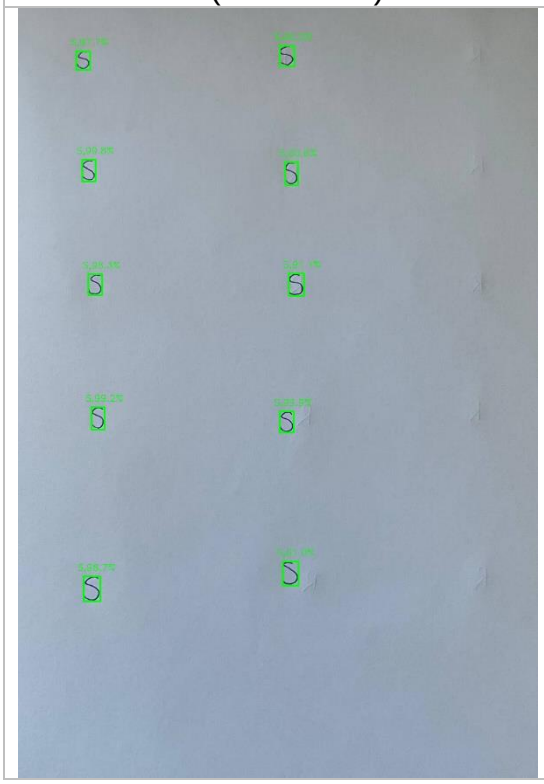
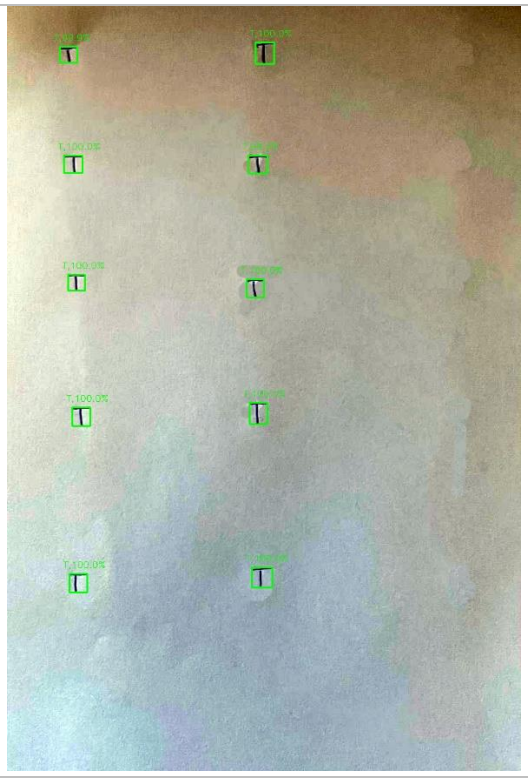
ANEXO 5: Problemas iniciales en la detección de texto con el modelo OCR generado

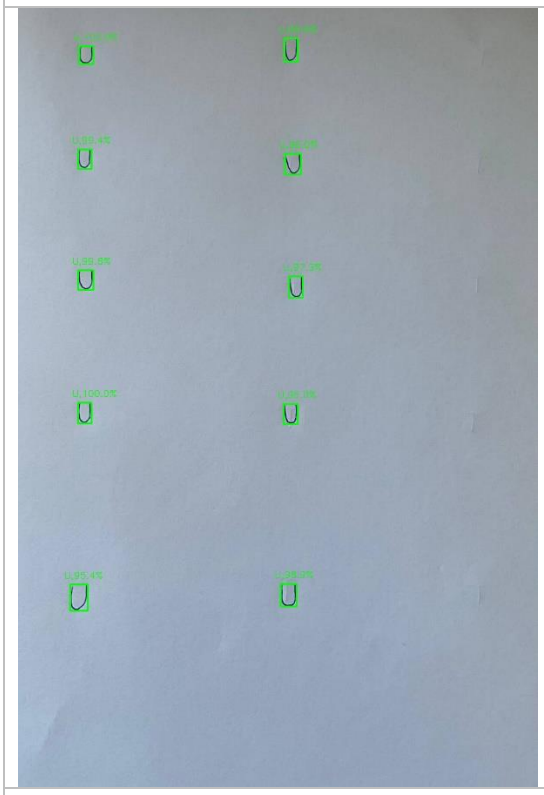
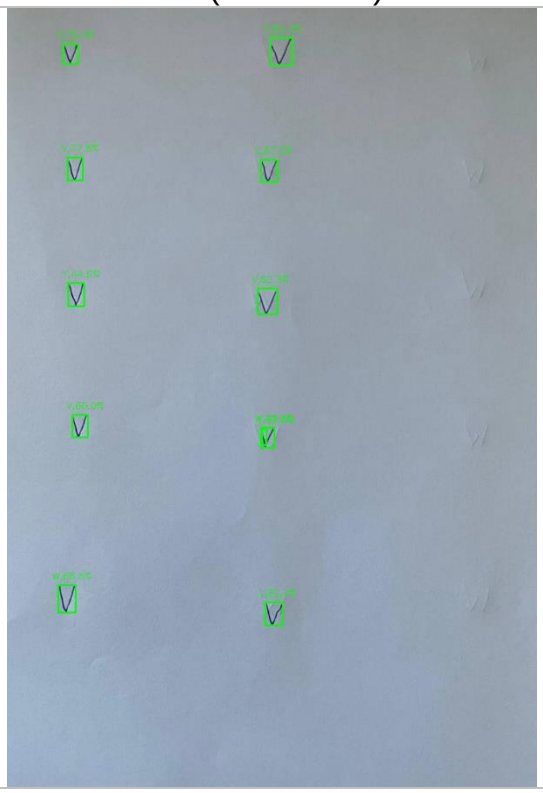
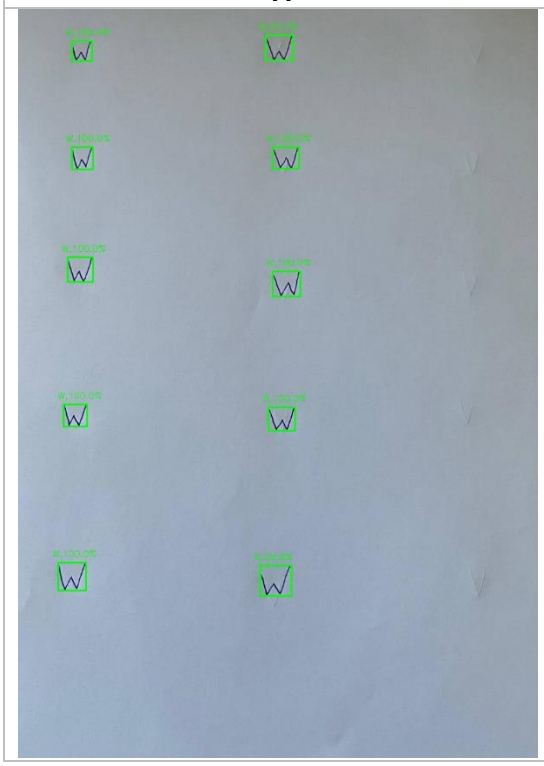
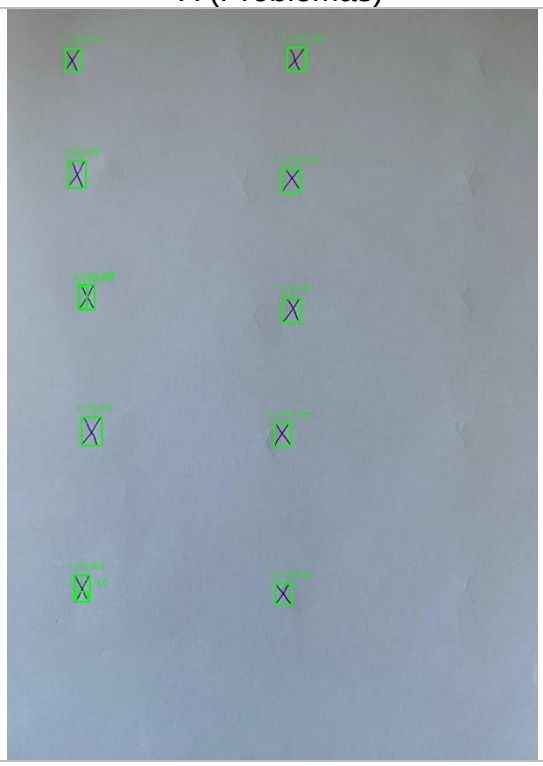
A (Problemas)	B (Problemas)
	
C	D
	

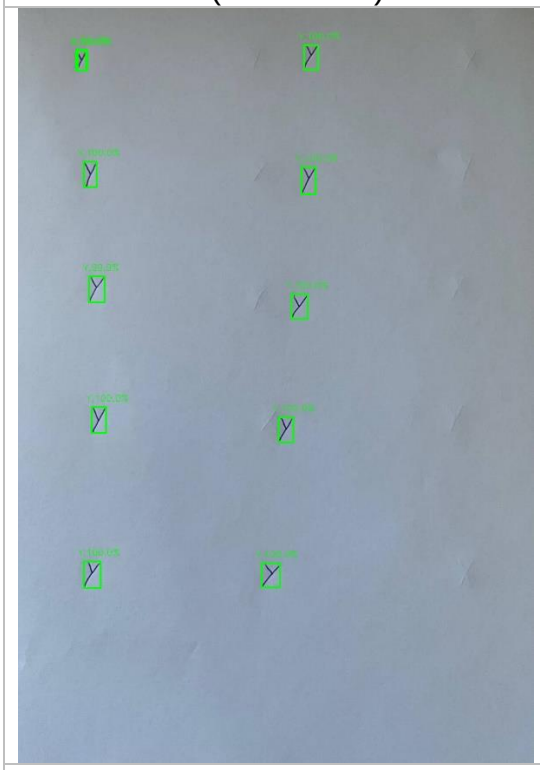
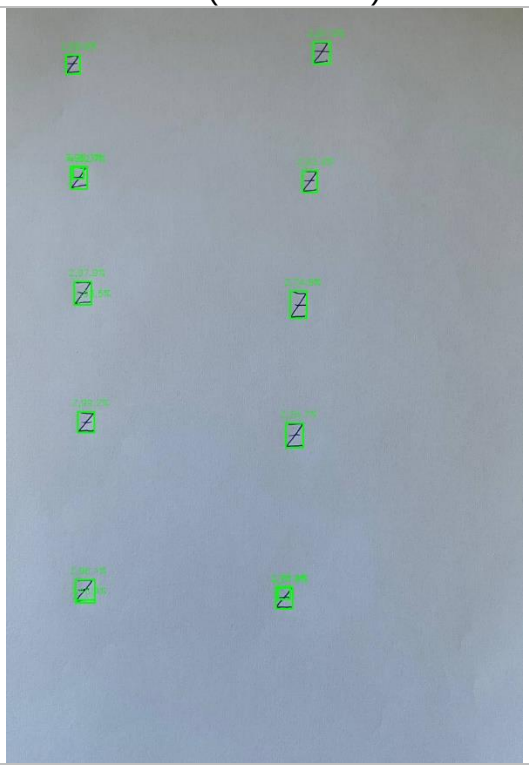
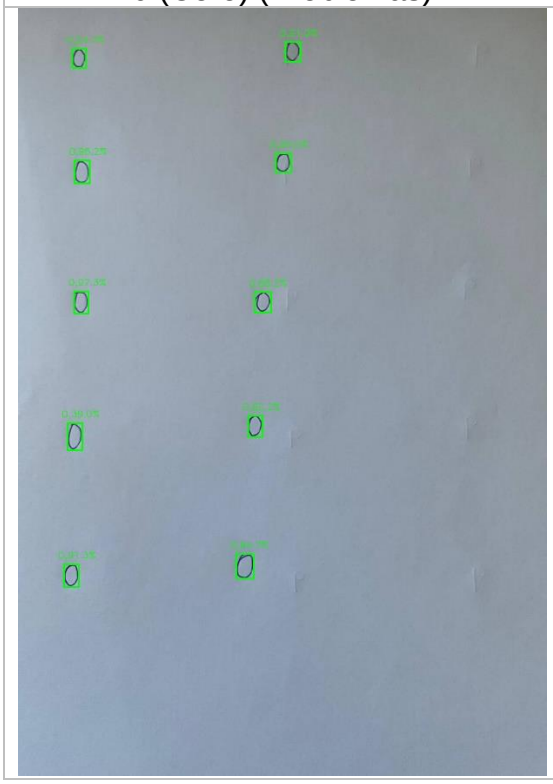
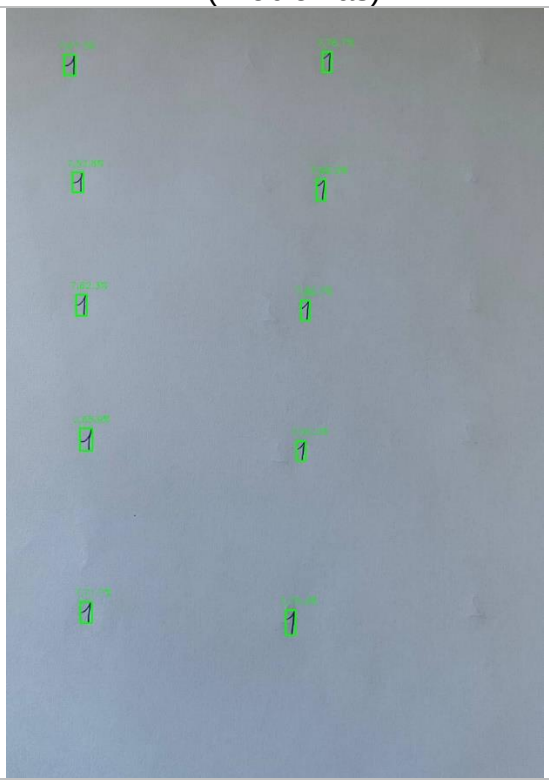
E (Problemas)	F
	
G (Problemas)	H (Problemas)
	

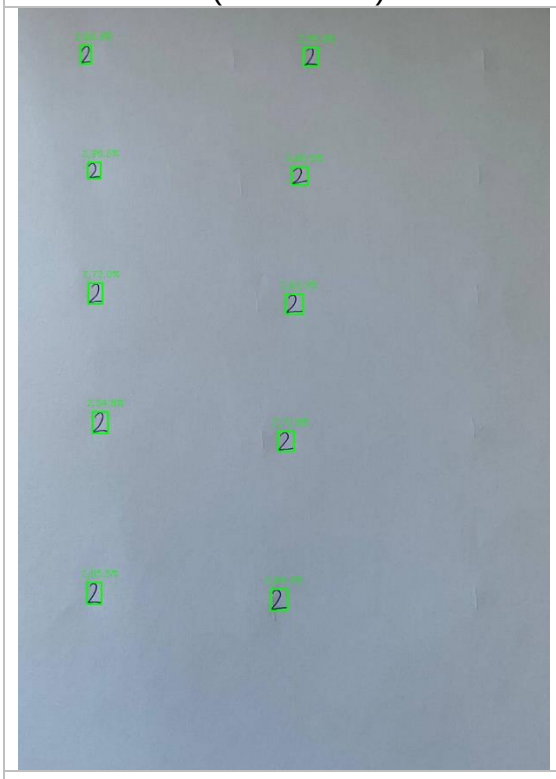
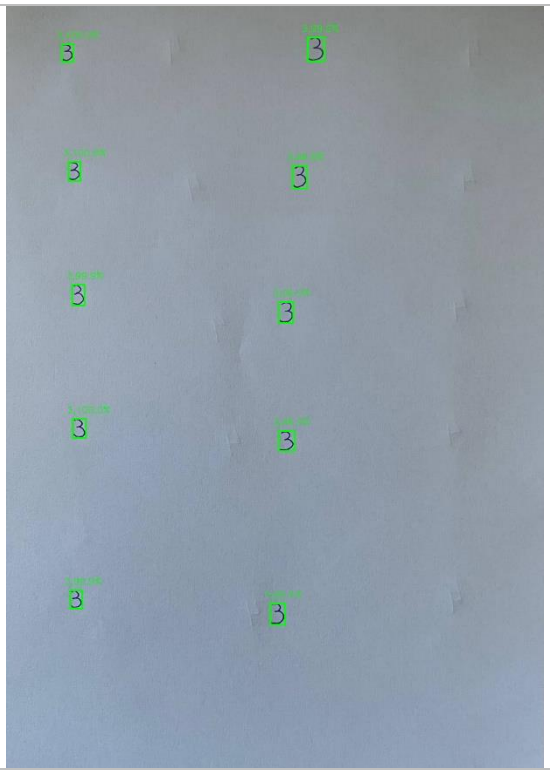
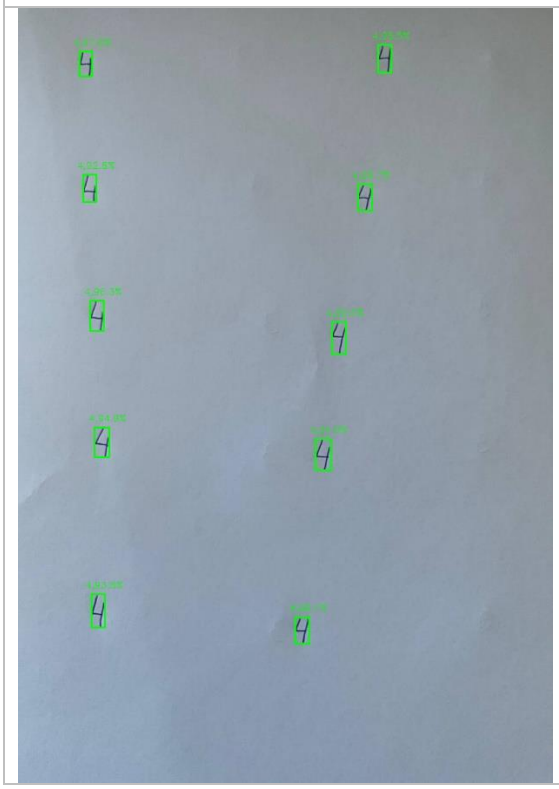
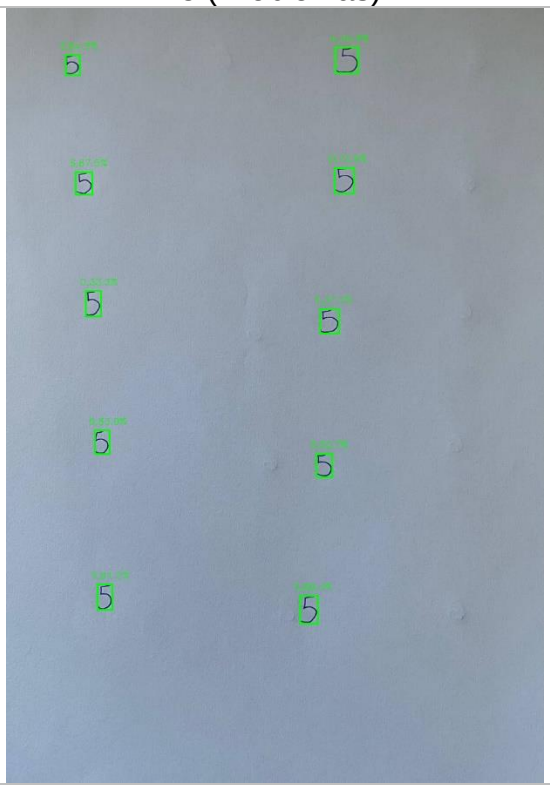
I (Problemas)	J (Problemas)
	
K (Problemas)	L
	

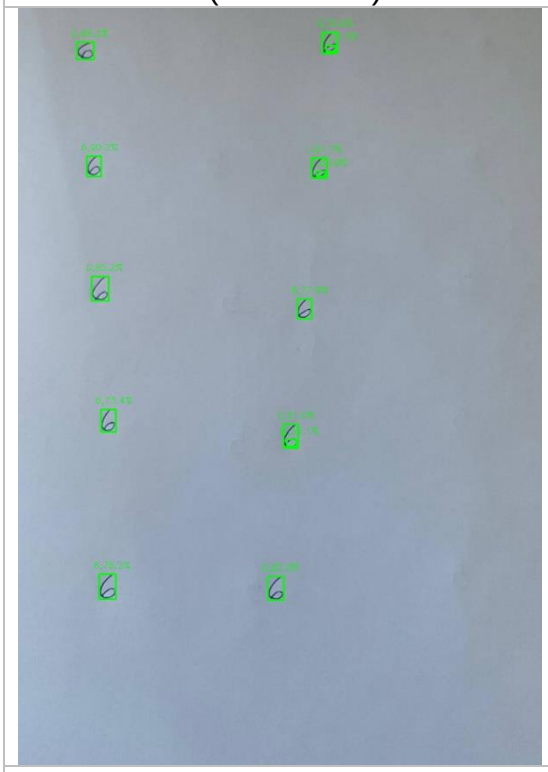
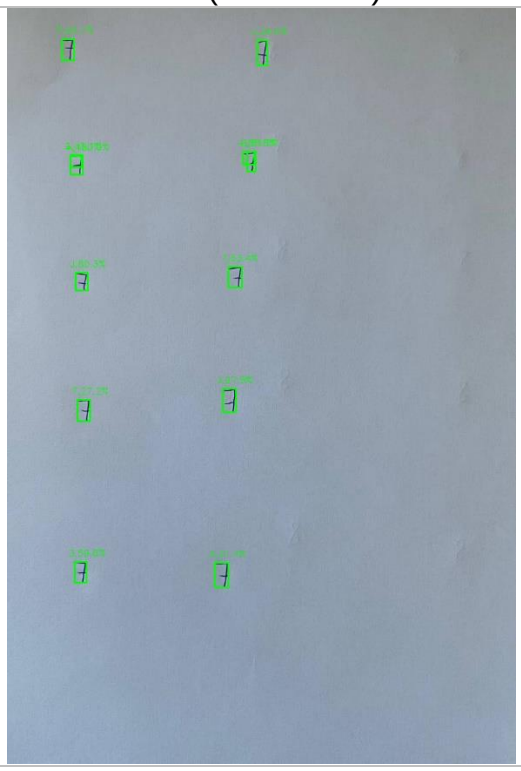
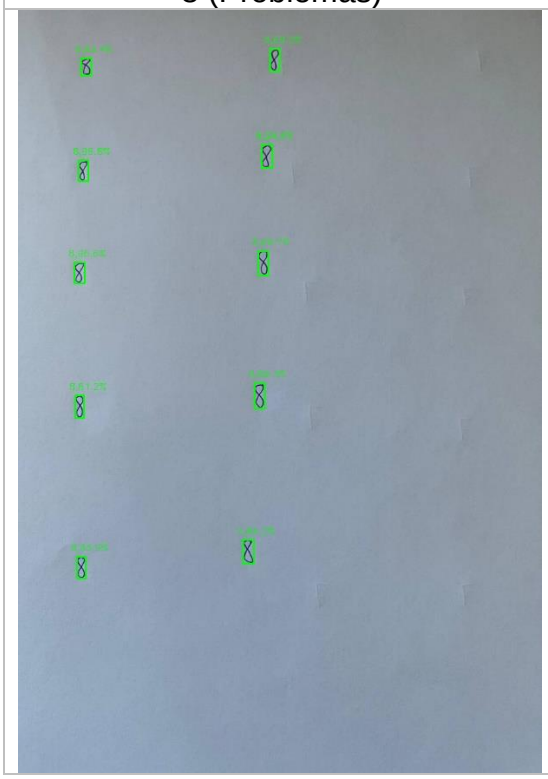
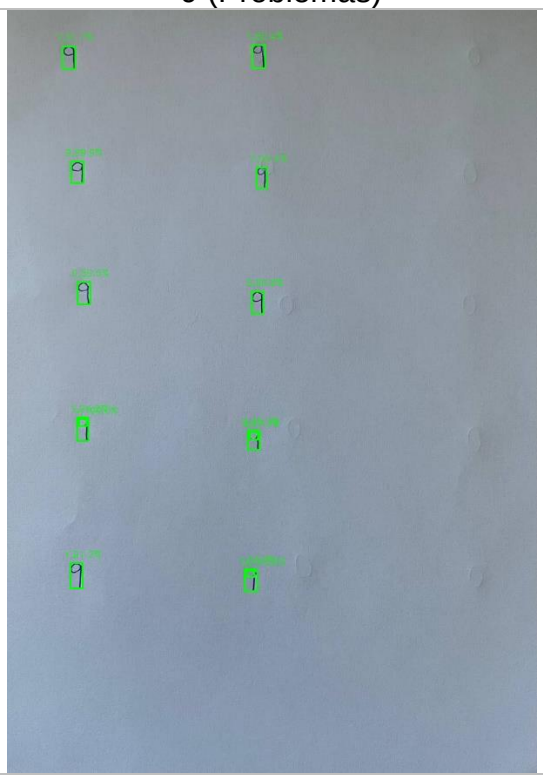
M (Problemas)	N (Problemas)
	
O	P
	

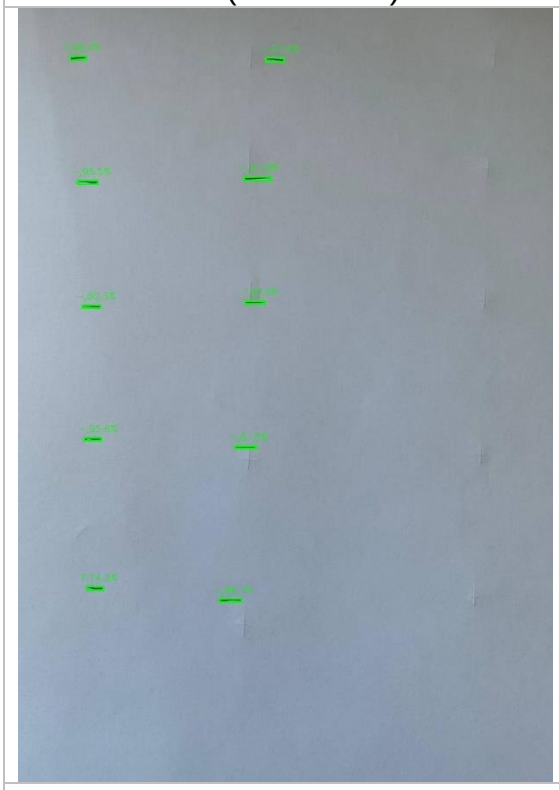
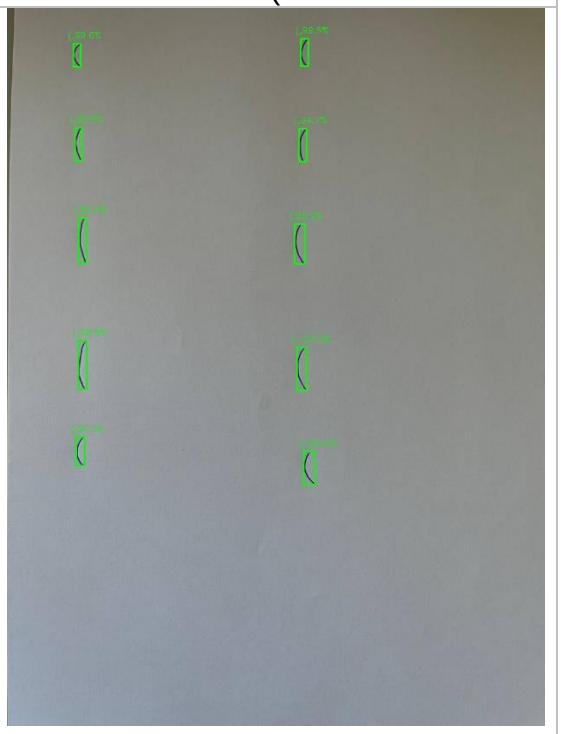
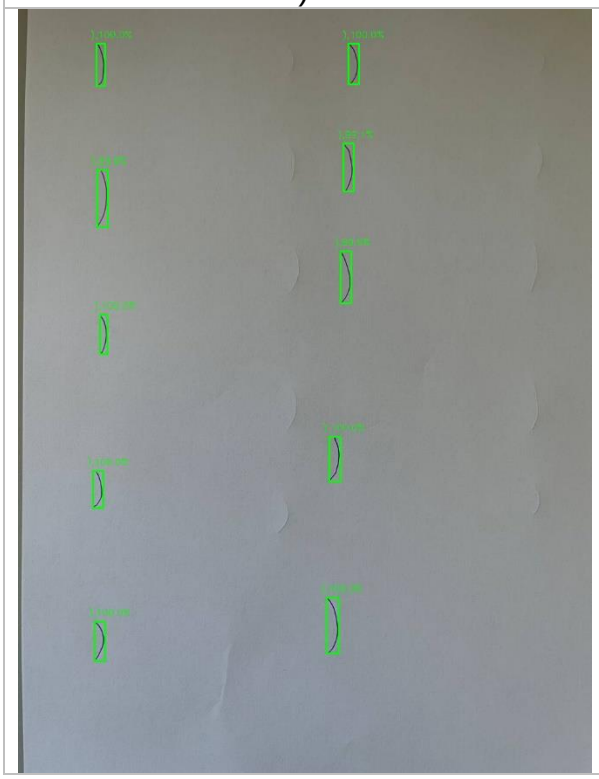
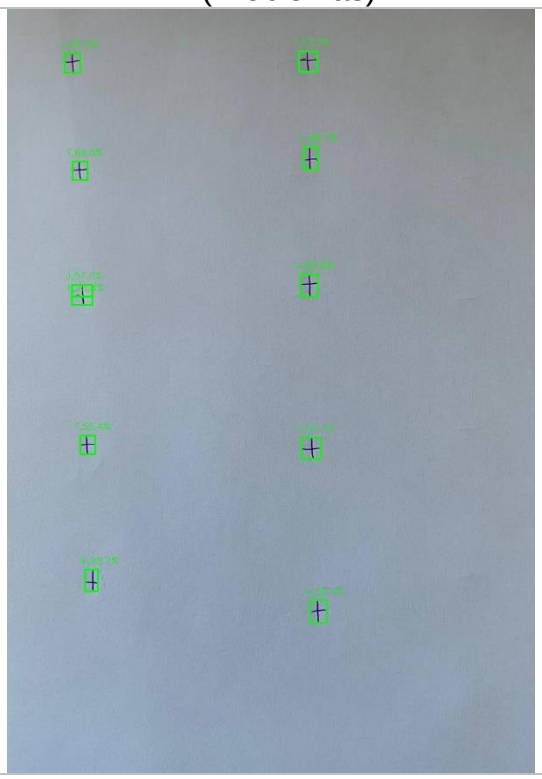
Q (Problemas)	R (Problemas)
	
S (Problemas)	**T**
	

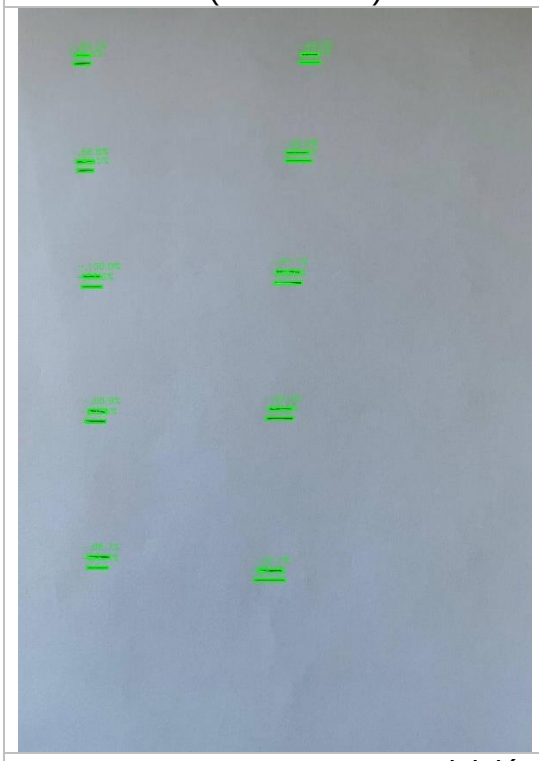
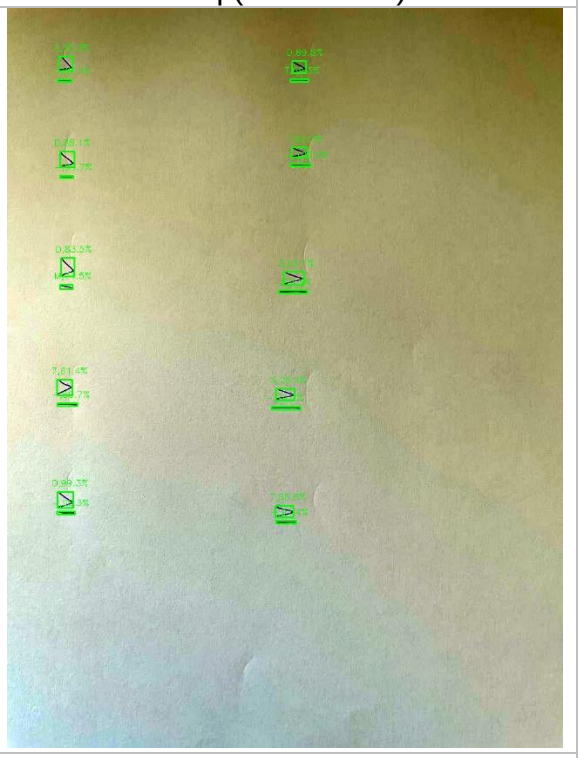
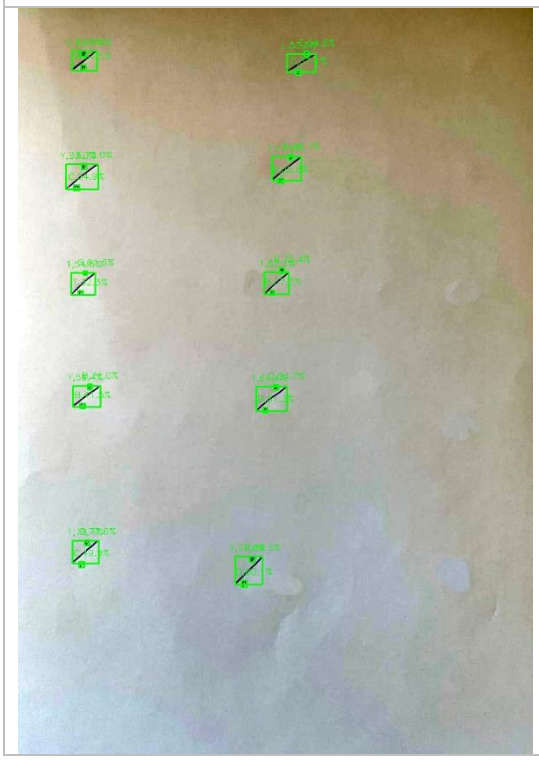
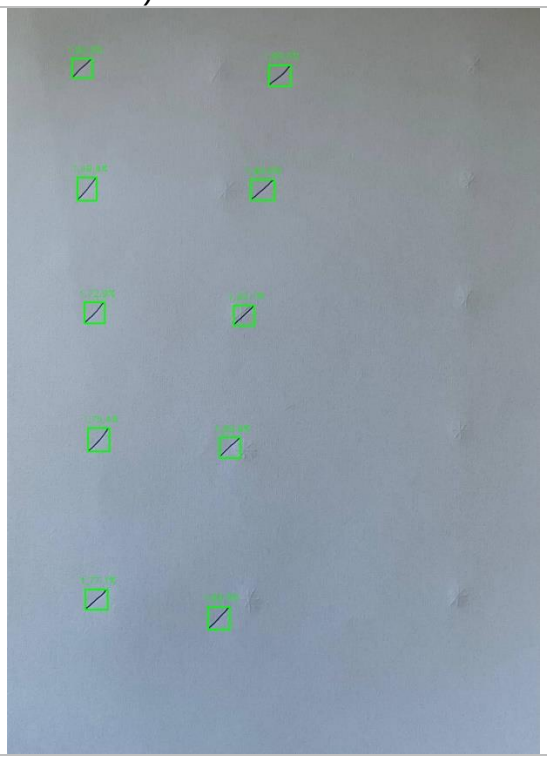
U	V (Problemas)
	
W	**X (Problemas)**
	

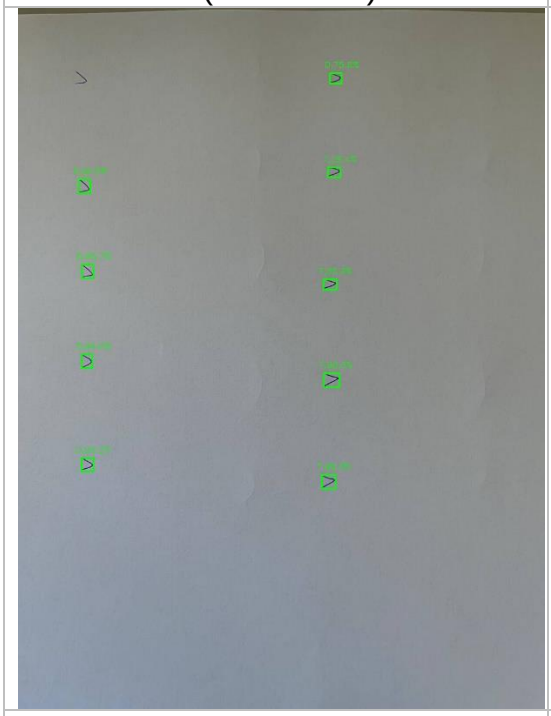
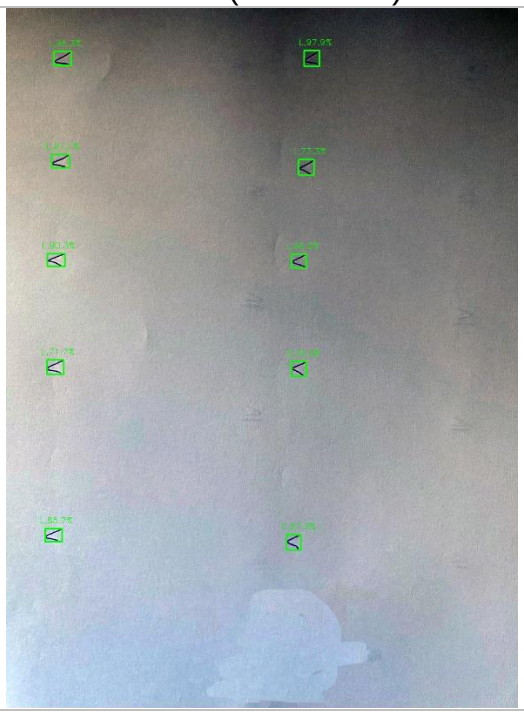
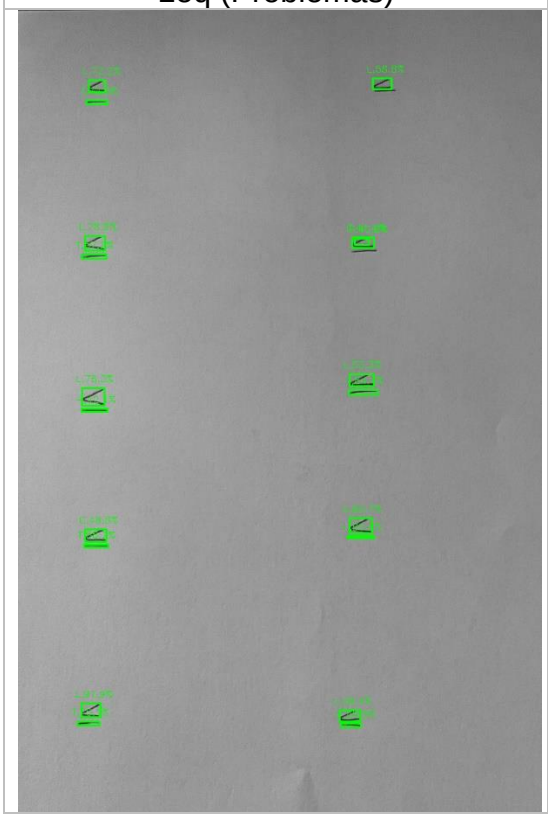
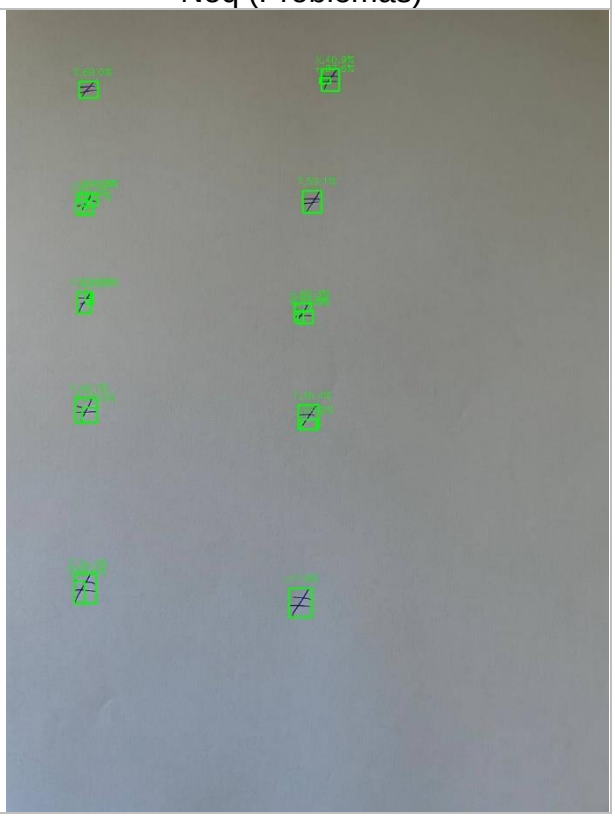
Y (Problemas)	Z (Problemas)
	
0 (Cero) (Problemas)	1 (Problemas)
	

2 (Problemas)	3
	
4	5 (Problemas)
	

6 (Problemas)	7 (Problemas)
	
8 (Problemas)	9 (Problemas)
	

- (Problemas)	(
	
)	+ (Problemas)
	

= (Problemas)	Geq (Problemas)
	
División (Problemas)	
	

> (Problemas)	< (Problemas)
	
Leq (Problemas)	Neq (Problemas)
	

ANEXO 6: Respuesta JSON imgBB

imgbb.json


```
{
  "data":{
    "id":"f2dsKK5",
    "title":"b3bfe3f0a3d6",
    "url_viewer":"https://ibb.co/f2dsKK5",
    "url":"https://i.ibb.co/Mskmjj4/b3bfe3f0a3d6.jpg",
    "display_url":"https://i.ibb.co/2y3TLLQ/b3bfe3f0a3d6.jpg",
    "width":893,
    "height":1280,
    "size":119576,
    "time":1673802671,
    "expiration":1800,
    "image":{
      "filename":"b3bfe3f0a3d6.jpg",
      "name":"b3bfe3f0a3d6",
      "mime":"image/jpeg",
      "extension":"jpg",
      "url":"https://i.ibb.co/Mskmjj4/b3bfe3f0a3d6.jpg"
    },
    "thumb":{
      "filename":"b3bfe3f0a3d6.jpg",
      "name":"b3bfe3f0a3d6",
      "mime":"image/jpeg",
      "extension":"jpg",
      "url":"https://i.ibb.co/f2dsKK5/b3bfe3f0a3d6.jpg"
    },
    "medium":{
      "filename":"b3bfe3f0a3d6.jpg",
      "name":"b3bfe3f0a3d6",
      "mime":"image/jpeg",
      "extension":"jpg",
      "url":"https://i.ibb.co/2y3TLLQ/b3bfe3f0a3d6.jpg"
    },

"delete_url":"https://ibb.co/f2dsKK5/a56ddd826ad99a46a8d74634f0880628"
  },
  "success":true,
  "status":200
}
```

ANEXO 7: Requisitos del programa

Hardware
Este software está testeado para funcionar con el sistema operativo Windows. Por limitaciones de las librerías de machine learning que se emplean en el mismo (TensorFlow), no es posible su empleo en equipos mac con arquitectura Apple Silicon.

Software
Es preciso instalar las librerías necesarias para el intérprete de Python descritas en el fichero “requirements.txt”. Para el empleo del sistema OCR “Tesseract” es necesario instalar sus kernel desde el siguiente enlace: https://tesseract-ocr.github.io/tessdoc/Installation.html

El programa necesita de una conexión activa a internet para su funcionamiento.

ANEXO 8: Limitaciones de los diagramas.

Según el tipo de bloque escrito, existen unas limitaciones de lo que se puede escribir dentro:

- Bloque de tipo “print”: solamente se podrá escribir el nombre de una variable.
- Bloque de tipo “scan”: solamente se podrá escribir el nombre de la variable en la que almacenar el input del usuario.
- Bloque de tipo “process”: se aceptan operaciones matemáticas con la siguiente estructura:
    - Variable = (variable/numero) operador matemático (variable/número)
        - Ejemplo: var1 = var2 + 5
    - Variable (operador matemático de asignación) (variable/número)
        - Ejemplo: var2 += 7
    - En cuanto a los operadores matemáticos que acepta son los siguientes: +, -, *, /.
    - En cuanto a los operadores matemáticos de asignación que acepta son los siguientes: +=, -=, *=, /=.
- Bloque de tipo “decision”: como mínimo deben especificarse dos opciones. Dentro del bloque se escribirá la condición, si es un condicional de tipo “if” contendrá una comparación, y si es un condicional de tipo “switch-case” se escribirá el nombre de la variable. Luego, en las flechas de la condición, en la más próxima al bloque “decision” se pondrá la condición de la rama, en caso de ser de tipo “if” se establecerá la opción “True” y la opción “False”, y en caso de ser de tipo “switch-case”, el valor de la variable. Además, en este último caso, si se quiere poner una rama por defecto se pondrá el texto “DEFAULT”.

Para una correcta interpretación de las figuras por parte del modelo de detección es preciso mantener una buena separación entre las flechas.

En cuanto a los textos escritos, se deben escribir en una sola línea por cada bloque.

Los bloques condicionales deben finalizar en el mismo bloque. Estos bloques tampoco aceptan condicionales dentro de otros.

Finalmente, el programa no admite operaciones sobre cadenas de texto.

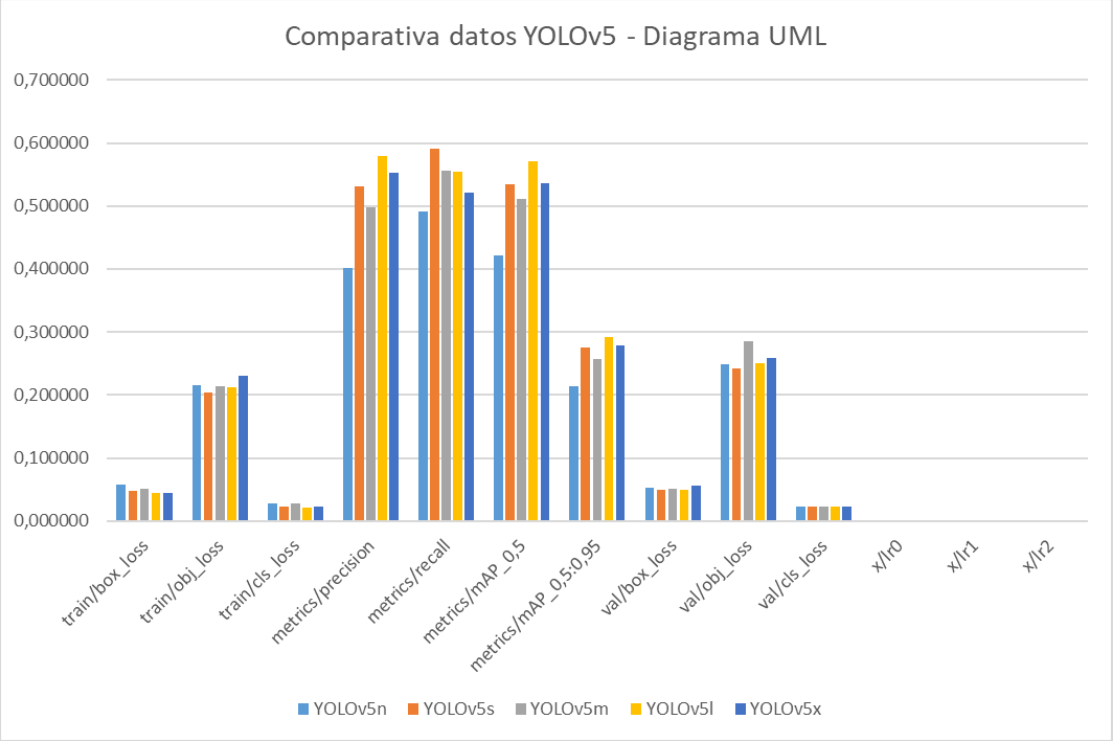
ANEXO 9: Manual de usuario

Una vez satisfechos los requisitos del programa según el Anexo 7, se pasa a la ejecución del programa. Para ello es necesario colocar la fotografía o escaneo del diagrama de flujo en la carpeta “. /User Files/Input Images”.

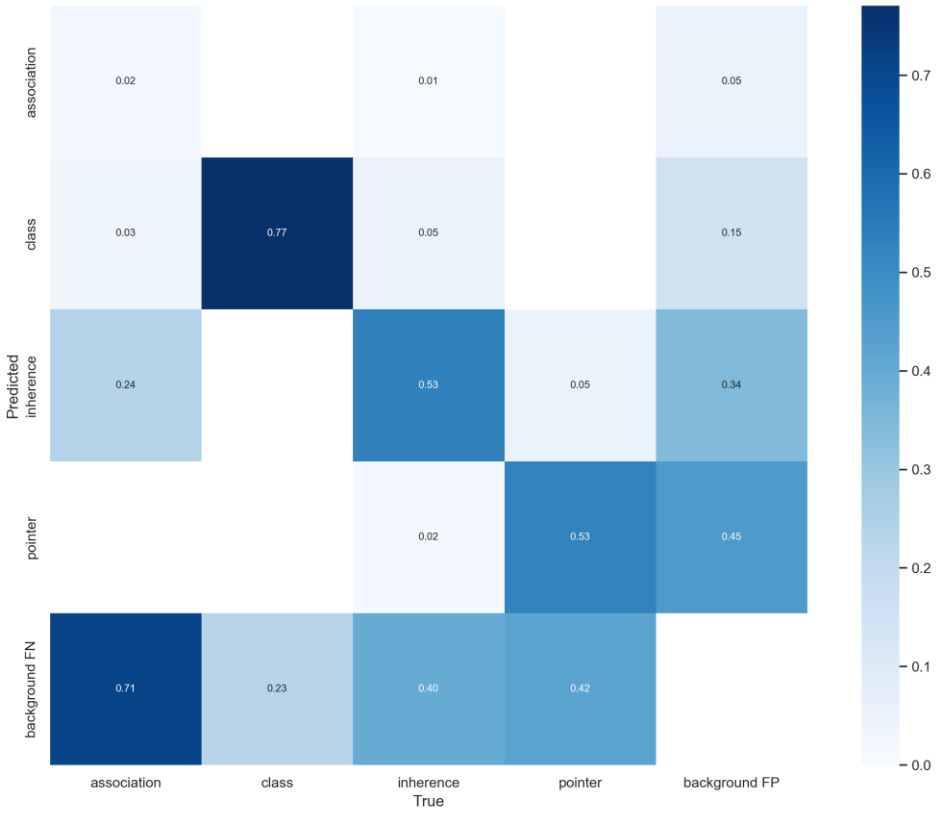
A continuación, se lanzará el script “main.py” desde el entorno de Python desde el que se hayan instalado los requisitos. Lo siguiente que tendrá que hacer el usuario es seguir las instrucciones de selección de diagrama a evaluar, sistema OCR que quiere emplear y finalmente el lenguaje de programación de salida.

Una vez finalizado la ejecución del programa se obtendrá el código fuente del diagrama de flujo en la carpeta “. /User Files/Output Files”.

ANEXO 10: Estadísticas Diagrama UML



Porcentaje de detección



Cuadro de confusión