Universidad Politécnica de Madrid

Proyecto Final:

RECONOCIMIENTO Y EXTRACCIÓN DE CÓDIGO

Fundamentos de Análisis de Imágenes

Ramos Osuna, Víctor
Vulpe, Beatriz Noelia

Curso 2023-2024

Contents

1 Introducción.

Los programadores, independientemente de su nivel de experiencia, dedican tiempo a buscar información para sus tareas diarias. Ya sea a través de medios contemporáneos como YouTube, que alberga una gran cantidad de vídeos de programación, o mediante materiales más tradicionales, todos los usuarios en la red recurren a los conocimientos compartidos por otros.

La práctica común de copiar y pegar código es esencial para los programadores. Sin embargo, las restricciones sobre el material digital a menudo obligan a los desarrolladores a realizar la transcripción manual del código.

Situaciones como presentaciones que no permiten copiar y pegar, o la necesidad de capturar una imagen del código, añaden barreras adicionales que contribuyen a la complejidad y lentitud en la incorporación de código a proyectos. Esto destaca la necesidad de abordar estos desafíos para mejorar la eficiencia y accesibilidad en el aprendizaje y aplicación de la programación.

Con el objetivo de simplificar este proceso, nuestra aplicación se centra en facilitar la extracción de texto a partir de imágenes que contienen código. Los usuarios podrán tomar fotografías o cargar imágenes con código fuente, y la aplicación analizará la imagen para extraer el texto relacionado con el código. Posteriormente, se presentará el texto resultante al usuario, permitiéndole copiar y pegar fácilmente en su propio script.

2 Tipología de los datos.

La aplicación se alimenta de imágenes que contienen fragmentos de código fuente, presentadas en formatos comunes como **JPG, JPEG o PNG**. Nos enfocamos en **material digital** relevante para programadores, incluyendo capturas de pantalla o fotografías tomadas con la cámara de un dispositivo móvil desde Entornos de Desarrollo Integrado (IDE), así como fragmentos de código extraídos de páginas web o presentaciones. Aunque inicialmente no se consideró relevante la inclusión de **fotografías de medios tradicionales**, actualmente se ofrece la opción de introducir imágenes de libros o material impreso, reconociendo la **importancia de abarcar todas las posibles fuentes de material programático**.

La variabilidad inherente en estas imágenes abarca aspectos cruciales como el formato del texto, el lenguaje de programación, la orientación, los colores, la iluminación y la calidad. En el diseño de la aplicación, hemos priorizado la capacidad de procesar eficientemente estas variaciones, entendiendo que la aplicación debe ser capaz de manejar esta diversidad en las imágenes de entrada para garantizar su robustez y utilidad en entornos prácticos y situaciones del mundo real.

2.1 Conjunto de datos de prueba

Para poner a prueba la aplicación y asegurar su eficacia, hemos construido un conjunto de datos de prueba representativo basado en criterios meticulosos:

- **Variedad de Lenguajes de Programación:** El conjunto de datos abarca ejemplos de código en diversos lenguajes de programación, desde aquellos más afines al lenguaje tradicional, como Python y R, hasta aquellos cercanos al lenguaje de máquina, como Java y C++. Esta inclusión estratégica asegura el análisis de la versatilidad de nuestra aplicación. Además, hemos considerado la aplicación en diversos formatos de contenido, incluyendo código y texto variado, como ficheros de Docker o de repositorios en GitHub y otros tipos de documentación programática. Al incorporar esta diversidad, se busca analizar con precisión la eficacia del programa frente a diferentes estilos de codificación y prácticas de documentación.

- **Diversidad de Fuentes y Entornos de Origen:** Se recopilaron muestras de código de diversas fuentes, como repositorios en línea, libros de programación, capturas de pantalla de IDEs y presentaciones. Esto permite que la aplicación sea aplicable en una variedad de situaciones del mundo real, presentando variabilidad en el tipo de texto, la presentación del código, la coloración y el tamaño de letra.

- **Desafíos Visuales y de Calidad:** El conjunto de datos abarca imágenes con desafíos visuales, como problemas de iluminación, presencia de ruido y variaciones en la perspectiva. Estos elementos son comunes en entornos reales y permiten evaluar la robustez de la aplicación ante condiciones menos que ideales.

- **Longitud y Complejidad Variable del Código:** Se han incorporado fragmentos de código de diferentes longitudes y complejidades, desde líneas cortas hasta funciones completas. Esto asegura el análisis del posterior manejo de diversos tamaños y niveles de complejidad en el código fuente.

Disponer de un conjunto de alrededor de 30-40 fotografías con las características detalladas se considera adecuado para realizar un análisis exhaustivo y evaluar el rendimiento de la aplicación. La diversidad y representatividad inherentes en este conjunto de datos proporcionan una base robusta para la evaluación y la mejora continua de la aplicación en situaciones del mundo real. Este conjunto de imágenes también sirve como punto de partida valioso para futuros ajustes, contribuyendo al fortalecimiento de la aplicación en escenarios específicos. Así, aseguramos su evolución constante y la capacidad de adaptarse a las cambiantes necesidades y expectativas de los usuarios.

3 Metodología.

A continuación, se expondrán detalladamente todos los pasos, técnicas y elementos programados que han sido empleados para lograr una aplicación robusta y efectiva que cumpla con el objetivo de facilitar la transcripción de código desde imágenes a texto para los usuarios.

3.1 Procesamiento de Imágenes:

Se han desarrollado múltiples archivos .py en el curso de esta tarea. A continuación, se proporciona una descripción ordenada de cada uno de ellos, junto con una explicación detallada de los métodos que los conforman.

recorte.py

En un principio, se optó por optimizar la aplicación para recortes de pantalla o screenshots. Este enfoque presenta un código más simple y rápido en comparación con la implementación destinada a fotografías. La elección se basa en la premisa de que las capturas de pantalla carecen de componentes de ruido, problemas de iluminación, entre otros, que podrían afectar negativamente al reconocimiento del texto.

La única función de este archivo es **ocr**, que realiza los siguientes pasos:

- Ajusta la escala de la imagen para alcanzar una resolución de 2000 píxeles en el lado de menor resolución.

- En caso de que el color predominante en la imagen (es decir, el fondo) sea el negro, se invierten los colores, ya que nuestro OCR seleccionado ofrece mejores resultados con fondo blanco y texto negro.

- Realiza un threshold con el fin de obtener una imagen binaria, utilizando el threshold de Otsu, especializado para texto, y añade un borde del 5% del lado de la imagen, pues se ha demostrado que esto facilita la detección de texto del ocr.

- Seguidamente se realiza el OCR, utilizando la configuración predeterminada para la segmentación de páginas (3) y el modelo de reconocimiento de la opción 3, que es un modelo LSTM entrenado con miles de textos.

- Los datos del modelo se obtienen mediante su método `image_to_data`, que devuelve un diccionario (que se convertirá a un DataFrame) del cual extraemos información jerárquica de los textos, desde bloques de texto hasta líneas individuales. Al iterar sobre estos datos, se crea un array que contiene líneas de texto junto con su posición con respecto al extremo izquierdo de la imagen, lo cual nos ayuda posteriormente a detectar la indentación en el código.

- Finalmente, dividimos cada una de las distancias por la menor, obteniendo una proporción. Luego, realizamos un redondeo y convertimos a entero, lo que nos proporciona un array con las tabulaciones de cada línea de texto. En algunos casos, las tabulaciones pueden saltarse órdenes en la jerarquía, por ejemplo, que de la tabulación 1 se pase a la 3; esto se soluciona con la función `substitute_with_index`. Una vez obtenidas las tabulaciones, se unen las líneas de texto del array, añadiendo al principio el número de tabulaciones correspondientes y agregando un salto de línea al final. Este string de texto será el valor de retorno de la función.

foto.py

Seguidamente se realizó la implementación para fotografías, dirigida a imágenes de pantallas e incluso de páginas de libros. Este archvo contiene 2 funciones, **correct_skew** y **ocr**, pues por lo general, el texto de imágenes tomadas por humanos no es perfectamente paralelo a su eje correspondiente, pudiendo afectar a la calidad del reconocimiento:

correct_skew

- Toma como argumentos la `imagen` (un numpy array en este momento), y los parámetros `delta y limit`. `Delta` define el paso de las rotaciones, es decir, cuántos grados gira la imagen en cada comprobación, mientras que `limit` determina el rango de grados a probar con un paso de delta. Debido a consideraciones de tiempo de ejecución y a la confianza de que el usuario capture imágenes con una rotación razonable, los valores por defecto son 1 para delta y 10 para `limit`. Esto significa que se prueban todos los valores de 1 en 1 desde -10 hasta 10 grados.

- La propia función tiene una función interna, `determine_score`, que en resumidas cuentas rota la imagen con el ángulo elegido, crea un histograma en función de los cambios entre blancos y negros (`axis 1`) y calcula las diferencias al cuadrado de los cambios de color, de forma que cambios más bruscos indican una mejor alineación del texto. Esta suma de diferencias cuadradas será el score que devuelva la función.

- Ángulo por ángulo, se calcula el score y se elige el ángulo con un mayor score. Finalmente se rota la imagen ese mismo ángulo con eje en el centro de la imagen, y esta imagen rotada será lo que devuelva.

ocr

- Ajusta la escala de la imagen para alcanzar una resolución de 2000 píxeles en el lado de menor resolución.

- En caso de que el color predominante en la imagen (es decir, el fondo) sea el negro, se invierten los colores, ya que nuestro OCR seleccionado ofrece mejores resultados con fondo blanco y texto negro.

- Aplica un blur Gaussiano para reducir el ruido.

- Realiza un threshold con el fin de obtener una imagen binaria, utilizando el threshold de Otsu y binario, seguidamente aplicando la operación OR entre ellos, dando en nuestros experimentos los mejores resultados.

- Crea un pequeño kernel de tamaño (2, 2) y dos iteraciones de erosión de la imagen, con el fin de eliminar el mayor ruido posible y rectas de tabulaciones de código no deseadas, seguidamente aplicando una operación de dilatación en caso de que el texto haya podido sufrir por la erosión.

- Añade un borde del 10% del lado de la imagen, de forma que la corrección de rotación no elimine texto deseado y se aplica `correct_skew` a la imagen resultante.

- Seguidamente se realiza el OCR, utilizando la configuración predeterminada para la segmentación de páginas (3) y el modelo de reconocimiento de la opción 3, que es un modelo LSTM entrenado con miles de textos.

- Los datos del modelo se obtienen mediante su método `image_to_data`, que devuelve un diccionario (que se convertirá a un DataFrame) del cual extraemos información jerárquica de los textos, desde bloques de texto hasta líneas individuales. Al iterar sobre estos datos, se crea un array que contiene líneas de texto junto con su posición con respecto al extremo izquierdo de la imagen, lo cual nos ayuda posteriormente a detectar la indentación en el código.

- Finalmente, dividimos cada una de las distancias por la menor, obteniendo una proporción. Luego, realizamos un redondeo y convertimos a entero, lo que nos proporciona un array con las tabulaciones de cada línea de texto. En algunos casos, las tabulaciones pueden saltarse órdenes en la jerarquía, por ejemplo, que de la tabulación 1 se pase a la 3; esto se soluciona con la función `substitute_with_index`. Una vez obtenidas las tabulaciones, se unen las líneas de texto del array, añadiendo al principio el número de tabulaciones correspondientes y agregando un salto de línea al final. Este string de texto será el valor de retorno de la función.

shared.py

Contiene las funciones auxiliares utilizadas por foto.py y recorte.py. Estas 2 funciones, **resize_image** y **substitute_with_index**, se encargan de los siguiente:

resize_image

- Descrito por su nombre, esta función se encarga de reescalar la imagen para que el ocr de mejores resultados. Las letras deben tener un tamaño de 32 píxeles como mínimo. Dado que desconocemos la cantidad de texto de la imagen podemos asumir que con 2000 píxeles en el lado más pequeño de la imagen será suficiente. Esta medida es completamente customizable en el parámetro `target_size`.

substitute_with_index

- Dado un array de números de diferente valor, los sustituye por el valor de su índice en un set ordenado ascendentemente, estableciendo una jerarquía para las identaciones del texto.

3.2 Reconocimiento Óptico de Caracteres (OCR):

Se ha utilizado como modelo **Tesseract de UB Mannheim**[1], un port de Tesseract OCR para Windows para convertir la información visualizada en las imágenes (código) en texto legible por la máquina (originalmente entrenada para reconocer textos antiguos en alemán).

Tanto en recorte.py como foto.py se hace una llamada al ejecutable del modelo para que evalúe la imagen procesada y detecte el texto con la mayor precisión posible, ofreciendo datos de posición de texto, valor, e incluso medidas de confianza.

En particular, se ha empleado la versión 5 del modelo, que suele ser considerablemente más efectiva y presenta una mejor capacidad de reconocimiento en textos con color negro sobre fondo blanco.

3.3 Interfaz de Usuario:

Se ha desarrollado una interfaz de usuario amigable que permite a los usuarios cargar imágenes de forma sencilla, visualizar los resultados del procesamiento y copiar el texto extraído. La interfaz está diseñada para ser intuitiva y centrada en la experiencia del usuario.

En pos de simplificar este proceso, se ha empleado **Streamlit**[2], un marco de desarrollo de código abierto que posibilita la creación rápida y fácil de aplicaciones web con Python. La presentación del proyecto en una página web ha sido adoptada para facilitar el acceso de los usuarios y cargar sus imágenes de manera conveniente.

---

[1]https://github.com/UB-Mannheim/tesseract/wiki
[2]https://streamlit.io/

A pesar de la intención inicial de ofrecer una interfaz fácil de usar para usuarios sin experiencia en programación Python y sin necesidad de utilizar contenedores virtuales, sino con un funcionamiento completo en línea desde cualquier navegador, es esencial destacar que la elección del modelo de OCR **Tesseract de UB Mannheim**, específicamente diseñado para Windows, ha influido en las opciones de presentación de la aplicación a los usuarios. Dado que este modelo está destinado a sistemas operativos Windows, se ha limitado la compatibilidad con infraestructuras en la nube, que suelen estar más orientadas hacia sistemas operativos basados en Linux. Además, la configuración local se ha vuelto esencial debido a posibles dependencias y bibliotecas específicas del modelo que podrían no estar correctamente configuradas en entornos de la nube.

Considerando todos estos factores, se ha implementado la integración de la aplicación web en dispositivos Windows, operando en un entorno local con un mínimo uso de comandos Python. Esta decisión garantiza un rendimiento óptimo y una experiencia fluida para los usuarios, simplificando así la interacción y asegurando un acceso fácil a la funcionalidad del programa sin la necesidad de complejos comandos o configuraciones.

Para la integración de la aplicación y la página web, se implementaron dos archivos clave:

- **code_extract.py:** Este archivo actúa como intermediario entre la interfaz del usuario y los módulos de procesamiento de imágenes. Recibe información del servidor de Streamlit sobre los datos de entrada del usuario, como la imagen y especifica si es un recorte o una foto. En función de esta información, determina el uso de las funciones en foto.py o recorte.py, ejecuta el OCR y envía los resultados al servidor.

- **main.py:** Este archivo constituye el frontend de la aplicación. Inicia un servidor web que posibilita la interacción del usuario con main.py, permitiendo visualizar los resultados de las ejecuciones de manera intuitiva a través de la interfaz web.

La combinación de estos dos archivos garantiza una experiencia completa y sencilla para el usuario, desde la carga de la imagen hasta la presentación de los resultados extraídos.

4 Integración en máquina del programa

La aplicación web ha sido integrada con éxito en dispositivos que operan bajo el sistema operativo Windows, funcionando de manera local y minimizando la necesidad de utilizar comandos en Python. Se ha simplificado significativamente la interacción, garantizando un acceso fácil a todas las funcionalidades del programa sin requerir que los usuarios realicen configuraciones complicadas o utilicen comandos complejos. Esta implementación ha sido diseñada para proporcionar una experiencia fluida a los usuarios, sin comprometer el rendimiento.

La aplicación es totalmente compatible con sistemas operativos Windows de 64 bits y su funcionamiento requiere la presencia de Python[3]. A continuación, proporcionamos detalladamente los pasos para la instalación y uso del programa:

- **Obtención de la Aplicación:** Descarga la última versión de la aplicación desde la fuente oficial de github[4].

---

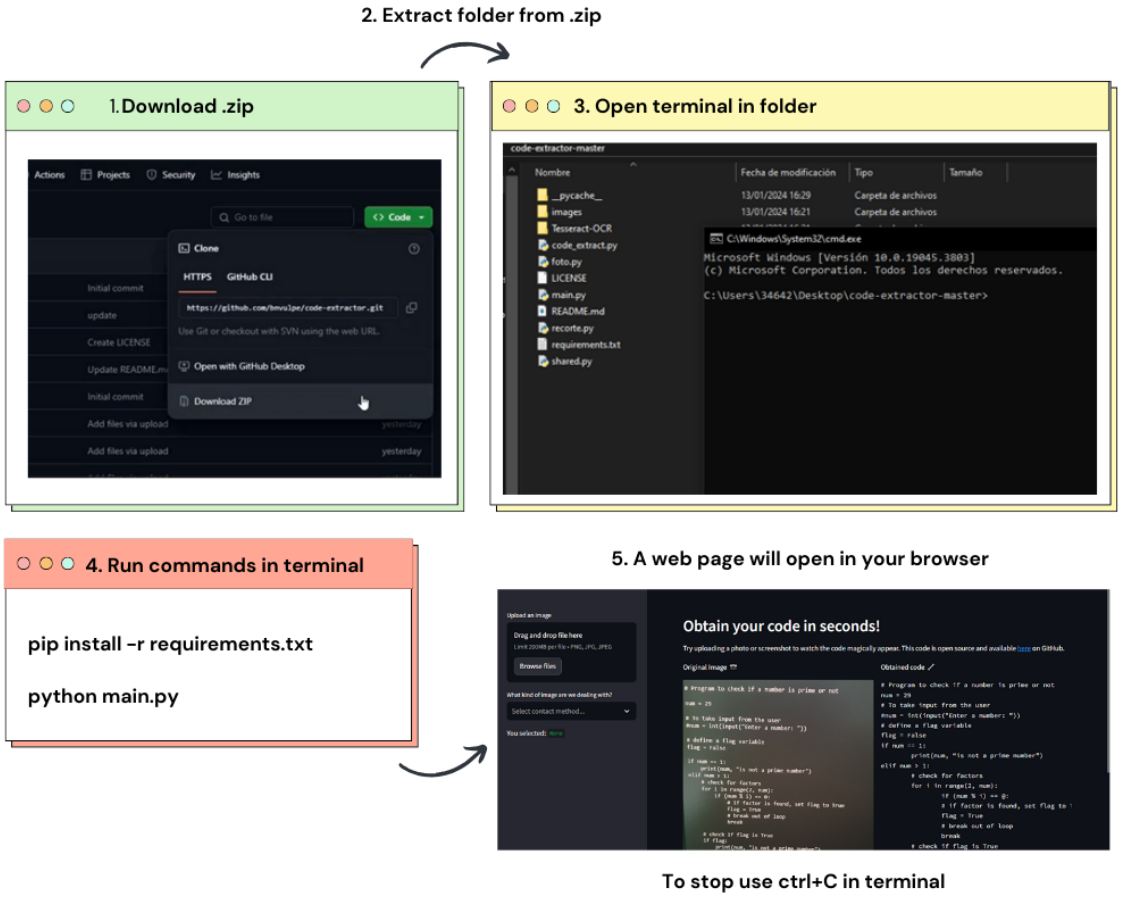
[3]https://www.python.org/downloads/
[4]https://github.com/bnvulpe/code-extractor

- **Descompresión del Archivo:** Descomprime el archivo .zip en una ubicación de tu elección.

- **Configuración del Entorno:**
  1. Accede al directorio de la aplicación.
  2. Verifica que Python esté añadido al PATH del sistema para facilitar la ejecución de comandos.

- **Ejecución de la Aplicación:**
  1. Abre una ventana de terminal en el directorio de la aplicación.
  2. Instala los requisitos con el comando pip install -r requirements.txt
  3. Ejecuta el programa con el comando 'python main.py'
  4. En caso de cerrar el programa introduce ctrl+c en la terminal

Siguiendo estos pasos, la aplicación se ejecutará sin complicaciones en tu entorno local de Windows, ofreciendo una experiencia de usuario intuitiva y eficiente.



5 Experimentaciones

La naturaleza de nuestro problema presenta desafíos que dificultan la aplicación de métricas matemáticas o estadísticas convencionales para evaluar el rendimiento. A diferencia de problemas que involucran la identificación binaria de objetos o la segmentación de un objeto en particular, nuestro enfoque aborda la transcripción de código fuente, que exhibe un rango ilimitado de variabilidad. En este contexto, limitarnos a una evaluación binaria de éxito o fracaso no sería adecuado.

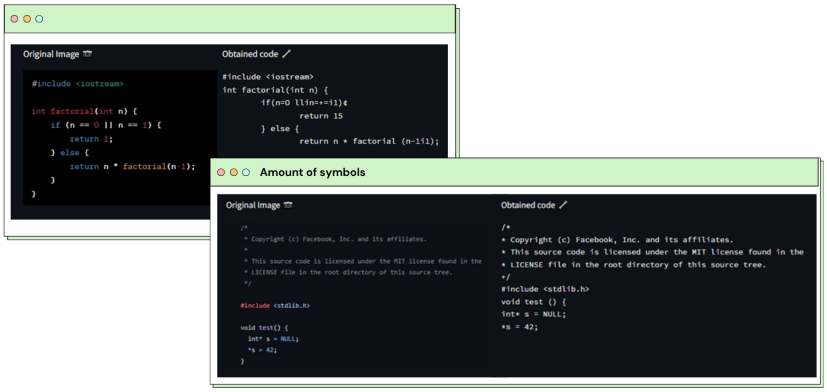
Una posible alternativa sería realizar una transcripción manual para etiquetar el código en todas nuestras muestras y, a partir de ese etiquetado, considerar el desarrollo de métricas específicas para la evaluación de errores en nuestro caso particular. Sin embargo, dado que nuestro objetivo principal es ofrecer a los usuarios la oportunidad de reducir el tiempo dedicado a la transcripción de código, así como concienciar sobre las posibles limitaciones de los enfoques clásicos de tratamiento de imágenes y sugerir mejoras futuras para nuestra aplicación, no consideramos relevante centrarnos en métricas matemáticas tradicionales para calcular el error.

En cambio, optamos por llevar a cabo nuestras experimentaciones con el propósito de presentar al usuario los mejores casos de uso y proporcionar una comprensión completa de las posibilidades de nuestra aplicación. Este enfoque nos permite destacar escenarios prácticos y situaciones del mundo real en las que nuestra aplicación puede ofrecer beneficios significativos, y al mismo tiempo, reconocer las limitaciones actuales y posibles áreas de mejora.

5.1 Evaluación en diferentes escenarios de uso

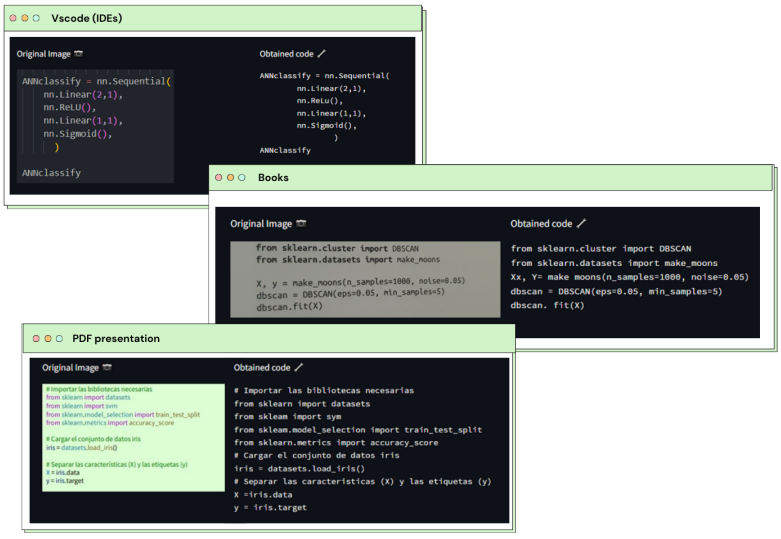
Durante el desarrollo de la aplicación, se llevaron a cabo experimentaciones exhaustivas para evaluar la efectividad y las limitaciones en diferentes escenarios. A continuación, se detallan las principales pruebas realizadas:

- **Variación de Lenguajes de Programación**: Se llevaron a cabo pruebas utilizando muestras de código en diversos lenguajes de programación con el objetivo de evaluar la capacidad de la aplicación para reconocer una amplia gama de sintaxis. Se observó que los lenguajes de programación más cercanos al lenguaje tradicional proporcionaron mejores resultados, gracias al material de entrenamiento del modelo de OCR utilizado. Esto se traduce en un rendimiento óptimo para lenguajes con sintaxis más sencilla, evitando complejidades derivadas de una simbolización excesiva en su escritura.

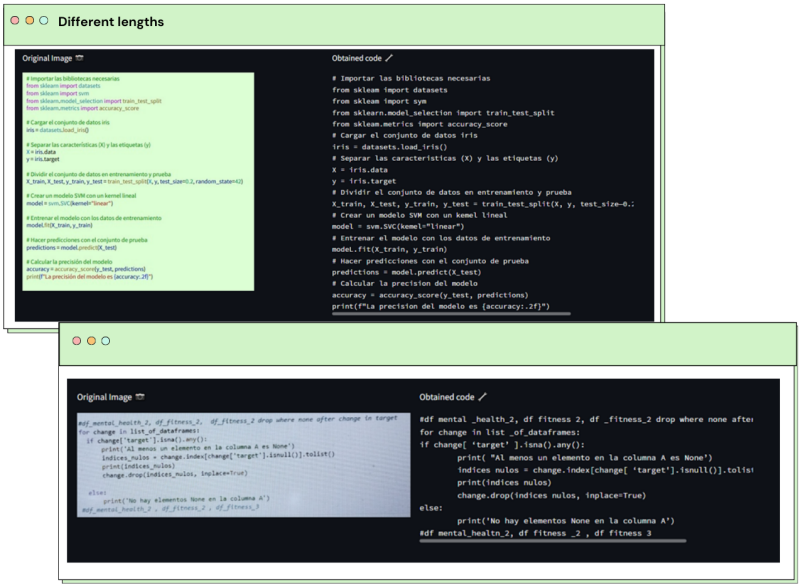


- **Iluminación y Condiciones Ambientales**: Diferentes configuraciones de iluminación fueron evaluadas, incluyendo luz directa y variaciones en la intensidad. Se examinaron los efectos de las condiciones ambientales en la precisión del reconocimiento. Es importante señalar que, contando con un procesamiento general de imágenes, la amplia variabilidad ambiental puede plantear desafíos significativos, afectando al tiempo de ejecución y a los resultados.

- **Ubicación del Código**: Se realizó una evaluación de la robustez del sistema ante diversas combinaciones de colores de texto y fondo. Se realizaron experimentos utilizando código obtenido de diversas fuentes, como repositorios en GitHub, presentaciones en PDF, libros de texto, entornos de desarrollo integrado (IDE) como Visual Studio Code y plataformas en línea como Google Colab. Es importante destacar que estos aspectos están influenciados por las mismas consideraciones que la variación en los lenguajes de programación.



- **Cantidad de Código en las Muestras**: Pruebas con muestras de diferentes longitudes se llevaron a cabo para entender cómo el tamaño del código afecta el rendimiento de la aplicación, tanto en términos de tiempo de procesamiento como de precisión en la transcripción. Dada la limitación del modelo en el tamaño de las letras, es comprensible que se observe un rendimiento inferior en muestras con un mayor número de líneas de código, considerando las proporciones finales de cada línea. Considerando el rápido tiempo de ejecución del programa, se recomienda capturar imágenes o recortes con una menor cantidad de líneas para asegurar una detección eficaz del texto. Esto ayudará a evitar en gran medida posibles correcciones que el usuario pueda tener que realizar.



- **Manipulación de Perspectiva y Ángulo**: Se examinaron los efectos de la manipulación de perspectiva y ángulo en la precisión de la aplicación, particularmente en el reconocimiento de indentaciones y símbolos complejos. Dada la implementación de un rango de rotación de -10 hasta 10 grados en nuestra función `correct_skew`, cualquier ángulo fuera de este rango podría ocasionar problemas. Es importante destacar que este aspecto es personalizable por parte del usuario al disponer del código completo de la aplicación.

- **Comparación entre Recortes y Fotografías**: Se llevó a cabo una comparación entre los resultados obtenidos al tratar las imágenes como recortes frente a fotografías, analizando la eficiencia y precisión de ambos enfoques en función de las necesidades del usuario. La aplicación enfrenta desafíos particulares en el reconocimiento de símbolos propios de la programación y en la gestión de diferentes tipos de paréntesis. Además, puede experimentar dificultades en la correcta interpretación de las indentaciones y el número '0' según la tipografía utilizada.

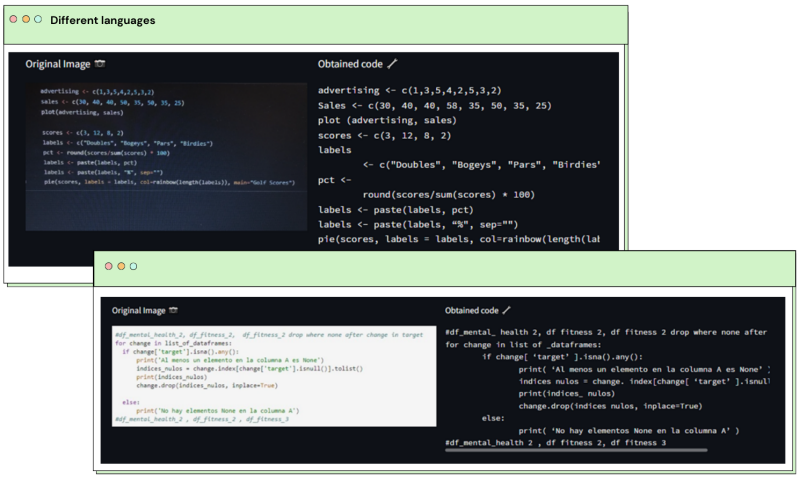
  A pesar de que los libros ofrecen mayor nitidez y calidad en comparación con las fotografías de pantallas, aún presentan fallos comunes inherentes al modelo de Tesseract. Por otro lado, las fotografías tienen una probabilidad mayor de fallo en comparación con los recortes. Además, la corrección de perspectiva, aunque beneficiosa para mejorar la legibilidad, puede aumentar significativamente el tiempo de cómputo, afectando la eficiencia general de la aplicación.

5.2 Limitaciones y Consideraciones

- **Corrección de Código:** La aplicación actual no incorpora funcionalidades de corrección de código, lo que significa que cualquier error de reconocimiento debe ser manejado manualmente por el usuario. Esta limitación destaca la importancia de verificar y corregir el código extraído según sea necesario.

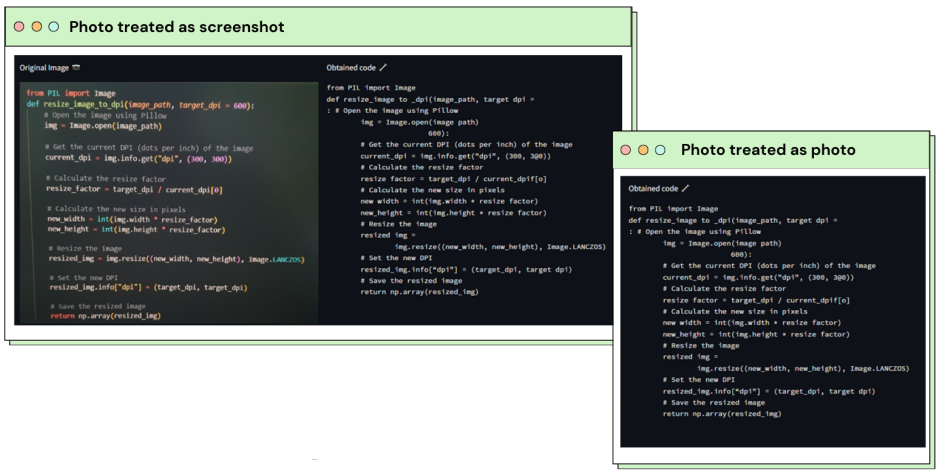
- **Restricciones de Tesseract:** Tesseract, siendo la base del reconocimiento óptico de caracteres (OCR), impone limitaciones en la detección de símbolos específicos de programación, como guiones bajos, comillas, dobles asteriscos, y acumulaciones de diferentes tipos de paréntesis.

- **Soporte para Otros Lenguajes de Programación:** La aplicación es capaz de reconocer diversos lenguajes de programación. Sin embargo, la detección puede ser menos precisa en lenguajes más abstractos o no tradicionales.



- **Condiciones Inapropiadas:** Errores pueden surgir en condiciones de luz, contraste, ángulo y perspectiva no apropiados. Se destaca la responsabilidad del usuario en garantizar una comprensión clara de la imagen y en ajustar las condiciones para mejorar la legibilidad.

- **Tratamiento como Fotografía vs. Recorte:** En condiciones donde la imagen no presenta desafíos visuales significativos para el ojo humano, la aplicación puede devolver un código adecuado con el tratamiento de la fotografía como un recorte. Sin embargo, el usuario debe considerar sus prioridades al elegir entre menor tiempo de ejecución o una disminución de errores en el código resultante.



Estas consideraciones resaltan la importancia de la interacción y supervisión del usuario para obtener resultados óptimos en una variedad de situaciones prácticas.

6 Conclusiones y posibles mejoras

Luego de numerosos experimentos y una cantidad considerable de modificaciones, hemos alcanzado varias conclusiones:

- **Cualquier método de preprocesamiento puede quedarse corto**: Esto es evidente en los experimentos, donde un ángulo inadecuado puede impedir la corrección de la rotación, y una perspectiva incorrecta puede resultar en tabulaciones erróneas. Dado que un archivo de código puede tener numerosas indentaciones no necesariamente alineadas y puede terminar con una indentación distinta de cero, no es posible prever la posición y perspectiva exactas del texto para su corrección, lo que dificulta una tabulación precisa. Incluso al realizar pruebas como recubrir el bloque de código con un polígono con el fin de encontrar la perspectiva, este desafío persiste.

  Para esta tarea pueden ser necesarios modelos convolucionales capaces de detectar posición de texto e indentación por contexto. Además, algunas interfaces como la de Google Colab implementan rectas verticales que indican identación, algo que puede ser beneficioso si todos los casos fueran así, en el nuestro, era necesaria su eliminación para que Tesseract no interpretara como símbolo "|" estas rectas, pero su eliminación provocaba pérdidas en símbolos del propio código. A esto hay que sumarle la iluminación; una luz apuntando directamente a la pantalla, demasiadas sombras o una frecuencia de refresco más baja que la de la cámara llevará a regiones de la imagen completamente blancas o negras que un median o gaussian blur junto con threshold no puede corregir, llevando a ruido o manchas de color en la imagen irreparables, es decir, es algo que no se puede automatizar para elegir los mejores parámetros en cada caso, al menos con las herramientas que tenemos a nuestra disposición.

- **Tesseract no es 100% preciso**: Es fácil ver que Tesseract puede cometer errores, pues al estar entrenado con plaintext, le es dificil detectar símbolos propios de la programación o combinaciones de los mismos. Con algunos de los ejemplos proporcionados vemos como numerosas veces no es capaz de detectar varios paréntesis juntos o ceros, sustituyéndolos con símbolos @. Incluso, en algunos casos, existiendo texto que anteriormente había detectado correctamente, puede llegar a fallar si las líneas de texto son lo suficiéntemente largas. Finalmente Tesseract aporta un problema paramétrico más, pues necesita que los caracteres tengan una altura de mínimo 32 píxeles, por tanto imágenes con muchas líneas no pueden ser sometidas al mismo reescalado que una con menos, llevando al dilema de en general subir el escalado de la imagen a costa de tiempos de ejecución mucho más elevados.

De esta manera, contamos con una aplicación que es aceptable en la mayoría de los casos, ofreciendo resultados en su generalmente confiables y que en caso de pequeños fallos pueden ser corregidos por el usuario, siempre que este haya aportado una foto lo suficientemente buena.

Existen posibles mejoras, como sistemas de procesamiento basados en inteligencia artificial que puedan aplicar filtros que mitiguen ruido e iluminación indeseada de manera inteligente, o modelos más efectivos como lo es Google Lens (aunque requiera un coste monetario).