universidad de león

Escuela de Ingenierías

Industrial, Informática y Aeroespacial

MÁSTER EN INVESTIGACIÓN EN CIBERSEGURIDAD

Trabajo de Fin de Máster

Enhancement of automatic URL extraction in Smishing

Autor: **Manuel Clerigué García**
Tutores: **Alicia Martínez Mendoza** y
Eduardo Fidalgo Fernández

(Julio, 2023)

UNIVERSIDAD DE LEÓN
Escuela de Ingenierías Industrial, Informática y Aeroespacial

MÁSTER EN INVESTIGACIÓN EN CIBERSEGURIDAD

Trabajo de Fin de Máster

ALUMNO: Manuel Clerigué García

TUTORES: Alicia Martínez Mendoza y Eduardo Fidalgo Fernández

TÍTULO: Enhancement of automatic URL extraction in Smishing

TITLE: Enhancement of automatic URL extraction in Smishing

CONVOCATORIA: Julio, 2023

RESUMEN:

La detección de ataques de *phishing* es un problema de cierta relevancia en ciberseguridad debido al uso de ingeniería social para engañar a las personas para que compartan información confidencial.

En este Trabajo de Fin de Máster (TFM) se aborda la detección de una variante del *phishing*: el *smishing*, que realiza este tipo de engaños mediante mensajes de texto (SMS) en los que se suplanta a una entidad legítima y se exige información confidencial o la realización de ciertos pagos.

Este TFM se desarrolla basándose en una necesidad real de INCIBE (Instituto Nacional de Ciberseguridad): la extracción de la URL presente en el texto de un mensaje de *smishing*, ya que la mayoría de SMS suelen contener un enlace que lleva a una copia de la web legítima de la entidad donde se pretende que la víctima realice un pago o ingrese datos confidenciales.

Una vez extraído el enlace de manera automática, este podría ser analizado para determinar si pertenece a un sitio web malicioso o a uno legítimo y por consiguiente identificando si el SMS es un caso de *smishing*.

Por este motivo, el objetivo que se plantea en este trabajo es mejorar la exactitud de un sistema previamente implementado capaz de realizar esa extracción y que fue proporcionado por el Grupo de Visión y Sistemas Inteligentes (GVIS) de la Universidad de León. El conjunto de datos utilizado también fue cedido por el grupo GVIS y está compuesto por 400 imágenes de capturas de pantalla de mensajes SMS, correos electrónicos y notificaciones con uno o varios enlaces en diferentes formatos.

Se investigó sobre la detección automática de URL y el *smishing* con el fin de conocer el estado del arte y de encontrar posibles cambios o mejoras a introducir en el sistema. Se detectó que la mayor parte de las publicaciones revisadas tratan sobre temas relacionados y no directamente sobre la extracción de URL, debido a la falta de estudios que se centren en este tema concreto.

Inicialmente, el sistema partía con una exactitud de 40.38 % evaluada sobre 400 imágenes que contenían 439 URL. Se analizó la solución, identificando aquellas partes del código o problemas que mayor influencia podrían tener en el resultado final si se mejoraran. Se identificaron y solucionaron, entre otros, problemas: en el algoritmo de detección de regiones de texto empleado, errores en el código como el uso de expresiones regulares y de un método de redimensionado de imágenes incorrectos, y se gestionaron excepciones previamente no contempladas, como qué hacer cuando falla la extracción de una URL inicialmente o cómo incrementar el rendimiento del programa cuando trabaja con un número muy elevado de imágenes.

También se realizó un análisis de las librerías empleadas, identificando que 5 de ellas debían ser actualizadas para solucionar vulnerabilidades presentes en las versiones con las que se trabajaba en el programa.

El sistema final con las mejoras implementadas logró una precisión del 45.79 %, 5.41 puntos porcentuales por encima de la inicial, y un tiempo de ejecución de 78 minutos, 9 veces menos que el tiempo de 12 horas original. Es importante destacar que si la solución admitiera un error de hasta 5 caracteres, la precisión se incrementaría hasta el 74.26 %.

Este trabajo también ha planteado varios puntos para futuros estudios que pueden abordarse para lograr una mayor precisión, como la detección de enlaces texto subrayados, o la introducción de un nuevo OCR.

ABSTRACT:

Detecting phishing attacks is a problem of some relevance in cybersecurity due to the use of social engineering to trick people into sharing confidential information.

This work addresses the detection of a variant of phishing: smishing, which performs this type of deception by means of text messages (SMS) in which a legitimate entity is

impersonated demanding confidential information or the completion of certain payments.

This Master's Thesis (TFM) is developed based on a real need of INCIBE (National Institute of Cybersecurity): extracting URL present in the text of a smishing message, as most SMS tend to contain a link that leads to a copy of the legitimate website of the entity being supplanted where the victim is asked to make a payment or enter confidential data.

Once the link is automatically extracted, it could be analyzed to determine whether it belongs to a malicious website or a legitimate one, and consequently identifying whether the SMS is a case of smishing.

Consequently, the objective of this work is to improve the accuracy of a previously implemented system capable of performing this extraction, which was provided by the *Group for Vision and Intelligent Systems* (GVIS) of the University of León. The dataset used was also provided by GVIS. It is composed of 400 images of screenshots of SMS messages, emails, and notifications with one or more links in different formats.

Research on automatic URL detection and smishing was initially carried out in order to know the state of the art and to find possible changes or improvements to be introduced in the system. It was found that most of the publications reviewed dealt with related topics, but they did not directly address URL extraction or smishing, due to the lack of studies focusing on this specific topic.

Initially, the system started with an accuracy of 40.38 % evaluated on 400 images containing 439 URLs. The solution was reviewed, identifying those parts of the code or problems that could have the greatest influence on the final result if improved. Some of the problems that were identified and solved were the text region detection algorithm used and errors in the code, such as the use of incorrect regular expressions and image resizing methods. Finally, exceptions not previously contemplated were addressed, including what to do when the extraction of a URL initially fails or how to increase the program's performance when working with a large number of images.

The final system with the implemented improvements achieved an accuracy of 45.79 %, 5.41 percentage points above the initial accuracy, and a run time of 78 minutes, 9

times less than the original time of 12 hours. It is important to note that if the solution admitted an error of up to 5 characters, the accuracy would increase to 74.26 %.

This paper has also raised several points for future studies that can be addressed to achieve greater accuracy, such as the detection of underlined text links or the introduction of a new OCR.

Palabras clave: Visión Artificial, Smishing, IA, SMS

Índice de contenidos

Índice de tablas

Índice de imágenes

Glosario

IA: ciencia encargada de la fabricación de máquinas o programas inteligentes capaces de replicar la inteligencia humana. [1]

Phishing: tipo de ataque informático que consiste en el envío de correos electrónicos a usuarios haciéndose pasar por una entidad legítima con un fin malicioso, como el robo de información privada. [2]

Smishing: tipo específico de *phishing*, que consiste en el envío de SMS que simulan ser una entidad legítima con objetivo malicioso, como robar información o demandar un supuesto pago pendiente.

F1 score: en español puntuación F1, es una métrica que mide la habilidad de un modelo para realizar ciertas predicciones o tareas, que combina exactitud (del inglés *precision*) y exhaustividad (del inglés *recall*) mediante la media armónica de ambos valores. [3]

Accuracy: la precisión es una métrica que mide la habilidad de un modelo para realizar ciertas predicciones o tareas, que relaciona el número de predicciones correctas con el número total de predicciones. [4]

Data augmentation: en español ampliación de datos, consiste en el uso de diferentes técnicas de modificación de imágenes (rotar la imagen, cambio de colores, etc.) para expandir la cantidad de elementos del set de datos. [5]

Super-resolution: super-resolución, es una rama de la IA que aborda el problema de mejorar la calidad de una imagen con baja resolución valiéndose principalmente de técnicas de aprendizaje automático. [6]

Machine Learning: en español aprendizaje automático, es una rama de la IA que se centra en el análisis de datos con el uso de algoritmos y otros sistemas para conseguir que un sistema aprenda de manera gradual y de forma similar a como lo haría una persona. [7]

OCR: el reconocimiento óptico de caracteres (del inglés, *Optical Character Recognition*) es una tecnología capaz de analizar imágenes y de extraer el texto presente en ellas. [8]

Thresholding: en español umbralización, es una técnica que permite segmentar imágenes o datos dado un determinado umbral o valor límite: los píxeles de una imagen se clasificarán en inferiores al umbral y superiores al umbral, dejándola en blanco y negro. [9]

Capítulo 1

1.1 INTRODUCCIÓN

La detección de ataques de *phishing* es una de las principales preocupaciones de la ciberseguridad, dado que constituye una de las formas más fáciles y comunes [10] de engañar a la gente para que revele información confidencial como su DNI, dirección de residencia, correo electrónico, contraseña..., que luego puede ser fácilmente utilizada con fines maliciosos. Además, esta técnica es efectiva sin importar los conocimientos específicos de la persona sean sobre el tema: estos ataques afectan tanto a personas con amplios conocimientos en la materia, como a personas con menos formación en el ámbito tecnológico.

Dentro del ámbito del *phishing* también se pueden encontrar diferentes variedades específicas, entre las que cabe mencionar el *smishing*.

Este tipo de ataque es uno de los más comunes y fáciles de llevar a cabo [11] [12]. Consiste en el envío de mensajes de texto (SMS), en los que alguien se hace pasar por una entidad de confianza con el fin de engañar a la víctima para que proporcione información confidencial, instale *malware* o realice otro tipo de acciones con algún fin malicioso [13].

En un ataque típico de *smishing*, el atacante envía un mensaje de texto a la víctima haciéndose pasar, por ejemplo, por un banco o una empresa de reparto, y le pide que facilite información personal o que haga clic en un enlace para efectuar un pago pendiente, empleando un lenguaje apremiante para crear en la víctima una sensación de urgencia que le haga actuar de inmediato sin darle mucho tiempo a analizar la situación.

Además, este tipo de situaciones suelen venir reforzadas, también, por ataques realizados previamente contra el usuario en los que se ha robado información, como el banco, los servicios o las webs con las que interactúa de manera más habitual, lo que puede dar una mayor legitimidad al mensaje a ojos de la víctima.

El objetivo de este Trabajo de Fin de Máster (TFM) es la extracción de la URL presente en el texto de un mensaje de *smishing*. Dicho mensaje estaría contenido en capturas de

pantalla recibidas por los CERT (Centro de Respuesta a Ciberincidentes) a través de ciudadanos y empresas.

Una vez la URL ha sido extraída de manera automática, podría ser analizada para determinar si pertenece a un sitio web de *phishing* o legítimo. Para lograr esto, es necesario trabajar con técnicas de extracción de texto en imágenes (del inglés, *Text Spotting*).

A la hora de realizar el reconocimiento automático de texto se suele dividir el problema en dos partes, por un lado, la detección de texto, que se encarga de encontrar regiones de una imagen que podrían contener caracteres, y por otro el reconocimiento de texto, que se encarga de intentar identificar qué caracteres hay en una determinada imagen o región de esta.

Dentro de la detección de texto existen dos métodos principales: uno basado en la regresión y otro en la segmentación:

El basado en la regresión consiste en algoritmos de aprendizaje automático que predicen las coordenadas de un cuadro que delimita una región que contiene texto en una imagen. En este enfoque, el algoritmo se entrena en un conjunto de datos de imágenes etiquetadas en el que cada una de ellas tiene un cuadro que delimita la ubicación del texto. Con esto el algoritmo aprende a predecir las cuatro coordenadas del cuadro delimitador basándose en las características de la imagen.

Por otro lado, basados en la segmentación consisten en dividir una imagen en diferentes regiones en función de sus características visuales, tales como color o textura presentes. Luego, se analiza cada píxel y se identifican regiones que contienen texto.

Finalmente, es necesario hablar sobre el reconocimiento de texto, también conocido como OCR (*Optical Character Recognition* por sus siglas en inglés). En esta etapa se usa como punto de entrada las regiones en las que se ha detectado que hay caracteres, para reconocer y transcribir el texto contenido en cada una.

El reconocimiento de texto ha suscitado un gran interés en los últimos años debido a su utilidad en muchos campos diferentes: coches autónomos, reconocimiento de matrículas [14], digitalización de textos históricos [15], traducción automática de imágenes [16], digitalización de documentos en papel [17], extracción de texto de imágenes [18], etc.

Sin embargo, aún queda mucho por hacer en este campo debido a las dificultades que todavía plantea el trabajar con imágenes capturadas en diferentes situaciones [19], ya que el texto de las escenas naturales suele estar deformado o curvado debido a la distorsión de la perspectiva, la iluminación irregular y otros factores, lo que hace dificulta mucha la tarea de realizar una detección precisa sin preparar un sistema para enfrentarse a un contexto específico.

Adicionalmente, la detección de texto en escenas complejas que contienen varios idiomas, estilos de fuente y colores también resulta difícil. Por ejemplo, los algoritmos de detección de texto pueden fallar en la detección de texto en imágenes con fondos complejos, en imágenes con múltiples objetos y texturas, o en aquellas en las que el color del texto tiene muy poco contraste con respecto al color del fondo [20].

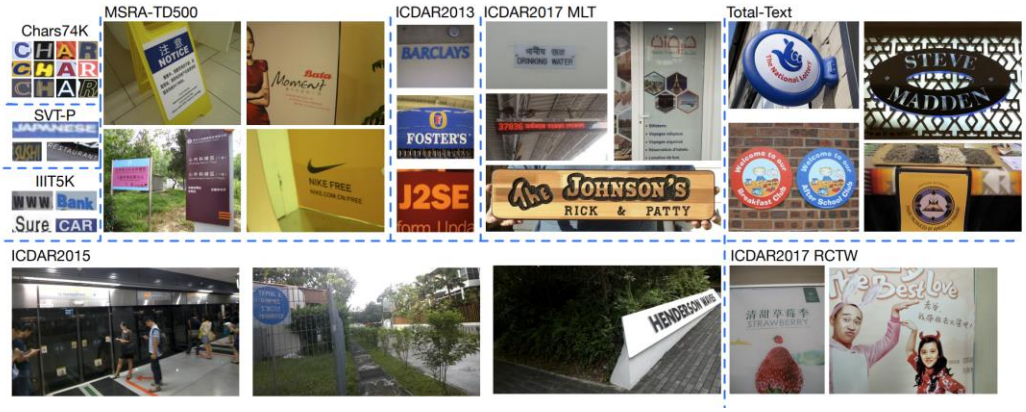

Figura 1.1 Ejemplos de texto en diferentes situaciones (Fuente: Scene Text Detection and Recognition: The Deep Learning Era [20])

Por otro lado, aunque hay muchos conjuntos de datos públicos disponibles relacionados con la detección de texto, a menudo tienen una cantidad limitada de datos que además no son representativos para muchos contextos diferentes, esto es porque se centran en proveer datos con características concretas relacionadas con un problema, como imágenes con texto curvado o con texto *in-the-wild* (en escenas del mundo real) [21].

Utilizar uno de estos conjuntos de datos para entrenar un sistema puede afectar a la capacidad de generalización de los algoritmos de detección de texto, ya que lo especializarían en la resolución de esos problemas concretos y no serían capaces de obtener buenos resultados al enfrentarse a otros diferentes [22] [23]. Volviendo al ejemplo

anterior, un modelo entrenado con imágenes con texto curvado y con texto en entornos reales tendría bastantes problemas para poder detectar texto manuscrito.

Otro problema de los conjuntos de datos es que requieren una anotación y un etiquetado exhaustivos, lo que conlleva mucho tiempo y hace que esta tarea sea difícilmente abordable por un equipo pequeño [24]. Curiosamente, una solución para este problema pasa por el mismo tema; el reconocimiento de texto, ya que un sistema fiable facilitaría bastante la tarea de etiquetar imágenes en función de, por ejemplo, si contienen enlaces o no, si contienen enlaces partidos en múltiples líneas, etc.

1.1.1 MOTIVACIÓN

Como se ha descrito anteriormente, hay varios motivos por los que este tema resulta de gran interés. Por un lado, al tratar sobre una cuestión de gran importancia para la ciberseguridad que tiene gran relevancia en la actualidad, presenta un gran atractivo, ya que es un campo que todavía no se ha explorado en su totalidad.

Como se expone posteriormente, aunque sí hay muchas investigaciones modernas relacionadas con el tema; detección de texto, detección de *phishing*, etc., no hay demasiadas que traten directamente sobre *smishing*, y aún menos que aborden, en concreto, la detección de enlaces.

Esto da pie a que se puedan proponer muchas ideas ya aplicadas a otros problemas para mejorar un sistema como este, por lo que se pueden encontrar rápidamente varios caminos que recorrer, investigar y probar, lo que da más dinamismo al trabajo.

Por otro lado, el hecho de que el proyecto recibido se desarrollara basándose en una necesidad real de INCIBE [1] (Instituto Nacional de Ciberseguridad) y su CERT es un indicativo de que puede ayudar a la lucha contra el cibercrimen en el mundo real.

Finalmente, otra motivación para hacer este trabajo es el hecho de que trate un tema que está a la orden del día: la inteligencia y la visión artificial están tomando mucha relevancia en el contexto actual, como se puede ver con el surgimiento de *ChatGPT* [25] y otros sistemas similares, por lo que trabajar con el problema en cuestión ofrece una oportunidad excelente para formarse y aprender a manejarse en este contexto.

1.1.2 OBJETIVOS

El objetivo principal del trabajo planteado inicialmente es mejorar la precisión (del inglés *accuracy)* de los problemas principales a analizar, esto es, los cinco problemas con más enlaces.

En base a eso, y a los requisitos establecidos por la normativa de TFM de la Escuela de Ingenierías [26], se plantearon varios objetivos para el desarrollo de este Trabajo de Fin de Máster:

1. Llevar a cabo un estudio del estado del arte de la literatura asociada a:
    a. Detección de Smishing.
    b. OCR - Reconocimiento de Texto.
        i. Postprocesado de texto.
    c. Detección de texto.
        i. Preprocesado de imágenes.
    d. Detección de URL.
2. Mejorar la precisión para los problemas principales de un sistema ya implementado:
    a. Estudiar el sistema previamente implementado.
    b. Análisis del rendimiento base.
    c. Análisis y solución de problemas.
    d. Introducción de nuevas técnicas y posibles mejoras.
        i. Planteamiento e implementación de posibles mejoras.
        ii. Análisis de rendimiento de los cambios propuestos.
3. Mejorar el tiempo de ejecución del sistema.

1.1.3 ESTRUCTURA DEL DOCUMENTO

El documento está dividido en 5 capítulos, siendo este donde se han tratado la introducción, motivación y objetivos el primero. El siguiente presenta el estudio del estado del arte, donde se analizan diferentes artículos relacionados con la temática y se expone la normativa aplicable a este trabajo.

El capítulo 3 aborda la metodología, herramientas y datos utilizados para la realización del trabajo. Tras esto, el 4 expone un análisis del estado del proyecto, los cambios introducidos y los resultados obtenidos, y por último el 5 describe el estado final del sistema y las conclusiones.

Adicionalmente, el anexo A contiene un manual de usuario que detalla cómo ejecutar el código, instrucciones adicionales para facilitar el despliegue y uso de este en un entorno de desarrollo.

Capítulo 2

2.1 ESTADO DEL ARTE

El *smishing* y la detección de URL son dos temas sobre los que no hay demasiados trabajos de investigación recientes, ya sea porque son subproblemas que no se suelen abordar de manera individual o porque han empezado a tomar mayor relevancia en el momento actual.

Por este motivo durante el desarrollo de este documento la mayoría de los artículos seleccionados para su análisis son de temáticas similares que podrían ser aplicables al caso, pero no tratan directamente del tema en cuestión, como pude ser la detección y reconocimiento de texto, los métodos de pre y postprocesado de imágenes y texto, etc.

A continuación se expone un análisis resumido sobre el contenido de cada uno de los artículos revisados.

2.1.1 DETECCIÓN DE SMISHING Y EXTRACCIÓN DE URL

Mishra & Soni [27] proponen un detector de *smishing* basado en el análisis del contenido de los SMS y del comportamiento de las URL, siendo esta última parte uno de los principales puntos diferenciadores respecto a otros sistemas similares.

La primera parte del sistema realiza un preprocesado del texto y luego lo pasa a un analizador de contenido de SMS, que extrae URL, números de teléfono y otros contenidos relevantes.

Todas las URL se pasan entonces por un filtro de lista negra para luego procesar los enlaces que no han sido eliminados analizando datos como la edad del dominio, o la longitud de la URL y basándose en eso determinar la probabilidad de que esté relacionado con el *smishing*.

Posteriormente los enlaces restantes y las otras características extraídas se analizan con un modelo de aprendizaje automático (del inglés *Machine Learning*, ML), para finalmente, pasar el mensaje por un sistema capaz de detectar descargas que comprueba

si se descarga un APK (fichero de instalación de una aplicación Android [5]) sin el consentimiento del usuario.

Para el entrenamiento y las pruebas se utiliza un conjunto de datos del trabajo de investigación de Almeida et al. [29], que se complementa con múltiples ejemplos más de textos de *smishing*. Los resultados muestran una precisión final del 96.29 % después de que el mensaje pase por todos los módulos.

Jain & Gupta [30] estudian qué aspectos de los mensajes SMS los hacen más propensos a ser maliciosos para crear un método de clasificación basado en reglas que permita predecir si hay *smishing* en un SMS.

Los primeros pasos del trabajo de los autores consistieron en recopilar SMS de The SMS Spam Dataset v.1 del trabajo de Almeida et al., conservando solo 5169 mensajes, de los cuales 362 eran casos reales de *smishing*. A continuación, todos estos datos se introducen en un módulo de preprocesamiento para eliminar la redundancia y facilitar su tratamiento en las siguientes partes del modelo.

La última parte del sistema busca coincidencias con las reglas establecidas previamente que identifican un mensaje como probablemente malicioso y lo clasifica en consecuencia. Cada regla fue probada individualmente, obteniendo que la que proporcionaba mejores resultados era la búsqueda de palabras usadas habitualmente en estafas, como *lotería*, *ganar*, *premio*…, que consiguió un 87.85 % de precisión, seguida de un 59 % y 53 % con las reglas que buscan caracteres especiales y el carácter dólar y libra. El modelo final obtuvo una precisión del 95.92 % con el algoritmo PRIMS [31] para la parte de clasificación.

Siddharth et al. [32] tratan de desarrollar un modelo especializado y eficiente que pueda ser utilizado para extraer URL de imágenes de PDF empleando un OCR. El sistema pasa imágenes preprocesadas a un módulo de segmentación que detecta y corrige imágenes inclinadas, separa el texto en líneas, luego en palabras y después en caracteres.

Por último, se pasan los datos a una red neuronal ConvNet y por una compleja expresión regular (*regex* [33]) que recorre todo el texto para extraer patrones que puedan ser URL. Se hizo uso de los sets de datos (*datasets* en inglés) DDI-100 [34] e Infty [35] para el entrenamiento y las pruebas, obteniendo una precisión final del 98.59 %.

C. Wang & Chen [36] plantean un modelo híbrido que combina autoatención y operaciones convolucionales para capturar las relaciones de dependencia internas de la estructura de frases y las correlaciones locales.

Con ello, pretenden solucionar problemas comunes encontrados en CNN (Redes Neuronales Convolucionales, *Convolutional Neural Networks*) y Transformadores (del inglés *Transformers*, otro modelo de aprendizaje profundo*)* combinando ambos con el fin de crear un modelo, denominado *TCURL,* que pueda detectar con certeza URL de phishing.

El diseño se compone de dos ramas, una de convolución y otra de transformación, que se alimentan de los mismos datos y producen sus propias salidas, que luego se combinan con una capa decodificadora.

Los conjuntos de datos empleados son Phishing Site URL [37], ISCX-URL2016 [38] y Phishing and Benign Websites Dataset [39], extrayendo de estas URL con una longitud máxima de 200 caracteres. Los resultados logrados con una puntuación F1 (del inglés, *F1 score)* del 96.91 %, 99.77 % y 89.73 % para cada set de datos respectivamente demostraron una mejora respecto a varios modelos base probados como BiLSTM [40].

2.1.2 PROPOSICIÓN DE NUEVOS MODELOS

Irimia et al. [18] presentan una solución extensible y de código abierto para el reconocimiento y la extracción precisa y rápida de texto de documentos de identidad. Los autores investigan el impacto de las transformaciones de imagen en los tiempos de procesamiento y exploran cuál puede ser el mejor enfoque para mejorar las imágenes para su uso con un OCR.

Aplican el método RASE (Error Espectral Promedio Relativo, del inglés *Relative Average Spectral Error*) [41] durante la fase de procesamiento con el que seleccionan un área que puede coincidir con una plantilla de un documento, para luego comparar con SSIM (Índice de similitud estructural, del inglés *Structural Similarity Index Measure*) [42] el área original con varias posibles redimensiones de esta para determinar si coinciden.

En el modelo final, las imágenes se redimensionan a 200x100 y se giran hasta que se ajustan aproximadamente con alguna de las plantillas de documentos de identidad

guardadas. El OCR aplica entonces esa plantilla para determinar las posiciones predeterminadas que deben escanearse en busca de texto.

La precisión alcanzada fue del 100 % con un conjunto de datos bastante reducido, proporcionando una mejora con respecto a otros modelos similares como el empleado por la empresa Sinosecu [43] y el Huawei Cloud Passport OCR [44], ya que es capaz de trabajar con imágenes de menor calidad y con imágenes rotadas desde hasta 359°.

Yousef et al. [45] exponen un nuevo modelo de reconocimiento de texto: una red totalmente convolucional que solo usa conexiones retroalimentación proyectada hacia el futuro (*feed forward en inglés)*, capaz de alcanzar un rendimiento a la altura de los mejores métodos en diferentes tareas (OCR, reconocimiento de escritura, reconocimiento de texto de escena...).

El sistema aplica normalización por capas, por lotes, y renormalización por lotes (de los términos en inglés *Batch Normalization*, *Layer Normalization*, y *Batch Renormalization*) para aumentar la velocidad de convergencia, y varias técnicas comunes para la ampliación de datos (en inglés *data augmentation*), pero que solo realizan modificaciones en un eje de la imagen (*x* o *y*) cada vez, para ayudar en la convergencia del modelo.

La precisión del modelo se evaluó en varios conjuntos de datos con la Tasa de Error de Caracteres (CER, *Character Error Rate*), entre todos los test cabe mencionar que con el conjunto de datos de la competición ICFHR2018 [46] este método obtuvo los mejores resultados de todas las propuestas presentadas, con una CER de 13.01, 0.4 puntos mejor que el segundo.

2.1.3 MEJORA DE LA DETECCIÓN DE TEXTO

H. Wang & Feng [47] estudian cómo mejorar el algoritmo DB (*Differentiable Binarization*, en español Binarización Diferenciable) de detección de texto por segmentación, abordando algunos de sus principales problemas, como la detección parcial de regiones de texto o la detección incorrecta de texto por problemas como las variaciones en las formas del texto y los fondos complejos.

Para esto, los investigadores proponen la introducción un módulo *FACR* (Feature Augmentation Constant Ratio, o módulo de mejora de Características de Ratio Constante)

al nivel de las capas superiores, que ayuda a propagar mejor la información semántica de las imágenes hacia las capas inferiores, reduciendo así el error causado por los píxeles que no son realmente texto y por la omisión de los textos más pequeños.

El modelo entrenado para esto emplea como base (*backbone* en inglés) el modelo ResNet-50 [48], que fue optimizado y entrenado posteriormente con el conjunto de datos ICPR Tianchi [49], con más de 20000 imágenes. El resultado final fue un modelo con una puntuación F1 de un 85.6 %, representando esto una mejora del 2.4 puntos porcentuales respecto al rendimiento del algoritmo DB estándar, y sobre el que cabe destacar que sigue teniendo problemas para lidiar con texto que sea difícilmente diferenciable del fondo.

Chen et al. [50] analizan la evolución de los métodos de detección de texto, partiendo de los métodos más antiguos basados en características diseñadas por el ser humano y llegando a los métodos basados en aprendizaje profundo que emplean enfoques basados en regresión y segmentación. En particular se analiza el modelo propuesto por Wang et al., [51] una Red de Agregación de Píxeles (del inglés *Pixel Aggregation Network* o PAN), y se introducen diferentes mejoras: un módulo para extraer las características del texto con mayor eficacia, añadiendo conexiones de salto (*skip connections* en inglés), cambiando la reducción de la dimensión de 1x1 a 3x3 para reducir la pérdida de información, y reemplazando la operación *suma* del modelo base por la operación *concatenación* para asegurar que las características e información extraídas se usan en mayor medida.

El modelo final se entrenó con los sets de datos ICDAR 2015 [52], MSRA-TD500 [53] y CTW1500 [54] y con cada una de las mejoras recomendadas por separado, obteniendo una puntuación F1 mínima de 80.72 %, y siendo en todos los casos superior a la del modelo base, siendo la mejor mejora la obtenida sobre el set de datos MSRA-TD500 con una puntuación F1 de 83.94 % frente al 78.6 % del modelo base.

Wödlinger & Sablatnig [55] presentan un nuevo método para detectar automáticamente resultados base o de partida (del inglés *baselines*) de texto manuscrito (líneas de referencia de dónde hay texto) a partir de imágenes de documentos sirviéndose de una CNN recurrente sin necesidad de reglas elaboradas a mano para cada caso. La entrada de ese modelo CNN es el punto de inicio y los ángulos de las líneas que se obtienen con otro modelo CNN basado en U-NET [56]. Para la segmentación de los resultados base

hay dos CNN recurrentes que predicen, por un lado, las coordenadas de inicio y el ángulo de la siguiente ventana o región de texto, y por otro las coordenadas de fin de la ventana actual.

El modelo se entrena sobre el conjunto de datos ICDAR 2019 (cBAD) [57], con las imágenes recortadas a 1024x1024, y rotadas. Las imágenes también se etiquetan dibujando las líneas, regiones de texto y puntos inicial y final de las palabras. Los resultados muestran que la primera configuración obtiene una puntuación F1 con un valor del 96.3 %, mejor que la ganadora de cBAD 2019.

2.1.4 POSTPROCESADO Y CORRECCIÓN DE ERRORES

R. Wang et al. [58] exponen un modelo de postprocesado para el reconocimiento de párrafos para OCR con redes convolucionales de grafos (GCN [59]), y un proceso de 2 pasos que primero divide las líneas de texto obtenidas en líneas más cortas con un clasificador de nodos GCN, y luego aplica otro clasificador GCN para agrupar esas nuevas líneas en párrafos. Con esto se obtiene un modelo más pequeño y rápido, que trabaja directamente con la salida del sistema OCR, en vez de con las imágenes originales como hacen otros métodos.

El modelo se probó en tres conjuntos de datos diferentes: PubLayNet [60], un conjunto de datos extraído de diferentes webs, y un conjunto de datos de párrafos anotados por humanos, y luego se comparó con otros modelos donde demuestra una ligera ventaja, terminando siempre con una puntuación F1 media superior a la de todos los demás ejemplos probados.

En los conjuntos de datos PubLayNet y el extraído de diferentes webs la puntuación F1 fue de 95.9 % y 83 % respectivamente, y en el conjunto de datos anotados por humanos la puntuación fue mucho menor, alcanzando un 67.1 % en el mejor de los casos.

Imam et al. [61] presentan un algoritmo para detectar texto malicioso incrustado en imágenes, tratando de detectar, corregir y clasificar los errores detectados en el texto (por ejemplo, caracteres repetidos, caracteres intercambiados, caracteres sustituidos, palabras fuera del vocabulario...).

Este proceso se realiza en tres pasos: primero se comparan las palabras con un diccionario; después, si no se encuentra la palabra, el algoritmo comprueba si hay caracteres o símbolos alterados y los corrige. Por último, si la palabra no puede corregirse se clasificará como fuera de vocabulario (OOV, *Out Of Vocabulary*) y se guarda y utiliza posteriormente para mejorar la detección de este tipo de errores.

El modelo final hace uso de un OCR para la detección de texto, TPS-S para el reconocimiento de texto combinado con el algoritmo propuesto y un sistema de clasificación de texto múltiple (MCS, *Multiple Classifier System*). La salida final tendrá en cuenta el voto de la mayoría de estos dos últimos módulos, es decir, la clase que se asigna al texto es la más común entre las que asignan esos módulos.

El algoritmo propuesto se probó en dos escenarios diferentes: primero comparándolo con otros correctores ortográficos, mostrando resultados similares, en los que los ataques de borrado causaron grandes problemas, siendo el algoritmo propuesto el segundo con mejor rendimiento con un 50 % de precisión. Y a continuación emparejando varios OCR con el algoritmo de corrección propuesto, logrando una mejora de 10 puntos porcentuales en uno de los problemas más complejos (errores de detección de 2 caracteres).

Mei et al. [62] desarrollan un modelo de postprocesado OCR que emplea un esquema de *tokenización* (del inglés *tokenization*: proceso de dividir un texto en unidades más pequeñas) en dos etapas y el uso de características lingüísticas de léxico, semántica y contexto para detectar si la palabra está en el vocabulario o si tiene coherencia, y corregir errores en textos OCR con mucho ruido, empleando el algoritmo de distancia Levenshtein Inverso (*Reverse-Levenshtein)* para determinar el candidato más apropiado según el contexto de la palabra.

El conjunto de datos usado se generó a partir del contenido del libro titulado *Birds of Great Britain and Ireland* aprovechando Tesseract 3.0.2 [63] para realizar una extracción preliminar que se corrigió manualmente, y se etiquetó para identificar todos los errores y sus correcciones.

La evaluación final del modelo se realiza comparándolo con Aspell [64], Noisy channel [65] y Kissos y Dershowitz [66] mostrando que supera a todos ellos alcanzando una puntuación F1 del 80.17 % en detección de errores y del 71.95 % en corrección de errores.

2.1.5 MODELOS SEMISUPERVISADOS

Zhou et al. [24] investigan cómo reducir la dependencia de los datos ya etiquetados, ya que es una tarea compleja y que requiere mucho tiempo y atención para su correcta realización. Para esto se propone un algoritmo semi-supervisado para la detección de texto en entornos naturales (SSTD, *Semi-supervised Scene Text Detection*), que emplea una pequeña cantidad de datos etiquetados y un gran conjunto de datos sin etiquetar.

Para predecir las etiquetas que deben llevar los datos el programa hace uso de un algoritmo de generación de pseudo-etiquetas de doble umbral (DPG, *dual-threshold pseudo-label generation algortihm*), y aplica diferentes técnicas de aumentación de datos, adquiriendo unos datos que pueden ser usados para entrenar un modelo de manera semi-supervisada. Con esto, los investigadores abordan las limitaciones de los métodos existentes que solo aplican un umbral, lo que da lugar a una clasificación errónea de textos o fondos complejos.

Los resultados finales se obtuvieron con un modelo de ResNet con una FPN (Red de Pirámide de Características, o *Feature Pyramid Network* en inglés) como columna vertebral o base (en inglés *backbone)*, entrenado con el set de datos CTW1500 como base, al que luego se le agregó el algoritmo anteriormente mencionado junto a otras mejoras.

Comparando el sistema con otros del estado del arte, se demuestra que un 50 % de datos anotados son suficientes para obtener resultados similares a los modelos que tienen el 100 %, logrando una puntuación F1 del 83.5 %, que es superada por 4 de los otros 12 modelos analizados.

Xue et al. [67] proponen un método de preentrenamiento poco supervisado de Visión-Lenguaje (*weakly supervised Vision-Language pre-training method*), denominado *oCLIP*, que facilita las tareas de detección y reconocimiento de texto y reduce la dependencia de conjuntos de datos completamente anotados. Para ello emplearon:

- Un codificador de texto, para capturar información semántica y sintáctica de las descripciones en lenguaje natural, ya que obtiene mejores representaciones textuales de las imágenes.

- Un decodificador visión-texto, que, con anotaciones parciales es capaz de aprender acerca de la relación entre la imagen de entrada y cada texto, con lo que ayuda a aprender en mayor medida de las representaciones visuales.

Los experimentos se llevaron a cabo en varios conjuntos de datos, como SynthText [68] y Total-Text [69], empleando varios sistemas de detección de texto combinados con el modelo oCLIP propuesto y comparándolos con varios modelos de referencia.

Los resultados del conjunto de datos mostraron mejoras en todos los casos, con el set de datos ICDAR 2015 el modelo DB-RN50 [70] mejoró en 4.5 puntos porcentuales, alcanzando una puntuación F1 del 85.4 %.

2.1.6 DETECCIÓN Y RECONOCIMIENTO DE TEXTO

Chiatti et al. [71] proponen un marco para la extracción y recuperación de texto a partir de capturas de pantalla de teléfonos móviles que aprovecha tanto la información textual como la visual de las capturas de pantalla con el fin de permitir diversas aplicaciones, como la creación de un sistema de búsqueda basado en los datos de la imagen.

El conjunto de datos de este artículo se creó recopilando 13172 capturas de pantalla de teléfonos móviles de múltiples usuarios en su día a día, de las que se extrajo texto con un OCR y luego fueron anotadas por un grupo de voluntarios humanos. Para permitir la búsqueda por información de la imagen se vinculó toda la información obtenida a cada imagen y un esquema de ponderación de términos Okapi BM25 [72].

El rendimiento del modelo, en particular hasta la parte de reconocimiento óptico de caracteres, se midió en varios pasos, primero sin aplicar ningún preprocesamiento de imágenes, y luego aplicando algunos, lo que aumentó la precisión global tanto a nivel de palabras como de caracteres, y finalmente aplicando correcciones a los guiones anotados por humanos, ya que estos contenían errores.

Además, se probaron las versiones 3 y 4 de Tesseract, y se observó un aumento inmediato del rendimiento con la última versión, con una precisión del 74.81 % a nivel de caracteres y del 71.24 % en cuanto a palabras.

Busa et al. [73] plantean como objetivo del artículo mejorar la precisión del reconocimiento de los textos con un tamaño muy pequeño, aplicando para ello varios

métodos, como técnicas de mejora de imagen (como super-resolución o *super-resolution*) como paso previo al procesamiento para el reconocimiento de texto, la segmentación de caracteres como método para mejorar la precisión y un modelo CRNN (*Convolutional Recurrent Neural Network*, en español Red Neuronal Convolucional Recurrente) con pérdida CTC (Clasificación Temporal Connectionista, en inglés *Connectionist Temporal Classification* [74]) para el reconocimiento de texto.

Se usaron los sets de datos ICDAR 2003 [75] y MJSynth [76] [77] para el entrenamiento y las pruebas. Los resultados muestran que el sistema final con todos los cambios consigue una mejora de casi 19 puntos porcentuales en la precisión en comparación con el modelo base de CRNN con pérdida CTC, pero, aun así, solo alcanza una del 57.2 %.

Harsha et al. [78] desarrollan un modelo de aprendizaje profundo que pueda reconocer el texto en una imagen e identificar la mejor técnica de extracción de características entre HOG (Histograma de Gradientes Orientados, *Histogram of Oriented Gradient* [79]) y LPB (Patrón Binario Local, *Local Binary Pattern* [80]).

El conjunto de datos usado estaba formado por imágenes aleatorias con texto extraído del set de datos ICDAR 2003 que fueron redimensionadas a un tamaño un poco menor que la imagen más grande obtenida (mejorando con esto la velocidad de rendimiento del algoritmo). El modelo base usado es una CNN, donde también se aplican técnicas de preprocesado para mejorar la calidad de cada imagen, con el que se usan por separado HOG y LBP para analizar sus resultados, logrando una puntuación F1 del 73 % y 77.85 % respectivamente, demostrando que LPB es mejor algoritmo para la extracción de características en el reconocimiento de texto.

Huang et al. [81] proponen un nuevo marco de trabajo (o *framework* en inglés) para el reconocimiento óptico de caracteres multilingüe de extremo a extremo que puede identificar automáticamente el idioma de las palabras detectadas y es capaz de elegir el mejor método base de reconocimiento para cada uno.

El proceso seguido por este modelo se divide en tres partes que se realizan de extremo a extremo: detección de texto, reconocimiento de texto e identificación de escritura.

Para las dos primeras tareas utiliza los mismos módulos de detección y segmentación que Mask TextSpotter V3 [82], y continuación, una Red de Predicción Lingüística (*Language*

Prediction Network, LPN) para seleccionar automáticamente el módulo de reconocimiento más apropiado para el texto dado. El modelo se entrena con más de 8 conjuntos de datos diferentes con un amplio conjunto de imágenes y texto en distintos idiomas y en tres fases. Los resultados finales muestran que, en comparación otros modelos con los sets de datos MLT17 [83] y ICDAR 2019 MLT19, el método propuesto da mejores resultados con una ventaja de aproximadamente un punto porcentual sobre el segundo mejor modelo (puntuación F1 de 69.41 % y 69.51 %).

Olejniczak & Šulc [84] analizan los principales sistemas de detección de texto existentes para poder medir si también consiguen resultados competitivos en la detección de texto de documentos, incluso si se comparan con otros modelos habituales de OCR para documentos.

Para ello, se probaron una amplia gama de modelos de detección de texto que adoptan diferentes enfoques (PAN, TextBPN++ [85], DCLNet [86], DBNet, DBNet++ [87] y CRAFT [88]), que obtienen los cuadros que delimitan el texto y los alimentan a un modelo de reconocimiento de texto en entornos reales, en escenas del mundo real, comparando los resultados obtenidos con tres OCR de documentos: Tesseract, Google Document AI y AWS Textract.

Los modelos fueron evaluados con tres conjuntos de datos diferentes (FUNSD [89], CORD [90] y XFUND [91]). En la parte de detección de texto los resultados obtenidos fueron al menos tan buenos como los obtenidos con los OCR de documentos, llegando a obtener las puntuaciones con el rendimiento más alto con los tres conjuntos de datos, siendo TextBPN++ con el set de datos CORD uno de los mejores sistemas, al lograr una puntuación F1 del 99.7 %.

En cuanto al reconocimiento de texto, entre los modelos con imágenes en condiciones reales, los que peores resultados consiguieron fueron los que empleaban PAN para la detección, quedando solo por delante de Tesseract. CRNN con DBNet++ obtienen los segundos mejores resultados en general, quedando por detrás de AWS Tesseract solo en el set de datos FUNSD con una puntuación F1 del 92.23 %.

Sofwan et al. [17] pretenden probar e identificar qué método de umbralización (del inglés, *thresholding*) y qué método de escaneo de documentos funcionan mejor en

documentos con el fin de determinar cuál debería aplicarse junto a un OCR para extraer con una alta precisión el texto de imágenes de documentos.

Para ello probaron siete métodos de umbralización diferentes (*binario*, *tozero*, *Otsu*, *binario+Otsu*...), combinados con un OCR y evaluando su precisión sobre diferentes sets de datos de imágenes propios con el algoritmo de la distancia de Levenshtein, que cuenta el número de caracteres que deben ser cambiados para que dos cadenas de texto sean iguales.

Los resultados muestran que los métodos *Adaptive Gaussian* y *Adaptive Mean* dan resultados casi inservibles, y el resto de métodos proporcionan resultados precisos, manteniéndose por encima de un 92 % para todos los casos, excepto con las imágenes tomadas directamente con la aplicación de cámara de un *smartphone*, siendo *Trunc* el mejor método en general.

Ravindra et al. [92] proponen un sistema de aprendizaje automático centrado en el aspecto del aprendizaje supervisado que hace uso de un conjunto de 4000 URL, donde la mitad de ellas son maliciosas.

El módulo principal de la arquitectura es un extractor de características que ayuda a analizar 9 características principales de las URL: longitud, número de subdominios y puntos, uso del protocolo http etc., también emplea el algoritmo de Bosque Aleatorio (del inglés *Random Forest* [93]*)* en parte del proceso debido a su alta precisión y alto rendimiento.

La implementación del sistema alcanzó una precisión del 86 %, que, aunque no es la mejor entre otras opciones estudiadas anteriormente, proporciona un sistema más simple y eficiente en términos de recursos que también alcanza buenos resultados.

Kumar et al. [94] describen una arquitectura completa para etiquetar imágenes con fines de búsqueda mediante métodos y herramientas de código abierto que puede ser ejecutado en un dispositivo del usuario final que tenga poca capacidad de procesamiento.

La parte de detección de texto tiene un modelo que soporta la detección de texto multilingüe, CTPN [95], pero con solo una red convolucional de cuatro capas en la extracción de características, con lo que logran una precisión del 91 %.

Por otro lado, el texto se clasifica según el lenguaje con una CNN de cuatro capas con una precisión del 90 %, pasando el resultado a la siguiente etapa para que seleccione el

mejor OCR dependiendo del idioma (MLKit [96] para los alfabetos latinos, y Tesseract 4 para el resto).

La última etapa del modelo extrae palabras clave y las clasifica con diferentes etiquetas que son posteriormente expandidas en base a lo propuesto por Feipeng Zhao et al. [97], siendo el sistema final capaz de superar 63 % de las pruebas de manera parcial y 12 % de manera total.

2.2 NORMATIVA APLICABLE

El análisis de datos y su tratamiento implican una serie de normativas y leyes que deben de tenerse en consideración cuando se trabaja con contenido cedido por usuarios, como las capturas de pantalla que se analizan en este TFM. Por este motivo cabe mencionar las normativas aplicables y su efecto sobre el desarrollo del mismo.

2.2.1 LEY ORGÁNICA DE PROTECCIÓN DE DATOS PERSONALES Y GARANTÍA DE LOS DERECHOS DIGITALES (LOPDGDD) y REGLAMENTO GENERAL DE PROTECCIÓN DE DATOS (RGPD)

Estas leyes [98] [99] regula la protección de datos personales y establece los principios y requisitos que deben seguirse al procesar datos personales.

Se consideran datos personales cualquier información relativa a una persona física, o aquella que, recopilada y contrastada pueda llevar a la identificación de una determinada persona. [100] Entre este tipo de información se incluyen, precisamente, las capturas de pantalla de SMS, ya que son susceptibles de contener información como nombres, apellidos, números de teléfono.

La colección de imágenes empleada en el proyecto ha sido proporcionada por el Grupo de Visión y Sistemas Inteligentes (GVIS [101]) de la Universidad de León [102], por lo que ya ha sido revisada anteriormente y requerirá un análisis menos exhaustivo para confirmar que cumple con esta ley.

2.2.4 LEY DE PROPIEDAD INTELECTUAL (LPI)

La LPI [103] es una normativa que protege los derechos de autor y otros derechos relacionados con la propiedad intelectual. En el contexto de un TFM que analiza capturas de pantalla hay varios puntos que pueden ser relevantes de la LPI.

Por un lado, los derechos de autor: cualquier creación original como textos, imágenes o gráficos está protegida por derechos de autor desde el momento de su creación. Esto significa que si las capturas de pantalla son o contienen contenido protegido por derechos de autor (mensajes de texto o imágenes), es necesario respetar los derechos de los autores y obtener las autorizaciones correspondientes para su uso dependiendo del fin con el que se vayan a utilizar.

Por otro lado, las bases de datos, los escritos científicos y los programas informáticos también están sometidos a derechos de autor, por lo que tanto el set de datos y el programa con el que se trabaja así como los artículos investigados pueden estar amparados bajo esta ley.

Sin embargo, la LPI también contempla excepciones y limitaciones, como el uso legítimo: que permite que las obras con derechos de autor puedan ser usadas en citas, o con fines educativos o de investigación científica, como es el caso de este trabajo.

Capítulo 3

3.1 GESTIÓN DEL PROYECTO

La planificación del trabajo a realizar se planteó en conjunto con los tutores del TFM teniendo en cuenta las circunstancias que podían afectar al desarrollo del mismo, de manera que pudiera compatibilizarse el desarrollo de este con el seguimiento de las asignaturas del Máster, y con el desempeño de mi trabajo.

Además, la división de las tareas se hizo teniendo en cuenta los objetivos propuestos, por lo que se plantearon diferentes etapas:

1. Definición de temática y planteamiento de trabajo inicial.
    a. Planteada como una etapa inicial, este periodo de tiempo se dedicó a la comunicación con los tutores para acordar tanto una temática concreta, como un plan temporal de trabajo que seguir para el desarrollo del TFM.
2. Revisión del estado del arte.
    a. Dividida a su vez en tres subtareas (búsqueda de artículos, análisis de artículos y planteamiento inicial de mejoras) este periodo se empleó para encontrar información actualizada relacionada con el problema, y para plantear algunas posibles mejoras iniciales en base a lo visto.
3. Implementación y optimización del sistema.
    a. Despliegue del entorno y análisis del código y del rendimiento inicial: Divido en dos tareas realizadas en paralelo, esta fase se centró en analizar el código ya existente proporcionado por los tutores, en crear un entorno en el cual ese programa se pudiera ejecutar, y en probarlo con el set de datos proporcionado.
    b. Planteamiento, implementación y análisis de mejoras: durante este periodo el trabajo se centró en plantear ideas para mejorar la precisión del sistema: posibles mejoras de código, correcciones de errores, sustitución de librerías por otras más modernas, nuevos métodos de análisis de imágenes, etc., en implementarlas y en analizarlas,

centrando los esfuerzos en mejorar la precisión alcanzada en los cinco problemas con mayor número de URL.

c. Optimización de tiempos de ejecución: esta tarea se propuso tras ver durante la etapa anterior que la ejecución del sistema con todas las imágenes proporcionadas tardaba varias horas.

Durante esta fase se buscaron alternativas que redujeran el tiempo de ejecución, como la reducción de tareas de preprocesado de imágenes o la paralelización de diferentes actividades, todo esto sin afectar negativamente a la precisión final.

4. Desarrollo de la documentación.
    a. Esta actividad se llevó a cabo durante todo el desarrollo del TFM, estando el grueso del trabajo en los días posteriores a la finalización de la etapa descrita en el punto anterior.

Las ilustraciones 3.1 y 3.2 de la siguiente página detallan en un cronograma las tareas planteadas para trabajar en el TFM y su extensión temporal:

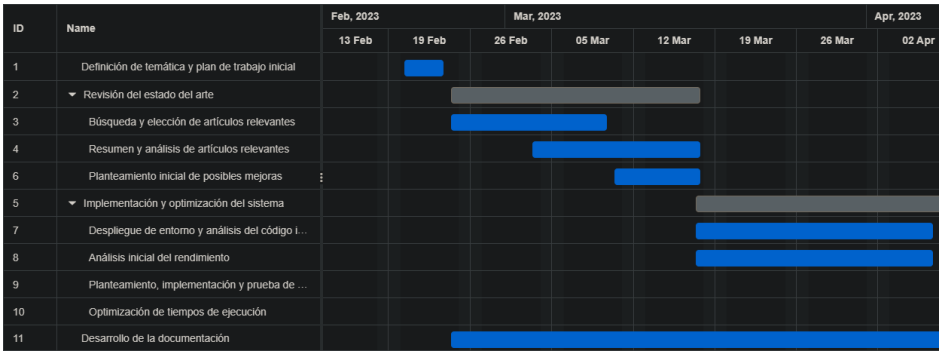

Figura 3.1 Diagrama de Gantt, elaborado con la herramienta https://www.onlinegantt.com/ (primera mitad) (Fuente: elaboración propia)

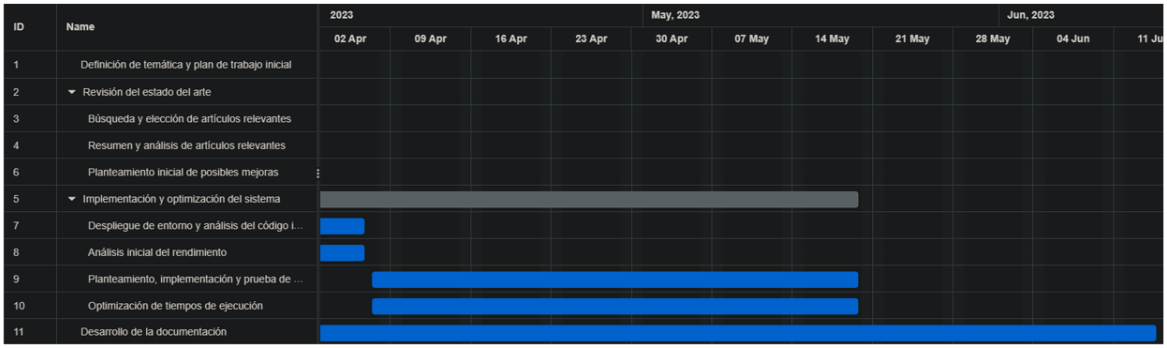

Figura 3.2 Diagrama de Gantt, elaborado con la herramienta https://www.onlinegantt.com/ (segunda mitad) (Fuente: elaboración propia)

3.2 HERRAMIENTAS UTILIZADAS

Durante la realización de este proyecto se han utilizado varias herramientas para la parte de implementación, edición de imágenes, documentación, etc., entre las cuales cabe destacar las siguientes:

- **Visual Studio Code:** editor de código fuente gratuito, con soporte para una amplia cantidad de lenguajes y fácilmente extensible, desarrollado por Microsoft [104]. Esta herramienta ofrece una alta versatilidad y facilidad de uso, lo que propició su elección para trabajar en la parte del código.
- **GIMP:** programa para la edición, retoque y creación de imágenes libre y gratuito desarrollado en el marco del proyecto GNU [105]. Este programa ofrece unas características similares a otros como Adobe Photoshop, pero a diferencia de este es de uso libre, por lo que se ha escogido para su uso en el proyecto.
- **Anaconda:** es una distribución de código abierto de Python y R, centrada principalmente en dar soporte y facilitar el trabajo en tareas como el aprendizaje automático o la ciencia de datos [106]. Ofrece una gran flexibilidad para crear diferentes entornos con distintas versiones tanto de Python como de los paquetes o librerías de este lenguaje.
- **OpenCV:** biblioteca de libre uso que provee funcionalidades para trabajar en el campo de la visión artificial [107].
- **Pan++:** sistema de detección y reconocimiento de texto propuesto en 2021 por Wang et al. [108] para mejorar el rendimiento y la precisión de los sistemas end-to-end.
- **EasyOCR:** popular OCR de código libre desarrollado por Jaided AI para detectar y reconocer texto en múltiples idiomas [109].

- **Tesseract:** popular OCR de código libre, cuyo desarrollo está financiado principalmente por Google [63].
- **Excel:** software de hojas de cálculo propietario de Microsoft que proporciona una manera sencilla de trabajar con grandes cantidades de datos [110].
- **Word:** software orientado a la creación y edición de textos o documentos, propiedad de Microsoft [111].
- **Overleaf:** editor de texto online centrado principalmente en la redacción de documento en línea para facilitar la colaboración, que da soporte para trabajar con LaTeX, un sistema de composición de textos con formato orientado al trabajo científico [112].
- **GitHub:** herramienta gratuita de Microsoft para el almacenamiento y la gestión de proyectos, principalmente código, que destaca por su sistema de gestión de versiones [113].

3.3 DATOS UTILIZADOS

Para el desarrollo de este trabajo se ha hecho uso de un set de datos principal que fue proporcionado por el Grupo de Visión y Sistemas Inteligentes (GVIS [101]) de la Universidad de León junto al código inicial. Este está compuesto por 400 imágenes que contienen un total de 439 enlaces y que pueden ser clasificadas como *smishing.*

La mayoría de estas imágenes son capturas de pantalla de mensajes SMS en *smartphones* sacadas en la aplicación de SMS por defecto del sistema, aunque también hay presente una pequeña cantidad de imágenes en emails y de notificaciones.

Además de esto, las fotografías están divididas según el tipo de problema o escenario que presentan, identificando si tiene uno o varios enlaces, si estos están en algún color, si están divididos en varias líneas…

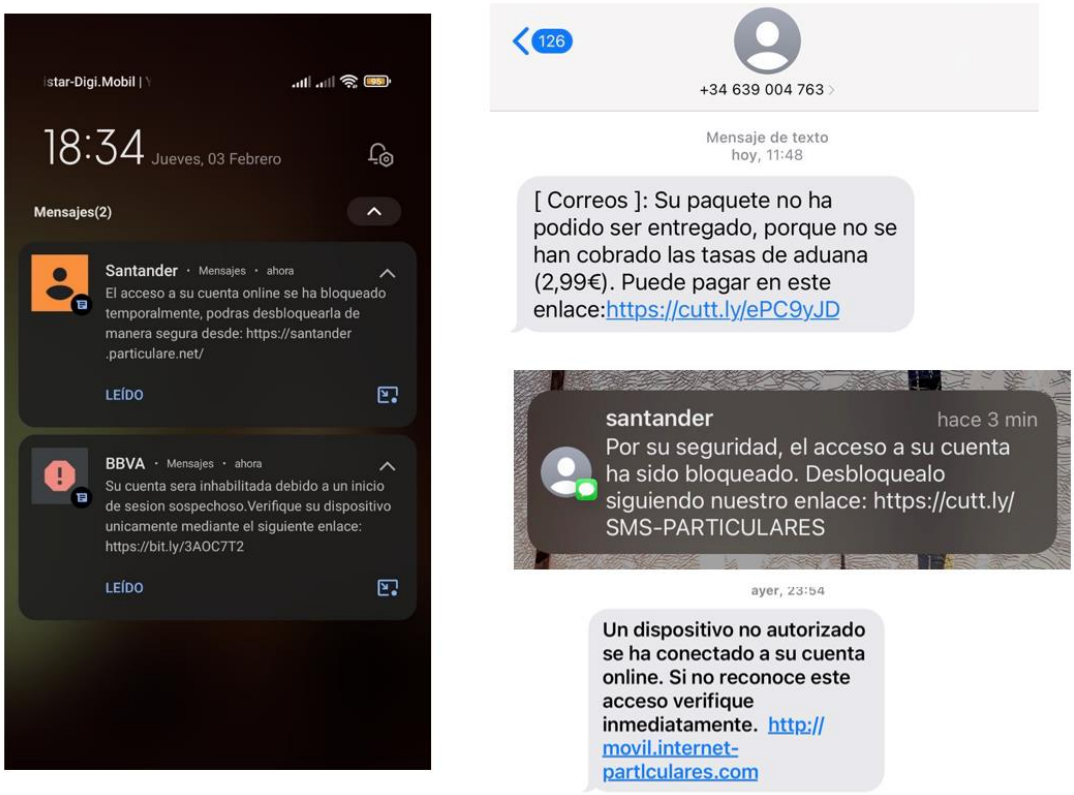

Figura 3.3 Ejemplos de imágenes del set de datos (Fuente: set de datos proporcionado por el grupo GVIS)

En la tabla 1, disponible en esta página, se puede observar más detalladamente como se ha realizado esta división y el número de enlaces e imágenes presentes para cada caso.

Problema	**Imágenes**	**Enlaces**
Enlace en azul en una sola línea	52	52
Enlace en azul en una sola línea con fondo negro	30	30
Enlace sin color en una sola línea	56	56
Enlace sin color en una sola línea con fondo negro	26	26
Enlace en azul sin http	22	22
Enlace sin color y sin http	10	10
Enlace en otro color	2	2
Enlace en azul en dos líneas	62	62
Enlace en azul en dos líneas con fondo negro	38	38
Enlace sin color dos líneas	27	27

Enlace sin color en dos líneas con fondo negro	9	9
Enlace en azul en tres líneas	9	9
Enlace en azul en tres líneas con fondo negro	7	7
Enlace sin color en tres líneas	7	7
Múltiples enlaces en azul y en dos líneas	21	48
Múltiples enlaces sin color y en dos líneas	6	25
Falso negativo	14	8
Difícil	1	1

Tabla 3.1 – División de problemas del set de datos (Fuente: elaboración propia)

De manera orientativa, las siguientes ilustraciones ofrecen un vistazo a cómo son las imágenes de los problemas con más enlaces que se analizarán en mayor profundidad más adelante:

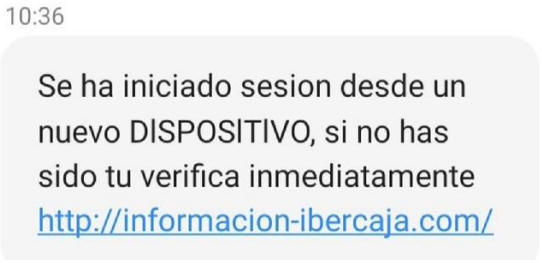

Figura 3.4 Ejemplo de imagen del problema "Enlace en azul en una sola línea" (Fuente: set de datos proporcionado por el grupo GVIS)

Figura 3.5 Ejemplo de imagen del problema "Enlace en azul en dos líneas" (Fuente: set de datos proporcionado por el grupo GVIS)

Figura 3.6 Ejemplo de imagen del problema "Enlace sin color en una sola línea" (Fuente: set de datos proporcionado por el grupo GVIS)

Figura 3.7 Ejemplo de imagen del problema "Múltiples enlaces en azul y en dos líneas" (Fuente: set de datos proporcionado por el grupo GVIS)

Figura 3.8 Ejemplo de imagen del problema "Enlace en azul en dos líneas con fondo negro" (Fuente: set de datos proporcionado por el grupo GVIS)

Capítulo 4

4.1 ANÁLISIS INICIAL DEL PROYECTO

Como se ha mencionado anteriormente, para el desarrollo de este TFM no se parte desde cero, si no que utiliza como base un proyecto escrito en Python proporcionado por el grupo GVIS. Es por este motivo que la primera tarea antes de poder empezar a trabajar con el sistema es realizar un análisis del código para identificar sus características, que librerías necesita, puntos débiles y puntos fuertes…

El programa está compuesto por una gran cantidad de ficheros y funciones diferentes, por lo que el análisis se centrará en varios puntos principales:

Sistema usado para detección de texto: el proyecto utiliza una implementación de Pan++ [108] para la tarea de detección de texto, combinado con una gran cantidad de preprocesado de imágenes. El análisis inicial de este modelo no es muy prometedor, ya que a la hora de crear las cajas delimitadoras (*bounding boxes)* para el texto detectado, lo hace casi siempre palabra por palabra, en lugar de por líneas.

Figura 4.1 Ejemplo de cajas delimitadoras detectadas con Pan++ (Fuente: elaboración propia)

Este comportamiento añade mucho tiempo de ejecución a todo el sistema, ya que todas las tareas de preprocesado para la extracción de enlaces se aplican a cada una de esas cajas delimitadoras. A parte de esto, esta división complica la tarea de composición de

URL, ya que en la siguiente etapa el OCR detecta muchos menos espacios, los cuales son, por lo general, bastante útiles a la hora de decidir si un bloque de texto forma parte del enlace o no. Estas características ponen de manifiesto que los cambios en este punto son susceptibles de ser los que mayor mejora pueden introducir inicialmente.

Sistema usado para reconocimiento de texto: para esta tarea se aplica Tesseract [63], que, según varios artículos del estado del arte analizados, no es el modelo que más precisión obtiene, lo que indica que sustituyéndolo por otra herramienta se podría alcanzar una mayor precisión. Sin embargo, Tesseract está altamente acoplado al sistema implementado, ya que la mayoría de las tareas de postprocesado aplicadas tienen como objetivo solucionar problemas específicos de este motor.

Procesamiento de imágenes: las imágenes se introducen inicialmente al sistema como tal, para pasarlas primero por Pan++ con el objetivo de detectar todos los puntos que contienen texto en la imagen. Una vez se obtienen todos esos puntos se extraen esas regiones y pasan por todos los métodos de procesado: cambio de tamaño de imagen, binarización, combinación con cajas delimitadoras próximas…

Sin embargo, cabe destacar que no todas las imágenes pasan por el mismo proceso, después de aplicar algunas de esas transformaciones se utiliza el OCR sobre la imagen y si el texto obtenido es el inicio de un enlace (*http, https*) o tiene apariencia de un enlace abreviado conocido (*bit.ly*, *t.co*, etc.), salta a la siguiente etapa de reconocimiento donde se procesarán las imágenes de diferente manera dependiendo de si contienen texto en azul, que típicamente pertenece a un enlace, o no.

Figura 4.2 Ejemplo de imagen de una URL binarizada, abajo la imagen original, arriba, convertida a blanco y negro (Fuente: elaboración propia)

Procesamiento de texto y captura de URL: para esta tarea se aplican una serie de expresiones regulares y simples comprobaciones de texto para identificar el tipo de enlace

(abreviado o completo), corregir errores comunes en los enlaces y para eliminar partes del texto que no pertenezcan al enlace. Esta parte de la solución está altamente acoplada al OCR utilizado, Tesseract, por lo que complica bastante la tarea de introducir otro motor en el sistema.

Tras este análisis inicial, se probó el programa con el set de datos al completo, dando una precisión base del 40.38 % y tardando en ejecutarse más de 12 horas, datos que se usarán como punto de partida para contrastar los efectos a nivel global de los cambios realizados posteriormente.

Por otro lado, con los 5 problemas con más enlaces (enlace en azul en dos líneas línea, enlace sin color en una sola línea, enlace en azul en una sola línea, múltiples enlaces en azul y en dos líneas y enlace en azul en dos líneas con fondo negro) se obtiene una precisión del 47.27 % y un tiempo de ejecución de algo más de 7 horas.

4.2 CAMBIOS INTRODUCIDOS

Una vez expuesto el comportamiento base del sistema, las tareas restantes se centran en mejorar dos aspectos: la precisión de los 5 casos con más enlaces y la velocidad de ejecución. Como nota importante, las referencias a la precisión y al tiempo de ejecución en esta sección solo hacen referencia a esos 5 casos a menos que se indique lo contrario.

4.2.1 EASYOCR

Con el objetivo de incrementar la velocidad de ejecución y facilitar el trabajo en la composición de URL, lo primero que se buscó fue un nuevo motor de detección de texto compatible con el programa y que pudiera dividir el texto en líneas. Debido a su eficiencia sin necesidad de una *GPU* (tarjeta gráfica) demasiado potente, y su disponibilidad como una librería de Python, EasyOCR [109] fue el escogido para reemplazar a Pan++. Este OCR posee tanto la capacidad de detectar como de reconocer el texto, sin embargo, el motor de reconocimiento de EasyOCR introducía bastantes problemas por lo que finalmente no se hace uso de él.

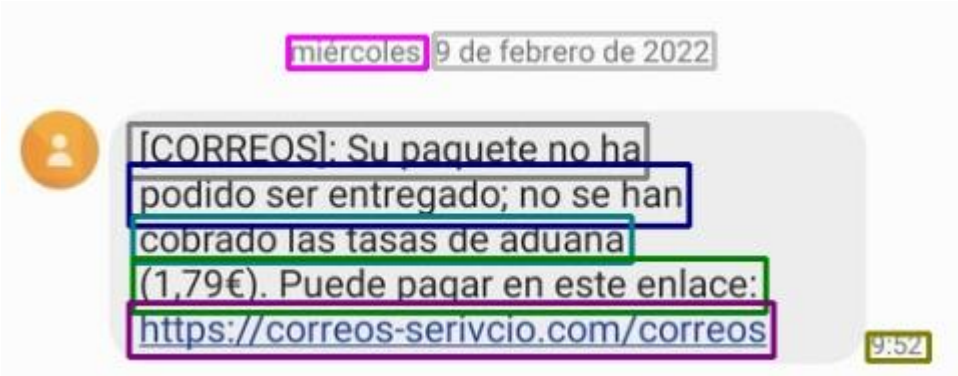

Figura 4.3 Ejemplo de bounding boxes detectadas con EasyOCR (Fuente: elaboración propia)

Con la introducción de este modelo los resultados mejoran en más de 6 puntos porcentuales, llegando al 53.52 % de precisión, y reduciéndose el tiempo de ejecución a aproximadamente la mitad, gracias a la gran reducción de la cantidad de cajas delimitadoras que se analizan por separado. Cabe destacar que, aunque el rendimiento general del sistema mejora con la introducción de EasyOCR, hay varios enlaces que se detectaban correctamente con Pan++ pero que ahora presentan errores. Para evitar que el rendimiento empeore se han aplicado una serie de cambios como se verá en las secciones 4.2.3 y 4.2.4 (conservar Pan++ como método de respaldo, corrección de caracteres…).

4.2.2 PROCESADO DE IMÁGENES

Las tareas de procesado de imágenes en el proyecto base son bastantes, sobre todo en cuanto al redimensionado. Para investigar el impacto de esta última se probó el proyecto eliminando algunos de los tamaños a los que se ampliaba o reducía la imagen, con lo que se determinó que aumentar la imagen un 300 % o reducirla un 50 % en la mayoría de los casos no producía cambios o solamente introducía problemas, sobre todo en el caso de que la imagen ya fuera de por sí bastante grande o pequeña. Por otro lado, también se observó que de manera general las imágenes a un 150 % de resolución obtenían mejores resultados con el OCR.

Por este motivo se eliminaron del proyecto las transformaciones que menos influencia tenían en el resultado, y se reordenaron las restantes para ejecutar siempre primero la ampliación al 150 % en caso de que no se identifique texto en primera instancia.

4.2.3 CORRECCIÓN DE ERRORES

Con los cambios listados anteriormente también se introdujeron algunos problemas nuevos, sobre todo a la hora de reconocer ciertos caracteres muy similares entre sí (como la *p* y la *g*, o el cero y la *O*). Tras realizar un análisis de algunos casos en los que esto ocurría se detectó que el error se producía principalmente al realizar la binarización de la imagen, ya que en este proceso se generaban artefactos en la imagen.

Para mejorar esta situación en primera instancia se introdujeron otros métodos de umbralización y binarización, sin embargo, ninguno produjo mejoras destacables con respecto al método ya empleado, y en bastantes casos incluso empeoraba el resultado final, por lo que no se realizó ningún cambio en este apartado. El siguiente método probado para solucionar este problema fue la utilización de las imágenes a color en vez de en blanco y negro, ya que de esta manera no se habrían creado artefactos que puedan suponer ruido para el OCR.

Atendiendo a los datos obtenidos anteriormente, este cambio debería dar peores resultados, ya que la precisión de los problemas en los que no se aplica binarización para obtener el enlace era del 60 % en los mejores casos, mientras que en el caso contrario llegaba al 80 %. No obstante, el error de los caracteres similares se subsana con este cambio, por lo que se implementó una función de decisión que compara ambos resultados y prioriza el valor del carácter obtenido en la imagen a color cuando son diferentes.

A parte de estas mejoras también se encontraron y arreglaron algunas partes del código que producían errores: la corrección de algunas faltas de ortografía producía fallos con enlaces que las contenían a propósito, algunas expresiones regulares estaban mal formuladas y reemplazaban caracteres que no debían, el redimensionado de algunas imágenes se estaba haciendo de manera incorrecta, etc.

También cabe destacar que se analizaron otros cambios con el objetivo de mejorar la detección de ciertos caracteres problemáticos, como la *j*, la *q* o la *g* cuando la URL está subrayada, y para acotar aún más la búsqueda de los elementos que componen un enlace, como buscar un elemento subrayado dentro de la imagen, pero estas propuestas finalmente fueron abandonadas debido a la falta de tiempo para implementarlas

correctamente de manera que no introdujeran errores adicionales. Estas propuestas se analizan posteriormente, en la sección de *Trabajo futuro* del Capítulo 5.

4.2.4 MÉTODO DE *FALLBACK*

Otro de los problemas encontrados era que la URL obtenida como resultado final presentaba caracteres incorrectos, estaba vacía, o presentaba alguna otra característica que podía considerarse un error.

Como se mencionó en el apartado de EasyOCR, Pan++ daba mejores resultados para algunas imágenes, es por esto por lo que se introdujo un sistema de respaldo (*fallback*) y de decisiones, similar a lo visto en algunos de los artículos vistos en el capítulo 2. Con este sistema se aplican una serie de expresiones regulares para analizar el enlace final (o enlaces), y en el caso de que algo extraño sea detectado se ejecuta el método de respaldo, es decir, el mismo programa, pero con Pan++ para la detección de texto y se reemplaza la URL incorrecta con la obtenida en esta segunda instancia.

4.5.5 DOMINIOS COMUNES

Uno de los problemas sobre el que menos optimización había en el proyecto son los enlaces abreviados sin *http* o *https*, con los que apenas se detectaba una URL entre más de 30 imágenes. Con el fin de mejorar estos resultados se arreglaron algunos errores en el código que se encargaba de detectar este tipo de links, y se introdujo un nuevo módulo de comprobación para que se admitieran como posible enlace las cadenas de texto que contengan algo seguido de una las extensiones de dominio más comunes [114] [115] [116], como *.es*, *.com*, *.to*… Con este cambio se pasa a conseguir 4 enlaces dentro de estos casos, 4 veces más de lo logrado inicialmente.

4.2.6 PARALELIZACIÓN

Todos los esfuerzos anteriores atienden, principalmente, a mejorar la precisión del sistema, y aunque alguno también reduce el tiempo de ejecución, en este punto, sigue siendo bastante elevado, y mejorar la velocidad para cada imagen resulta difícil.

Por ese motivo se explora una última alternativa para reducirlo: paralelizar el análisis de cada imagen. Para esto se crea una nueva función que se puede ejecutar de manera asíncrona con un *ThreadPoolExecutor*, con lo que se puede llamar a la función encargada de analizar una imagen múltiple veces al mismo tiempo. Junto a esto se añaden nuevos cambios en todas las funciones del código que trabajan con ficheros, para evitar la colisión entre procesos cuando escriben en un archivo que tenga el mismo nombre, y la posibilidad de configurar fácilmente el número de hilos con los que se ejecutará el programa.

Con esta paralelización y ejecutando el análisis de los 5 problemas con más URL en 4 hilos a la vez, se disminuye el tiempo de ejecución de unos 105 a 40 minutos, reduciendo el tiempo a menos de la mitad.

4.2.7 ACTUALIZACIÓN DE LIBRERÍAS

Dado que el programa base fue creado y desarrollado hace más de un año, existe la posibilidad de que dependa de librerías que recientemente se haya descubierto que presentan vulnerabilidades. Por este motivo, se realizó una actividad adicional: el análisis de todas las librerías y la actualización de aquellas que tuvieran vulnerabilidades, para lo que se configuró y utilizó *GitHub Dependabot* [117].

El resultado de este análisis llevó a identificar que había 5 librerías con vulnerabilidades de diferente nivel:

- urllib3: CVE-2021-33503 – Nivel de amenaza alto, actualizada a la versión 1.25.9.
- pillow: CVE-2022-45198 – Nivel de amenaza alto, actualizada a la versión 9.2.0.
- torch: CVE-2022-45907 – Nivel de amenaza crítico, actualizada a la versión 1.13.1.
    - torchvision: actualizada a la versión 0.14.1 para ser compatible con torch 1.13.1.
- certifi: CVE-2022-23491 – Nivel de amenaza moderado, actualizada a la versión 2022.13.07.
- requests: CVE-2023-32681 – Nivel de amenaza moderado, actualizada a la versión 2.31.0.

Este cambio en las librerías no tuvo un efecto en el rendimiento final del sistema, aunque sí fue necesario realizar pequeños cambios en el código para mantener la compatibilidad.

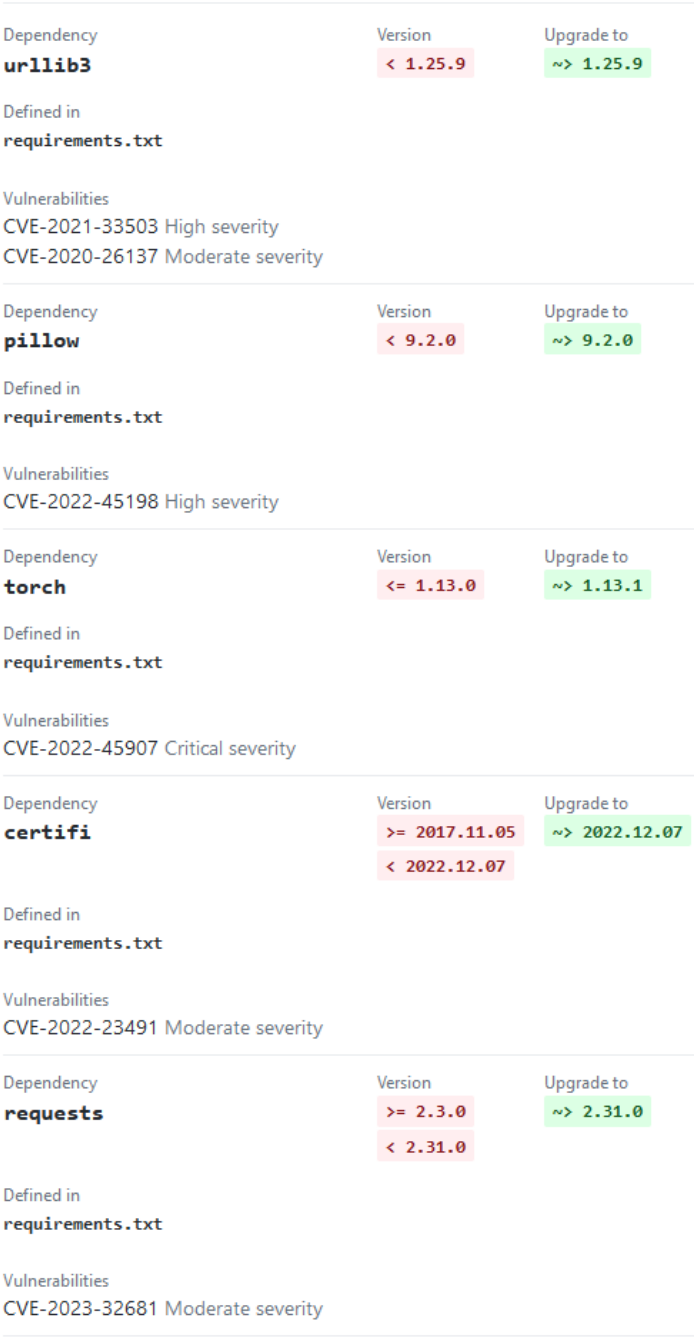

Figura 4.4 Análisis de seguridad realizado con GitHub Dependabot (Fuente: elaboración propia)

4.2.8 OTROS CAMBIOS

El resto de los cambios introducidos en el proyecto fueron mejoras centradas en la estructura del código y en la facilidad de ejecución y uso. Por un lado, hay un nuevo fichero de configuración denominado *config.py* que se encarga de almacenar datos y funcionalidades comunes de configuración: ruta a directorio del set de datos, al de los resultados, número de hilos que deben usarse para la ejecución, etc.

Por otro, hay un fichero *main.py* que, acompañado de una serie de perfiles de ejecución de Visual Studio Code, facilita bastante la ejecución del programa, admitiendo parámetros para configurar un prefijo para el nombre de los ficheros con el resultado, la cantidad de hilos, el caso o imagen a analizar, etc. También es el que contienen las funciones principales que ejecutan el análisis de las imágenes, y toman decisiones sobre si el enlace final es correcto o no.

También hay ficheros auxiliares que añaden, por ejemplo, la posibilidad de enviar un mensaje o fichero por Telegram cuando la ejecución termina. Finalmente, está *tfm_utils.*py que contiene el resto de las funciones auxiliares, como el redimensionado de imágenes, la corrección de caracteres similares, etc.

4.3 RENDIMIENTO FINAL DEL PROYECTO

El objetivo final de mejorar el resultado sobre los 5 casos con más imágenes se cumple, subiendo desde el 47.27 % original hasta a un 56.25 % de precisión, lo que supone un incremento de casi 9 puntos porcentuales. La velocidad también mejora en gran medida, pasando de unas 7 horas a 40 minutos.

El rendimiento global del sistema pasa de un 40.38 % a un 45.79 % de precisión. El tiempo de ejecución se reduce considerablemente, siendo inicialmente más de 12 horas y tardando 78 minutos con las mejoras implementadas.

Modelo	Descripción	Precisión
Pan++ y Tesseract	Modelo base, resultados obtenidos sin modificar el programa base	47.27

EasyOCR y Tesseract	Programa base, con Pan++ reemplazado por EasyOCR	53.50
EasyOCR, Tesseract y correcciones	Programa base, con Pan++ reemplazado por EasyOCR y con las correcciones mencionadas en el punto 4.2.3	54.68
EasyOCR, Tesseract, correcciones y Pan++	Programa con todas las modificaciones mencionadas: EasyOCR como sistema de detección de texto principal, Pan++ como método de respaldo, y corrección de problemas mencionados	56.25

Tabla 4.1 Precisión en los 5 casos principales del set de datos (Fuente: elaboración propia)

Problema	**URL**	**URL recuperadas correctamente**	**Precisión**
Enlace en azul en dos líneas línea	62	33	53.22
Enlace sin color en una sola línea	56	32	57.14
Enlace en azul en una sola línea	52	41	78.46
Múltiples enlaces en azul y en dos líneas	48	22	45.83
Enlace en azul en dos líneas con fondo negro	38	16	42.11
Enlace en azul en una sola línea con fondo negro	30	19	63.33
Enlace sin color dos líneas	27	12	44.44
Enlace sin color en una sola línea con fondo negro	26	9	34.62
Múltiples enlaces sin color y en dos líneas	25	7	28
Enlace sin color en dos líneas con fondo negro	9	3	33.33
Enlace en azul en tres líneas	9	2	22.22
Enlace en azul en tres líneas con fondo negro	7	0	0
Enlace sin color en tres líneas	7	0	0
Difícil	1	0	0
Enlace en otro color	2	1	50
Enlace en azul sin http	22	4	18
Enlace sin color y sin http	10	0	0

Falso negativo	8	0	0

Tabla 4.2 Precisión en cada caso del set de datos (Fuente: elaboración propia)

Por otro lado, también cabe realizar un análisis de la precisión teniendo en cuenta que se puede admitir un cierto número incorrecto de caracteres, ya que, según indicó el INCIBE para el desarrollo del proyecto original, se puede admitir un cierto número de errores sin que esto suponga un problema para los usos que se darán posteriormente a los enlaces extraídos:

Problema	**Precisión con 1 carácter mal**	**Precisión con 2 caracteres mal**	**Precisión con 3 caracteres mal**	**Precisión con 4 caracteres mal**	**Precisión con 5 caracteres mal**
Enlace en azul en dos líneas línea	76.67	83.33	83.33	86.67	88.33
Enlace sin color en una sola línea	80.36	83.93	83.93	83.93	92.86
Enlace en azul en una sola línea	92.31	92.31	94.23	96.15	98.08
Múltiples enlaces en azul y en dos líneas	66.67	72.79	75	77.08	77.08
Enlace en azul en dos líneas con fondo negro	63.16	73.68	86.84	86.84	86.84
Enlace en azul en una sola línea con fondo negro	86.67	86.67	86.67	90	90.33
Enlace sin color dos líneas	81.48	92.59	92.59	92.59	92.59
Enlace sin color en una sola línea con fondo negro	65.38	69.23	76.92	76.92	76.92
Múltiples enlaces sin color y en dos líneas	36	36	36	36	36
Enlace sin color en dos líneas con fondo negro	55.56	66.67	66.67	66.67	66.67
Enlace en azul en tres líneas	22.22	22.22	22.22	22.22	22.22
Enlace en azul en tres líneas con fondo negro	0	0	0	0	0

Enlace sin color en tres líneas	0	0	0	14.29	14.29
Difícil	0	0	0	0	0
Enlace en otro color	50	50	50	50	50
Enlace en azul sin http	31.82	31.82	31.82	31.82	31.82
Enlace sin color y sin http	0	0	0	0	0
Falso negativo	0	0	0	0	0

Tabla 4.3 Precisión con hasta 5 caracteres mal en cada problema del set de datos (Fuente: elaboración propia)

Con estos datos se puede observar que la precisión se incrementa en gran medida cuando se admiten hasta 5 caracteres mal: en los 5 casos con más URL pasa del 56.25 % al 88.67 %, un incremento de más de 23 puntos porcentuales, mientras que la precisión global del sistema sube desde el 45.79 % hasta un 74.26 %.

Capítulo 5

5.1 CONCLUSIÓN

En este trabajo se ha expuesto una aparente escasez de estudios dedicados a la detección de URL, lo cual refuerza las motivaciones planteadas para realizar una investigación de este tipo, ya que ayuda a reforzar y establecer un proyecto base para futuros avances o estudios en este mismo ámbito. Además, las mejoras implementadas (introducción de EasyOCR, sistema de respaldo, corrección de errores…) logran un incremento tanto en la precisión del sistema, pasando del 40.38 % a un 45.79 % en términos globales, como en su tiempo de ejecución, reduciéndose a una fracción, 78 minutos, en comparación con los tiempos iniciales de 12 horas.

Los resultados obtenidos permiten cumplir con la mayoría de los objetivos establecidos, tanto en términos de mejora del rendimiento y del resto de objetivos planteados formalmente, como en los objetivos personales, como la participación en un proyecto con aplicaciones reales y el aprendizaje sobre el uso de sistemas de IA y Visión Artificial.

También es importante destacar que aún existen muchas oportunidades para seguir mejorando el programa. El siguiente apartado se enfocará en mencionar algunas de estas posibilidades que no se han explorado en profundidad en este trabajo, pero que podrían contribuir significativamente en futuras investigaciones sobre este mismo problema.

5.2 TRABAJO FUTURO

Dentro de todos los cambios propuestos durante el desarrollo de este TFM para mejorar la precisión del sistema ha habido varios que, debido a las restricciones de tiempo y alcance del proyecto, no pudieron ser implementados o estudiados en profundidad. No obstante, es importante destacar que varias de estas propuestas tienen potencial para poder generar mejoras significativas si se dedica el tiempo necesario para su estudio e

implementación. Por lo tanto, sería pertinente considerar la posibilidad de explorar y evaluar su viabilidad en futuros trabajos.

5.2.1 DETECCIÓN DE SUBRAYADO

Una de las características más comunes de los enlaces es que casi siempre tienen un color diferente al del texto circundante, siendo el azul el color que se emplea más comúnmente para resaltarlos. Por ese motivo, una de las cosas que utiliza este programa para encontrar enlaces es comprobar si el color mayoritario del texto es el azul, y la efectividad de este método se ve reflejada en los resultados finales, los problemas en los que se detecta este color tienen una mayor precisión.

Con esta misma idea, una de las propuestas probadas es la detección de regiones con texto subrayado, ya que esta también es una característica común en casi todas las URL del set de datos.

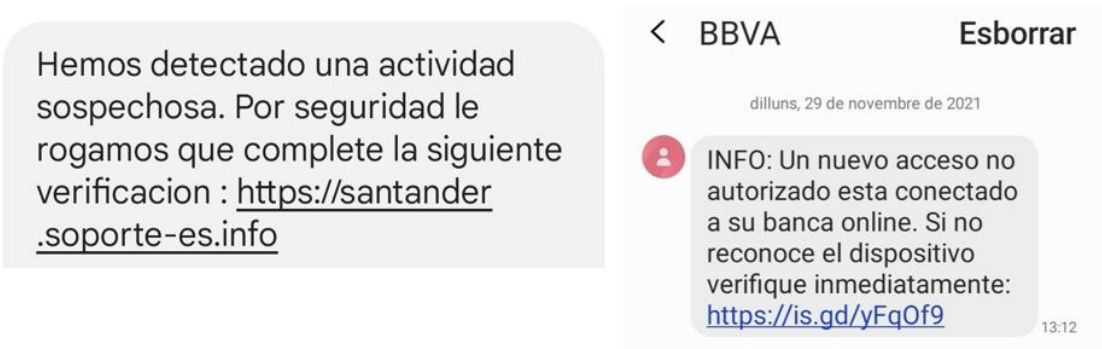

Figura 5.1 Izquierda, mensaje con enlace subrayado y en el mismo color que el texto, derecha, mensaje con el enlace en azul y subrayado (Fuente: set de datos proporcionado por el grupo GVIS)

Recortando el ancho de las regiones donde se ha detectado texto hasta que solo incluyan palabras subrayadas facilitaría bastante el trabajo en la fase de selección de texto que conforma un enlace, ya que no dependería de la detección de patrones comunes de texto (*http*, *https*, *.com*, *.net*…), todo el texto restante formaría parte de un enlace. La única parte que quedaría pendiente es identificar si las secciones de texto subrayado pertenecen a un único enlace o a varios. Esto se podría comprobar contrastando si la caja delimitadora

está directamente adyacente a otra ya identificada como enlace o no, si no la hubiera, se consideraría parte de otro enlace, y en caso contrario, parte del mismo.

De igual forma, también sería necesario añadir las mismas comprobaciones ya empeladas anteriormente, como que no esté presente la cadena de texto *http* dos veces, para asegurar que realmente es un enlace independiente.

5.2.2 ELIMINACIÓN DE SUBRAYADO

Gracias al análisis realizado para la detección de subrayado se encontró otro fenómeno común en el texto, y es que en muchas ocasiones la línea pasa por encima del texto, dificultando bastante el reconocimiento de ciertos caracteres, como la *q*, la *g*, o la *j*.

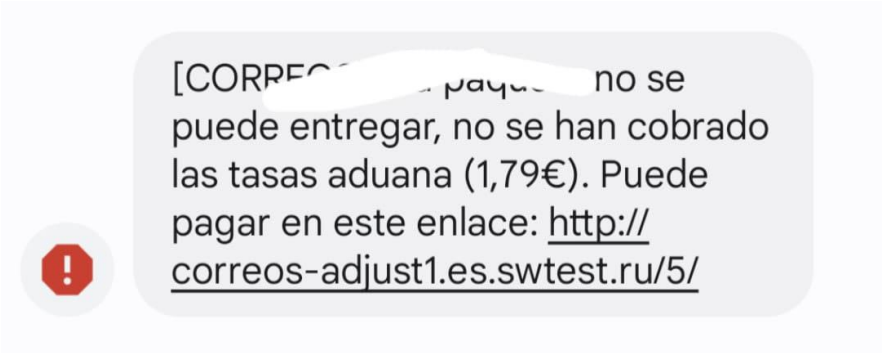

Figura 5.2 Enlace con una "j" tapada por la línea de subrayado (Fuente: set de datos proporcionado por el grupo GVIS)

Para acabar con este problema uno de los métodos más directos es la eliminación del subrayado, sin embargo, esta tarea no es tan sencilla como puede parecer en primera instancia.

Por un lado, al eliminar la línea de subrayado los caracteres que estaban parcialmente tapados por ella pasarán a quedar separados en varios trozos, por lo que será necesario aplicar un método de reconstrucción, como por ejemplo usando funciones de OpenCV como Erosión (*Erosion*) o Dilatación (*Dilation*). Esto complica más el proceso, ya que será necesario convertir la imagen a blanco y negro y el uso de esas funciones crea artefactos adicionales.

Figura 5.3 Ejemplos de reconstrucción de caracteres (Fuente: elaboración propia)

Por otro lado, detectar las líneas de subrayado de manera exacta puede resultar bastante complejo, ya existe el problema de que no todas tiene el mismo formato, por ejemplo, algunas son discontinuas u otras tienen mayor grosor.

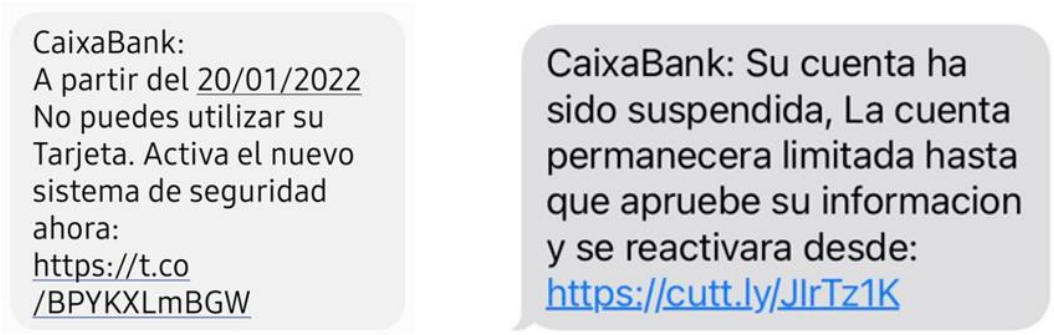

Figura 5.4 enlaces con diferentes tipos de subrayado (Fuente: set de datos proporcionado por el grupo GVIS)

5.2.3 NUEVO OCR

Otra cosa que se podría mejorar es la parte de reconocimiento del texto, ya que actualmente da problemas con varios caracteres. Hay varios motores de reconocimiento de texto que podrían ser probados con todas las imágenes para comprobar si tienen una menor tasa de error, pero este trabajo también necesitaría de bastantes cambios en el código para desacoplar las transformaciones de texto e imágenes actuales, ya que están adaptadas a las características concretas de Tesseract.

Esta idea también surge, en gran medida, debido a que EasyOCR tiene instrucciones para poder entrenar un modelo propio de manera más o menos sencilla, lo que permitiría especializar mucho más el OCR para resolver el problema actual.

Bibliografía y referencias

[1] «¿Qué es la inteligencia artificial (IA)? | IBM». https://www.ibm.com/es-es/topics/artificial-intelligence (accedido 29 de junio de 2023).

[2] «Phishing | INCIBE | INCIBE». https://www.incibe.es/aprendeciberseguridad/phishing (accedido 29 de junio de 2023).

[3] «F-Score», *DeepAI*, 17 de mayo de 2019. https://deepai.org/machine-learning-glossary-and-terms/f-score (accedido 29 de junio de 2023).

[4] «Classification: Accuracy | Machine Learning», *Google for Developers*. https://developers.google.com/machine-learning/crash-course/classification/accuracy (accedido 29 de junio de 2023).

[5] «Papers with Code - Data Augmentation». https://paperswithcode.com/task/data-augmentation (accedido 29 de junio de 2023).

[6] C. Galea, «What is 'Image Super Resolution', and why do we need it?», *Medium*, 20 de octubre de 2022. https://towardsdatascience.com/what-is-image-super-resolution-and-why-do-we-need-it-9c3bd9dc233e (accedido 29 de junio de 2023).

[7] «What is Machine Learning? | IBM». https://www.ibm.com/topics/machine-learning (accedido 29 de junio de 2023).

[8] «What Is Optical Character Recognition (OCR)?», 18 de febrero de 2022. https://www.ibm.com/cloud/blog/optical-character-recognition (accedido 29 de junio de 2023).

[9] «Thresholding (image processing)», *Wikipedia*. 19 de febrero de 2023. Accedido: 29 de junio de 2023. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Thresholding_(image_processing)&oldid=1140317997

[10] «Principales Formas De Estafa Traves Del Email Phishing Mas Comunes | Empresas | INCIBE». https://www.incibe.es/empresas/blog/principales-formas-de-estafa-traves-del-email-phishing-mas-comunes (accedido 29 de junio de 2023).

[11] «Conoce Uno De Los Ataques Mas Replicados En La Red El Phishing Y Sus Variantes | Empresas | INCIBE». https://www.incibe.es/empresas/blog/conoce-uno-de-los-ataques-mas-replicados-en-la-red-el-phishing-y-sus-variantes (accedido 27 de junio de 2023).

[12] «Sabes Funciona Ciberataque Utiliza Ingenieria Social | Empresas | INCIBE». https://www.incibe.es/empresas/blog/sabes-funciona-ciberataque-utiliza-ingenieria-socia (accedido 27 de junio de 2023).

[13] «Smishing | INCIBE | INCIBE». https://www.incibe.es/aprendeciberseguridad/smishing (accedido 27 de junio de 2023).

[14] A. Balasubramaniam y S. Pasricha, «Object Detection in Autonomous Vehicles: Status and Open Challenges». arXiv, 19 de enero de 2022. doi: 10.48550/arXiv.2201.07706.

[15] S. Cascianelli, M. Cornia, L. Baraldi, y R. Cucchiara, «Boosting Modern and Historical Handwritten Text Recognition with Deformable Convolutions». arXiv, 17 de agosto de 2022. doi: 10.48550/arXiv.2208.08109.

[16] C. Ma, Y. Zhang, M. Tu, Y. Zhao, Y. Zhou, y C. Zong, «E2TIMT: Efficient and Effective Modal Adapter for Text Image Machine Translation». arXiv, 9 de mayo de 2023. doi: 10.48550/arXiv.2305.05166.

[17] A. Sofwan, A. Y. Sumardi, I. Santoso, Y. A. A. Soetrisno, M. Arfan, y E. Handoyo, «Optimization of OCR in detecting research proposal and lecturer community service documents using thresholding method», en 2021 8th International Conference on Information Technology, Computer and Electrical Engineering, ICITACEE 2021. 2021. doi: 10.1109/ICITACEE53184.2021.9617487.

[18] C. Irimia, F. Harbuzariu, I. Hazi, y A. Iftene, «Official document identification and data extraction using templates and OCR», en Procedia Computer Science, vol. 207. 2022. doi: 10.1016/j.procs.2022.09.214.

[19] F. Naiemi, V. Ghods, y H. Khalesi, «Scene text detection and recognition: a survey», *Multimed Tools Appl*, vol. 81, n.º 14, pp. 20255-20290, jun. 2022, doi: 10.1007/s11042-022-12693-7.

[20] S. Long, X. He, y C. Yao, «Scene Text Detection and Recognition: The Deep Learning Era». arXiv, 9 de agosto de 2020. doi: 10.48550/arXiv.1811.04256.

[21] C.-K. Ch'ng, C. S. Chan, y C.-L. Liu, «Total-Text: toward orientation robustness in scene text detection», *IJDAR*, vol. 23, n.º 1, pp. 31-52, mar. 2020, doi: 10.1007/s10032-019-00334-z.

[22] «3 big problems with datasets in AI and machine learning», *VentureBeat*, 17 de diciembre de 2021. https://venturebeat.com/uncategorized/3-big-problems-with-datasets-in-ai-and-machine-learning/ (accedido 27 de junio de 2023).

[23] «What is Overfitting? | IBM». https://www.ibm.com/topics/overfitting (accedido 27 de junio de 2023).

[24] Y. Zhou, H. Xie, S. Fang, y Y. Zhang, «Semi-supervised text detection with accurate pseudo-labels», *IEEE Signal Processing Letters*, vol. 29, 2022, doi: 10.1109/LSP.2022.3175667.

[25] «Introducing ChatGPT». https://openai.com/blog/chatgpt (accedido 27 de junio de 2023).

[26] «20210125_Reglamento-TFM_EIIIA_aprobado_JE.pdf». Accedido: 29 de junio de 2023. [En línea]. Disponible en: https://ingenierias.unileon.es/wp-content/uploads/2021/02/20210125_Reglamento-TFM_EIIIA_aprobado_JE.pdf

[27] S. Mishra y D. Soni, «Smishing Detector: A security model to detect smishing through SMS content analysis and URL behavior analysis», *Future Generation Computer Systems*, vol. 108, 2020, doi: 10.1016/j.future.2020.03.021.

[28] «apk (file format)», *Wikipedia*. 17 de junio de 2023. Accedido: 26 de junio de 2023. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Apk_(file_format)&oldid=1160578715

[29] T. A. Almeida, J. M. G. Hidalgo, y A. Yamakami, «Contributions to the study of SMS spam filtering: new collection and results», en *Proceedings of the 11th ACM symposium on Document engineering*, en DocEng '11. New York, NY, USA: Association for Computing Machinery, sep. 2011, pp. 259-262. doi: 10.1145/2034691.2034742.

[30] A. K. Jain y B. B. Gupta, «Rule-based framework for detection of smishing messages in mobile environment», en Procedia Computer Science, vol. 125. 2018. doi: 10.1016/j.procs.2017.12.079.

[31] J. Cendrowska, «PRISM: An algorithm for inducing modular rules», *International Journal of Man-Machine Studies*, vol. 27, n.º 4, pp. 349-370, oct. 1987, doi: 10.1016/S0020-7373(87)80003-2.

[32] S. Siddharth, S. Chaudhary, y J. Meena, «URL scanner using optical character recognition», 2021. doi: 10.1007/978-981-33-4367-2_25.

[33] adegeo, «Regular Expression Language - Quick Reference», 18 de junio de 2022. https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference (accedido 27 de junio de 2023).

[34] I. Zharikov, F. Nikitin, I. Vasiliev, y V. Dokholyan, «DDI-100: Dataset for Text Detection and Recognition», en *Proceedings of the 2020 4th International Symposium on Computer Science and Intelligent Control*, nov. 2020, pp. 1-5. doi: 10.1145/3440084.3441192.

[35] M. Suzuki, S. Uchida, y A. Nomura, «A ground-truthed mathematical character and symbol image database», en *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, ago. 2005, pp. 675-679 Vol. 2. doi: 10.1109/ICDAR.2005.14.

[36] C. Wang y Y. Chen, «TCURL: Exploring hybrid transformer and convolutional neural network on phishing URL detection», *Knowledge-Based Systems*, vol. 258, 2022, doi: 10.1016/j.knosys.2022.109955.

[37] «Phishing Site URLs». https://www.kaggle.com/datasets/taruntiwarihp/phishing-site-urls (accedido 26 de junio de 2023).

[38] M. S. I. Mamun, M. A. Rathore, A. H. Lashkari, N. Stakhanova, y A. A. Ghorbani, «Detecting Malicious URLs Using Lexical Analysis», en *Network and System Security*, J. Chen, V. Piuri, C. Su, y M. Yung, Eds., en Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 467-482. doi: 10.1007/978-3-319-46298-1_30.

[39] P. Mowar y M. Jain, «Fishing out the Phishing Websites», en *2021 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, jun. 2021, pp. 1-6. doi: 10.1109/CyberSA52016.2021.9478237.

[40] Y. Liang, J. Deng, y B. Cui, «Bidirectional LSTM: An Innovative Approach for Phishing URL Identification», en *Innovative Mobile and Internet Services in Ubiquitous Computing*, L. Barolli, F. Xhafa, y O. K. Hussain, Eds., en Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2020, pp. 326-337. doi: 10.1007/978-3-030-22263-5_32.

[41] P. S. Chavez, «Comparison of Three Different Methods to Merge Multiresolution and Multispectral Data:Landsat TM and SPOT Panchromatic», *PHOTOGRAMMETRIC ENGINEERING*, 1991.

[42] J. Nilsson y T. Akenine-Möller, *Understanding SSIM*. 2020.

[43] Y.-M. Su, H.-W. Peng, K.-W. Huang, y C.-S. Yang, «Image processing technology for text recognition», en *2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, nov. 2019, pp. 1-5. doi: 10.1109/TAAI48200.2019.8959877.

[44] «Optical Character Recognition (OCR) - Image into Text | Huawei Cloud». https://www.huaweicloud.com/intl/en-us/product/ocr.html (accedido 26 de junio de 2023).

[45] M. Yousef, K. F. Hussain, y U. S. Mohammed, «Accurate, data-efficient, unconstrained text recognition with convolutional neural networks», *Pattern Recognition*, vol. 108, 2020, doi: 10.1016/j.patcog.2020.107482.

[46] T. Strauss, G. Leifert, R. Labahn, T. Hodel, y G. Mühlberger, «Dataset for ICFHR2018 Competition on Automated Text Recognition on a READ Dataset». Zenodo, 1 de octubre de 2018. doi: 10.5281/zenodo.1442182.

[47] H. Wang y S. Feng, «Research on text detection algorithm based on improved FPN», en IEEE Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), vol. 2022- October. 2022. doi: 10.1109/IAEAC54830.2022.9929716.

[48] K. He, X. Zhang, S. Ren, y J. Sun, «Deep Residual Learning for Image Recognition». arXiv, 10 de diciembre de 2015. doi: 10.48550/arXiv.1512.03385.

[49] M. He *et al.*, «ICPR2018 Contest on Robust Reading for Multi-Type Web Images», en *2018 24th International Conference on Pattern Recognition (ICPR)*, ago. 2018, pp. 7-12. doi: 10.1109/ICPR.2018.8546143.

[50] M. Chen, M. Ibrayim, y A. Hamdulla, «Research of scene text detection algorithms», en 5th International Conference on Intelligent Robotics and Control Engineering, IRCE 2022. 2022. doi: 10.1109/IRCE55557.2022.9963115.

[51] W. Wang *et al.*, «Efficient and Accurate Arbitrary-Shaped Text Detection with Pixel Aggregation Network». arXiv, 1 de agosto de 2020. doi: 10.48550/arXiv.1908.05900.

[52] D. Karatzas *et al.*, «ICDAR 2015 competition on Robust Reading», en *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, ago. 2015, pp. 1156-1160. doi: 10.1109/ICDAR.2015.7333942.

[53] «MSRA Text Detection 500 Database (MSRA-TD500) - TC11». http://www.iapr-tc11.org/mediawiki/index.php/MSRA_Text_Detection_500_Database_(MSRA-TD500) (accedido 28 de junio de 2023).

[54] L. Yuliang, J. Lianwen, Z. Shuaitao, y Z. Sheng, «Detecting Curve Text in the Wild: New Dataset and New Solution». arXiv, 6 de diciembre de 2017. doi: 10.48550/arXiv.1712.02170.

[55] M. Wödlinger y R. Sablatnig, «Text baseline recognition using a recurrent convolutional neural network», en Proceedings - International Conference on Pattern Recognition. 2020. doi: 10.1109/ICPR48806.2021.9412624.

[56] «U-Net», *Wikipedia*. 8 de septiembre de 2022. Accedido: 28 de junio de 2023. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=U-Net&oldid=1109255426

[57] C. Rigaud, A. Doucet, M. Coustaty, y J.-P. Moreux, «Dataset of ICDAR 2019 Competition on Post-OCR Text Correction». Zenodo, Sydney, Australia, 21 de octubre de 2019. doi: 10.5281/zenodo.3515403.

[58] R. Wang, Y. Fujii, y A. C. Popat, «Post-OCR paragraph recognition by graph convolutional networks», en 2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV). 2022, pp. 2533-2542. doi: 10.1109/WACV51458.2022.00259.

[59] T. N. Kipf y M. Welling, «Semi-Supervised Classification with Graph Convolutional Networks». arXiv, 22 de febrero de 2017. doi: 10.48550/arXiv.1609.02907.

[60] X. Zhong, J. Tang, y A. J. Yepes, «PubLayNet: largest dataset ever for document layout analysis». arXiv, 15 de agosto de 2019. doi: 10.48550/arXiv.1908.07836.

[61] N. H. Imam, V. G. Vassilakis, y D. Kolovos, «OCR post-correction for detecting adversarial text images», *Journal of Information Security and Applications*, vol. 66, 2022, doi: 10.1016/j.jisa.2022.103170.

[62] J. Mei, A. Islam, A. Moh'd, Y. Wu, y E. Milios, «Statistical learning for OCR error correction», *Information Processing and Management*, vol. 54, n.º 6, 2018, doi: 10.1016/j.ipm.2018.06.001.

[63] «Tesseract OCR». tesseract-ocr, 28 de junio de 2023. Accedido: 28 de junio de 2023. [En línea]. Disponible en: https://github.com/tesseract-ocr/tesseract

[64] «GNU Aspell». http://aspell.net/ (accedido 28 de junio de 2023).

[65] «Noisy channel model», *Wikipedia*. 26 de febrero de 2023. Accedido: 28 de junio de 2023. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Noisy_channel_model&oldid=1141756107

[66] I. Kissos y N. Dershowitz, «OCR Error Correction Using Character Correction and Feature-Based Word Classification». arXiv, 21 de abril de 2016. doi: 10.48550/arXiv.1604.06225.

[67] C. Xue, W. Zhang, Y. Hao, S. Lu, P. H. S. Torr, y S. Bai, «Language matters: A weakly supervised vision-language pre-training approach for scene text detection and spotting», en Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 13688 LNCS. 2022. doi: 10.1007/978-3-031-19815-1_17.

[68] A. Gupta, A. Vedaldi, y A. Zisserman, «Synthetic Data for Text Localisation in Natural Images». arXiv, 22 de abril de 2016. doi: 10.48550/arXiv.1604.06646.

[69] C. S. Chan, «Total-Text-Dataset (Official site)». 19 de junio de 2023. Accedido: 28 de junio de 2023. [En línea]. Disponible en: https://github.com/cs-chan/Total-Text-Dataset

[70] M. Liao, Z. Wan, C. Yao, K. Chen, y X. Bai, «Real-time Scene Text Detection with Differentiable Binarization». arXiv, 3 de diciembre de 2019. doi: 10.48550/arXiv.1911.08947.

[71] A. Chiatti *et al.*, «Text extraction and retrieval from smartphone screenshots», 2018. doi: 10.1145/3167132.3167236.

[72] «Okapi BM25», *Wikipedia, la enciclopedia libre*. 27 de junio de 2021. Accedido: 28 de junio de 2023. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Okapi_BM25&oldid=136620963

[73] R. Busa, K. C. Shahira, y A. Lijiya, «Small text extraction from documents and chart images», en INDICON 2022 - 2022 IEEE 19th India Council International Conference. 2022. doi: 10.1109/INDICON56171.2022.10039990.

[74] «Connectionist temporal classification», *Wikipedia*. 6 de mayo de 2023. Accedido: 28 de junio de 2023. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Connectionist_temporal_classification&oldid=1153458815

[75] S. M. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, y R. Young, «ICDAR 2003 robust reading competitions», en *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, ago. 2003, pp. 682-687. doi: 10.1109/ICDAR.2003.1227749.

[76] M. Jaderberg, K. Simonyan, A. Vedaldi, y A. Zisserman, «Synthetic data and artificial neural networks for natural scene text recognition», en *Workshop on deep learning, NIPS*, 2014.

[77] M. Jaderberg, K. Simonyan, A. Vedaldi, y A. Zisserman, «Reading text in the wild with convolutional neural networks», *International Journal of Computer Vision*, vol. 116, n.º 1, pp. 1-20, ene. 2016.

[78] S. S. Harsha, B. P. N. M. Kumar, R. S. S. R. Battula, P. J. Augustine, S. Sudha, y T. Divya, «Text recognition from images using a deep learning model», en 6th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), I-SMAC 2022 - Proceedings. 2022. doi: 10.1109/I-SMAC55078.2022.9987404.

[79] M. Tyagi, «HOG(Histogram of Oriented Gradients)», *Medium*, 24 de julio de 2021. https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f (accedido 28 de junio de 2023).

[80] «Local binary patterns», *Wikipedia*. 13 de octubre de 2022. Accedido: 28 de junio de 2023. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Local_binary_patterns&oldid=1115831394

[81] J. Huang *et al.*, «A multiplexed network for end-to-end, multilingual OCR», en Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2021. doi: 10.1109/CVPR46437.2021.00452.

[82] M. Liao, P. Lyu, M. He, C. Yao, W. Wu, y X. Bai, «Mask TextSpotter: An End-to-End Trainable Neural Network for Spotting Text with Arbitrary Shapes». arXiv, 22 de agosto de 2019. doi: 10.48550/arXiv.1908.08207.

[83] J. andreu Sánchez, V. Romero, A. H. Toselli, M. Villegas, y E. Vidal, «Dataset for ICDAR2017 Competition on Handwritten Text Recognition on the READ Dataset (ICDAR2017 HTR)». Zenodo, 27 de julio de 2017. doi: 10.5281/zenodo.835489.

[84] K. Olejniczak y M. Šulc, «Text detection forgot about document OCR», en CEUR Workshop Proceedings, vol. 3349. 2023.

[85] S.-X. Zhang, C. Yang, X. Zhu, y X.-C. Yin, «Arbitrary Shape Text Detection via Boundary Transformer». arXiv, 19 de junio de 2023. doi: 10.48550/arXiv.2205.05320.

[86] Y. Bi y Z. Hu, «Disentangled Contour Learning for Quadrilateral Text Detection», en *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, ene. 2021, pp. 908-917. doi: 10.1109/WACV48630.2021.00095.

[87] M. Liao, Z. Zou, Z. Wan, C. Yao, y X. Bai, «Real-Time Scene Text Detection with Differentiable Binarization and Adaptive Scale Fusion». arXiv, 21 de febrero de 2022. doi: 10.48550/arXiv.2202.10304.

[88] X. Sui *et al.*, «CRAFT: Cross-Attentional Flow Transformer for Robust Optical Flow». arXiv, 31 de marzo de 2022. doi: 10.48550/arXiv.2203.16896.

[89] G. Jaume, H. K. Ekenel, y J.-P. Thiran, «FUNSD: A Dataset for Form Understanding in Noisy Scanned Documents». arXiv, 29 de octubre de 2019. doi: 10.48550/arXiv.1905.13538.

[90] L. L. Wang *et al.*, «CORD-19: The COVID-19 Open Research Dataset». arXiv, 10 de julio de 2020. doi: 10.48550/arXiv.2004.10706.

[91] Y. Xu *et al.*, «LayoutXLM: Multimodal Pre-training for Multilingual Visually-rich Document Understanding». arXiv, 9 de septiembre de 2021. doi: 10.48550/arXiv.2104.08836.

[92] S. S. Ravindra, S. J. Sanjay, S. N. A. Gulzar, y K. Pallavi, «Phishing website detection based on URL», *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 2021, doi: 10.32628/cseit2173124.

[93] «¿Qué es un bosque aleatorio? | IBM». https://www.ibm.com/es-es/topics/random-forest (accedido 28 de junio de 2023).

[94] S. Kumar *et al.*, «On- device information extraction from screenshots in form of tags», *CoRR*, vol. abs/2001.06094, 2020, [En línea]. Disponible en: https://arxiv.org/abs/2001.06094

[95] Z. Tian, W. Huang, T. He, P. He, y Y. Qiao, «Detecting Text in Natural Image with Connectionist Text Proposal Network». arXiv, 12 de septiembre de 2016. doi: 10.48550/arXiv.1609.03605.

[96] «ML Kit», *Google for Developers*. https://developers.google.com/ml-kit (accedido 28 de junio de 2023).

[97] F. Zhao, M. R. Min, C. Shen, y A. Chakraborty, «Convolutional Neural Knowledge Graph Learning». arXiv, 29 de marzo de 2018. doi: 10.48550/arXiv.1710.08502.

[98] Jefatura del Estado, *Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales*, vol. BOE-A-2018-16673. 2018, pp. 119788-119857. Accedido: 29 de junio de 2023. [En línea]. Disponible en: https://www.boe.es/eli/es/lo/2018/12/05/3

[99] «Normativa», *AEPD*, 7 de octubre de 2022. https://www.aepd.es/es/informes-y-resoluciones/normativa (accedido 29 de junio de 2023).

[100] «¿Qué son los datos personales?» https://commission.europa.eu/law/law-topic/data-protection/reform/what-personal-data_es (accedido 29 de junio de 2023).

[101] «GRUPO DE VISIÓN Y SISTEMAS INTELIGENTES | Research groups - Universidad de León». https://portalcientifico.unileon.es/grupos/8538/detalle (accedido 28 de junio de 2023).

[102] «Inicio | Universidad de León». https://www.unileon.es/ (accedido 29 de junio de 2023).

[103] Ministerio de Cultura, *Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia*, vol. BOE-A-1996-8930.

1996, pp. 14369-14396. Accedido: 29 de junio de 2023. [En línea]. Disponible en: https://www.boe.es/eli/es/rdlg/1996/04/12/1

[104] «Visual Studio Code - Code Editing. Redefined». https://code.visualstudio.com/ (accedido 29 de junio de 2023).

[105] «GIMP», *GIMP*. https://www.gimp.org/ (accedido 29 de junio de 2023).

[106] «Anaconda | The World's Most Popular Data Science Platform», *Anaconda*. https://www.anaconda.com (accedido 29 de junio de 2023).

[107] «Home», *OpenCV*. https://opencv.org/ (accedido 29 de junio de 2023).

[108] W. Wang *et al.*, «PAN++: Towards Efficient and Accurate End-to-End Spotting of Arbitrarily-Shaped Text». arXiv, 2 de agosto de 2021. doi: 10.48550/arXiv.2105.00405.

[109] «EasyOCR». Jaided AI, 28 de junio de 2023. Accedido: 28 de junio de 2023. [En línea]. Disponible en: https://github.com/JaidedAI/EasyOCR

[110] «Software de hojas de cálculo Microsoft Excel | Microsoft 365». https://www.microsoft.com/es-es/microsoft-365/excel (accedido 29 de junio de 2023).

[111] «Microsoft 365 online gratuito | Word, Excel y PowerPoint». https://www.microsoft.com/es-es/microsoft-365/free-office-online-for-the-web (accedido 29 de junio de 2023).

[112] «Overleaf, Online LaTeX Editor». https://www.overleaf.com (accedido 29 de junio de 2023).

[113] «GitHub: Let's build from here», *GitHub*. https://github.com/ (accedido 29 de junio de 2023).

[114] «What Are the Most Common Domain Extensions in 2023? - GoDaddy», *GoDaddy Blog*, 14 de julio de 2022. https://www.godaddy.com/resources/skills/most-common-domain-extensions (accedido 29 de junio de 2023).

[115] K. Rohtela, «38 Popular Domain Name Extension in 2023», *Unboxfame*, 10 de abril de 2023. https://www.unboxfame.com/blog/which-are-the-popular-domain-name-extension-in-2023/ (accedido 29 de junio de 2023).

[116] «List of Internet top-level domains», *Wikipedia*. 28 de junio de 2023. Accedido: 29 de junio de 2023. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=List_of_Internet_top-level_domains&oldid=1162349937

[117] «Dependabot», *GitHub*. https://github.com/dependabot (accedido 4 de julio de 2023).

Anexo A

MANUAL DE USUARIO

El entregable proporcionado junto a este TFM es el programa sobre el que se ha trabajado. Está escrito en Python, y hace uso de múltiples librerías y herramientas externas, por lo que en esta sección se expondrá brevemente cómo crear un entorno en el que se pueda ejecutar y probar con el sistema operativo Ubuntu.

A.1 PYTHON Y LIBRERÍAS

La versión de Python recomendada para este proyecto es 3.8, concretamente durante el desarrollo y las pruebas se ha usado la 3.8.16. Para facilitar el uso de esta versión se puede usar un sistema como Conda [106], que permite crear diferentes entornos con la versión deseada tanto de Python como de las librerías de este.

Las librerías requeridas están incluidas en el fichero *requirementes.txt* y pueden ser instaladas automáticamente con el comando *pip install -r requirements.txt*. Adicionalmente es necesario instalar Tesseract con el comando *sudo apt install tesseract-ocr*.

A.2 CONFIGURACIÓN

El fichero *config.py* (*varpcore_textSpottingSMS/main/config.py*) incluye varios parámetros de configuración que deben ser cambiados dependiendo de la localización de diferentes ficheros y otros factores, a continuación se exponen los valores que deben ser configurados:

- tesseractPath: indica la ruta al ejecutable de Tesseract, por defecto */usr/bin/tesseract* en Ubuntu.
- datasetPath: ruta donde está el set de datos a usar.
- pretrainedPath: ruta donde se encuentran los ficheros base como *resnet18-imagenet.pth*.
- resultsPath: directorio en el que se deben escribir los resultados de la ejecución.
- easyOCRUseGPU: indica si EasyOCR debe usar la GPU o no.

- threadPrefix: prefijos de hilos, se debe añadir uno para cada hilo que se quiera usar para ejecutar el programa, por defecto es *['0-', '1-', '2-', '3-']*.
- threaded: indica si el programa debe ejecutarse con varios hilos o no.

A.3 USO DEL PROGRAMA

Se recomienda instalar Visual Studio Code (VS Code) [104], ya que el proyecto contiene varios perfiles de ejecución hechos para este programa que facilitan la realización de pruebas. Para emplear estos perfiles se ha de abrir la carpeta *tfm-url-detection*, la carpeta raíz, con VS Code, y posteriormente, desde el panel de "Ejecutar y depurar" se debe seleccionar uno de los perfiles:

- *Main - Run default image*: ejecuta el código sobre una imagen de prueba.
- *Main - Run default image*: ejecuta el código sobre una imagen indicada en el fichero de configuración del perfil.
- *Main - Run All*: ejecuta el código sobre todas las imágenes del set de datos.
- *Main – Run top 5*: ejecuta el código sobre las imágenes los cinco problemas con más URL.
- *Main - Run the rest*: ejecuta el código sobre las imágenes todos los problemas excepto las de los 5 con más imágenes.

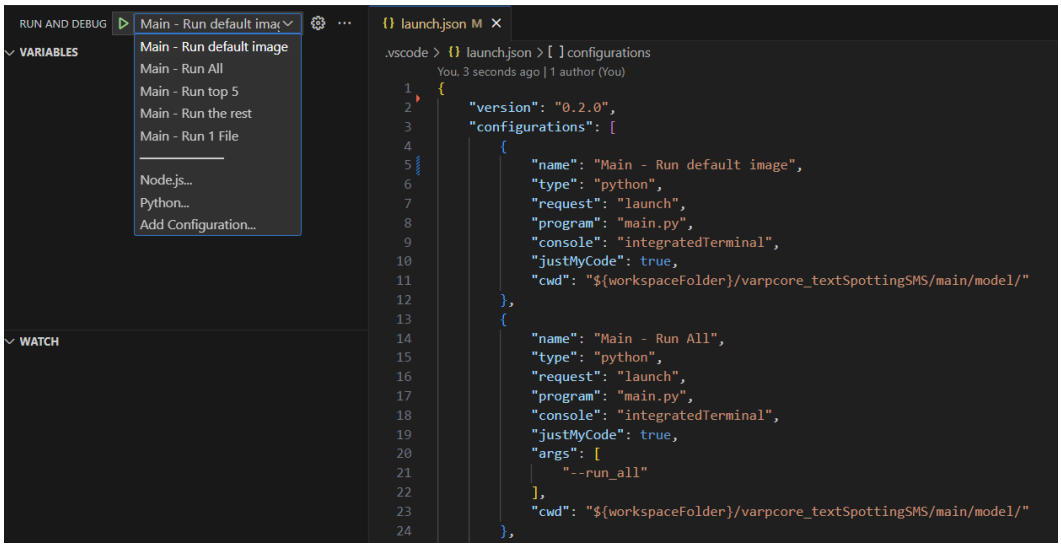

Figura A.1 Perfiles de ejecución creados para Visual Studio Code (Fuente: elaboración propia)

Como alternativa a esto se puede utilizar directamente la terminal del sistema operativo, para lo cual será necesario navegar hasta la carpeta que contiene el fichero *main.py* (*varpcore_textSpottingSMS/main/model/main.py*). Una vez la terminal tiene abierta esa ruta, basta con introducir el comando *Python main.py* para que se ejecute el programa sobre una imagen de prueba. Para poder darle otras órdenes se pueden añadir los siguientes argumentos al comando:

- --prefix: prefijo que se añadirá al nombre del fichero que almacenará los resultados de la ejecución.
- --no_threads: desactiva la ejecución multihilo del programa.
- --file <ruta_fichero>: ejecuta el programa sobre la imagen pasada en *ruta_fichero*. Si *ruta_fichero* es un directorio, el programa analizará todas las imágenes presentes en este.
- --run_all: ejecuta el código sobre todas las imágenes del set de datos.
- --run_5: ejecuta el código sobre las imágenes de los cinco problemas con más URL.
- --run_remaining: ejecuta el código sobre las imágenes todos los problemas excepto las de los 5 con más imágenes.
- --panpp: cambia el modelo de detección de texto poniendo Pan++ como principal y EasyOCR como respaldo.

Al terminar el programa guardará los enlaces extraídos en la carpeta indicada en *config.py*, creando para cada problema o carpeta analizada un directorio con su nombre donde se guardan los resultados y el tiempo de ejecución en los ficheros *…-results.txt* y *…-time.txt*.

El formato de estos resultados es el siguiente:

- NombreImagen Enalce1 || Enlace2 || Enalce3… ||
    - Twitter_SMS_Screenshot_00128.jpg https://s.id/RBLQ ||