UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Aprendizaje profundo para la gestión de formularios

Autor/es

MARÍA VILLOTA MIRANDA

Director/es

JÓNATAN HERAS VICENTE y María Vico Pascual Martínez Losa

Facultad

Escuela de Máster y Doctorado de la Universidad de La Rioja

Titulación

Máster universitario en Ciencia de Datos y Aprendizaje Automático

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2020-21

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE MÁSTER

Máster en Ciencia del dato y Aprendizaje Automático

Aprendizaje profundo para la gestión de formularios

Realizado por:

María Villota Miranda

Tutelado por:

Jónathan Heras Vicente

María Vico Pascual Martínez Losa

7 de julio de 2021

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación y los fondos FEDER con un contrato con cargo al proyecto "HOLMS: Técnicas avanzadas de análisis de imágenes para la categorización y extracción de información en documentos"[RTC-2017-6640-7].

Resumen

Debido al creciente volumen de formularios que se generan diariamente, la extracción automática de la información incluida en estos documentos está muy demandada. Sin embargo, no es una tarea sencilla debido a la gran diversidad de plantillas, a las distintas ubicaciones de las entidades de los formularios, y a la calidad de los documentos escaneados. En este trabajo, hemos dado un primer paso hacia el reconocimiento de entidades en formularios (por ejemplo, pares clave-valor) combinando técnicas de visión por computador y de procesamiento del lenguaje natural. En primer lugar, hemos aplicado modelos de detección de objetos y segmentación semántica de última generación para localizar la posición de las entidades basándonos únicamente en las características visuales. También hemos creado un modelo de localización utilizando únicamente técnicas de procesamiento de imágenes. Después, hemos estudiado diferentes enfoques de *transfer learning* (ajuste y extracción de características) para clasificar el contenido textual de las entidades localizadas. Los modelos estudiados requieren recursos computacionales y de imagen bajos, lo que los convierte en una alternativa factible a los modelos de última generación, aunque su rendimiento sea ligeramente peor.

Abstract

Due to the increasing volume of forms that are generated in a daily basis, the automatic extraction of the information included in these documents is greatly demanded. However, this is not a straightforward task due to the great diversity of templates with different location of form entities, and the quality of the scanned documents. In this work, we have made a first step towards form entity recognition by combining computer vision and natural language processing techniques. We have also created a location model using only image processing techniques. First, we have applied state-of-the-art deep object detection and semantic segmentation models for localising the position of form entities based only on visual features. Afterwards, we have studied different transfer learning approaches (fine-tuning and feature extraction) for classifying the text content of the localised form entities. The studied models require low computational and image resources, making them a feasible alternative to state-of-the-art models even if their performance is slightly worse.

Índice general

Capítulo 1

Introducción

Los formularios son un tipo de documento muy extendido en muchos campos como la administración, la medicina, las finanzas, los seguros, etc. Este tipo de documento se utiliza para recoger y comunicar datos siguiendo un formato estructurado. Hoy en día, existe una gran demanda para digitalizar formularios e interpretar los datos que se incluyen en ellos, con el fin de conocer mejor los datos y poder hacer estudios sobre ellos. Generalmente, encontramos estos formularios en dos formatos distintos: en formato digital (como PDFs, ficheros HTML, o documentos incluidos en aplicaciones web), o como una imagen obtenida al escanear un formulario escrito o impreso en papel. Independientemente del formato de los formularios, existen infinidad de plantillas, estructuras y diseños que pueden variar mucho de un formulario a otro. Lo único que es común a todos ellos es la manera en la que se presenta la información, que se hace principalmente a través de pares clave-valor o pregunta-respuesta. En facturas es habitual encontrar pares clave-valor como *Fecha: 19/04/12* o *TOTAL 365.00€*, en estos casos, la clave o la pregunta sería *Fecha:* y *TOTAL*; y el valor o respuesta sería *19/04/12* y *365.00€*.

El coste en tiempo y recursos necesarios dentro de las organizaciones para manejar y estructurar la información efectiva que contienen todos estos tipos de documentos es, hoy en día, demasiada elevada. Tradicionalmente, los sistemas inteligentes de captura o análisis de documentos han tratado de minimizar estos costes mediante técnicas avanzadas de procesamiento de texto e imágenes. Estas técnicas permiten extraer la información contenida en las imágenes escaneadas en los documentos. Sin embargo, en lo que se refiere a formularios o facturas, todos los sistemas comerciales actuales tienen serias limitaciones para alcanzar un grado de automatización óptimo en los procesos de extracción de información.

Surgue así el proyecto *"HOLMS: Técnicas Avanzadas de Análisis de Imágenes para*

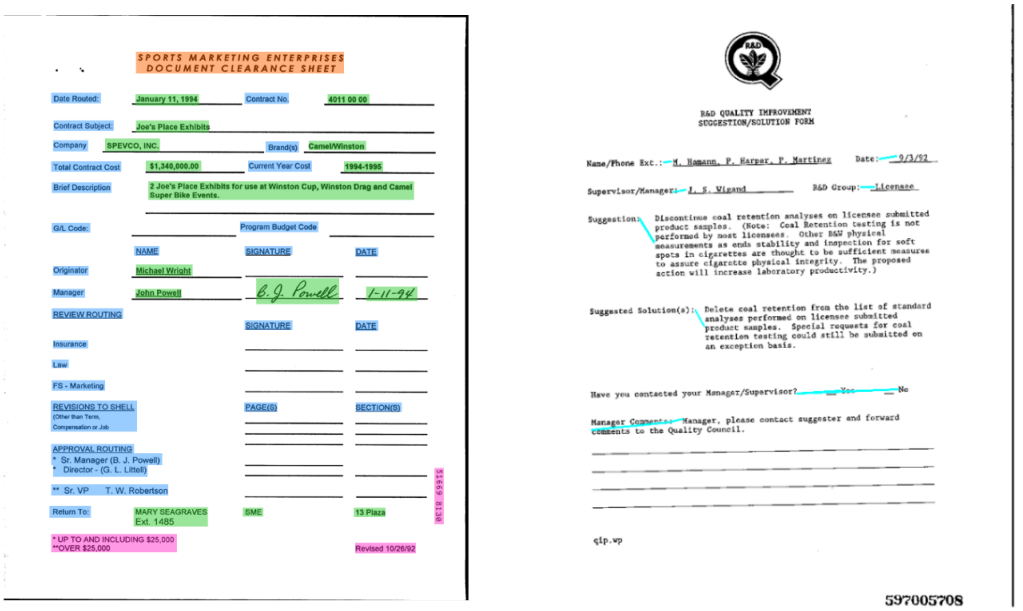
Categorización y Extracción de Información de Documentos" llevado a cabo por la empresa Ysengineers S.L. y el Grupo de Informática de la Universidad de La Rioja, con el objetivo de automatizar al máximo los procesos de clasificación y extracción de datos en documentos de tipo contable y formularios, reduciendo también al máximos el esfuerzo humano necesario en la gestión de los mismos. Desde un punto de vista tecnológico, el objetivo del proyecto es ser capaces de convertir meras imágenes de documentos en una representación estructurada y codificada en un lenguaje de marcado basado en XML que sea posteriormente procesable y reusable por sistemas automáticos. De esta forma, la información contenida en los documentos sería más fácil de interpretar, compartir e incluso integrar con los sistemas finales de contabilidad o sistemas de información.

Esta tarea se denomina *comprensión de formularios* y puede definirse como el proceso de extracción automática de esos pares clave-valor, y suele abordarse analizando tanto el contenido textual como las estructuras organizativas del documento. La comprensión de formularios en documentos escaneados es una tarea difícil debido a la diversidad de plantillas y formatos que pueden variar mucho de un formulario a otro; y, también, debido a la calidad de las imágenes de los documentos escaneados. Podemos dividir esta tarea en dos: el *reconocimiento de entidades* y el *enlazado de entidades*. La primera consiste en localizar e identificar los pares de pregunta-respuesta y las demás entidades contenidas en el documento (encabezados u otros). Una vez extraídas dichas entidades, hace falta entender la relación que existe entre ellas, este paso tiene lugar en la tarea de enlazado de entidades. En la figura 1.1a vemos un ejemplo de lo que esperamos en la tarea de reconocimiento de entidades. Los rectángulos se corresponden con la localización y el color con la identificación: las preguntas se corresponden con los rectángulos azules, las respuestas con los verdes; los encabezados aparecen en naranja y las demás entidades se identifican todas como "otro" y se corresponden con los rectángulos rosas. En la figura 1.1b vemos cómo con la línea azul asociamos cada pregunta con su respuesta.

Como ya hemos dicho, el objetivo del proyecto sería abordar estas dos tareas. No obstante, para este trabajo nos centraremos en la tarea de la figura 1.1a, es decir, en el reconocimiento de entidades. Se trata de un problema de detección de objetos y más adelante veremos cómo lo abordaremos.

El código asociado a este trabajo se encuentra en un repositorio de GitHub. Su dirección es: `https://github.com/mavillot/FUNSD-Information-Extraction`.

A continuación veremos la distribución del trabajo. En el siguiente capítulo, el capítulo 2, introduciremos el dataset que hemos utilizado para construir nuestros modelos. El resto del trabajo está dividido en dos grandes bloques. El primero está concentrado en el capítulo 3, en el que explicaremos los distintos modelos de visión por computador.



(a) Reconocimiento de entidades

(b) Enlazado de entidades

Figura 1.1: Comprensión de formularios

El segundo gran bloque se corresponde con el capítulo 4 en el que veremos técnicas de procesamiento de lenguaje natural para crear clasificadores de texto. En el capítulo 5 veremos otra aproximación basada en procesamiento de imágenes y evaluaremos nuestros modelos en un dataset nuevo. Y, por último, en el capítulo 6, veremos unas conclusiones finales.

Capítulo 2

FUNSD

Como ya hemos dicho antes, los formularios son documentos que sirven para recoger y comunicar información. Sin embargo, la información contenida suele ser sensible. Esta es la razón de la inexistencia de datasets libres para trabajar con la comprensión de formularios. Por lo que la publicación del dataset de FUNSD [1] en 2019 fue un gran hito para la investigación en este campo.

2.1. Descripción del dataset

El dataset de FUNSD es un dataset libre y gratuito formado por 199 documentos antiguos escaneados con ruido que varían con respecto a la estructura y apariencia. Los documentos se tomaron del dataset RVL-CDIP [2] que está compuesto por imágenes reales en escala de grises de la época de los 80 y 90. Los documentos tienen una calidad con distintos tipos de ruido añadido a partir de sucesivos escaneados e impresiones, y una baja resolución (alrededor de 100 ppp - *puntos por pulgada*).

Las anotaciones de cada una de las imágenes están codificadas en archivos JSON. Cada formulario está representado por un lista de entidades semánticas que están interconectadas. Una entidad semántica está descrita por un único identificador, una etiqueta (que pertenece a una de las siguientes categorías: pregunta, respuesta, encabezado, u otro), un rectángulo con la posición de la entidad, una lista de palabras y una lista de enlaces con la relación entre las entidades. En la figura 2.2, podemos ver un ejemplo de anotación de la imagen que aparece en la figura 2.1.

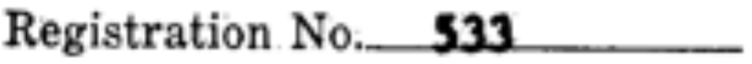


Figura 2.1: Ejemplo de anotación incluida en el dataset FUNSD

```
{
    "form": [{
                "id": 0,
                "text": "Registration No.",
                "box": [94,169,191,186],
                "linking": [[0,1]],
                "label": "question",
                "words": [{
                            "text": "Registration",
                            "box": [94,169,168,186]
                          },
                          {
                            "text": "No.",
                            "box": [170,169,191,183]
                          }]
            },
            {
                "id": 1,
                "text": "533",
                "box": [209,169,236,182],
                "label": "answer",
                "words": [{
                            "text": "533"
                            "box": [209,169,236,182],
                          }],
                "linking": [[0,1]]
        }]
}
```


Figura 2.2: Anotación JSON de la figura 2.1

Como vemos, el archivo JSON está formado por una única clave "*form*", que se corresponde con el tipo de documento. Los 199 documentos que componen el dataset de FUNSD son de tipo "*form*". El valor asociado con la clave "*form*" es una lista y, cada elemento de dicha lista, se corresponde con la anotación de una entidad semántica. En la figura 2.1, existen dos entidades: *Registration No.* y *533*. En la anotación vemos como a cada una de ellas se le asigna un "id", se marca el texto contenido en dicha entidad, el rectángulo que la contiene (al cual llamaremos a partir de ahora *bounding*

	Encabezado	Pregunta	Respuesta	Otro	TOTAL
Conjunto de entramiento	441	3.266	2.802	902	7.411
Conjunto de test	122	1.077	821	312	2.332

Tabla 2.1: Distribución de las clases de las entidades semánticas de FUNSD

box), la etiqueta y el enlace que indica si la entidad está relacionada con otra. También encontramos la anotación a nivel de palabra. Se trata de una lista de palabras con sus *bounding boxes*. Nuestro objetivo es encontrar la posición y la clase de cada una de las entidades, por lo que, la información de los enlaces y la anotación a nivel de palabra no la consideraremos.

Los 199 formularios anotados contienen más de 30.000 palabras anotadas y alrededor de 10.000 entidades. El dataset aparece ya separado en conjunto de entrenamiento y conjunto de test con 149 y 50 imágenes respectivamente. Los estadísticos del dataset y la distribución de las entidades semánticas, como se describen en [1], se incluyen en la tabla 2.1. Como podemos observar, las entidades no están balanceadas, esto quiere decir, que hay muchas más entidades de las clases pregunta y respuesta que de las de encabezado y otro. Esto es algo que tendremos que tener en cuenta a la hora de calcular las métricas. En el apartado siguiente explicaremos las métricas que se utilizan en este dataset.

2.2. Métricas

Con la publicación del propio dataset se proporcionaron métricas de partida, que también han sido utilizadas en artículos posteriores. Estas métricas son la *Precision*, el *Recall* y el *F1-score*. Por lo tanto, nosotros para poder comparar los resultados obtenidos con los que ya existen utilizaremos también dichas métricas, además de la métrica *mAP: mean Average Precision*, que se utiliza habitualmente para la detección de objetos. Todas ellas se explican a continuación.

Los algoritmos de detección de objetos en imágenes, que son los usados para localizar las entidades en los formularios, producen un conjunto de *bounding boxes*, unas etiquetas asociadas a los mismos y el nivel de confianza con el que se predicen ambos. Para evaluar dichos algoritmos es necesario en primer lugar determinar si los objetos de las imágenes han sido localizados correctamente con los *bounding boxes*. Para ello definimos la *IoU (Intersection over union)* [3], que mide cuánto se superpone el rectángulo predicho con el real.

El valor del IoU se calcula de la siguiente manera:

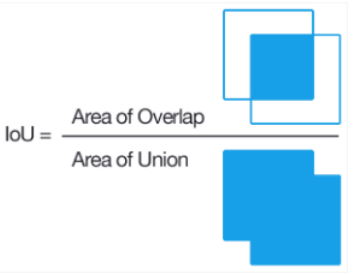


Figura 2.3: Representación gráfica del IoU [4]

$$IoU = \frac{area\ interseccion}{area\ union}$$

En la figura 2.3 mostramos una pequeña ilustración de lo que es el IoU. Consideramos que el rectángulo predicho es correcto si tiene un $IoU \geq 0.5$, aunque dependiendo del contexto, este valor puede variar.

Una vez se ha determinado si un objeto es localizado mediante uno de los *bounding boxes* predichos, debemos determinar si la clase predicha por el modelo es la correcta. Para ello calcularemos la *Precision*, el *Recall* y el mAP [3].

$$Precision = \frac{predicciones\ correctas}{Total\ predichas}$$

$$Recall = \frac{predicciones\ correctas}{Total\ verdaderas}$$

La *Precision* responde a ¿cuántos de los objetos predichos son correctos? y el *Recall* ¿cuantos de los objetos reales hemos predicho? Para cada clase obtendremos un valor para la *Precision* y el *Recall*. Por lo tanto, para calcular la *Precision* y el *Recall* total hacemos la media. A veces es difícil saber qué modelo es mejor cuando obtenemos valores parecidos para la *Precision* o el *Recall*. Para poder comparar los modelos más fácilmente usamos la métrica *F1-score* que relaciona estas dos:

$$F1\text{-}score = 2\ \frac{Precision \cdot Recall}{Precision + Recall}$$

Por último, para calcular el mAP [3] necesitamos calcular la curva de precision-recall para cada clase. Esta curva se consigue graficando la *Precision* y el *Recall* según vamos analizando los rectángulos. Una vez que tenemos la curva para cada clase calculamos el AP *(Average Precision)* que es el área debajo de la curva. El mAP no es más que hacer la media de los distintos AP.

2.3. Trabajo relacionado

Con la publicación del dataset FUNSD se proporcionaron métricas de partida tanto para la localización de texto como para el enlazado de entidades, un *F1-score* de 0.57 y 0.04 respectivamente [5]. Desde la publicación de ese trabajo, este dataset ha sido un punto de referencia para muchos proyectos. De todos ellos, destacan 3 trabajos. LayoutLM [6], LayoutLMv2 [7] y BROS [8] son el nombre de los tres modelos más relevantes y con mejores resultados. Los tres se centran en la tarea de reconocimiento de entidades, aunque el último, también se encarga del enlazado de entidades. Todos ellos fueron preentrenados con una gran cantidad de documentos, más concretamente, la colección de IIT-CDIP [9] formada por 6 millones de documentos que se traducen en 11 millones de páginas. Los resultados de estos modelos los veremos a continuación.

2.3.1. Resultados

En la tabla 2.2 mostramos los resultados que se obtienen con los modelos existentes para la tarea de reconocimiento de entidades.

	Precision	Recall	F1-score
Original	-	-	0.57
BROS	0.81	0.82	0.81
LayoutLM	0.76	0.81	0.79
LayoutLMv2	0.83	0.85	0.84

Tabla 2.2: Métricas reconocimiento de entidades

En la tabla 2.3 encontramos los resultados del modelo de BROS que es el único que se dedica también al enlazado de entidades.

	Precision	Recall	F1-score
Original	-	-	0.04
BROS	0.64	0.70	0.67

Tabla 2.3: Métricas enlazado de entidades

Observamos como en la tarea de reconocimiento de entidades se obtiene un *F1-score* de 0.84 con el modelo de LayoutLMv2. Para el enlazado de entidades, el modelo de BROS obtiene un *F1-score* de 0.67. Se mejora, entonces, los resultados de partida obtenidos en ambas tareas.

La desventaja principal de estos modelos es que hace falta preentrenarlos con 11 millones de imágenes de documentos antiguos. Si queremos que estos modelos funcionen bien con documentos más modernos se necesitaría un preentrenamiento parecido al anterior, pero, con datasets actuales. Sin embargo, hasta la fecha, no conocemos de la existencia de ningún dataset de documentos modernos de tal tamaño. En nuestro trabajo, hemos diseñado una aproximación para el reconocimiento de entidades que no requiere ese paso de preentreno. Subdividimos esta tarea en dos, la primera consiste en localizar las entidades basándonos simplemente en las características visuales, y, una vez realizado este paso, etiquetaremos semánticamente cada una de las entidades usando modelos de clasificación de texto.

Capítulo 3

Modelos de visión

Como hemos visto en el dataset de FUNSD, los formularios constan de entidades como son los encabezados, pequeñas imágenes, logotipos o pares clave-valor entre otros. Algunas de estas entidades pueden reconocerse a simple vista sin entrar a valorar su contenido. En este capítulo abordamos la tarea de reconocer entidades en formularios usando técnicas de visión por computador. El reconocimiento de entidades en formularios puede ser visto como una tarea de detección de objetos o como una de segmentación semántica. Daremos unas breves nociones básicas de ambos tipos de algoritmos en este capítulo. Seguidamente, expondremos el flujo de trabajo para construir los distintos modelos que comienza con la obtención del dataset y termina con la evaluación de los distintos modelos. Por último, daremos unas pequeñas conclusiones.

3.1. Detección y segmentación

Debido a la variedad de tareas relacionadas con la visión por computador, puede ser difícil diferenciar las distintas tareas existentes. Por lo tanto, empezaremos describiendo la más general: el reconocimiento de objetos. Este es un término que describe una colección de tareas de visión por computador que involucra la identificación de objetos en imágenes digitales [10]. Entre ellas se encuentran las tareas de detección y segmentación.

Los algoritmos de detección de objetos determinan la posición mediante un rectángulo o un *bounding box* (localización) e identifican a cada uno de los objetos localizados con una etiqueta (clasificación). Por lo tanto, la detección de objetos combina la tarea de localización con la clasificación. En la figura 3.1 podemos ver como a cada objeto se le asigna un *bounding box* y que a cada uno de ellos se le da una etiqueta.

Por otro lado, los modelos de segmentación realizan una localización a nivel de píxel en vez de utilizar un rectángulo. Dentro de la segmentación diferenciamos la segmentación

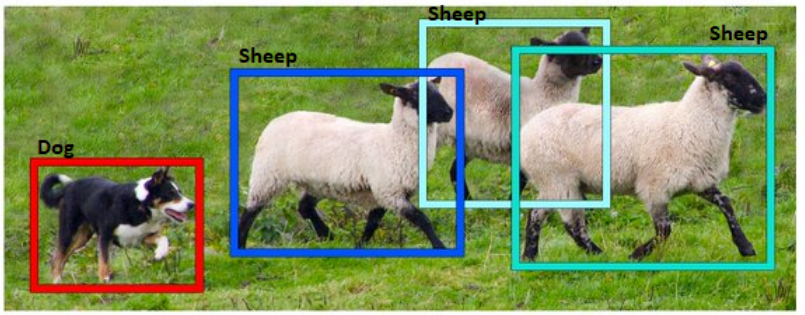


Figura 3.1: Detección de objetos

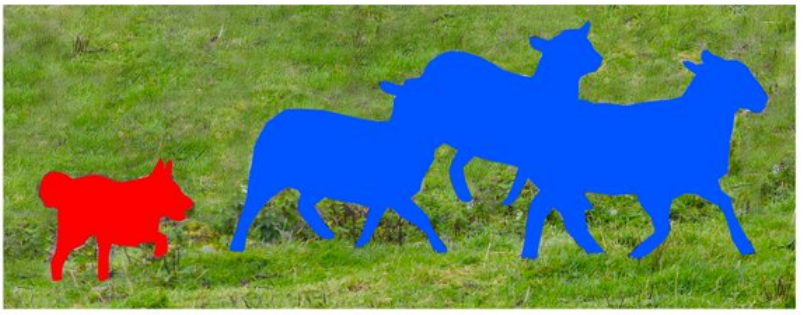


Figura 3.2: Segmentación semántica

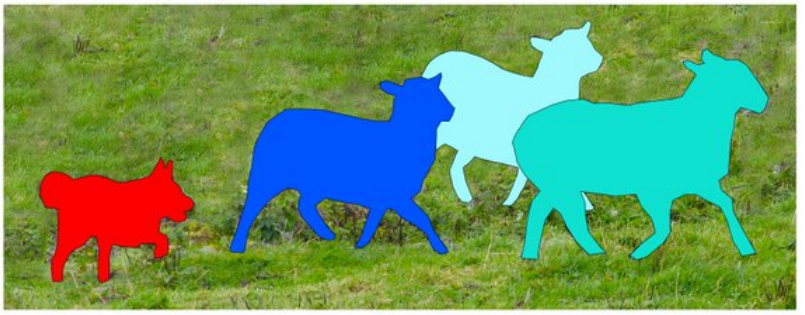


Figura 3.3: Segmentación de instancias

semántica y la segmentación de instancias que se distinguen en la manera de identificar los objetos. La segmentación semántica no distingue instancias de la misma clase y la segmentación de instancias sí. Esto lo vemos en las figuras 3.2 y 3.3.

Una vez explicados los algoritmos que utilizaremos para construir los distintos modelos de visión por computador, pasamos ahora a explicar el flujo de trabajo.

3.2. Flujo de trabajo

Para construir tanto los modelos de detección como los de segmentación, seguiremos el esquema que aparece en la figura 3.4. En él, diferenciamos los siguientes pasos que son

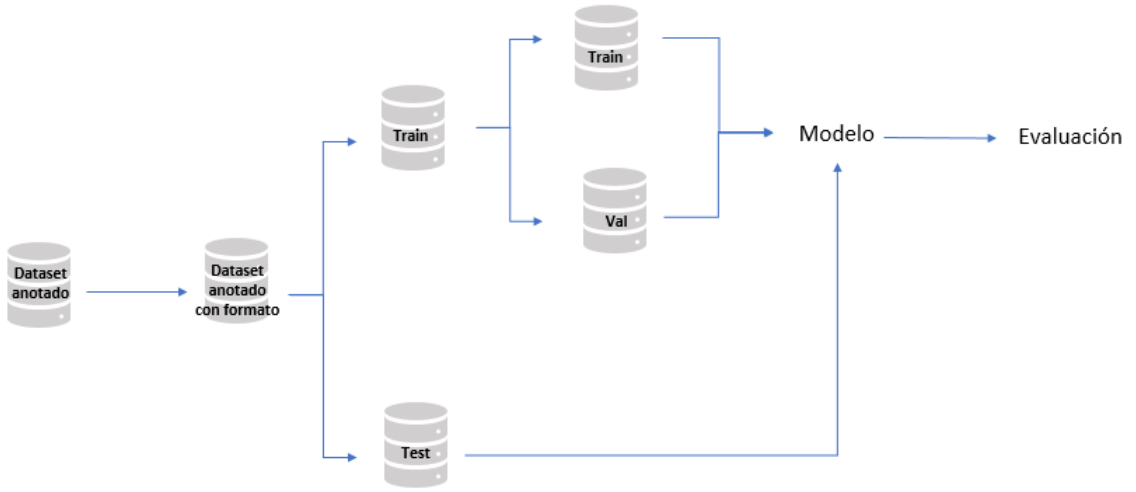


Figura 3.4: Flujo de trabajo

comunes para cualquier proyecto de estas características.

El primer paso es conseguir un dataset anotado y apropiado para la tarea que queremos llevar a cabo. Según el modelo o la librería que utilicemos tendremos que transformar las anotaciones del dataset a un formato que el modelo entienda. Una vez ya con el dataset preparado, lo dividimos en dos conjuntos: el conjunto de entrenamiento y el conjunto de test. Esto nos permitirá entrenar y evaluar cómo de bien lleva a cabo la tarea nuestro modelo. Para ajustar los hiperparámetros de los modelos se suelen tomar unas pocas instancias del conjunto de entrenamiento (entre un 10 y un 30 %). Le proporcionamos al algoritmo el conjunto de entrenamiento y el de validación, y, una vez que el modelo haya sido entrenado, le pasamos el conjunto de test para evaluar y obtener los resultados. Vemos a continuación el detalle de los pasos que acabamos de listar para nuestro caso concreto.

3.2.1. Data preparation

En este apartado veremos todos los pasos que hay que seguir para preparar el dataset de forma que el modelo sea capaz de entenderlo.

Dataset anotado

Este paso consiste en obtener un dataset adecuado para la tarea que queremos desarrollar. En el caso en el que no estuviera anotado, tendríamos que producirlas nosotros. Como ya hemos visto en el capítulo 2, nuestro dataset es el de FUNSD que está completamente anotado.

Anotaciones con formato

El siguiente paso es generar anotaciones en un formato adecuado. Este formato depende de la librería que utilicemos. Para la detección usaremos las librerías IceVision [11] y Darknet [12]. Para la primera no hace falta cambiar el formato de las anotaciones, simplemente tendremos que construir una clase, llamada `Parser`, en la que se especifique el *bounding box* de cada objeto y su etiqueta. Para la librería de Darknet las anotaciones son archivos *.txt* en los que cada línea de texto se corresponde con un objeto. Cada línea está formada por la etiqueta del objeto y sus coordenadas. El formato de estas coordenadas es distinto al que aparece en las anotaciones de FUNSD o el utilizado en la librería de IceVision, en estos casos, las coordenadas de cada *bounding box* venían dados por 4 valores: *xmin, ymin, xmax, ymax* (representando las coordenadas de la esquina superior izquierda y de la esquina inferior derecha respectivamente). En Darknet, estos valores son distintos: centro del objeto en X, centro del objeto en Y, ancho del objeto en X y ancho del objeto en Y. Por último, para la segmentación, utilizaremos las librerías SemTorch [13] y FastAI [14], que también requieren que las anotaciones tengan un formato especial. En este caso, las anotaciones son imágenes en escala de grises y cada clase tiene asociado un nivel de pixel, entre 0 y 255. Por lo tanto, por cada *bounding box* pintaremos un rectángulo con el nivel de píxel marcado por la clase del objeto.

Separación del dataset

Separamos el dataset en dos conjuntos: el de entrenamiento y el de test. El conjunto de entrenamiento consta de 149 imágenes, esto supone un 75 % sobre el total. Por lo que el conjunto de test, con 50 imágenes supone el 25 %. Esta partición es la proporcionada en [1]. A su vez, del conjunto de entrenamiento tomaremos un porcentaje pequeño de datos, un 20 % que se corresponde con 30 imágenes, para validar. Este conjunto de validación nos será útil para la búsqueda y elección de hiperparámetros y para el entrenamiento de los distintos modelos.

Notar que nuestro dataset de entrenamiento consta solo de 149 imágenes, lo que no suele ser suficiente para entrenar modelos de *deep learning*, por lo que, vamos a utilizar una técnica llamada *data augmentation*.

Data augmentation

El *data augmentation* [15] es una técnica que consiste en aplicar transformaciones en las imágenes para aumentar el número de instancias. En nuestro caso es necesario ya que nuestro conjunto de entrenamiento posee solamente 149 imágenes y para que el modelo consiga aprender necesita muchas más. En general, se aplican recortes, flips, rotaciones y muchas otras. Sin embargo, muchas de estas transformaciones no tienen

sentido en nuestros documentos. Por ejemplo, no tiene sentido que un formulario esté dado la vuelta, por lo que aplicaremos cambios en el color y ruidos.

3.2.2. Modelos

Una vez que tenemos una separación del dataset adecuada, utilizamos los conjuntos de entrenamiento y de validación para construir y entrenar distintos modelos. Los modelos que hemos utilizado para la tarea de detección son: de la librería de IceVision, Faster R-CNN y EfficientDet; y de la librería de Darknet, YOLOv4. Los modelos utilizados para realizar la segmentación semántica pertenecen a las librerías SemTorch y FastAI y son: DeepLab, HRNet-Seg y U-Net.

Modelos de detección

A continuación damos una pequeña explicación de los modelos de detección elegidos.

- **Faster R-CNN** [10] es un modelo de *deep learning* especializado en detección de objetos en imágenes. Es una versión mejorada de los modelos R-CNN [16] y Fast R-CNN [17]. Todos ellos son modelos de dos fases: la primera se encarga de generar regiones de interés a través de una búsqueda selectiva y en la segunda se clasifican dichas regiones.
- **EfficientDet** [18] es una red convolucional que surge de la mejora de la red de *EfficientNet* [19]. Fue creada por el equipo de Google Brain y ha conseguido la mejor *accuracy* con muy poco entrenamiento en tareas de detección de objetos.
- **YOLO** [10] o *"You Only Look Once"* es una familia de algoritmos muy popular para el reconocimiento de objetos. Son mucho más rápidos que los modelos de R-CNN. Se trata de una única red neuronal entrenada (de principio a fin) que toma una imagen de entrada y predice los *bounding boxes* y la clase de cada uno de ellos directamente. La imagen de entrada se divide en una cuadrícula de celdas donde cada celda es responsable de predecir un *bounding box* si el centro de un *bounding box* cae en él.

Modelos de segmentación

Pasamos ahora a explicar los modelos de segmentación semántica que vamos a utilizar.

- **U-Net** [20] es una red neuronal convolucional que proporciona una segmentación rápida y precisa de imágenes. La arquitectura U-Net consta de dos caminos: el codificador y el decodificador. El codificador, extrae las características que tienen información sobre lo que hay en la imagen, es decir, el contexto de la imagen. Esto lo hace utilizando capas convolucionales y capas de *max pooling*. El decodificador

o camino de expansión, aumentan progresivamente el tamaño de la imagen, para ello se utilizan capas de deconvolución.

Con el proceso de decodificación, se pierden algunas características que aprendió el codificador y es por eso por lo que la arquitectura U-Net tiene *skip connections* [20]. Con ellas se busca mejorar el detalle de la segmentación, y conseguir formas y bordes mucho más precisos. Estas conexiones consisten en que las salidas de las capas de codificación se pasan directamente a las capas de decodificación para que se puedan preservar todos los datos importantes.

- **DeepLab** [21] es una arquitectura general de segmentación semántica que puede considerarse, a grandes rasgos, como una red que codifica seguida de una red que decodifica. Hemos visto que la arquitectura U-Net utiliza capas de deconvolución para la decodificación, sin embargo, DeepLab utiliza capas de convolución *"atrous"*, estas permiten ampliar eficazmente el campo de visión de los filtros sin aumentar el número de parámetros ni la cantidad de cálculos.
- **HRNet-Seg** [22] surge de la necesidad de una red capaz de realizar tareas sensibles a la posición. Esto lo consigue manteniendo las representaciones de alta resolución a través de todo el proceso a diferencia de otras redes convolucionales que las representaciones de alta resolución se recuperan de las codificaciones de las representaciones de baja resolución.

Entrenamiento de modelos

Para entrenar estos modelos hemos utilizado algunas técnicas que nos ayudan a obtener mejores resultados en el entrenamiento. Entre ellas está el *transfer learning*, la búsqueda del *learning rate* y los *callbacks*.

El *transfer learning* [23] es un método muy común para cualquier tarea de visión por computador que nos permite crear modelos precisos en muy poco tiempo. Es impensable entrenar un modelo de *deep learning* desde cero, a no ser que se tenga muchos datos y abundantes recursos computacionales, y por eso, gracias al *transfer learning*, se parte de patrones que se han aprendido al resolver un problema diferente. De este modo, se aprovechan los aprendizajes anteriores y se evita empezar desde cero. En la visión por computador, el *transfer learning* suele expresarse mediante el uso de modelos preentrenados. Un modelo preentrenado es un modelo que fue entrenado con un gran dataset para resolver un problema similar al que queremos resolver. Como el coste computacional de entrenar estos modelos es muy alto, el *transfer learning* es una práctica muy común.

Antes de entrenar cualquier modelo es necesario fijar el *learning rate*, en español, el ratio

de aprendizaje. Se trata de un hiperparámetro que indica cuánto de rápido se modifican los pesos. Para determinar ese valor, utilizamos la función del `lr-finder` que calcula la función de pérdida para un intervalo de valores. De esta manera seremos capaces de seleccionar el valor más apropiado para este hiperparámetro [24]. Esta técnica sólo se aplicará a los modelos de IceVision y FastAI. En los modelos de YOLOv4 se aplicará el *learning rate* por defecto ya que la librería Darknet no proporciona dicha funcionalidad.

Por último, los *callbacks* son una herramienta muy poderosa que sirve para personalizar el comportamiento de un modelo durante el entrenamiento. Entre ellos está el `EarlyStopping`, que finaliza el entrenamiento si el mAP del modelo no aumenta en n épocas, diremos entonces que tiene una paciencia de n épocas; `SaveModel` guarda el modelo que obtenga el mayor mAP durante el entrenamiento y `ReduceLROnPlateau` que reduce el *learning rate* si en n épocas no hay mejora, es decir si el mAP obtenido en el conjunto de validación no amenta. Estas tres técnicas son muy útiles ya que nos permiten mejorar el entrenamiento. De nuevo, estas técnicas han sido utilizadas para los modelos de IceVision y FastAI ya que en Darknet no están disponibles.

Para el entrenamiento de los modelos de IceVision y FastAI hemos utilizado Google Colab [25]. Se trata de un servicio cloud que permite ejecutar y programar en el navegador con Python. Una de sus grandes ventajas es que da acceso gratuito a GPUs a cualquier usuario con una cuenta de Google. La asignación del modelo de GPU es aleatoria pero cualquiera de ellas es suficiente para entrenar nuestros modelos. El único inconveniente es que el entorno de ejecución de Colab se reinicia cada 12h y este tiempo no es suficiente para entrenar el modelo de YOLOv4. Por lo tanto, para entrenar este último modelo, hemos hecho uso de una GPU Nvidia RTX 2080 Ti del servidor de Simba de la Universidad de La Rioja.

Los modelos de IceVision fueron entrenados durante 50 épocas y usando un *early stopping* con una paciencia de 3 épocas. Repetimos este proceso varias veces, buscando cada vez un *learning rate* adecuado. Para el modelo de YOLOv4 se utilizaron los hiperparámetros por defecto, por lo tanto, el número de épocas alcanzadas fueron 4000. Los modelos de segmentación se entrenaron durante 30 épocas aplicando *early stopping* con una paciencia de 3 épocas.

3.2.3. Resultados

Una vez entrenados los modelos, el último paso es la evaluación. Tomamos los modelos y pasamos el conjunto de test a través de ellos. Utilizaremos las métricas vistas en la sección 2.2 para la evaluación: la *Precision*, el *Recall*, el *F1-score* y el *mAP*. En la tabla 3.1 encontramos los resultados para la tarea de detección.

	Precision	Recall	F1-score	mAP
Faster R-CNN	0.16	0.14	0.15	12.5
EfficientDet	0.09	0.01	0.01	4.83
YOLOv4	**0.7**	**0.72**	**0.71**	**60.19**
DeepLab	0.36	0.42	0.39	17.99
HRNet-Seg	0.43	0.48	0.45	20.44
U-Net	0.27	0.43	0.33	13.57
BROS	0.81	0.82	0.81	-
LayoutLM	0.76	0.81	0.79	-
LayoutLMv2	0.83	0.85	0.84	-

Tabla 3.1: Evaluación en la detección

Vemos que el modelo que obtiene los mejores resultado es el de YOLOv4 con un F1-score de 0.71. Los siguientes son los modelos de segmentación, que aunque no consigan acercarse a los resultados de YOLO, consiguen un F1-score entre 0.33 y 0.45. Faster R-CNN y EfficientDet se quedan muy atrás con un F1-score de 0.15 y 0.01 respectivamente. A pesar de que los resultados de nuestro modelo de YOLO son buenos, no consiguen superar a los modelos de otros trabajos anteriores como BROS o LayoutLMv2. Estos obtenían un F1-score de 0.81 y 0.84 respectivamente. El punto a a favor de nuestros modelos es que no han necesitado ser preentrenados con 11 millones de páginas de documentos.

Estos resultados pertenecen a la tarea de detección que, si recordamos, consistía en localizar los bloques de texto y etiquetarlos. Por lo tanto, estos modelos están etiquetando basándose simplemente en los aspectos gráficos (la disposición en el documento, en el tamaño, etc). Si consideramos simplemente la localización, resulta evidente que obtendremos mejores resultados. Estos resultados los encontramos en la tabla 3.2.

	Precision	Recall	F1-score	mAP
Faster R-CNN	0.73	0.63	0.68	13.05
EfficientDet	0.76	0.06	0.11	2.27
YOLOv4	**0.80**	**0.81**	**0.80**	**63.05**
DeepLav	0.76	0.55	0.64	11.27
HRNet	0.75	0.53	0.62	11.19
U-Net	0.74	0.54	0.63	11.10

Tabla 3.2: Evaluación en la localización

Como era de esperar, los resultados de la localización mejoran con respecto a los de detección. De nuevo, el modelo con mejores resultados es el de YOLOv4 con un F1-score de 0.8. No obstante, Faster R-CNN adelanta a los modelos de segmentación consiguiendo un F1-score de 0.68 aunque no por mucho ya que estos modelos obtienen un F1-score entre 0.62 y 0.64. EfficientDet, que obtiene un F1-score de 0.11, continúa siendo el modelo con peores resultados.

Observando los resultados tanto de la detección como los de la localización vemos que el etiquetado que realizan los modelos puede mejorarse. Lo que nos hace buscar otras soluciones para realizar el etiquetado. En concreto, en el próximo capítulo emplearemos modelos de clasificación de texto. Pero antes de pasar a ello vamos a presentar algunos resultados cualitativos de nuestros modelos de detección.

3.3. Predicciones

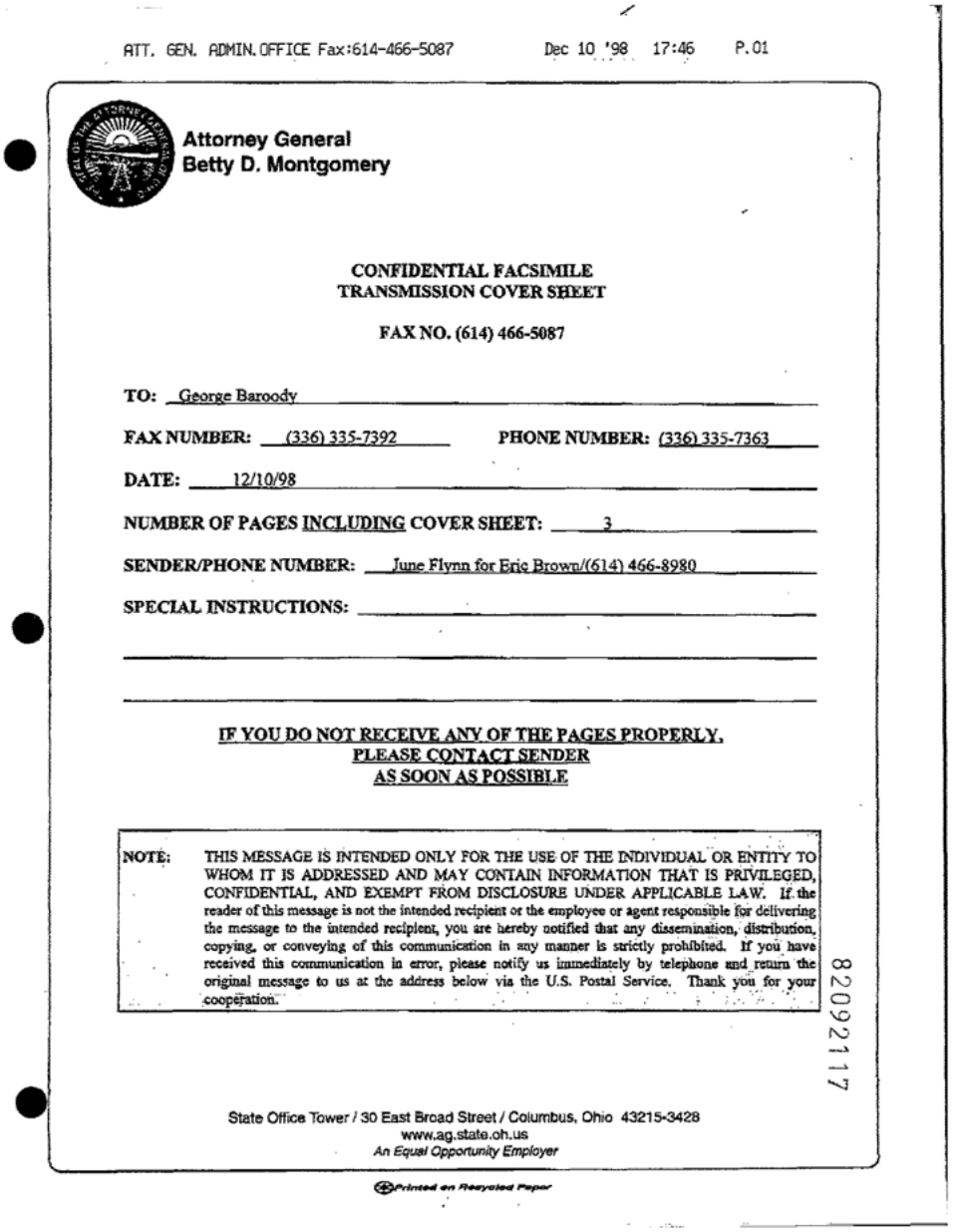
En esta sección utilizaremos los modelos entrenados para generar predicciones. Tomamos una imagen del conjunto de test que, para nuestros modelos, se trata de una imagen vista por primera vez. En la figura 3.5 se muestra la imagen seleccionada.

Pasamos dicha imagen por nuestros modelos entrenados. Las predicciones generadas aparecen en la figura 3.6. En ella encontramos lo siguiente:

- Predicción de Faster R-CNN (figura 3.6a). En ella observamos los distintos *bounding boxes* de colores. Cada uno tiene una pequeña etiqueta que los clasifica en una de las cuatro clases. Para simplificar, se ha utilizado la *Q* para la clase *Question*, la *A* para *Answer*, la *H* para *Header* y *o* para *Other*. Si nos fijamos en la localización, vemos que casi todos los objetos están identificados. Cuando el modelo intenta clasificarlos vemos que hay errores. Por ejemplo, en la parte superior derecha clasifica los bloques de texto como pregunta.

- Predicción de EfficientDet (figura 3.6b). Vemos que solo ha sido capaz de detectar dos entidades, sin embargo, la clasificación de las dos es perfecta. Esta predicción concuerda con los resultados obtenidos. Con el modelo de EfficientDet obteníamos un 0.06 en *Recall* y un 0.76 de *Precision*, es decir, que detectamos sólamente un 6 % de las entidades, no obstante, este 6 % es correcto un 76 % de las ocasiones.

- Predicción de YOLOv4 (figura 3.6c). Al igual que la predicción de Faster R-CNN y EfficientDet, consiste en un conjunto de *bounding boxes* con etiquetas. En la etiqueta también aparecen unos números que se corresponden con el nivel de confianza con el que se detecta dicho objeto. La predicción que hace el modelo de YOLO en este formulario es bastante buena.


ATT. GEN. ADMIN. OFFICE Fax:614-466-5087 Dec 10 '98 17:46 P.01

Attorney General
Betty D. Montgomery

CONFIDENTIAL FACSIMILE
TRANSMISSION COVER SHEET

FAX NO. (614) 466-5087

TO: George Baroody

FAX NUMBER: (336) 335-7392 PHONE NUMBER: (336) 335-7363

DATE: 12/10/98

NUMBER OF PAGES INCLUDING COVER SHEET: 3

SENDER/PHONE NUMBER: June Flynn for Eric Brown/(614) 466-8980

SPECIAL INSTRUCTIONS:

IF YOU DO NOT RECEIVE ANY OF THE PAGES PROPERLY,
PLEASE CONTACT SENDER
AS SOON AS POSSIBLE

NOTE: THIS MESSAGE IS INTENDED ONLY FOR THE USE OF THE INDIVIDUAL OR ENTITY TO WHOM IT IS ADDRESSED AND MAY CONTAIN INFORMATION THAT IS PRIVILEGED, CONFIDENTIAL, AND EXEMPT FROM DISCLOSURE UNDER APPLICABLE LAW. If the reader of this message is not the intended recipient or the employee or agent responsible for delivering the message to the intended recipient, you are hereby notified that any dissemination, distribution, copying, or conveying of this communication in any manner is strictly prohibited. If you have received this communication in error, please notify us immediately by telephone and return the original message to us at the address below via the U.S. Postal Service. Thank you for your cooperation.

82092117

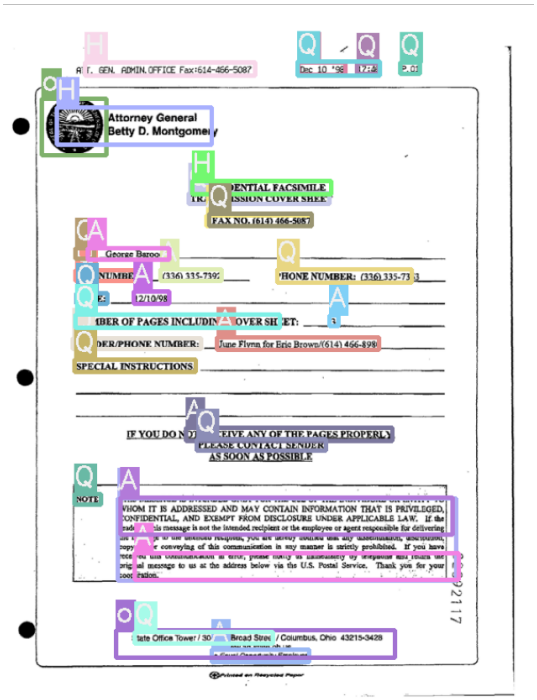
State Office Tower / 30 East Broad Street / Columbus, Ohio 43215-3428
www.ag.state.oh.us
An Equal Opportunity Employer

Printed on Recycled Paper

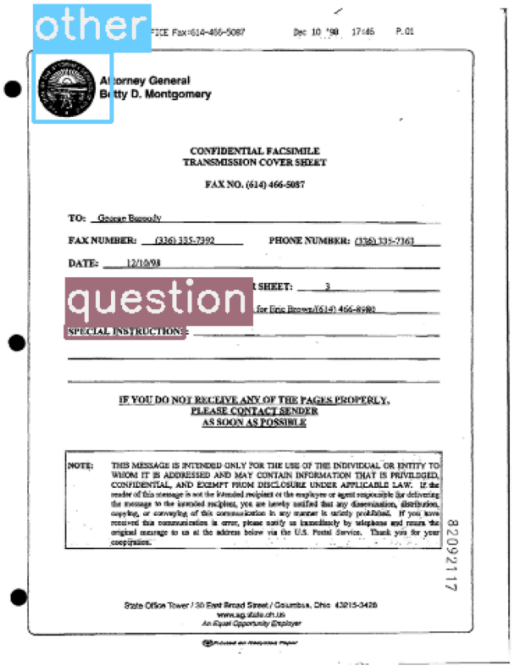
Figura 3.5: Formulario

- Predicción de DeepLab (figura 3.6d). Como ya hemos explicado anteriormente, los modelos de segmentación hacen predicciones a nivel de píxel, y esa es la razón por la que en esta predicción no encontramos *bounding boxes*. Los colores marcan la etiqueta de los píxeles. El verde se corresponde con la etiqueta *question*, el rojo con *answer*, el azul con *header* y el gris con *other*. Aunque para esta tarea no necesitemos una detección a nivel de pixel, este modelo es capaz de generalizar y ofrecer predicciones útiles.

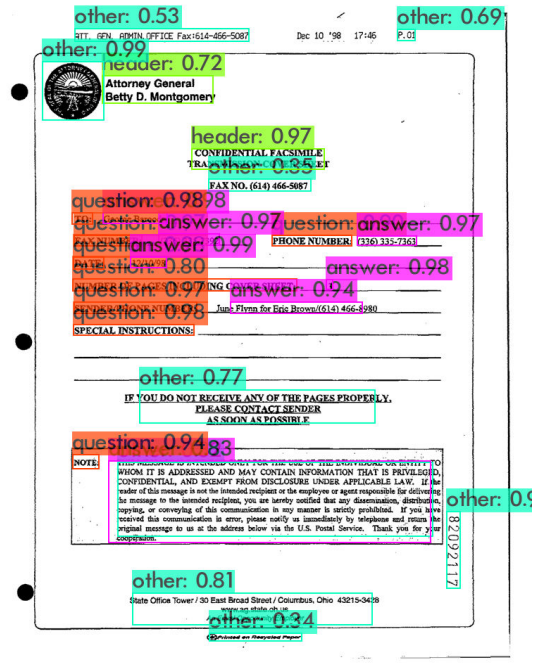
- Predicción de HRNet (figura 3.6e). Las etiquetas se corresponden con los mismos colores que con las del modelo de DeepLab. Hace una predicción bastante parecida a la anterior.
- Predicción de U-Net (figura 3.6f). Al igual que los otros dos modelos de segmen-



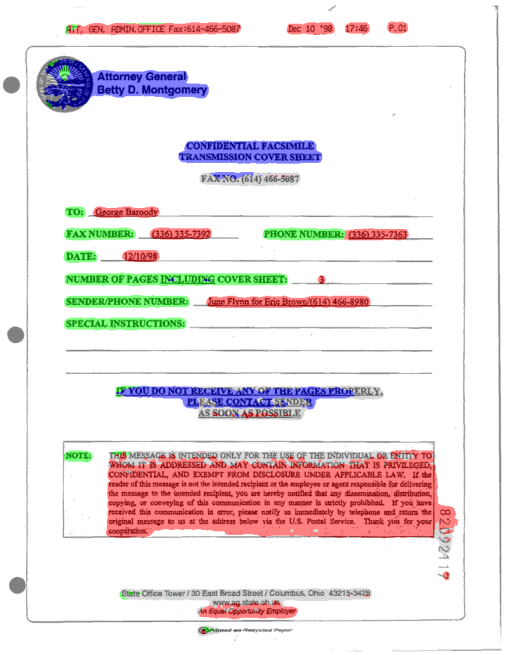
(a) Predicción Faster R-CNN



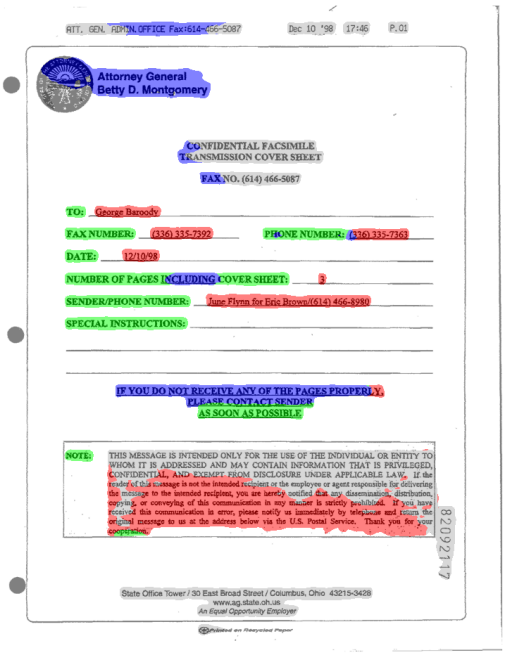
(b) Predicción EfficientDet



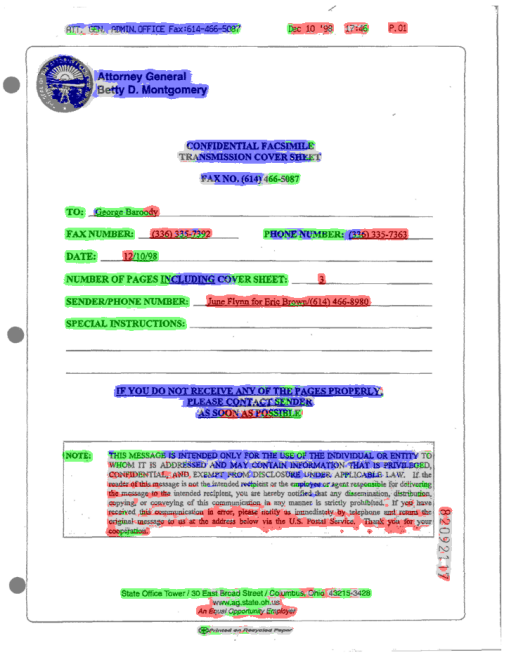
(c) Predicción YOLO



(d) Predicción DeepLab



(e) Predicción HRNet-Seg



(f) Predicción U-Net

Figura 3.6: Predicciones de los modelos de detección

tación, los colores representan las mismas etiquetas. En este caso, U-Net hace una predicción un poco peor que DeepLab y HRNet.

3.4. Conclusiones

Hemos visto como a partir de distintos modelos de visión por computador somos capaces de detectar los bloques de texto de formularios. Sin embargo, el modelo de YOLO es el único capaz de generalizar y proporcionar resultados aceptables. Cuando consideramos solamente la tarea de localización vemos que los resultados mejoran considerablemente. Esto se debe a que los modelos de visión por computador solo se fijan en la forma y no en el contenido, por lo que cuando clasifican los distintos bloques de texto, solo tienen en cuenta el aspecto gráfico. En el capítulo siguiente trataremos de utilizar la información que nos proporciona el texto para mejorar la clasificación. Transformamos entonces nuestro problema en uno de localización y de clasificación de texto. Con los modelos de visión por computador obtendremos los *bounding boxes* y aplicando alguna técnica de reconocimiento de caracteres extraeremos el texto y utilizaremos un modelo de clasificación de texto para obtener la etiqueta de dicho bloque de texto.

Capítulo 4

Clasificación de texto

En el capítulo anterior vimos que eramos capaces de localizar correctamente los distintos bloques de texto dentro de los formularios. Sin embargo, al sumar a la localización la tarea de clasificación (es decir, la tarea de detección), los resultados empeoraban drásticamente. Y es que, para clasificar un bloque de texto no solo necesitamos fijarnos en su aspecto gráfico o en su disposición en la hoja, si no también en su contenido. De esta manera, parece adecuado utilizar modelos de clasificación de texto. Por lo tanto, necesitaremos extraer el texto de la imagen. Esto lo conseguimos utilizando herramientas de reconocimiento de caracteres, como el OCR (*Optical Character Recognition*). Aplicaremos el OCR a cada *bounding box* para obtener su contenido y pasaremos cada uno de los fragmentos por nuestro modelo de clasificación de texto.

En este capítulo veremos una introducción a la clasificación de texto y los pasos seguidos para construir los distintos modelos junto con sus evaluaciones. Para finalizar, veremos unas conclusiones.

4.1. ¿Qué es la clasificación de texto?

La clasificación de texto es una tarea del procesado natural del lenguaje que consiste en categorizar un texto a partir de un conjunto de clases predefinidas [26]. Tiene infinidad de aplicaciones y lo encontramos en nuestro día a día. Por ejemplo, se utiliza para identificar los correos electrónicos que son spam, categorizar noticias, o para hacer un análisis de sentimiento en comentarios [26]. Hoy en día, esta tarea se realiza principalmente a través de modelos de *deep learning* [26]. Nosotros lo utilizaremos para categorizar cada bloque de texto que aparezca en el documento como *pregunta, respuesta, encabezado* u *otro*. A continuación, veremos los pasos que hemos tomado en la construcción del modelo.

4.2. Flujo de trabajo

En esta sección veremos como hemos construido el modelo de clasificación de texto. En la figura 4.1 vemos los pasos que hemos dado para construir dicho clasificador.

Aprovechamos las anotaciones de texto de cada una de las entidades del dataset de FUNSD para crear nuestro dataset anotado. Dividimos el dataset de la misma manera que lo hicimos para el modelo de detección. Lo siguiente que hacemos es construir un *tokenizador*, que transforma el texto en un vector de números para que el modelo lo entienda. Por último entrenamos el modelo y utilizamos el conjunto de test para evaluar. Describimos con mayor detalle cada fase a continuación.

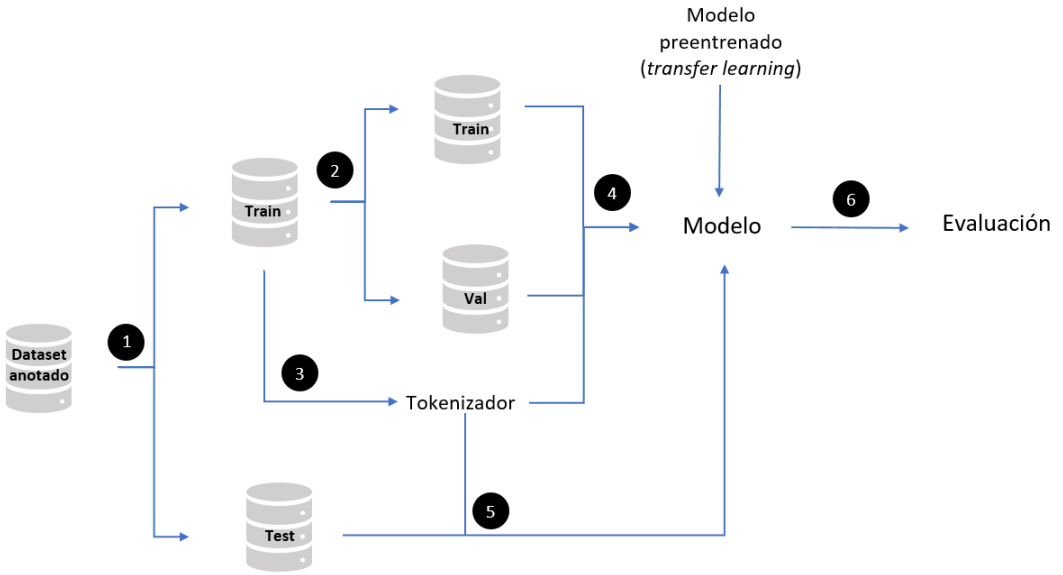


Figura 4.1: Flujo de trabajo para la clasificación

4.2.1. Data preparation

Como ya vimos, este paso consiste en la preparación del dataset de tal manera que el modelo lo entienda y pueda entrenar con él.

Dataset anotado

Como hemos visto en el capítulo 2, el dataset de FUNSD posee las anotaciones de texto de cada una de las entidades que aparecen en los 199 documentos. Utilizaremos dichas anotaciones para construir el dataset anotado.

Separación del dataset

Dividimos el dataset en un conjunto de entrenamiento y otro de test. Utilizaremos la división que nos proporciona el propio dataset de FUNSD (75-25 %). Además, el conjunto de entrenamiento lo subdividiremos en dos grupos como hicimos en la detección (80-20 %). Con esto conseguiremos tener un conjunto de validación y conseguir mejores resultados en el entrenamiento.

4.2.2. Tokenizador

El siguiente paso para nuestro modelo de clasificación de texto es construir un *tokenizador*. Este se encarga de transformar cualquier cadena de texto sin procesar a una representación comprensible para el modelo. Este proceso no es más que un preprocesamiento de los datos denominado *tokenización* y consiste en codificar el texto en *tokens*. Existen distintas codificaciones. Una muy simple consiste en dividir el texto en las palabras que la componen y asignar un identificador a cada palabra. Este tipo de codificación funciona bien cuando el vocabulario es pequeño. El inconveniente de esta *tokenización* es que palabras similares, como "gato" y "gatos", obtienen identificadores diferentes.

Para solventar este problema se han desarrollado técnicas de subtokenización en las que se amplía la estrategia de división de manera que se descomponga cada palabra en subcomponentes gramaticales aprendidos de los datos [27]. De esta forma, tomando el ejemplo anterior de "gato" y "gatos", la subtokenización de esta última sería: [gato, ##s] donde ## indica un subtoken de la palabra inicial.

Nuestro tokenizador se basa en esta segunda técnica, pasamos los conjuntos de entrenamiento y validación a través de él y así transformarlos en entradas con una representación comprensible para nuestro modelo.

4.2.3. Modelos

A continuación veremos las técnicas utilizadas y los pasos que hemos seguido para entrenar los distintos modelos.

Transfer learning

En este punto, lo único que nos queda por construir es el modelo de clasificación. Al igual que en los modelos de detección y segmentación, en este tipo de tareas también hace falta utilizar el *transfer learning*. Esto se debe a que la tarea es complicada y entrenar un modelo desde cero supondría un coste muy elevado en recursos y en tiempo.

Para aplicar esta técnica será necesario usar un modelo de lenguaje [28]. Se trata de un modelo encargado de predecir la siguiente palabra de un texto a partir de las que han aparecido antes. A este tipo de tareas se las conoce como *self-supervised learning* [28] ya que no es necesario proporcionar etiquetas a nuestro modelo, sino simplemente proporcionarle grandes cantidades de texto, por ejemplo, la Wikipedia. Este proceso obtiene las etiquetas de manera automática a partir de los datos. Es una tarea que está lejos de ser trivial ya que adivinar la siguiente palabra de un texto requiere una comprensión del lenguaje.

Por lo tanto, para construir nuestro clasificador de texto primero buscaremos un modelo de lenguaje entrenado con un dataset de tamaño considerable. Utilizaremos los modelos siguientes: Albert [29], BERT [30], DistilBert [31], Roberta [32] y ULMFit [33]. Los modelos de Albert, BERT y DistilBert están entrenados con la Wikipedia y los 11,038 libros inéditos que componen el dataset de BookCorpus [29, 30, 31]. Roberta también está entrenado con estos dos dataset además del dataset CC-News [32], que contiene 63 millones de artículos en inglés, y otros dos grandes datasets. El modelo de ULMFit está entrenado solo con la Wikipedia [33]. Estos modelos sirven para conocer los fundamentos del lenguaje con el cual se está trabajando.

Entrenamiento de modelos

A la hora de construir un modelo de clasificación es conveniente que el modelo comprenda el estilo que se usa para escribir esos textos. En nuestro caso nombres de empresas, nombres propios, fechas, etc. Es en este paso en el que a partir de los modelos anteriores entrenamos un modelo de lenguaje para FUNSD. A continuación, reemplazamos la última capa de nuestro modelo de lenguaje de FUNSD para convertirlo en un clasificador adaptando dicha capa al número de clases de nuestro dataset. Por último, entrenamos dicho modelo durante 5 épocas con los hiperparámetros por defecto que nos da la librería. La razón de las 5 épocas se debe a que a partir de la sexta el modelo se sobreajustaba. De nuevo, volvemos a utilizar las GPUs que nos presta Google Colab.

El *transfer learning* visto es el habitual para los modelos de clasificación de texto. No obstante, también hemos probado una técnica distinta de *transfer learning*: utilizar la última capa del modelo de BERT adaptado a FUNSD como extractor de características y pasárselo a 7 algoritmos distintos de machine learning para realizar la clasificación (KNN, arbol de decisión, SVM, Naive Bayes, random forest, Redes neuronales y regresión logística).

4.2.4. Resultados

Por último, pasando el conjunto de test por el tokenizador y por el modelo obtenemos los resultados que encontramos en la tabla 4.1. Las métricas utilizadas son las habituales: *Accuracy*, *Precision*, *Recall* y *F1-score*. La única que todavía no conocemos es la *Accuracy*. Esta métrica es muy parecida al *Recall*, ya que se define como las predicciones correctas dividido por las instancias totales. La única diferencia es que con el *Recall* consideramos solamente las instancias de una clase y con la *Accuracy* consideramos todas las instancias. Al ser un problema multiclase y al no estar balanceado (hay más instancias de una clase que de otras) utilizaremos las métricas ponderadas. No es más que calcular dicha métrica para cada clase y hacer una media ponderada, siendo los pesos el porcentaje de instancias de cada clase sobre el total.

La única métrica que todavía no

	Accuracy	Precision	Recall	F1-score
Albert	0.78	0.78	0.78	0.78
BERT	**0.82**	**0.81**	**0.82**	**0.81**
DistilBert	0.72	0.71	0.72	0.71
Roberta	0.71	0.69	0.71	0.69
ULMFit	0.75	0.76	0.75	0.76
KNN	**0.80**	**0.81**	**0.80**	**0.81**
Arbol de decisión	0.79	0.79	0.79	0.79
SVM	0.80	0.80	0.80	0.80
Naive Bayes	0.79	0.79	0.79	0.79
Random Forest	0.79	0.80	0.79	0.79
Redes Neuronales	**0.81**	**0.81**	**0.81**	**0.81**
Regresión logística	**0.81**	**0.81**	**0.81**	**0.81**
BROS	-	0.80	0.81	0.81
LayoutLM	-	0.76	0.81	0.79
LayoutLMv2	-	0.83	0.85	0.84

Tabla 4.1: Evaluación de la clasificación de texto

En la tabla 4.1 observamos que el mejor modelo utilizando la técnica de *transfer learning* habitual es el de BERT con el que obtenemos un F1-score del 0.81. Todos los demás se quedan muy por debajo. Utilizando la segunda técnica igualamos este resultado con KNN, redes neuronales y regresión logística. Los demás obtienen un F1-score cercano a 0.8 superando así a los modelos de *deep learning*. La gran ventaja de los modelos de

machine learning y diferencia con los de *deep learning* es que no se precisa el uso de GPU.

Si ahora comparamos nuestros resultados con los que ya existen vemos que solo son superados por el modelo de LayoutLMv2 con un F1-score de 0.84. Si recordamos, estos modelos fueron preentrenados con 11 millones de páginas de documentos, por lo que, comparado con nuestros modelos que no requieren de ningún entrenamiento especial, la diferencia es mínima.

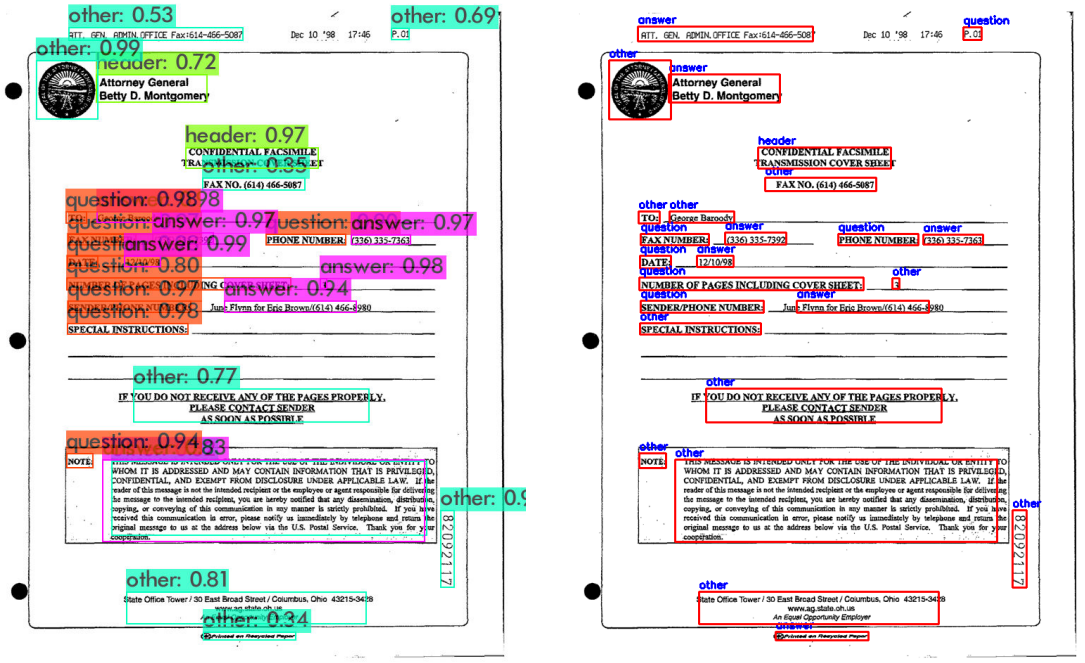
4.3. Predicciones

A lo largo del trabajo hemos construido distintos modelos de detección o de localización (ignorando las etiquetas de los objetos localizados) y clasificadores de texto. Es hora de entrelazar ambos modelos con el uso de alguna herramienta de reconocimiento de caracteres. Para ello utilizaremos el OCR (del inglés, *Optical Character Recognition*), esta herramienta es capaz de identificar automáticamente caracteres o símbolos a partir de una imagen. El OCR es una de las primeras tareas de visión por computador que se abordaron [34].

En esta sección utilizaremos el modelo entrenado de localización de YOLOv4, de esta manera, obtendremos los *bounding boxes* de cada documento. Haremos uso de la técnica *Non Max Suppresion* [35] que consiste en eliminar todos aquellos *bounding boxes* que estén duplicados o que tengan un IoU superior a un valor umbral respecto a otro rectángulo detectado. No obstante, esta técnica se centra en la coordenada de *ymin* sin importar cual de los rectángulos es más grande. Nosotros buscamos que el rectángulo sea el mayor, para hacer posteriormente una mejor lectura. Por lo tanto hemos programado una versión de este *Non Max Suppresion* basándonos en el área de los *bounding boxes*, quedándonos así con el que contenga a los demás. A continuación, obtenemos el texto contenido en dichos *bounding boxes* haciendo uso de la librería de Tesseract [36], un motor de OCR ya entrenado. Y, por último pasaremos el texto por el clasificador construido para obtener su etiqueta.

En la figura 4.2 encontramos las predicciones del mismo documento que utilizamos de ejemplo en el capítulo 3. En la figura 4.2b, encontramos la predicción que ofrece YOLOv4 con la clasificación de nuestro modelo BERT. A su derecha, figura 4.2a, encontramos la predicción que nos proporciona el propio modelo de YOLOv4. Podemos observar que aunque haya bastantes entidades bien clasificadas, no conseguimos mejorar la clasificación de YOLOv4.

Calculamos las métricas para así poder comparar el modelo de YOLOv4 con el modelo de localización de YOLOv4 y el clasificador BERT. Los resultados parecen recogidos en



(a) Predicción YOLOv4 (b) Predicción YOLOv4 + BERT

Figura 4.2: Predicción YOLOv4 vs YOLOv4+BERT

la tabla 4.2. Recordar que, en este último modelo, hemos utilizado la técnica del *Non Max Suppresion* por lo que los resultados de la localización también son distintos con respecto a los de YOLOv4. Vemos que en la localización obtenemos mejores resultados para el mAP, la *Precision* y el *F1-score*. Sin embargo, si consideramos también la clasificación, los resultados de este nuevo modelo son muchos peores que los que obteníamos solo con YOLOv4. Un 0.44 frente a un 0.71 de F1-score.

	Localización				Detección			
	Precision	Recall	F1-score	mAP	Precision	Recall	F1-score	mAP
YOLOv4 + BERT	**0.85**	**0.78**	**0.81**	**68.16**	0.49	0.40	0.44	24.11
YOLOv4	0.80	0.81	0.80	63.05	**0.7**	**0.72**	**0.71**	**60.19**

Tabla 4.2: Métricas modelo YOLOv4 + BERT

Analizamos el porqué de estos malos resultados. Una de las razones es que a veces el *bounding box* no recoge correctamente el bloque de texto dejando así trozos de texto fuera. A pesar de que hemos aumentado los márgenes de cada *bounding box* 3 píxeles a la izquierda y otros 3 a la derecha, la lectura posterior no es perfecta. La segunda razón tiene más peso y es que al tratarse de documentos antiguos, con baja resolución y con ruido, el OCR no es capaz de reconocer los caracteres correctamente por lo que cuando se proporciona el texto al modelo de BERT a veces dicho texto carece de sentido. Esto

lo analizamos con más detalle a continuación.

Tomando los *bounding boxes* de las anotaciones del dataset de FUNSD comparamos el texto anotado con el que obtenemos al hacer OCR. Para comparar el texto utilizamos una métrica de distancia de edición [37]. En teoría de la información, lingüística e informática, esta distancia es una métrica que sirve para medir la diferencia entre dos secuencias de caracteres. Informalmente, la distancia de edición entre dos palabras es el número mínimo de ediciones u operaciones de un solo carácter (inserciones, eliminaciones o sustituciones) necesarias para cambiar una palabra por la otra [37, 38]. En [39] encontramos una definición formal en la que a cada una de las operaciones se le asigna un coste. Entonces, la distancia de edición entre dos palabras x e y es el coste mínimo de transformar x en y.

En particular, utilizaremos la métrica *Levenshtein*, en la que a las tres operaciones se le asigna el mismo coste, 1, y se define de la siguiente manera:

La distancia *Levenshtein* [40] entre dos palabras x e y viene dada por *lev(x,y)* donde

$$lev(x,y) = \begin{cases} |x| & si \quad |y| = 0 \\ |y| & si \quad |x| = 0 \\ lev(cola(x), cola(y)) & si \quad x[0] = y[0] \\ 1 + min \begin{cases} lev(x, cola(y)) \\ lev(cola(x), y) \\ lev(cola(x), cola(y)) \end{cases} & otro\ caso \end{cases}$$

Siendo $|x|$ la longitud de la palabra x, la $cola(x)$ la palabra formada por todos los caracteres de x menos el primero y $x[0]$ es el primer carácter de la palabra x.

Existen otras menos comunes como la de *Hamming*, Alineación Óptima de cadenas u *OSA*, y la de *Damerau-Levenshtein* [38].

Veamos algunos ejemplos:

- La distancia Levenshtein entre "*formulario*" y "*fomulario*" es 1 ya que hace falta una operación de inserción para transformar la segunda en la primera. Se necesita introducir una "*r*" entre el segundo y tercer carácter de "*fomulario*".
- La distancia Levenshtein entre "*formulario*" y "*forrmulario*" es 1 ya que hace falta una operación de eliminación para transformar la segunda en la primera. Se

necesita eliminar una "*r*" entre el tercer y cuarto carácter de "*forrmulario*".

- La distancia Levenshtein entre "*formulario*" y "*farmulario*" es 1 ya que hace falta una operación de sustitución. Se necesita sustituir la "*a*" por la "*o*".

El número de errores cometidos en una palabra se corresponde con la distancia Levenshtein. Si dividimos la distancia de edición entre el número de palabras obtendremos la distancia normalizada, de esta manera podremos comparar el porcentaje de error entre palabras cortas con otras con mayor número de caracteres.

Por último, utilizaremos la similitud que es el inverso de la distancia normalizada. Este valor representa cuanto de parecido son las dos cadenas de texto siendo el máximo la unidad.

En general, el porcentaje de errores tipográficos varía entre el 1 % y el 3,2 % y los de ortografía entre el 1,5 % y el 2,5 % [37]. Sin embargo, este porcentaje aumenta bastante cuando se trata de colecciones de texto digitalizadas mediante reconocimiento óptico de caracteres (OCR), entre el 7 % al 16 % [37].

Calculamos a continuación las métricas al extraer el texto de los *bounding boxes*. Dichos resultados aparecen recogidos en la tabla 4.3.

	Distancia	Distancia normalizada	Similitud
OCR	5.39	0.40	0.60

Tabla 4.3: Métricas OCR

De la tabla de resultados observamos que el número medio de ediciones en cada bloque de texto es aproximadamente 5. Esto representa un porcentaje de error del 40 %, más del doble con respecto al error esperado. Lo que nos deja una similitud del 60 %, solo 6 de cada 10 caracteres son correctos. Estos resultados los obtenemos suponiendo que la localización del bloque texto es perfecta, así que, habría que sumar a este error la inexactitud de los modelos de localización.

4.4. Conclusiones

En este capítulo hemos visto que nuestros modelos de clasificación de texto no se quedan atrás con respecto a los tres modelos más competitivos para el dataset FUNSD. Es más, estos modelos requieren un preentrenamiento con datasets de grandes dimensiones además de un uso exhaustivo de la GPU. Para entrenar nuestros modelos hemos utilizado las GPUs que nos presta Google Colab y en los casos de los algoritmos de *machine learning* ni eso. Sin embargo, es importante remarcar que para conectar los modelos de

localización con el modelo de clasificación de texto hace falta buscar alguna solución al reconocimiento de caracteres. Ya que en las mejores condiciones solo somos capaces de extraer el 60 % del texto correctamente, y como hemos visto en el capítulo 3, nuestro mejor modelo de localización obtiene un F1-score de 0.80, es decir, es bueno pero no perfecto.

Capítulo 5

Otras alternativas

En este capítulo veremos otras alternativas que hemos considerado para llevar a cabo nuestra tarea. Entre ellas está la construcción de un modelo distinto para localizar las entidades y la preparación de un dataset más actual con facturas propias.

5.1. Modelo nuevo de localización

No siempre es necesario entrenar modelos de *deep learning* para las tareas de visión por computador. Es más, durante muchos años, el OCR no ha necesitado del aprendizaje profundo para poder extraer los caracteres de una imagen [34]. Y es que, muchas veces, haciendo uso simplemente de las propiedades de las imágenes podemos clasificarlas sin necesidad de estar largas horas entrenando un modelo y sin hacer ningún uso de GPUs.

A continuación introduciremos las técnicas utilizadas para más tarde ver el procedimiento y los resultados.

5.1.1. Técnicas empleadas

Lo primero que vamos a estudiar son los filtros morfológicos, que son técnicas de análisis de imágenes basadas en la geometría y en la forma [41]. Tienen muchas aplicaciones, entre ellas suavizar bordes, separar regiones unidas y unir regiones. Estos filtros se aplican a imágenes en blanco y negro, pero son generalizables a imágenes en escala de grises. Por lo tanto, nuestro primer paso será transformar la imagen a escala de grises.

Los filtros que vamos a utilizar son:

- Erosión: quita una capa de píxeles de una región a lo largo de sus bordes. Sirve para separar objetos

- Dilatación: añade una capa de píxeles alrededor de sus bordes y sirve para reparar roturas
- Apertura: consiste en aplicar la erosión y seguida una dilatación y sirve para eliminar ruido.
- Cierre: consiste en aplicar una dilatación primero y después una erosión. Sirve para eliminar huecos dentro de objetos o regiones.
- Blackhat: consiste en la diferencia entre el cierre y la imagen (en escala de grises o en blanco y negro). Con este filtro somos capaces de detectar regiones oscuras sobre fondos claros.

Además de filtros morfológicos, también se aplican filtros para la corrección de imágenes, la detección de bordes y una técnica llamada umbralización. Nuestro dataset está formado por documentos que tienen ruido y manchas de impresión por lo que conviene aplicar un filtro para suavizar la imagen y eliminar ruido y de esta manera intentar corregir la imagen. Con el filtro Sobel además de remarcar las regiones oscuras sobre fondo claro conseguimos obtener los bordes que tengan cambios verticales. En cuanto a la umbralización, convierte una imagen en escala de grises a una binaria.

Una vez vista una pequeña introducción a los filtros que utilizaremos, presentamos el procedimiento seguido para obtener los mejores resultados.

5.1.2. Procedimiento

El procedimiento consiste en aplicar los filtros vistos con el objetivo de localizar los bloques de texto. Utilizaremos la librería de OpenCV [42] en la que están implementados todos ellos.

A continuación expondremos los pasos que hay que seguir para conseguir los *bounding boxes* de los bloques de texto. En la figura 5.3 encontramos un ejemplo de cada uno de los pasos a seguir.

1. Transformar la imagen a escala de grises.
2. Aplicar un filtro para la corrección de imágenes (Gaussian blurr o el de la mediana).
3. Filtro blackhat para detectar regiones oscuras sobre fondo blanco.
4. Aplicar el filtro Sobel para detectar los bordes.
5. Operación de cierre para quitar los huecos entre las letras.

6. Erosionar la imagen con un kernel rectangular con el fin de quedarnos con las líneas verticales.

7. Eliminar de la imagen las líneas verticales de la imagen restando la imagen obtenida en el paso 6 con la del paso 7.

8. Umbralizar la imagen que queda utilizando el umbral calculado mediante el método de Otsu [42].

En la figura 5.1 encontramos los pasos del 1 al 4 y los pasos del 5 al 8 se corresponde con la figura 5.2. En el paso 8, figura 5.2d, nos quedan pequeñas regiones que se corresponden con los distintos bloques de texto. Para obtener las coordenadas de los contornos de los bloques utilizamos la función *findContours* de OpenCV. Dibujamos los rectángulos que nos proporciona dicha función en la figura 5.3a. Con esto tendríamos ya la localización de los objetos. Para obtener la clasificación de cada objeto localizado volvemos a hacer uso de la librería Tesseract para realizar el OCR en cada *bounding box* y así pasarlo por el clasificador. El resultado al usar el clasificador de BERT lo vemos en la figura 5.3b.

5.1.3. Resultados

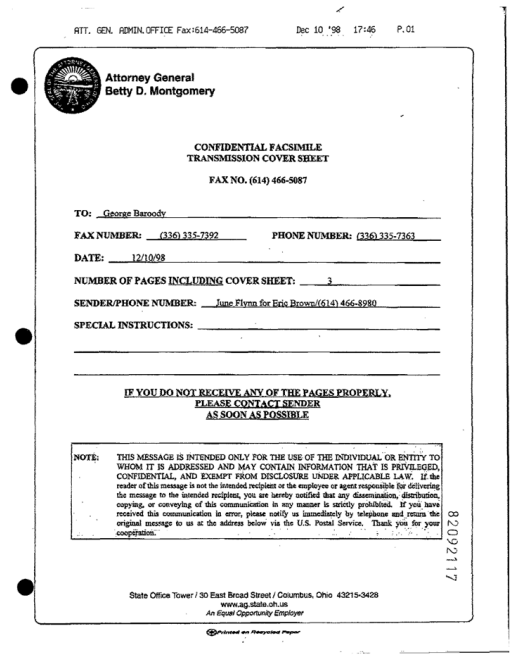
Una vez aplicado nuestro procedimiento al conjunto de test del dataset FUNSD, procedemos a calcular las mismas métricas que utilizamos en el capítulo 3, de esta manera, podremos comparar los resultados. En la tabla 5.1 aparecen recogidos los resultados de todos los modelos que he hemos visto hasta ahora.

	Localización				Detección			
	Precision	Recall	F1-score	mAP	Precision	Recall	F1-score	mAP
Procesamiento-img	0.67	0.63	0.65	43.60	0.41	0.34	0.37	17.69
Faster R-CNN	0.73	0.63	0.68	13.05	0.16	0.14	0.15	12.5
EfficientDet	0.76	0.06	0.11	2.27	0.09	0.01	0.01	4.83
YOLOv4	0.80	0.81	0.80	63.05	**0.7**	**0.72**	**0.71**	**60.19**
DeepLab	0.76	0.55	0.64	11.27	0.36	0.42	0.39	17.99
HRNet-Seg	0.75	0.53	0.62	11.19	0.43	0.48	0.45	20.44
U-Net	0.74	0.54	0.63	11.10	0.27	0.43	0.33	13.57
YOLOv4 + BERT	**0.85**	**0.78**	**0.81**	**68.16**	0.49	0.40	0.44	24.11
BROS	0.81	0.82	0.81	-	0.81	0.82	0.81	-
LayoutLM	0.76	0.81	0.79	-	0.76	0.81	0.79	-
LayoutLMv2	0.83	0.85	0.84	-	0.83	0.85	0.84	-

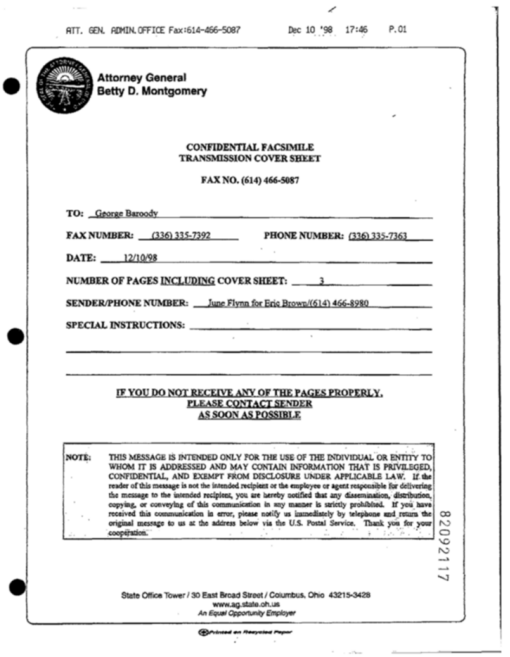
Tabla 5.1: Métricas resumen de todos los modelos

En cuanto a la localización, observamos que con el procesamiento de imágenes visto somos capaces de llevar a cabo esta tarea bastante bien. Este procedimiento se posiciona

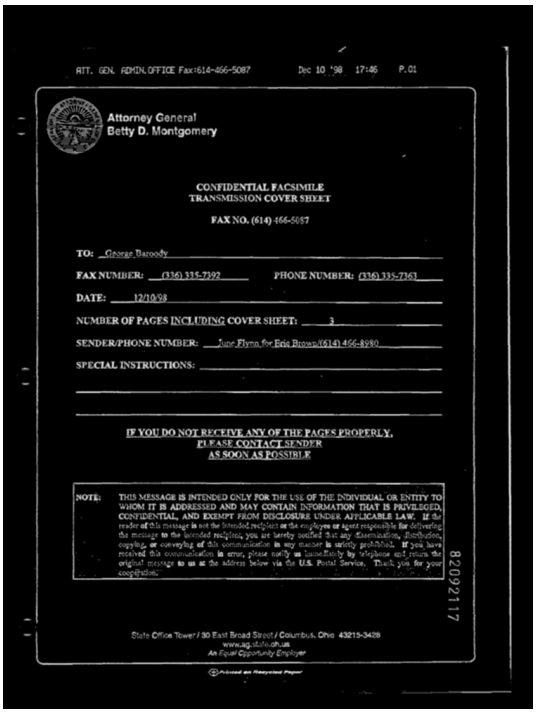
entre los tres mejores con un F1-score de 0.65. Se queda muy cerca del de Faster R-CNN, pero lejos del modelo ganador. Si consideramos ahora la detección, el F1-score desciende a 0.37 superando solamente a los modelos de EfficientDet, Faster R-CNN y



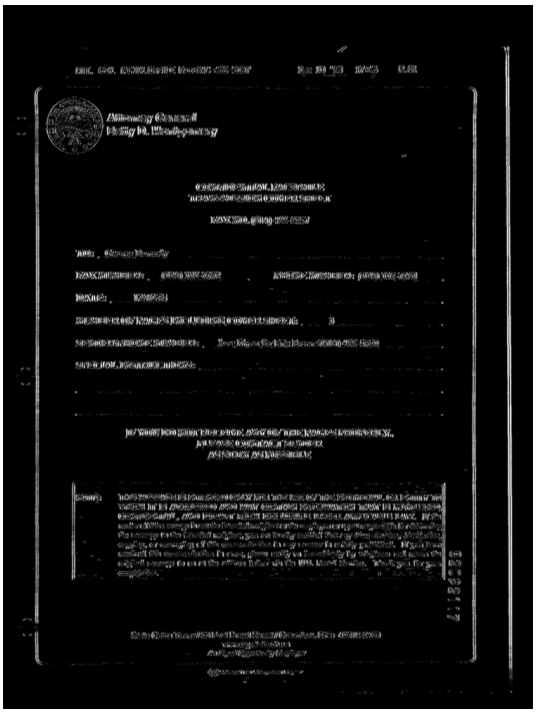
(a) Escala de grises



(b) Gaussian Blurr

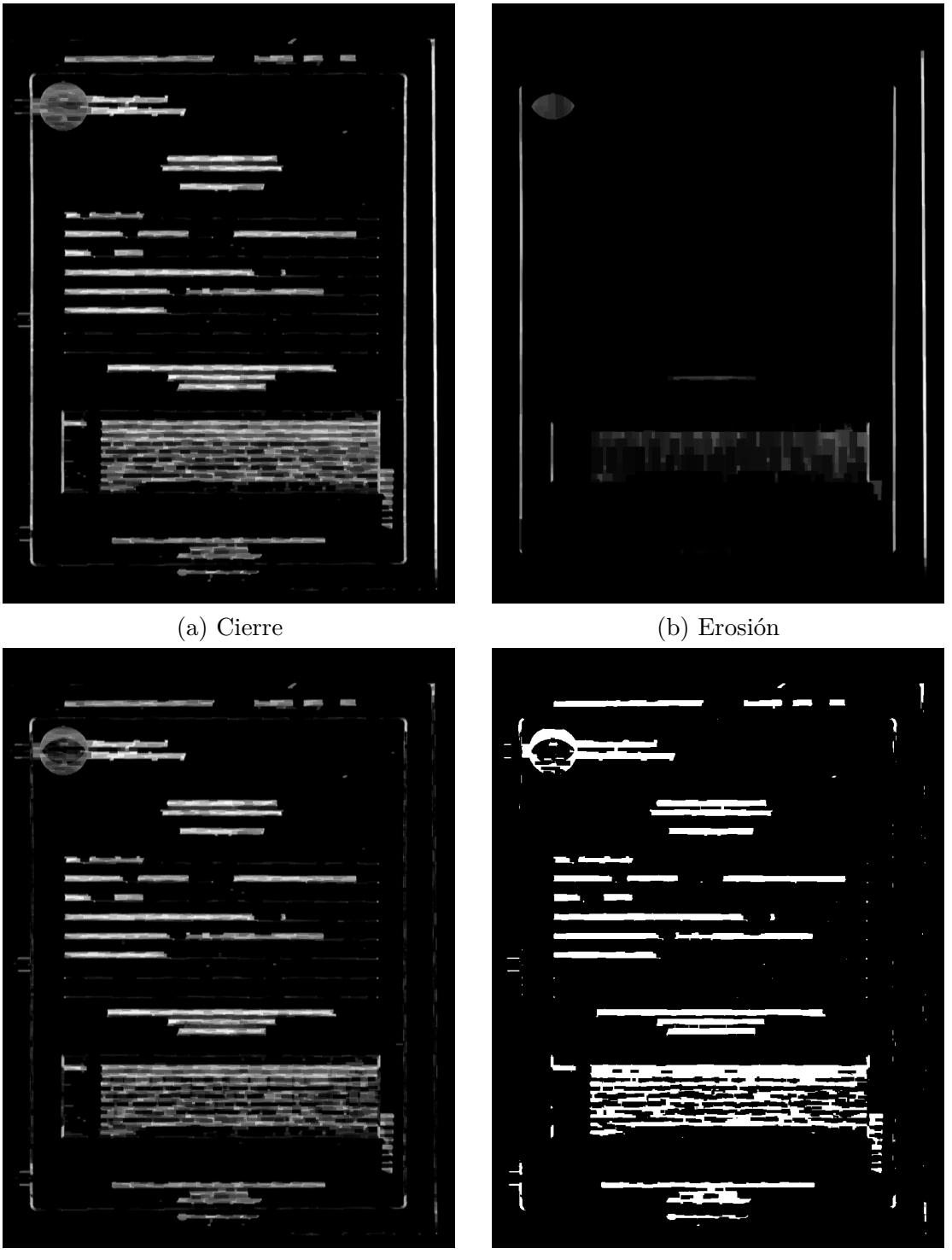


(c) Blackhat



(d) Sobel

Figura 5.1: Cuatro primeros pasos del procedimiento



(a) Cierre

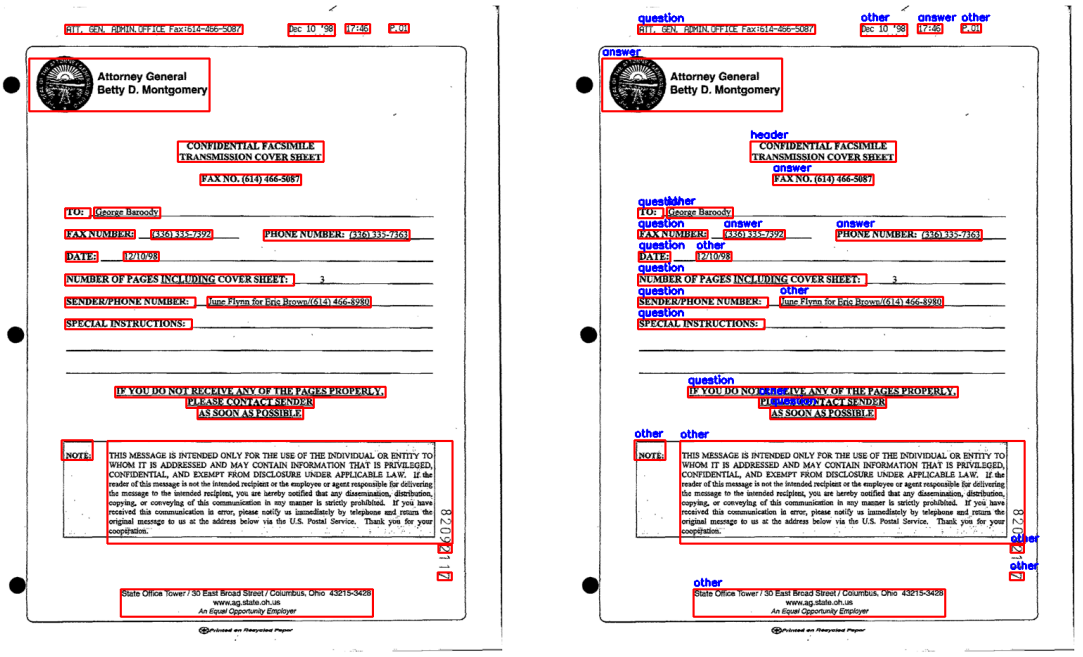
(b) Erosión

(c) Eliminar lineas

(d) Umbralización

Figura 5.2: Pasos del 5 al 8 del procedimiento

U-Net. Obtenemos resultados muy parecidos a los que obteníamos con DeepLab, muy por debajo de YOLOv4.



(a) Localización (b) Detección

Figura 5.3: Localización y detección del procedimiento

5.2. Nuevo dataset

Lo que más nos interesa de nuestros modelos es la capacidad de generalizar ante nuevas facturas o formularios, es decir, que el modelo sea capaz de predecir las preguntas y respuestas de cualquier formulario que le presentemos. Para ello nos surge la necesidad de crear un nuevo dataset propio con facturas reales y actuales y, así, ver cómo de bien se desenvuelven los modelos.

Este nuevo dataset está formado por 27 documentos escaneados de tipo financiero y formularios que fechan desde el 2015. Han sido escaneados con una resolución de 200 ppp. En cuanto a las anotaciones, hemos utilizado el programa de LabelImg [43]. Este programa abre cada una de las imágenes que componen el dataset y el usuario marca con un rectángulo las coordenadas y la clase de cada uno de los objetos. LabelImg almacena esta información en archivos *.xml* o *.txt* según el formato de las coordenadas elegido por el usuario. Si se ha elegido el formato de YOLO, se guardarán en archivos *.txt* y si se ha elegido que se almacenen en formato *xmin, ymin, xmax, ymax* se guardarán en archivos *.xml*. Nosotros utilizaremos ambos formatos para evaluar los distintos modelos.

Es importante remarcar que este dataset, a diferencia del de FUNSD, está en español. Esto supone un problema para nuestro clasificador de texto ya que ha sido entrenado en la lengua inglesa. Para que el modelo entienda el texto, haremos uso del traductor

de Google mediante la libría Googletrans [44]. No obstante, si no se hace una lectura perfecta, el traductor es incapaz de traducir el texto con errores, por lo que solo se llegan a traducir correctamente muy pocos bloques de texto. Para intentar mejorar la lectura que se hace de las palabras, pasaremos los bloques de textos por un corrector. Utilizaremos el de la librería Autocorrect [45]. Sin embargo, obtenemos el mismo F1-score al no utilizar traductor, al utilizar únicamente el traductor y al utilizar el corrector y el traductor.

A continuación, presentaremos los resultados obtenidos con los modelos entrenados en el dataset FUNSD al aplicárselo a este nuevo dataset.

5.2.1. Resultados

Es hora de probar todos los modelos construidos y entrenados hasta ahora. Aunque YOLOv4 haya sido el modelo que mejor resultados nos ha proporcionado para el conjunto de test de FUNSD, veremos si con el nuevo dataset sigue estando en cabeza. En la tabla 5.2 encontramos la tabla de resultados para este nuevo dataset.

	Precision	Recall	F1-score	mAP
Procesamiento-img	0.20	0.27	0.23	6.26
Faster R-CNN	0.35	0.22	0.27	12.50
EfficientDet	0.34	0.08	0.13	0.70
YOLOv4	**0.31**	**0.34**	**0.32**	**16.60**
DeepLab	0.11	0.22	0.15	3.40
HRNet-Seg	0.14	0.24	0.18	4.45
U-Net	0.09	0.22	0.13	3.43
YOLOv4 + BERT	0.35	0.29	0.31	12.61

Tabla 5.2: Métricas para el nuevo dataset

YOLOv4 vuelve a ser el modelo con el que obtenemos los mejores resultados, sin embargo, estos son mucho más bajos que los que obteníamos con el dataset de FUNSD. Obtenemos un F1-score del 0.32 frente al 0.71 que obteníamos antes. Ahora sí, parece que nuestro clasificador no lo hace tan mal. Los resultados del modelo no distan mucho del de YOLOv4, obtenemos un F1-score del 0.31. Para obtener estos resultados no hemos hecho uso ni del traductor ni del corrector, sin embargo en ambos casos obteníamos un F1-score de 0.31. El siguiente es el de Faster R-CNN que consigue un 0.27 en dicha métrica, y seguido, el proceso basado en análisis de imagen con un 0.23. Todos los demás modelos no consiguen llegar al 0.20.

El modelo que hemos construido a partir de la aplicación de filtros de procesamiento de imágenes ya no está entre los tres mejores. Y es que, este modelo necesita un mínimo de interacción con el usuario para obtener mejores resultados. En esta interacción le decimos al modelo si queremos que los bloques de texto sean más o menos grandes. Al pasar de imágenes con baja resolución al nuevo dataset con mayor resolución, los rectángulos obtenidos se nos quedan pequeños. Sin embargo, modificando ligeramente los parámetros obtenemos una *Precision* de 0.31, un *Recall* de 0.29, un F1-score de 0.30 y un mAP de 7.65. Se posiciona entonces entre el modelo de YOLOv4 y el de YOLOv4 con el clasificador BERT.

Sería interesante ver cómo se comportan los modelos de LayoutLM y BROS en este nuevo dataset. Sin embargo, la documentación de estos modelos es escasa y aunque disponen de página web, su uso no es directo. Quedaría como trabajo futuro explorar el uso de estos modelos. En la siguiente sección veremos algunas predicciones de los distintos modelos sobre una de las facturas de este nuevo dataset.

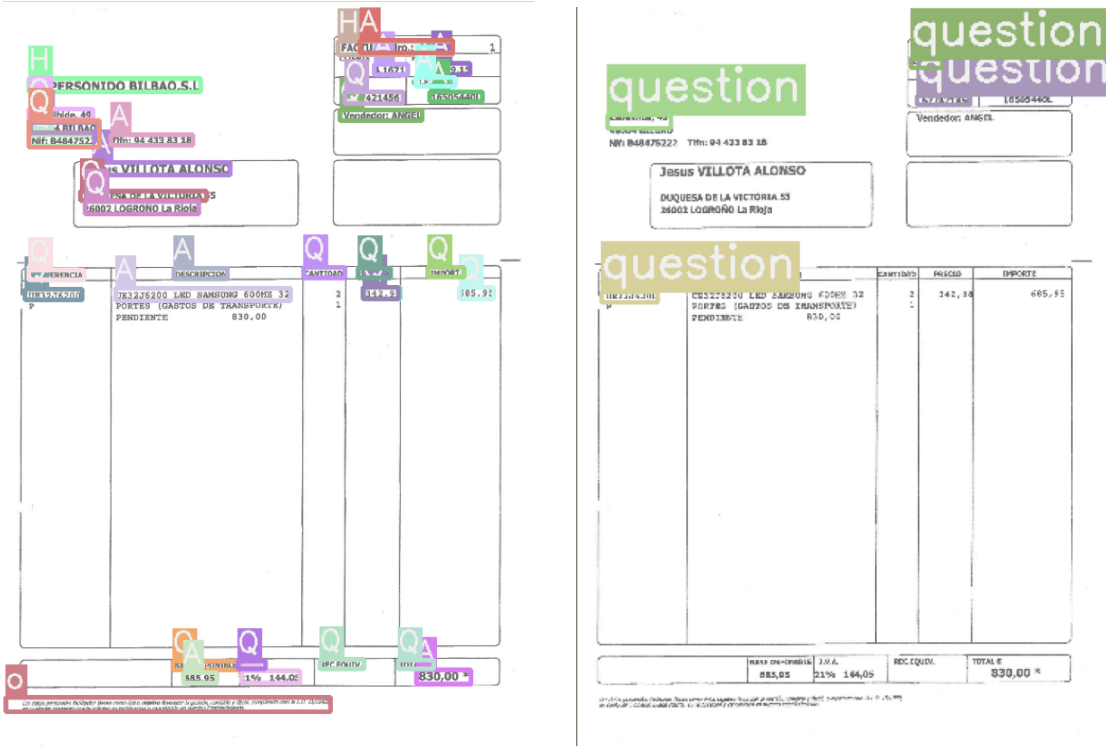
5.2.2. Predicciones

Al igual que en los anteriores capítulos, mostraremos a continuación un ejemplo de cómo se desenvuelven nuestros modelos ante facturas y formularios nuevos. Tomamos una factura aleatoria de nuestro nuevo dataset y lo pasamos por los modelos. El resultado obtenido lo encontramos en las figuras 5.4, 5.5, 5.6 y 5.7. Utilizamos las mismas etiquetas y colores que usamos para determinar las predicciones del capítulo 3.

En la figura 5.4 encontramos los modelos de detección de objetos de la librería de IceVision. Empezamos con la predicción de Faster R-CNN, figura 5.4a. En la tarea de localización se deja algunos bloques de texto por localizar, y en la tarea de clasificación, identifica erróneamente bastantes entidades. El modelo de EfficientDet, figura 5.4b, solo consigue detectar cuatro bloques de texto, de entre los cuales, tres de ellos están mal clasificados.

Observamos ahora la predicción de YOLOv4, figura 5.5a, la localización y la clasificación es bastante buena. Si nos fijamos ahora en el modelo YOLOv4+BERT, figura 5.5b, vemos que nos hemos deshecho de muchos objetos que estaban localizados varias veces. En cuanto a la clasificación, nuestro modelo no es capaz de superar la de YOLOv4.

Recordamos las representación de las etiquetas que utilizamos en los modelos de segmentación. En verde encontramos los objetos de tipo *question*, en rojo los de tipo *answer*, en azul los *header* y en gris los objetos de tipo *other*. Los modelos de segmentación, figuras 5.6, no son capaces de realizar una buena tarea de clasificación en esta factura ya que clasifica a la mayoría de las respuestas como encabezado y a los encabezados



(a) Predicción Faster R-CNN

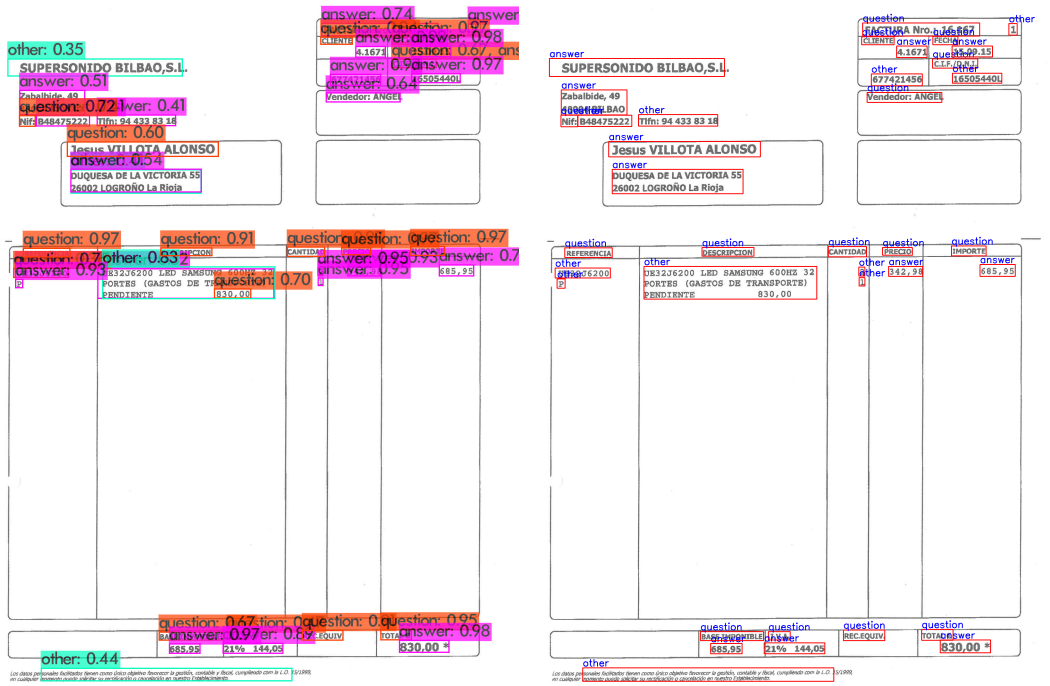
(b) Predicción EfficientDet

Figura 5.4: Predicciones de los modelos de IceVision

como respuestas.

Por último tenemos los procesamientos de imágenes, figuras 5.7. En el primero se utilizan los valores que utilizamos en el dataset de FUNSD. La localización de las entidades solo se realiza correctamente cuando se trata de palabras sueltas, sin embargo, cuando la entidad comprende varias palabras, este procedimiento no es capaz de detectarlas como un todo. Adaptando algunos valores conseguimos con el nuevo procedimiento realizar una mejor localización. En cuanto a la clasificación, las entidades de tipo *question* están bastante bien clasificadas, en cambio, la mayoría de los bloques de tipo *question* se identifican como *other*.

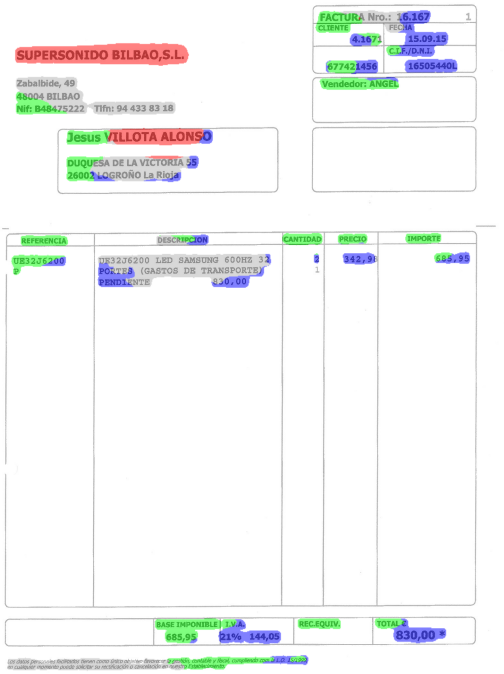
Una vez visto cómo se comportan nuestros modelos ante una nueva factura, cerramos este capítulo y procedemos a las conclusiones.



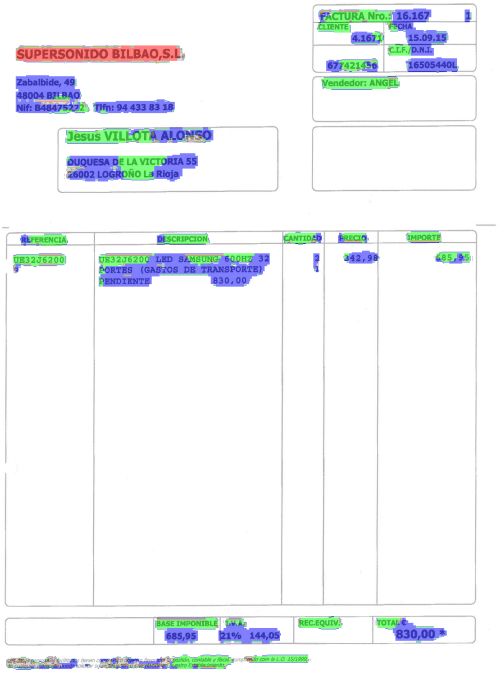
(a) Predicción YOLOv4 (b) Predicción YOLOv4 + BERT

Figura 5.5: Predicciones de YOLOv4 y nuestro clasificador

(a) Predicción DeepLab

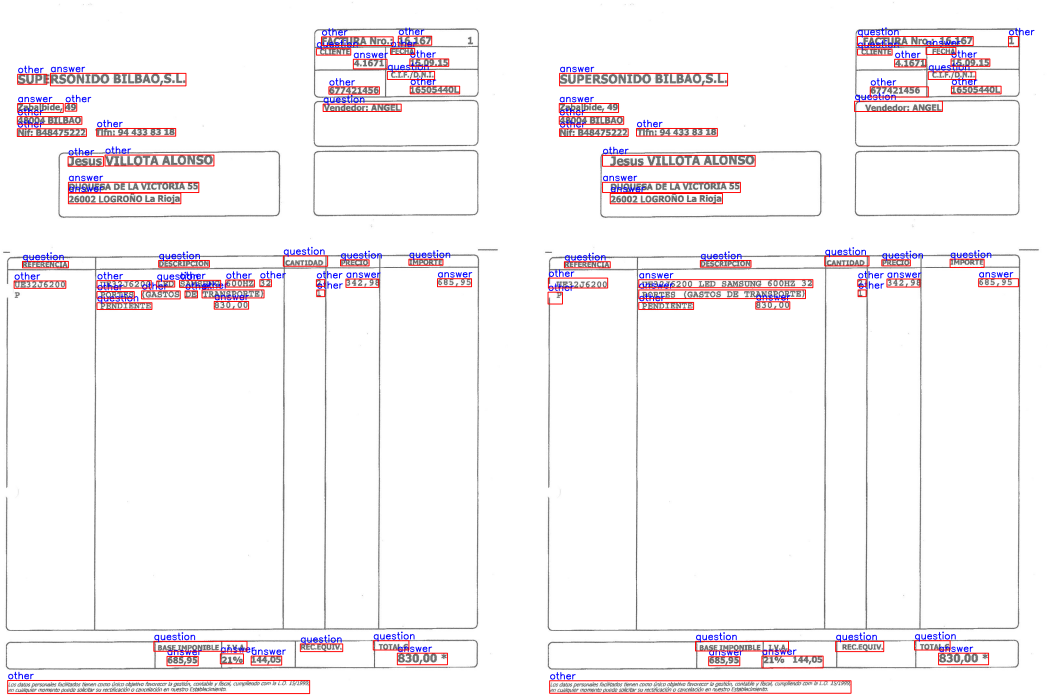


(b) Predicción HRNet-Seg



(c) Predicción U-Net

Figura 5.6: Predicciones de los modelos de segmentación



(a) Predicción procesamiento imágenes (b) Predicción procesamiento adaptado

Figura 5.7: Predicciones del procesamiento de imágenes

Capítulo 6

Conclusiones

Este trabajo es un primer paso hacia el reconocimiento de entidades de formularios basado en la combinación de técnicas de visión por computador y de procesamiento de lenguaje natural. Hemos probado distintos modelos de aprendizaje profundo para la detección y la segmentación semántica en la tarea de identificar entidades en los formularios utilizando únicamente características visuales. El modelo de detección YOLOv4 ha demostrado ser el más competitivo, con un F1-score de 0.71, comparado con los modelos más complejos presentados en la literatura.

Además, hemos explorado la aplicación de dos técnicas de *transfer learning* para clasificar el texto de las entidades. En dicho estudio, llegamos a la conclusión de que el modelo de BERT es el que produce el mejor resultado, con un F1-score de 0.81.

Los modelos estudiados no requieren un paso de preentrenamiento, esto los convierte en una alternativa factible a los modelos más avanzados, aunque su rendimiento sea ligeramente peor. También es importante destacar que estos modelos, salvo el de YOLOv4, han sido entrenados con las GPUs que nos presta Google Colab, disponibles para cualquier usuario que tenga una cuenta de Google.

Hasta ahora, hemos probado a combinar el modelo de localización ganador con el modelo de BERT y de esta manera aportar las ventajas de cada uno de ellos. Sin embargo, los resultados obtenidos descienden a un F1-score del 0.44. La causa principal es la incorrecta lectura del texto incluido en los bloques de texto. Esto se debe a que la localización no es perfecta y que las técnicas de reconocimiento óptico de caracteres no funcionan correctamente en el conjunto de datos FUNSD [5]. Esta tarea queda como reto para el futuro.

En este trabajo también hemos construido un modelo basado en aplicar distintos filtros a las imágenes, sin necesidad de entrenar ningún modelo de *deep learning*. Este mo-

delo consiguió posicionarse entre los 3 mejores en la tarea de localización del dataset de FUNSD. Sin embargo, para obtener resultados mejores con documentos nuevos, es necesario una pequeña interacción con el usuario. Cuando se realiza esta interacción, los resultados obtenidos se aproximan mucho a los de YOLOv4.

Por último, hemos creado un nuevo dataset de facturas actuales donde hemos podido ver cómo se desenvuelven los distintos modelos construidos. Vuelve a ser YOLOv4 el modelo con el que obtenemos los mejores resultados, un F1-score de 0.31.

Como trabajo futuro además de continuar mejorando los modelos de reconocimiento de entidades, nos quedaría abordar la tarea de enlazado de entidades para así finalizar con nuestra tarea de comprensión de formularios.

Bibliografía

[1] Guillaume Jaume. Funsd: Form understanding in noisy scanned documents. https://guillaumejaume.github.io/FUNSD/.

[2] Adam W Harley, Alex Ufkes, and Konstantinos G Derpanis. Evaluation of deep convolutional nets for document image classification and retrieval. In *International Conference on Document Analysis and Recognition (ICDAR)*.

[3] Jonathan Hui. map (mean average precision) for object detection. https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173.

[4] J. Cartucho, R. Ventura, and M. Veloso. Robust object recognition through symbiotic deep learning in mobile robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2336–2341, 2018.

[5] Guillaume Jaume, Hazim Kemal Ekenel, and Jean-Philippe Thiran. Funsd: A dataset for form understanding in noisy scanned documents. In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, volume 2, pages 1–6, 2019.

[6] Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. Layoutlm: Pre-training of text and layout for document image understanding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 1192–1200, New York, NY, USA, 2020. Association for Computing Machinery.

[7] Yang Xu, Yiheng Xu, Tengchao Lv, Lei Cui, Furu Wei, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, Wanxiang Che, Min Zhang, and Lidong Zhou. Layoutlmv2: Multi-modal pre-training for visually-rich document understanding, 2021.

[8] Teakgyu Hong, DongHyun Kim, Mingi Ji, Wonseok Hwang, Daehyun Nam, and Sungrae Park. Bros: A pre-trained language model for understanding texts in document, 2021.

[9] D. Lewis, G. Agam, S. Argamon, O. Frieder, D. Grossman, and J. Heard. Building a test collection for complex document information processing. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, page 665–666, New York, NY, USA, 2006. Association for Computing Machinery.

[10] Jason Brownlee. A gentle introduction to object recognition with deep learning. `https://machinelearningmastery.com/object-recognition-with-deep-learning/`, 2019.

[11] airctic. Icevision. `https://airctic.com/0.7.0/`.

[12] AlexeyAB. Darknet. `https://github.com/AlexeyAB/darknet`.

[13] Semtorch 0.1.1. `https://pypi.org/project/SemTorch/`, SemTorch 0.1.1.

[14] Fastai python library v1.0.57. `https://docs.fast.ai/`, 2019.

[15] Connor Shorten and Taghi Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6, 07 2019.

[16] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. 2014.

[17] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[18] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. *CoRR*, abs/1911.09070, 2019.

[19] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[20] Harshall Lamba. Understanding semantic segmentation with unet. `https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47`, 2019.

[21] Beeren Sahu. The evolution of deeplab for semantic segmentation. `https://towardsdatascience.com/the-evolution-of-deeplab-for-semantic-segmentation-95082b025571`, 2019.

[22] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. Deep high-resolution representation learning for visual recognition. *CoRR*, abs/1908.07919, 2019.

[23] Pedro Marcelino. Transfer learning from pre-trained models. `https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751`.

[24] Leslie N. Smith. No more pesky learning rate guessing games. *CoRR*, abs/1506.01186, 2015.

[25] Google colab. `https://colab.research.google.com/`.

[26] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning based text classification: A comprehensive review. *CoRR*, abs/2004.03705, 2020.

[27] Summary of the tokenizers. `https://huggingface.co/transformers/tokenizer_summary.html`.

[28] Jónathan Heras. Apuntes de inteligencia artificial. Universidad de La Rioja, 2020.

[29] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019.

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[31] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

[32] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[33] Jeremy Howard and Sebastian Ruder. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146, 2018.

[34] Gidi Shperber. A gentle introduction to ocr. `https://towardsdatascience.com/a-gentle-introduction-to-ocr-ee1469a201aa`, 2018.

[35] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Improving object detection with one line of code. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[36] R. Smith. An overview of the tesseract ocr engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 629–633, 2007.

[37] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33:31–88, 2001.

[38] Diego Campos Sobrino. Métricas de similitud para cadenas de texto. `https://soldai.com/metricas-comparacion-cadenas-texto/`, 2019.

[39] Yo-Sub Han, Sang-Ki Ko, and Kai Salomaa. Computing the edit-distance between a regular language and a context-free language. *International Journal of Foundations of Computer Science*, 24:85–96, 08 2012.

[40] Klaus Schulz and Stoyan Mihov. Fast string correction with levenshtein-automata. *INTERNATIONAL JOURNAL OF DOCUMENT ANALYSIS AND RECOGNITION*, 5:67–85, 2002.

[41] Ana Romero. Apuntes de procesamiento de imágenes digitales. Universidad de La Rioja, 2021.

[42] OpenCV. `https://opencv.org/`.

[43] Darren TzuTa. Labelimg. `https://github.com/tzutalin/labelImg`.

[44] Googletrans. `https://pypi.org/project/googletrans/`.

[45] Autocorrect. `https://github.com/fsondej/autocorrect`.