AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

Una propuesta basada en Python para la extracción de información electoral a partir de códigos QR

Fernández, Joaquín
Giorgi, Daiana
Universidad Tecnológica Nacional, Facultad Regional Santa Fe. Departamento de Ingeniería en Sistemas de Información

Abstract
El trabajo describe el desarrollo de parte del proceso definido para lograr la extracción de caracteres de un telegrama de escrutinios, a partir de la lectura de códigos QR. El presente trabajo tiene por objetivo la transformación del documento original en formato pdf y la posterior lectura y reconocimiento de dos códigos QR ubicados en extremos inversos del documento, que contienen información sobre el mismo y demarcan el área donde posteriormente debe realizarse el reconocimiento de caracteres.

Palabras Clave
reconocimiento, código QR, python.

1. Introducción

El proyecto en el que los autores participan tiene como objetivo la extracción de caracteres numéricos presentes en un documento electoral denominado "telegrama". El telegrama es una herramienta que se les brinda a los presidentes de mesa en los actos electorales o elecciones, para poder comunicar y procesar información sobre los resultados de cada mesa, y que ésta pueda llegar al conocimiento público.

Este telegrama contiene una serie de elementos presentes que cumplen una función específica: el encabezado con la información para los presidentes de mesa, los campos numéricos (registro de cantidad de votos obtenidos por partido), los campos de firmas de los encargados y los códigos QR para facilitar el escaneo digital por el sistema SETT (Sistema de Escaneo y Transmisión de Telegramas) [5] que se encarga del envío digital de los mismos. Para cumplir con el objetivo del proyecto, teniendo en cuenta el contexto en el que se iba a trabajar, se planificó un hilo conductor o 'pipeline' de desarrollo que se muestra en la Figura 1. El mismo contiene actividades procedurales para lograr el objetivo final, que es el reconocimiento y la extracción de los datos numéricos que representan la cantidad de votos escrutados y los resultados de cada mesa.

Tal como muestra la Figura 1, la entrada al proceso es un archivo en formato PDF (elemento 1) que contiene la información sobre los resultados de los comicios en cada una de las mesas de la jurisdicción donde se llevó adelante la elección.

La actividad 1 *"Conversión a imagen"* consiste en la conversión del documento PDF a PNG para poder tratar el archivo como imagen (elemento 2). Luego de eso, se realiza la actividad 2 *"Detección y lectura de QR"*.

Esta primera parte del proceso (primera fila de la Figura 1) son las actividades que serán descritas a continuación y en las que los responsables somos los autores de este trabajo a través de una Beca de Investigación para alumnos y una Beca BINID.

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

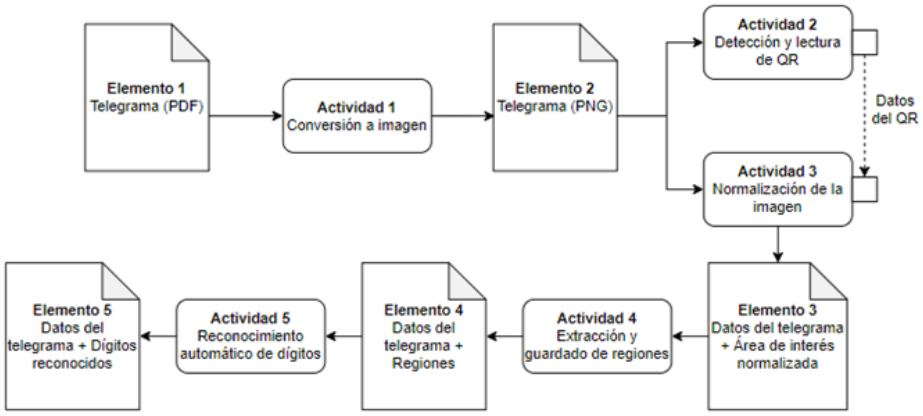

Figura 1 - Pipeline de actividades del proyecto.

Posteriormente, con esta información se realizará la actividad 3 *"Normalización de la imagen*" para poder luego obtener un recorte de la misma, generando así los *"Datos del telegrama + Área de interés normalizada"* (elemento 2) y así lograr extraer la región de interés. Luego, la actividad 4 *"Extracción y guardado de regiones"* procesa estos datos (elemento 2), en donde mediante un algoritmo de reconocimiento de márgenes, se extraerá la región principal del telegrama, en donde se encuentran los caracteres numéricos. Así se generará el elemento 3 *"Datos del telegrama + Regiones"*. Por último, la actividad 5 *"Reconocimiento automático de dígitos"* procesa estos datos del telegrama y las regiones (elemento 3) mediante una estrategia de inteligencia artificial de reconocimiento de caracteres que fue obtenida de un espacio público de Google y fue entrenada por más de mil usuarios. Finalmente se genera el elemento 4 y salida resultante del pipeline *"Datos del telegrama + Dígitos reconocidos"*. Esta salida servirá de entrada para un siguiente proyecto en el que se procesarán los datos extraídos y, mediante una secuencia lógica de comparación de documentos, los datos serán subidos a un servidor público en donde se podrán visualizar los resultados parciales de las elecciones.

2. Consideraciones generales

En cada proceso electoral que se realiza en alguna jurisdicción, los presidentes de cada mesa electoral deben generar un telegrama que resuma, organice y contenga la información de los resultados de su mesa.

En cada telegrama perteneciente a una mesa, cada página de todos los telegramas impresos y enviados con la información sobre los resultados y cantidad de votantes se puede reconocer inequívocamente mediante los datos presentes en los códigos QR. En la Figura 2 se presenta un telegrama real extraído de las elecciones generales 2021, en el que se muestran cada QR y su ubicación general. El mismo tiene doble función: por un lado, es un identificador único de cada telegrama y, por otro lado, brinda información acerca del mismo. Es por este motivo, que este código necesita ser identificado y reconocido no sólo para extraer la información del telegrama, sino también para verificar la misma en las primeras etapas del proceso de recuento.

Figura 2. Códigos QR en el telegrama electoral.

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

Como puede verse en la Figura 2, cada hoja de un telegrama posee dos códigos QR en vértices opuestos, lo que sirve para delimitar el tamaño y el área donde se encuentra la información útil que luego deberá ser reconocida y extraída.

En cada elección y para cada zona o región, los telegramas cambian su formato y diseño debido a que varían los cargos que se votan y también los partidos que presentan candidatos para cada cargo. Debido a esto, los telegramas cambian su formato y diseño, ya que pueden existir filas y/o columnas sin utilizar en alguna jurisdicción. Por ello, será necesario reconocer las diversas regiones de interés para lograr extraer la información útil de cada formato posible (Figura 1, actividad 4). Para lograr esto, es necesario generar una "máscara" que permita realizar este reconocimiento de regiones, según el diseño de los telegramas a procesar cada vez. Sin embargo, esta generación y utilización de una máscara está fuera del área de interés de este trabajo.

3. Detección y lectura del código QR del telegrama

3.1 Análisis inicial del QR

Para realizar el análisis de los telegramas, se obtuvieron cuatro datasets correspondientes a elecciones PASO y GENERALES de los años 2019 y 2021. Para la definición y desarrollo del algoritmo, se tomó como ejemplo el telegrama genérico que se muestra en la Figura 2. Como se mencionó, cada hoja del telegrama tiene dos códigos QR: el primero se encuentra en la parte superior izquierda y el segundo en la parte inferior derecha. Una vez detectada la ubicación de los códigos, se procedió a leer cada uno para lograr acceder a la información asociada. La Figura 3 muestra a modo de ejemplo un QR de la parte superior del telegrama superior.

Figura 3. Código QR superior y su identificación.

3.2 Significado del código QR

Del análisis pormenorizado de la cadena de números descifrados de los códigos QR, los datos ya impresos en el telegrama y la información plasmada en la base de datos, se pudo identificar qué significaba cada uno de los números extraídos del código.

Cada código QR se asocia con una secuencia de 9 dígitos numéricos en total. Tal como muestra la Figura 4, los primeros 5 dígitos pertenecen al número de mesa de la elección, la sexta posición de la secuencia indica si la mesa es de personas residiendo en el extranjero o nacionales (representando al extranjero con un número cero y al nacional con un número uno). El dígito ubicado en la séptima posición indica la página actual del telegrama (el mismo puede contener más de una página). La octava posición representa la cantidad de páginas y el último dígito indica si el QR de referencia es el de la esquina superior izquierda (1) o la esquina inferior derecha (0) del telegrama.

Figura 4. Código QR inferior y su identificación.

3.3 Decisiones para la actividad de lectura de códigos QR

Para lograr ejecutar las actividades 1 y 2, se desarrolló un algoritmo que es capaz de tomar una imagen en formato .pdf, luego convertirla a .png, para posteriormente escanear la misma, descifrar los datos almacenados en el código QR y finalmente guardar la información brindada por el mismo.

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

Una vez planteado el problema inicial y definido el pipeline del proyecto (Figura 1), se analizó qué lenguaje de programación utilizar para automatizar el procedimiento resultante. Luego de una primera etapa de investigación y de considerar las habilidades previamente obtenidas de los investigadores involucrados, se eligió el lenguaje de programación Python [3] para llevar a cabo el desarrollo del módulo de lectura, debido a su gran capacidad de manipulación y procesamiento de datos, especialmente imágenes.

Desde el primer momento se investigó cómo este lenguaje gestiona el tratamiento de imágenes y se encontraron dos librerías muy interesantes: *OpenCV* (*'cv2*' en adelante) y *pyzbar*. *Cv2* [2] es una gran herramienta para el procesamiento de imágenes y la realización de tareas de visión artificial y *pyzbar* [4] es una librería que lee códigos de barras unidimensionales y códigos QR utilizando la biblioteca zbar.

Para llevar adelante la actividad 2 del pipeline que se muestra en la Figura 1, *"Detección y lectura de QR"*, se generaron dos algoritmos posibles que pueden dar solución al objetivo de la actividad.

En primer lugar, se propuso localizar ambos códigos QR, extraer y analizar cada uno como imágenes individuales, guardando posteriormente esas imágenes para finalmente tratar cada una de manera independiente para la extracción de la información.

Como segunda solución, se propuso extraer en bucle los datos de ambos códigos QR en una sola imagen íntegra que se corresponda con el telegrama original, para luego enviarlos como retorno de una función que se encargue de extraer la información.

Si bien se propusieron, desarrollaron y probaron ambas opciones, se optó por avanzar con la segunda solución, por tres motivos:

1. *La velocidad de procesamiento*. Detectar y extraer datos directamente desde la imagen incrementó la velocidad en que los telegramas eran procesados y, al disminuir el tiempo de lectura, se disminuye también el tiempo de ejecución de todas las actividades del pipeline en su totalidad.
2. *Propósito*. El objetivo de la lectura de los códigos QR está vinculado con sus datos y no con su contenido visual. Dado que un código QR es una forma de codificar datos, el interés del trabajo es lograr acceder a esos datos, y no la imagen en sí.
3. *Simplicidad de desarrollo*. Dado que el módulo de *cv2* trae incluido una gran cantidad de funciones eficientes, la lectura de los códigos QR se puede realizar en muy pocas líneas de código. Esto, junto a la decodificación de *pyzbar* (siendo simplemente una línea de código) mejora los tiempos y uso de memoria para el procesamiento, lo que agiliza el acceso a la información.

3.4 Desarrollo de la detección y lectura de los códigos QR

Para iniciar el desarrollo de la solución que se muestra en la Figura 5, se comenzó definiendo un contador '**counter**' con un valor igual a cero. Este contador sólo tomará los valores 0 o 1, haciendo referencia a cada uno de los códigos QR que pueden estar presentes en una página.

```
counter = 0

qrs = ["", ""]

for barcode in decode(img, symbols=[ZBarSymbol.QRCODE]):

    qrs[counter] = barcode.data.decode('utf-8')

    counter += 1
```

Figura 5. Código de la solución.

Luego se declara un arreglo de dos dimensiones ('**qrs**'), donde se almacenarán posteriormente los datos que serán decodificados. El bucle '**for**' itera sobre la imagen '**img**' que se pasa como primer

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

parámetro y busca y almacena parcialmente en '**barcode**' sólo cifrados presentes en la imagen que se detecten como tipo QR. Esto se logra por el segundo parámetro en el bucle '**symbols=[ZBarSymbol.QRCODE]**'. Este parámetro hace que la búsqueda ignore todos los otros tipos de codificaciones que la librería *pyzbar* soporta, disminuyendo los tiempos de búsqueda cerca de un 10%. En cada '**barcode**' que se encuentra sobre '**decode**', se extraen los datos mediante la decodificación '**barcode.data.decode('utf-8')**' y se almacenan en '**decoded_page**'. Luego se pasan a '**qrs[counter]**' en la posición correspondiente y se aumenta el contador para una posible siguiente iteración.

El código podría tener algunos casos de error. Dado a que la imagen no es escaneada automáticamente, sino que requiere de un operador humano para su envío, la imagen puede llegar deformada (como se muestra en la Figura 6) y consecuentemente no se logren decodificar los QRs.

Figura 6. Ejemplo de telegrama problemático.

4. Métricas y prueba del algoritmo

Para poder verificar el código generado y controlar que cumpla con el objetivo de la actividad se armaron diversas muestras de telegramas. Se generaron distintos conjuntos de telegramas aleatorios, comenzando con una muestra de 1 telegrama y ascendiendo hasta muestras de 1000 telegramas distintos. En particular, se armaron muestras de 1, 10, 100, 500 y 1000 telegramas.

Para la confección de los lotes de prueba se usaron telegramas del dataset de las elecciones generales del 2021. Las pruebas se realizaron de la siguiente manera: se generó el código para que sea capaz de recorrer una carpeta que contenía un conjunto de telegramas convertidos a archivos de tipo png. Entonces, uno a uno iba procesando esos archivos y realizaba un registro de los datos que podía detectar del código QR del telegrama y un registro a modo de advertencia en el caso de que no hubiera podido leerlo. Es decir, se creó como salida un archivo log en el cual se registró tanto el resultado favorable de la lectura del código QR (entiéndase como favorable al código QR que pudo ser descifrado), como así también los códigos que no pudieron leerse. Es decir, se registró la lectura y obtención de los datos de todos los QR presentes en el telegrama, como así también, si alguno de los QR no pudo ser leído, se logró informar en qué archivo sucedió y si fue el QR superior o inferior (o bien, en algunas ocasiones, se informó que ninguno de los dos códigos pudo ser descifrado).

En la Figura 7 se muestra una parte del log en donde se ven diferenciados los casos anteriormente mencionados.

```
El QR del 05991-N-1636931699220.pdf.png no pudo ser leido
Pudo leer el QR SUPERIOR
 Numero de mesa leido desde QR: 05992N      Cantidad ocurrencias QR: 1
 Numero de mesa leido desde QR: 05993N      Cantidad ocurrencias QR: 2
```

Figura 7. Log de reporte de pruebas.

Una vez ejecutado el algoritmo para todas las muestras constituidas, se realizó un análisis a nivel estadístico, en el cual de acuerdo a la cantidad de telegramas y números totales de códigos QR (entendiéndose que por cada hoja que presenta un telegrama existen dos códigos QR) procesados se pueda analizar qué cantidad de códigos pudieron ser descifrados y cuáles no se lograron leer para poder visualizar que tan efectivo era el algoritmo desarrollado y si era lo que el equipo necesitaba y esperaba. Los resultados de totales y valores estadísticos se ven reflejados en la Tabla 1.

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

Analizando los resultados obtenidos, entiéndase por esto a la información resultante (tanto los códigos QR descifrados como a los que no se logró descifrar) de cada uno de los telegramas de las distintas muestras, los mismos en su mayoría fueron aceptables. Esto significa que se logró descifrar la mayoría de los códigos, obteniendo así toda la información que contenían dentro.

Cant. telegramas por lote	Cant. total QR	Cant. de QR descifra-dos	Cant. QR sin poder descifrar	% éxito (QR descifra-dos)	Perfor-mance (seg.)
1	2	2	0	100%	0.075
10	20	16	4	80%	0.721
100	200	187	13	93.5%	7.25
500	1000	921	79	92.1%	38.16
1000	2000	1838	162	91.9%	76.16

Tabla 1. Resultados de totales y porcentajes de las distintas muestras ejecutadas por el algoritmo.

Se alcanzaron altos porcentajes de éxito (91,5% en promedio) al momento de descifrar cada uno de estos códigos QR. Se observó también el pequeño porcentaje de falla (imposibilidad de descifrado) y se detectó que en casi todos los códigos QR que no fueron reconocidos, se debió a que los mismos no se encontraban en las condiciones óptimas para ser leídos, ya que la imagen tenía el código QR "cortado" por dobleces en el papel o hubo mala posición de la cámara para capturar el telegrama. Es decir, no era un error del algoritmo, sino de la imagen (Figura 6).

Otro dato que se tuvo en cuenta fue el tiempo de demora o performance del algoritmo, es decir, el tiempo que necesitaba para poder desencriptar los códigos QR de las distintas muestras. Los tiempos resultaron realmente alentadores dado que, en promedio, necesitaba de menos de medio segundo de lectura por telegrama para poder descifrar sus códigos.

4.1 Criterio de aceptación

Cuando se analizó el porcentaje estadístico que arrojaban las muestras, se detectó que en todos los casos los porcentajes de acierto (posibilidad de descifrar el código QR) superó el 90%. Por lo tanto, con este resultado, se pudo dar por finalizada la etapa, ya que se consideró que estos resultados cumplían con las expectativas para continuar con el pipeline que muestra la Figura 1. En la etapa del análisis de los códigos QR, se logró desde las primeras versiones un porcentaje de aceptación mayor al 90%, llegando a alcanzar un 91,6% en la versión final.

Luego del procesamiento del lote mayor se realizó un análisis de los resultados obtenidos. Este análisis se llevó a cabo observando el log y comparando con el lote. Si una entrada del log indicaba que se leyeron ambos códigos QR, era el mejor resultado posible. Sin embargo, cuando una entrada en el log registraba que se logró leer un QR (o ninguno) del archivo, se obtenía la imagen original y se analizaba para determinar cuál era el problema. En la mayoría de los casos, la imagen era defectuosa (con sombras, doblada, fuera de foco). Se logró desde las primeras versiones un porcentaje de aceptación mayor al 90%, llegando a alcanzar un 91,6% en la versión final. Este porcentaje significa total de QRs leídos sobre el total de QRs existentes en el lote.

Conclusiones y Trabajos Futuros

La división de las actividades mediante un pipeline de producción logró segmentar las partes del proceso y poder trabajar cada una de manera independiente. En particular, el análisis y extracción de información de los códigos QR fue logrado exitosamente mediante el uso de librerías de Python, logrando un porcentaje de éxito de 91.6% del total del mayor lote de pruebas sobre el cual el algoritmo fue ejecutado.

Como primera experiencia en un proyecto de investigación y de la propuesta que

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

implicó este trabajo de a dos, se puede afirmar que fue positiva y fructífera, ya que permitió complementar la experiencia y el conocimiento de ambos autores.

Como trabajo futuro relacionado a los temas desarrollados en este trabajo, se propone continuar ejecutando las tareas del pipeline mostrado en la Figura 1, que implican las actividades 3, 4 y 5 (*Normalización de la imagen*, *Extracción y guardado de regiones* y *Reconocimiento automático de dígitos*), a fin de lograr el reconocimiento de dígitos que se incluyen en los telegramas y agilizar así no sólo los conteos provisorios y definitivos, sino también, la difusión de la información vinculada a las elecciones.

Referencias

[1] Autor Desconocido (2021) “LA PROVINCIA REALIZÓ EL ESCRUTINIO PROVISORIO 100% DIGITAL”. Portal de noticias de la Provincia de Santa Fe. Disponible en <https://www.santafe.gob.ar/noticias/noticia/273086/>. Visitada por última vez el 14/10/2022.
[2] TensorFlow Datasets. (2022) “TensorFlow Datasets: a collection of ready-to-use datasets”. Página de repositorios de la organización TensorFlow. Disponible en <https://www.tensorflow.org/datasets/catalog/mnist>. Visitada por última vez el 14/10/2022.
[3] Python Software Foundation. (2022). Disponible en <https://www.python.org/>. Visitada por última vez el 14/10/2022.
[4] OpenCV Team. (2022). Disponible en <https://opencv.org/>. Visitada por última vez el 14/10/2022.
[5] Python Software Foundation. (2022) “pyzbar 0.1.9”. Hudson, Lawrence a.k.a “quicklizard” (2022). Disponible en <https://pypi.org/project/pyzbar/>. Visitada por última vez el 14/10/2022.

Datos de Contacto:

Joaquín Fernández. Universidad Tecnológica Nacional, Facultad Regional Santa Fe. 3000. joaco.fernandez.2212@gmail.com
Daiana Giorgi. Universidad Tecnológica Nacional, Facultad Regional Santa Fe. 3000. daiana.giorgi@gmail.com