Object Character Recognition: Handwritten text

Luis F. Méndez-Rodríguez, Juan C. Velazco-Rossell,
Agustín García-Rodríguez, David E. Hernández-Castañeda

Visión para robots

Feb - Jun 2020

Dr. Cesar Torres Huitzil

***Abstract*— Vivimos en una era donde la digitalización de documentos se ha vuelto algo indispensable, no solo para tenerlos disponibles desde cualquier parte del mundo, a un click de distancia, sino que además es una oportunidad de ayudar a personas que no sepan leer o que tengan dificultades leves de la vista. Aunque existen diferentes aplicaciones de reconocimiento de texto, aún hay un gran campo de oportunidad con el texto escrito a mano y sobre todo con aquel que no tiene una letra de fácil lectura. Es por ello que un transductor de escritos era necesario, se trata de programa diseñado para realizar una mejora en la imagen de entrada, procesar dicha imagen con una separación de palabras y caracteres, interpretarla a través de deep learning y finalmente producir un archivo con el texto que se ha logrado reconocer. En este documento se presentan los pasos necesarios para construir un transductor de este tipo, así como algunos de los retos que uno puede encontrarse a la hora de crearlo y lo que se seleccionó para superarlos.**

***Index Terms*— Machine Learning, Boxing, Clustering, Deep Learning, Wolf, Niblack, Tensorflow, Keras, OpenCV, OCR, Aprendizaje supervisado, Neural Network, CNN, Gradient, Optical character recognition, Backpropagation.**

I. Introducción

La escritura es uno de los medios de comunicación masivos más utilizados, lo cual propone un reto para lograr digitalizar dichos textos. Esta digitalización puede ser motivada por la necesidad de compartir información de manera más versátil y global. El reconocimiento de patrones es una disciplina en continuo desarrollo, que ha tomado relevancia desde los años cincuenta, sobre todo en el campo del análisis de imágenes y la visión por computador. Además de la motivación por la alta demanda de documentos digitalizados, existen otros campos de oportunidad como el apoyo de lectura de texto escrito a mano o la transformación de texto a voz inclusive. El campo de la visión por computadora avanza rápidamente y gracias a sus herramientas, este y otros proyectos se pueden desarrollar.

En este proyecto se busca interpretar texto escrito a mano a partir de imágenes de diferente naturaleza, este texto puede estar en diferente tipografía, orientación y tener ruido a su alrededor, por lo tanto se aplicarán diferentes transformaciones a las imágenes para eliminar este ruido, mejorar la calidad e identificar caracteres y palabras eficientemente para posteriormente utilizar la información obtenida y utilizarla en diferentes aplicaciones.

El texto escrito a mano puede ser encontrado en diferentes tipos de imágenes, como lo pueden ser notas, letreros, anuncios, calendarios, publicidad, recetas médicas, entre muchas otras. Por ello una solución integral de reconocimiento de texto tiene que incluir un mecanismo de reconocimiento de texto escrito a mano en imágenes.

Estudios recientes han demostrado estar trabajando en nuevas técnicas y métodos que puedan reducir el tiempo de procesamiento, mientras que se intenta mejorar la precisión del reconocimiento de texto. OCR, (Optical Character Recognition) por sus siglas en inglés, es el proceso por el cual se convierten imágenes escaneadas, de texto escrito o impreso, a un producto digitalizado en formato compatible con los computadores.

El reconocimiento óptico de caracteres tiene 5 etapas que son fundamentales para lograr una transformación exitosa:

1. **El pre-procesamiento**
2. **La segmentación**
3. **La extracción de características**
4. **Clasificación**
5. **Procesamiento de conversión**

EL preprocesamiento toma como entrada una imagen con extensión `.jpg`, `.jpeg` o `.png`, por mencionar algunos formatos compatibles, y revisa que las dimensiones de dicha imagen sean adecuadas para el programa; de no ser así lo convierte a una imagen compatible en escala y resolución, para posteriormente aplicar una serie de transformaciones que convierten la imagen en una que es más legible a simple vista y que además se muestra casi sin fondo, es decir con un tratamiento de enfoque y retoque de luz.

En la etapa de segmentación se busca separar dicha imagen en pequeñas imágenes que representen palabras, caracteres o inclusive frases completas, aquí se hace el

recorte de las imágenes, se coloca un fondo sólido y se produce el encapsulamiento por palabras o letras. además de estandarizar el tamaño de todas las imágenes a un formato $N \times M$, para que en la siguiente etapa sea más sencillo su manejo.

En una etapa posterior se busca extraer ciertas características de las imágenes que puedan mostrar una tendencia hacia algún tipo de caracter previamente aprendido por el computador. Esto hace que las neuronas ficticias logren reconocer ciertos patrones, figuras o contornos que sean más representativos de una letra en particular.

Siguiendo esta línea de pensamiento, el siguiente paso sería tratar de clasificar dicha información extraída de las imágenes, para así lograr un reconocimiento del texto. Esto implica tomar una decisión sobre la mayor probabilidad de que un caracter pertenezca a una familia o grupo parecido de caracteres.

Uno de los pasos más difíciles es precisamente la extracción de las características, ya que es de gran dificultad elegir el conjunto óptimo; de hecho, muchos algoritmos no tienen un sistema concreto que muestre siempre la misma recopilación de características de una imagen, ya que de ser así esto empeoraría las probabilidades de encontrar características óptimas. De manera general, para que una característica sea considerada útil debe poseer:

- **Discriminación: Deben ser características que diferencien de manera clara una clase de otra.**
- **Deben obtener igual valor para mismas clases.**
- **Independencia: Las características no deben contener correlaciones unas de otras.**
- **Pocas pero significativas características: El número de características debe ser pequeño por términos de rapidez y eficiencia de clasificación.**

Además las características deben contar con otros requerimientos, como que tengan un bajo costo computacional, tanto en tiempo como en complejidad. Debido a estos motivos es muy difícil conseguir características óptimas. Aunque actualmente en el mundo existe hardware especializado que ha vuelto estas tareas un poco más amigables, siempre hay que intentar hacer el proceso lo más eficiente posible.

Finalmente una vez que el proceso anterior ha sido completado con éxito y se tiene una respuesta del sistema ahora debemos convertir todas y cada una de las salidas de las imágenes de entrada para reconstruir un texto completo que además tenga sentido en espacio y forma.

Hasta ahora se ha hablado a grandes rasgos de lo que un procesamiento digital representa y sus etapas; sin embargo, aún existen diferentes métodos por los que esta línea de pensamiento pudiera cambiar y ser más o menos eficiente dependiendo siempre de algunos factores clave, como pueden ser, si el texto es impreso o escrito a mano, si se encuentra en un buen contraste con respecto de su fondo, si tiene buena iluminación de fondo, si es texto de molde o si es texto con texturas o formas distintas a las habituales de escritura, entre otros factores que alteran de manera directa el reconocimiento digital de caracteres.

Actualmente el problema del reconocimiento de caracteres ha encontrado algunas repuestas a través del aprendizaje por computador, además se han creado dos grandes vertientes o tipos de sistemas de reconocimiento: Los sistemas fuera de línea y los sistemas en línea. En este trabajo se estudia el desempeño de un sistema fuera de línea. Además existen herramientas como OpenCV que nos permiten manipular las imágenes con cierta facilidad en un entorno Linux, herramientas como Tensorflow o Keras nos ayudan fabricar redes neuronales en pocos pasos y sobre todo las bibliotecas con los dataset de escritos nos permiten una gran cantidad de imágenes ejemplo para entrenar nuestra red.

La tecnología de reconocimiento de caracteres, OCR, engloba además a un conjunto de técnicas basadas en estadísticas, en las formas de los caracteres, transformadas y en comparaciones, que complementándose entre sí, se emplean para distinguir de forma automática entre los diferentes caracteres alfanuméricos existentes. En realidad no se reconocen exactamente los caracteres de un determinado alfabeto, sino que es posible distinguir entre cualquier conjunto de formas o símbolos. Sin embargo, se debe tener en cuenta que la precisión que se obtiene en la práctica al intentar distinguir entre un conjunto de símbolos no es del 100 %. Por lo tanto, es fácil deducir que cuanto más numeroso es el conjunto de símbolos

entre los que se debe decidir, mayor es la probabilidad de que se produzca un fallo en la clasificación.

El enfoque del presente reporte es proponer una solución al problema de reconocimiento de caracteres a través de tres aproximaciones dirigidas primero al reconocimiento de dígitos numéricos para así seleccionar la mejor manera de implementar redes neuronales. Las técnicas empleadas fueron con el respaldo de Keras, Tensorflow y Sklearn. De estas tres se probó solo con aprendizaje no supervisado usando la biblioteca de Sklearn, específicamente con el modelo de KMeans que esta librería presenta; las otras dos aproximaciones fueron supervisadas, es decir, que en las imágenes de entrenamiento se tuvo que hacer un etiquetado preliminar de dichas imágenes.

El desarrollo de este documento se centra principalmente en un primer acercamiento sobre texto impreso con contraste bien definido para terminar con un acercamiento mucho más complejo a texto escrito a mano escaneado o con muy buena calidad y que no contenga sombras. Es importante también denotar la importancia de seleccionar un buen banco de datos de imágenes que representen las imágenes de entrenamiento pues de esto dependerá en gran medida la precisión que pueda tener este sistema.

Luego pasamos a reconocimiento de letras, siguiendo la misma técnica de implementación de redes neuronales logramos conseguir un porcentaje de alrededor de 75 por ciento de efectividad de reconocimiento y correcta clasificación de palabras. Además en este proyecto se tuvo el interés de reconstruir el texto completo través de una técnica de concatenado de palabras. Sin embargo, también se aprovecharon las herramientas de OpenCV que nos permitieron implementar algoritmos de mejora de imágenes, pasando por umbralizado, "boxing" y segmentación de palabras.

En resumen a lo largo de este documento se analizarán diferentes métodos y técnicas de mejoramiento, clasificación e identificación de texto, a través de algunas de las solucione más populares actualmente en el mundo del aprendizaje por computador. Finalmente se mostrarán los resultados obtenidos de estos algoritmos implementados y se hablará sobre el trabajo futuro en la mejora de estos algoritmos.

II. Estado del arte

Los sistemas de OCR son típicamente desarrollados y pensados para soportar un lenguaje en especifico, escritos y fuentes. Si en un caso concreto el material de entrada no es soportado por el sistema, se requiere complementar dicho sistema con modificaciones pertinentes. Por ejemplo si el material de entrada contiene caracteres o símbolos nuevos, como lo pueden ser de otro lenguaje distinto al de entrenamiento, se tendría que incluir un nuevo entrenamiento con estos nuevos símbolos.

Estudios recientes han demostrado el potencial de ganancia en la precisión de detección y reconocimiento de caracteres no solo al entrenar sistemas OCR, sino en el mismo reconocimiento del texto inscrito en dichas imágenes. Estos avances pueden inclusive ser obtenidos por sistemas en los que fueron diseñados para seguir una sola fuente de caracteres.

Actualmente exponentes como Google Tesseract, Tensorflow o inclusive Keras nos muestran un gran acercamiento a las redes neuronales utilizando software libre, nos proporcionan herramientas de uso libre que permiten formar redes de cualquier tipo, además de dotar de gran libertad al desarrollador, pues se muestra el código fuente de dichos algoritmos, el tiempo de implementación, entre otros. Sin embargo, suele ser más tardado, pues al no contar con una solución integral, casi todo se tiene que crear desde cero, escribiendo cada capa, cada interconexión, métodos de activación, entre muchos otros factores que pueden mejorar o empeorar el desempeño de nuestra red neuronal. Esto hace que el tiempo que se requiere para implementar esta red sea muchísimo mayor, pero con la ventaja de tener absoluto control sobre el proceso.

Otros grandes exponentes son Amazon Reckognition, Amazon Textract, Google Vision Cloud, Microsoft Cognitive Services; son los mayores exponentes de visión por computadora, mostrando grandes ventajas por sobre el software libre, pues están dotados de decenas de plantillas, códigos, foros e inclusive dotan una solución integral en algunos casos asignando a personas físicas a realizar una segunda revisión de los resultados obtenidos

por sus algoritmos, lo que vuelve este tipo de soluciones excelente para empresas que buscan una solución con el respaldo de estos grandes corporativos. Sin embargo, estos servicios tiene un precio por imágenes o por tiempo de uso lo que vuelve a esta solución alcanzable solo para aquellos que puedan invertir en ella. A continuación se muestran estas soluciones con un pequeña descripción, ventajas y desventajas que se tienen con el uso de cada una de estas herramientas y finalmente la dificultad para lograr implementar estas soluciones.

II-A. Amazon Reckognition

Amazon Rekognition facilita la adición de análisis de imagen y vídeo a sus aplicaciones con tecnología probada, altamente escalable y de aprendizaje profundo que no requiere experiencia en aprendizaje automático para su uso. Con Amazon Rekognition se puede identificar objetos, personas, texto, escenas y actividades en imágenes y vídeos, además de detectar cualquier contenido en la imagen. Amazon Rekognition también proporciona análisis faciales de alta precisión y capacidades de búsqueda facial que puede usar para detectar, analizar y comparar rostros. Es posible implementar estos recursos en una amplia variedad de casos de uso vinculados con la verificación de usuarios, el conteo de personas y la seguridad pública.

Con las etiquetas personalizadas de Amazon Rekognition, se puede identificar objetos y escenas en imágenes específicas para casi cualquier necesidad. Las etiquetas personalizadas de Amazon Rekognition se encargan de hacer el trabajo duro de moldear el desarrollo para usted, gracias a esto no se necesita experiencia en aprendizaje automático. Simplemente necesita abastecer imágenes de objetos o escenas que quiera identificar y el servicio se encargará del resto.

EL incoveniente con este sistema es la limitante de reconocer únicamente 50 palabras por imagen, por lo que este sistema no es buena opción para un sistema de reconocimiento de texto pues se encuentra limitado a su salida.

II-B. Amazon Textract

Amazon Textract es un servicio que extrae automáticamente texto y datos de documentos escaneados. Amazon Textract no se limita al simple reconocimiento óptico de caracteres (OCR), sino que también identifica el contenido de campos en formularios e información almacenada en tablas.

Amazon Textract utiliza el aprendizaje automático para "leer" de manera instantánea prácticamente cualquier tipo de documento a fin de extraer texto y datos de forma precisa sin necesidad de cambios manuales ni utilizar código personalizado.

Gracias a la incorporación de Amazon Augmented AI (Amazon A2I), puede integrar revisiones humanas para administrar flujos de trabajo confidenciales o exigentes que requieran del juicio humano para lograr predicciones de alta fiabilidad, o para auditar predicciones de forma regular.

Este servicio es especializado en el reconocimiento óptico de caracteres que es capaz de extraer texto de documentos y formularios con una increíble precisión, aunque hay herramientas gratis como Google Tesseract y MODI para el caso de Microsoft. Esas soluciones requieren muchísimo arreglos, retoques y escenarios específicos para llegar a una respuesta decente, además de que el tiempo necesario para hacer funcionar esas herramientas es muchísimo mayor, se puede pasar horas atorado con Tesseract además de mostrar la paciencia del desarrollador mientras que con soluciones como textract poseen miles de plantillas gratis para usar y adecuarse a más problemas de manera fácil.

El problema principal con esta herramienta es el costo, pues no ofrece un modo de empleo gratuito ni por un periodo de tiempo ni por número de imágenes de entrada, para que así pequeñas empresas o los mismos desarrolladores se familiaricen con esta herramienta, además de poder contribuir a mejorar este servicio.

II-C. Google vision

Los servicios en la nube de Google también ofrecen una herramienta para el reconocimiento de texto OCR, cabe destacar que, de todas las plataformas, Google Vision demostró tener los mejores resultados en la extracción de texto de imágenes con poca resolución. Son muy pocos los pasos que se deben seguir para tener esta solución funcionando, además de estar muy

bien documentada esta herramienta lo cual facilita la implementación. Existe muchísima ayuda por parte de la empresa, además de miles de foros que dan pronta solución a cualquier problema que pueda surgir, sin embargo, si se siguen las instrucciones al pie de letra no debería haber ningún problema en poner en funcionamiento este sistema. Se debe crear una cuenta en Google services, para acceder a la consola. Una vez dentro se tiene acceso a muchas librerías disponibles directamente de Google.

El precio por otra parte es gratuito para los primero 1000 archivos al mes, sin embargo si se requiere más archivos por procesar se tendrá que pagar 1.5 dolares por cada 100 imágenes extra. Además actualmente Gooogle Cloud Vision ofrece un periodo de prueba que permite procesar hasta 200 mil archivos en un mes.

II-D. Microsoft Cognitive Services

Microsoft congnitive services son APIS, SDKs, y servicios disponibles para ayudar a desarrolladores a crear soluciones que involucren inteligencia artificial o (IA), sin tener un acercamiento puro a la IA o habilidades o conocimientos en la ciencia de los datos. Esta solución permite a los desarrolladores crear de manera fácil e intuitiva procesos de IA en sus aplicaciones. La meta de estos servicios es ayudar a desarrolladores a crear aplicaciones que puedan ver, oír, hablar, entender y hasta tener capacidad de la razón. El catálogo de servicios dentro de esta librería puede ser categorizada en cinco principales pilares: la visión, el habla, el lenguaje, la búsqueda en internet y el raciocinio.

Esta solución además viene con ejemplos de implementación a través de Jupyter Notebooks, lo cual es de gran ayuda. Es importante notar que esta solución no retorna texto plano como resultado. La única manera de obtener dicho texto es sacándolo directamente de un procesos previo desde el boxing de las palabras.

Hay mucha ayuda por parte de la empresa en foros, esta información es necesaria para poder avanzar en la implementación de este sistema, además existe una guía de inicio rápido para probar el algoritmo, pero antes de ello se requiere crear una cuenta en Azure Cloud, pasar por una serie de pasos para obtener las credenciales de uso, este paso no es tan complicado pero aún así es necesario, después de esto entonces ya muestra una interfaz en donde se puede empezar a crear código.

Un punto a tomar en cuenta siempre es el precio. Esta solución es gratuita para las primeras 5000 páginas por mes. Si se quisiera pasar más archivos, se tendrían que pagar 1.5 dolares por cada 100 archivos extras.

III. Marco Teórico

El objetivo de este proyecto es concretamente la extracción y reconocimiento de texto. Se busca, dado una imagen cuyo contenido sea texto con alfabeto latino o inglés, una extracción por partes segmentadas en las que podemos discriminar entre palabras y caracteres, luego podremos pasarlo por un programa de reconocimiento de patrones para establecer el caracter al que pertenece el objeto encontrado en la imagen. Este método de asignación deberá ser capaz de reconocer dicho caracter o arreglo de caracteres (palabra). Dicho proceso se lleva a cabo mediante un proceso llamado "boxing" con el cual podemos encerrar en una especie de contenedor las letras con el objetivo de tener claro dónde termina y dónde inicia un caracter con respecto a otro.

III-A. Imagen

Las imágenes digitales son capturadas mediante los dispositivos apropiados para su almacenamiento en un ordenador. Estos dispositivos poseen una distribución en forma de matriz de elementos denominados píxeles. Cada elemento se activa con la incidencia de la radiación espectral, generando un valor de tensión eléctrica a la salida que se convierte a un valor numérico. Todos estos valores, organizados igualmente en forma de matriz se almacenan en el ordenador. Este almacenamiento se lleva a cabo a través de algún formato característico de imágenes (`.tiff`, `.bmp`, `.jpeg`).

La imagen se representa desde el punto de vista de su tratamiento computacional como una matriz numérica de dimensión $M \times N$, es decir, con M filas y N columnas. El contenido de esa matriz se refiere a valores enteros situados en las localizaciones espaciales (x, y) o píxeles, dicho valor es el resultado de la cuantificación de intensidad o nivel de gris.

Si la imagen es en blanco y negro, se almacena un valor por cada píxel. Este valor es el nivel de

intensidad o nivel de gris comentado anteriormente. Se suele utilizar un rango de valores para su representación, que generalmente es de 0 a 2^n-1. Uno de los valores más utilizados es con $n = 8$; esto significa que el rango de valores para este caso varía entre 0 y 255. En este caso, el 0 representa el negro absoluto y 255 blanco absoluto.

Sobre cada matriz se establece un sistema de coordenadas con origen normalmente en la esquina superior izquierda y con los ejes x y y tales que su orientación es positiva hacia la derecha en el caso del eje x y hacia abajo en el caso del eje y. Cuando la imagen es en color, para cada localización espacial existen tres valores de intensidad asociados, es decir, los elementos de la matriz vienen dados por tres valores que representan cada uno de los componentes básicos del color en cuestión. Estos componentes son el rojo (R), verde (G), y azul (B) y es lo que conocemos como código RGB. En este caso el conjunto de valores $(0, 0, 0)$ representa al negro, mientras que los valores $(255, 255, 255)$ es el blanco absoluto. La combinación de distintos valores representa otros colores. Debido a lo anterior, una imagen en color posee tres bandas espectrales: rojo, verde, azul; cada una de ellas viene representada por una matriz de números con valores en el rango $[0, 255]$ para imágenes de 8 bits.El píxel que se encuentra en la localización espacial (x, y) posee componentes (R, G, B). Los tres valores de cada píxel para las imágenes en color corresponden al modelo de color RGB.

Servicio	Mano escritos	Promedio
A. Rekognition	41 %	45 %
A. Textract	39 %	54 %
Google	83 %	84 %
Microsoft	90 %	94 %

TABLE I

TABLA DE EMPRESAS CON SOLUCIÓN OCR.

III-B. Visión por computadora

Se trata de un campo de la informática dedicado a replicar las partes complejas de la visión del ser humano computacionalmente, mediante algoritmos que emulan los resultados que tendría el ser humano, para reconocer objetos, figuras, tamaños, orientación, todo aquello que damos por sentado por ser parte del ser humano, la visión por computadora además tiene el poder de hacer estas tareas algunas veces mejor que el ser humano, ya que mediante técnicas de mejoramiento de imagen puede obtener mucha más información, aunque esto no siempre signifique algo bueno para el procesamiento de la imagen.

III-C. Machine Learning

Se trata de un campo dentro de la inteligencia artificial (IA), en donde el objetivo de un programa es que aprenda por sí mismo, es decir, que mediante la experiencia acumulada en el tiempo de ejecución pueda llegar a una solución intrínseca, haciendo prueba y error. Mediante esta técnica se puede hacer que la computadora aprenda por sí sola, casi sin ayuda humana, sin embargo, el tiempo que toma y las reglas que se deben escribir deben ser muy restrictivas si se quiere tener un buen rendimiento.

III-D. Deep Learning

Se trata de otra rama de IA en la que se hace uso de una base datos, llamada dataset, con lo que se logra entregar a un algoritmo de computador a reconocer ciertos patrones, figuras, o inclusive figuras compuestas. El término deep proviene del hecho en el que el algoritmo es segmentado por capas que se encuentran apiladas una encima de otra.

III-E. OpenCV

Es una herramienta de software libre que nos permite tener una mejor manipulación de las imágenes convertidas en matrices, además de tener libre acceso a software existente sobre umbralizado, detección de bordes, detección de objetos, detección de dimensiones de una imagen, en general es una herramienta poderosa de libre uso. OpenCV soporta varios modelos de deep learning.

III-F. TensorFlow

Tensorflow es una de las librerías más completas de aprendizaje por computador en donde se puede crear con relativa facilidad redes neuronales, entrenar dicha red y comenzar a obtener resultados de una implementación bastante flexible, pues permite tener completo control

sobre el procesos de diseño, de las capas y los métodos de activación. Además al ser software libre no se debe pagar licencia por su uso.

III-G. Google Tesseract

Google ofrece una solución integral que ya provee una gran precisión y facilidad de uso, sin embargo, al ser una solución integral no se tiene acceso al código que da vida a este algoritmo, pues al igual que la solución que se describe en este reporte hace uso de un preprocesamiento de la imagen, luego genera el boxing, hace los recortes y transformaciones pertinentes para finalmente introducir estas pequeñas imágenes en su red neuronal a la que no se tiene acceso. Finalmente, se despliegan los resultados obtenidos del texto extraído.

III-H. Google Colab

Colab es una herramienta gratuita de Google que permite ejecutar y programar en Python desde un entorno de navegador, algunas de las ventajas de su uso son, por mencionar algunas, que no requiere configuración, ofrece acceso gratuito a GPUs, se puede compartir fácilmente, tiene completa compatibilidad con toda la suit de Google; esto quiere decir que se puede tener almacenamiento en Drive para estos trabajos. Por otro lado facilita y mejora los tiempos de entrenamiento, ya que un computador normal no está preparada para un entrenamiento excesivo. Además, no solo se ofrecen GPUs, sino también se ofrecen TPUs, que es la tecnología patentada de Google, basada en el uso de procesadores con una arquitectura enfocada en tensores. Esto facilita este tipo de problemas, pues se trata de hardware especializado para dichas tareas, y, por si no fuese suficiente, además se trabaja con un sistema operativo que te permite instalar librerías, hacer uso de repositorios, uso de Github, entre otros beneficios. Cabe mencionar que la función imshow de la librería de OpenCV no está soportada, pues esta requiere una ventana emergente, pero se puede cambiar este tipo de ventanas a gráficas con la librería Matplotlib.

IV. METODOLOGIA

Para esta implementación es necesario detallar las distintas etapas del proyecto que se llevan hasta el momento. Estas etapas incluyen la binarización de la imagen, búsqueda de objetos, inserción de efectos visuales para la evaluación de la efectividad del método y el uso de la plataforma GitHub para mantener un mejor desarrollo del proyecto. En el proyecto se ocupan distintos métodos y distintas funciones, por lo que se cuentan con códigos de pruebas y códigos que se ocupan a modo de librerías. La implementación se está realizando con las herramientas de OpenCV y se está desarrollando enteramente en Python.

Como segunda instancia se emplearon herramientas como TensorFlow y Keras. Se hizo una comparación resultados contra librerías completas como Google Tesseract, para contrastar la eficiencia y presión de este sistema que se está desarrollando. Para la solución de este sistema se siguió una linea de 5 etapas en las que cada etapa aporta un proceso crítico para el sistema. El modelo propuesto para el entrenamiento e identificación de los caracteres dentro de la imagen fue ejecutado con Google Colab.

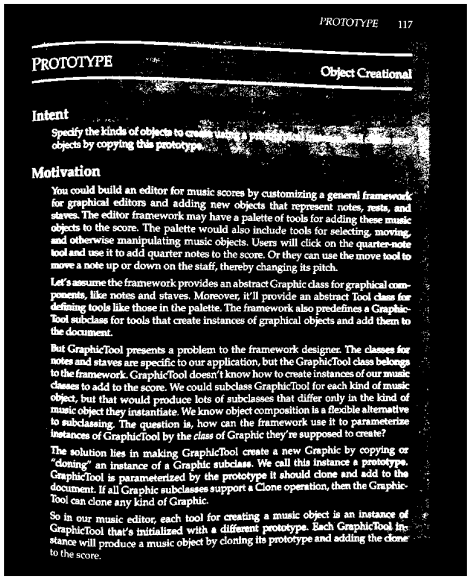
Se implementaron varios métodos durante las distintas etapas del proyecto. En la primer etapa, binarización, se implementaron métodos como Niblack, Wolf y Sauvola, y se tomaron en cuenta parámetros como el costo computacional que tenía el algoritmo y los resultados mismos que ofrecían. En la segunda etapa, búsqueda de objetos, se implementó el algoritmo de etiquetado secuencial de componentes. Para la tercer etapa se busca hacer una agrupación de objetos, ya que los objetos que se encuentran en la imagen suelen ser únicamente los caracteres y es necesario poder hacer segmentaciones bajo distintos criterios, es decir, poder obtener regiones donde se englobe una sola palabra, una sola línea de texto o un párrafo. Posteriormente, en la tercer etapa, se añaden efectos visuales simples en los que se colorean los objetos de modo que es más sencillo para el usuario diferenciarlos y evaluarlos, ya que de esta forma se obtiene con una manera gráfica de diferenciar los distintos objetos identificados en la etapa previa, ya sean caracteres, líneas o párrafos. Finalmente llega la etapa de obtener los caracteres escritos y poder saber clasificarlos correctamente. Se propone utilizar redes neuronales para hacer esta decodificación y clasificación de los caracteres que hay dentro de las imágenes. Para esto se implementó

código que se manda a llamar cuando se quiera inferir lo que se encuentra escrito dentro de la imagen.

Se validarán los métodos utilizados con imágenes generadas por los algoritmos desarrollados, así como los resultados obtenidos de proyectos que ya están completados y permiten obtener el reconocimiento de caracteres de manera fácil, de forma que se analizarán los resultados entregados por nuestro método y se contrastan contra los resultados obtenidos por otros métodos. Es por esta validación que se hizo la implementación de la etapa 4, efectos visuales, ya que sin esto no sería tan sencillo evaluar la efectividad de los distintos algoritmos que se implementaron. Aunque en la sección V se hace contrastan los resultados de la implementación y se hace la validación de la respuesta al proyecto, en la descripción de la metodología se irán desarrollando los resultados de cada una de las etapas previas a la etapa final, a modo de que el documento sea lo más comprensible y su replicación pueda ser ejecutada con sencillez, así como poder ofrecer una análisis más detallado de las decisiones respecto a los algoritmos a utilizar.

IV-A. Preprocesamiento

En esta etapa se evaluaron varios métodos en los que se usa un umbral adaptativo. El primer método a revisar es un median filter, el cual filtra la imagen y elimina ruido en la misma. Esto permite tener una mejor binarización de la imagen. En la binarización de la imagen se utilizan los métodos de Niblack (Fig. 1) y Wolf (Fig. 2). Ambos métodos buscan hacer la binarización de la imagen mediante el uso de ventanas y la convolución con ellas, sin embargo, las reglas para Niblack y para Wolf son distintas. De acuerdo a [2], Sauvola se desarrolla como una mejora a Niblack y Wolf, sin embargo, se decidió utilizar el método de Wolf, ya que mostró ser el más robusto con las imágenes con las que se hicieron las pruebas. Ya que se tiene la imagen binarizada, se pueden distinguir las letras en color blanco y el fondo de color negro, así permitiéndonos una mejor futura detección de las letras. Es importante destacar que aunque la imagen que se obtiene de entrada, la imagen que tenemos actualmente y con la que se trabajará de ahora en adelante se encuentra en un solo canal, es decir, escala de grises.


PROTOTYPE 117

PROTOTYPE

Object Creational

Intent

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Motivation

You could build an editor for music scores by customizing a general framework for graphical editors and adding new objects that represent notes, rests, and staves. The editor framework may have a palette of tools for adding these music objects to the score. The palette would also include tools for selecting, moving, and otherwise manipulating music objects. Users will click on the quarter-note tool and use it to add quarter notes to the score. Or they can use the move tool to move a note up or down on the staff, thereby changing its pitch.

Let's assume the framework provides an abstract Graphic class for graphical components, like notes and staves. Moreover, it'll provide an abstract Tool class for defining tools like those in the palette. The framework also predefines a GraphicTool subclass for tools that create instances of graphical objects and add them to the document.

But GraphicTool presents a problem to the framework designer. The classes for notes and staves are specific to our application, but the GraphicTool class belongs to the framework. GraphicTool doesn't know how to create instances of our music classes to add to the score. We could subclass GraphicTool for each kind of music object, but that would produce lots of subclasses that differ only in the kind of music object they instantiate. We know object composition is a flexible alternative to subclassing. The question is, how can the framework use it to parameterize instances of GraphicTool by the *class* of Graphic they're supposed to create?

The solution lies in making GraphicTool create a new Graphic by copying or "cloning" an instance of a Graphic subclass. We call this instance a prototype. GraphicTool is parameterized by the prototype it should clone and add to the document. If all Graphic subclasses support a Clone operation, then the GraphicTool can clone any kind of Graphic.

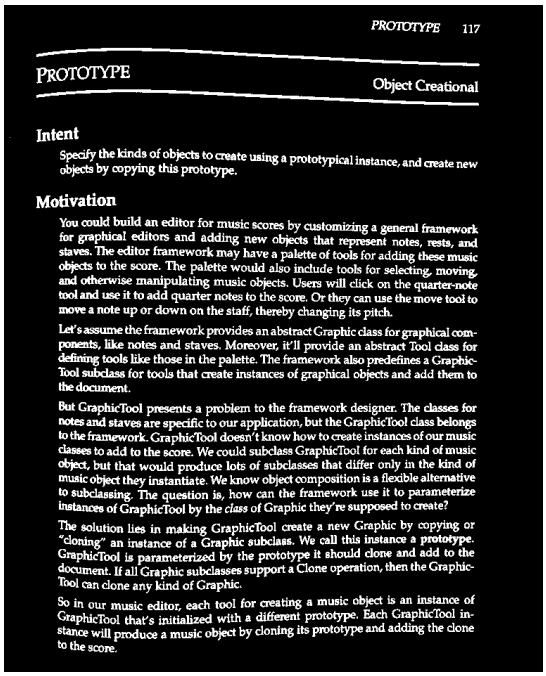
So in our music editor, each tool for creating a music object is an instance of GraphicTool that's initialized with a different prototype. Each GraphicTool instance will produce a music object by cloning its prototype and adding the clone to the score.

Fig. 1. Imagen Binarizada ocupando Niblack.

El algoritmo que se está ocupando en la etapa actual del proyecto es Wolf, esto debido a que por mucho es el mejor en los casos en los que la foto a la imagen es la mejor posible (sin demasiado ruido en el fondo), para dicho algoritmo se está usando la implementación de OpenCV. Un componente crítico que se utilizó para decidirse por este algoritmo, fue que el proyecto fue delimitado a tener imágenes de entrada en las que el fondo sea liso, es decir, que no tenga una cuadrícula o que no sea una hoja rayada. Así mismo, cabe destacar que se hicieron pruebas de binarización aplicando múltiples Niblack, como se puede ver en la Fig. 3, con ventanas adaptativas con un incremento en cada etapa, sin embargo, el coste computacional, calculado en una computadora Dell G7 7790 con 16GB de RAM y un Intel Core I7 de novena generación, era de alrededor $28s$, mientras que el método de binarización de Wolf ofrecido en las librerías de OpenCV tiene un costo computacional de $20ms$.

IV-B. Segmentación

El proceso de segmentación busca crear particiones de la imagen en regiones donde puedan existir objetos de



PROTOTYPE 117

PROTOTYPE Object Creational

Intent

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Motivation

You could build an editor for music scores by customizing a general framework for graphical editors and adding new objects that represent notes, rests, and staves. The editor framework may have a palette of tools for adding these music objects to the score. The palette would also include tools for selecting, moving, and otherwise manipulating music objects. Users will click on the quarter-note tool and use it to add quarter notes to the score. Or they can use the move tool to move a note up or down on the staff, thereby changing its pitch.

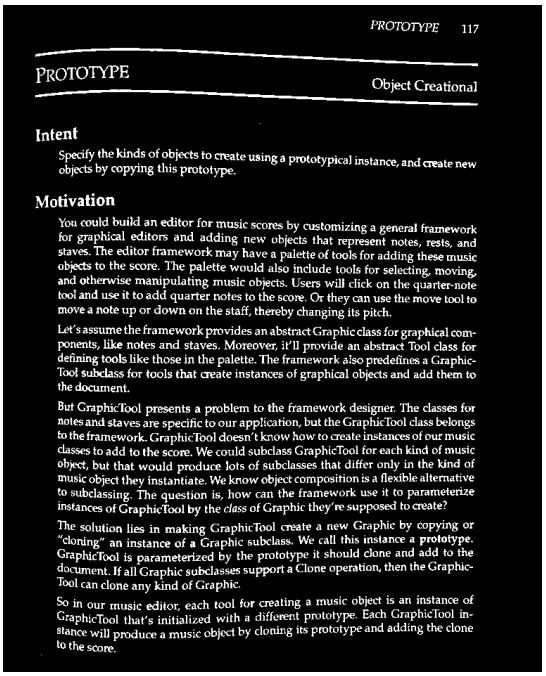
Let's assume the framework provides an abstract Graphic class for graphical components, like notes and staves. Moreover, it'll provide an abstract Tool class for defining tools like those in the palette. The framework also predefines a GraphicTool subclass for tools that create instances of graphical objects and add them to the document.

But GraphicTool presents a problem to the framework designer. The classes for notes and staves are specific to our application, but the GraphicTool class belongs to the framework. GraphicTool doesn't know how to create instances of our music classes to add to the score. We could subclass GraphicTool for each kind of music object, but that would produce lots of subclasses that differ only in the kind of music object they instantiate. We know object composition is a flexible alternative to subclassing. The question is, how can the framework use it to parameterize instances of GraphicTool by the *class* of Graphic they're supposed to create?

The solution lies in making GraphicTool create a new Graphic by copying or "cloning" an instance of a Graphic subclass. We call this instance a prototype. GraphicTool is parameterized by the prototype it should clone and add to the document. If all Graphic subclasses support a Clone operation, then the GraphicTool can clone any kind of Graphic.

So in our music editor, each tool for creating a music object is an instance of GraphicTool that's initialized with a different prototype. Each GraphicTool instance will produce a music object by cloning its prototype and adding the clone to the score.

Fig. 2. Imagen Binarizada ocupando Wolf.



PROTOTYPE 117

PROTOTYPE Object Creational

Intent

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Motivation

You could build an editor for music scores by customizing a general framework for graphical editors and adding new objects that represent notes, rests, and staves. The editor framework may have a palette of tools for adding these music objects to the score. The palette would also include tools for selecting, moving, and otherwise manipulating music objects. Users will click on the quarter-note tool and use it to add quarter notes to the score. Or they can use the move tool to move a note up or down on the staff, thereby changing its pitch.

Let's assume the framework provides an abstract Graphic class for graphical components, like notes and staves. Moreover, it'll provide an abstract Tool class for defining tools like those in the palette. The framework also predefines a GraphicTool subclass for tools that create instances of graphical objects and add them to the document.

But GraphicTool presents a problem to the framework designer. The classes for notes and staves are specific to our application, but the GraphicTool class belongs to the framework. GraphicTool doesn't know how to create instances of our music classes to add to the score. We could subclass GraphicTool for each kind of music object, but that would produce lots of subclasses that differ only in the kind of music object they instantiate. We know object composition is a flexible alternative to subclassing. The question is, how can the framework use it to parameterize instances of GraphicTool by the *class* of Graphic they're supposed to create?

The solution lies in making GraphicTool create a new Graphic by copying or "cloning" an instance of a Graphic subclass. We call this instance a prototype. GraphicTool is parameterized by the prototype it should clone and add to the document. If all Graphic subclasses support a Clone operation, then the GraphicTool can clone any kind of Graphic.

So in our music editor, each tool for creating a music object is an instance of GraphicTool that's initialized with a different prototype. Each GraphicTool instance will produce a music object by cloning its prototype and adding the clone to the score.

Fig. 3. Imagen Binarizada ocupando cinco Niblacks.

interés. Esto se hace para poder obtener interpretaciones semánticas de las imágenes, o imágenes que sean más sencillas de analizar. Para relizar esta tarea existen múltiples algoritmos que pueden ser empleados, como etiquetado secuencial de componentes 4-connectivity. El método que se escogió fue etiquetado secuencial ya que es muy simple de entender y relativamente sencillo de implementar en código.

En este algoritmo se escanea la imagen binarizada de izquierda a derecha, de arriba a abajo, y, en caso de encontrar un píxel con el valor deseado y solamente uno de los vecinos superior o izquierdo tienen etiqueta o la etiqueta sea la misma, se copia la etiqueta; en caso de que tengan etiquetas distintas, se copia la etiqueta del vecino superior y se asocian ambas etiquetas en una tabla de equivalencias como etiquetas equivalentes, es decir, que ambas etiquetas pertenecen al mismo objeto. En caso de que ninguna de las condiciones anteriores se cumplan, se asigna una etiqueta nueva y se introduce la nueva etiqueta en la tabla de equivalencias. Esto nos indica que es un nuevo objeto. Esto se repetirá hasta analizar todos los píxeles que hay en la imagen y se ejemplifica en la Fig. 4.

Una vez terminado el análisis descrito anteriormente,

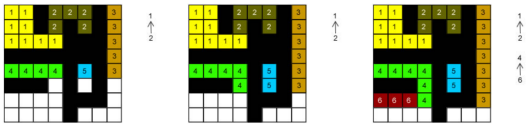


Fig. 4. Ejemplo simple del etiquetado de componentes.

se procede reemplazar las etiquetas asociadas como equivalentes por la etiqueta menor, homogeneizando así las etiquetas de los objetos encontrados y obteniendo una segmentación adecuada. Esto se muestra en la Fig. 5.

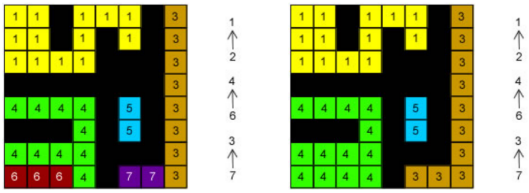


Fig. 5. Equivalencia etiquetas de los objetos.

IV-B.1. Propiedades de objetos: **Al buscar los objetos se interesa encontrar también las características que los definen. Una de estas características son los puntos que delimitan el rectángulo, o caja, más pequeño en el que cabe el caracter en cuestión. Posterior a que los objetos han sido encontrados y que se han delimitado**

sus cajas, se pueden tratar con la misma filosofía detrás de la programación orientada a objetos, pues todos estos tendrán las mismas características y pueden ser englobados en una clase que le llamaremos CropBox, y constará de los componentes x y y mínimas y máximas.

IV-C. Clustering

Una vez que ya se cuenta con una lista de los objetos encontrados (también conocidos como Crop Boxes), ahora se buscará hacer agrupaciones de dichos objetos con el fin de así poder delimitar qué objeto corresponde a cierta región. A estas agrupaciones las denominamos clusters. Las regiones que más nos importan serán: caracter forma parte de una palabra, palabra forma parte de una línea, línea forma parte de un párrafo y párrafo forma parte de una sección de texto. Esto es importante para posteriormente poder mandar los datos a la red neuronal en el formato que esta lo requiera para que pueda realizar una correcta decodificación de los caracteres.

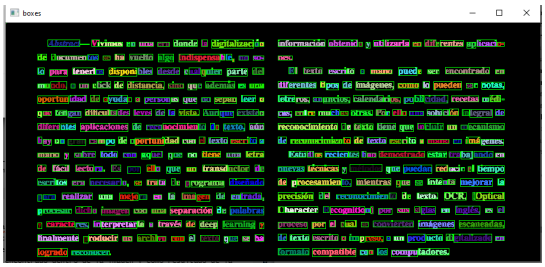


Fig. 6. Ejemplo de clustering a doble columna.

Para lograr el clustering, es necesario identificar algunos parámetros y, de alguna forma, limitaciones que tendrá nuestro algoritmo. Para identificar que un caracter forma parte de una palabra, es necesario hacer un promedio de la separación horizontal que hay entre cada uno de los Crop Boxes que se encuentran en una misma línea. Para identificar si un CropBox está dentro de una misma línea, basta con comparar los atributos Y mínima y máxima en la lista de Crop Boxes que se ha encontrado dentro de la imagen y como resultado de la sección IV-B. Si están dentro de cierto rango, se puede asumir que esos objetos pertenecen a la misma línea. Los párrafos se construyen bajo esta misma filosofía, pero la separación entre líneas debe de ser mayor al promedio de separación entre líneas.

Retomando la separación en X, se separan las líneas en palabras obteniendo la media y comparando la distancia entre los dos objetos a agrupar. En caso de que la distancia sea superior al promedio más un pequeño umbral, se determina que el caracter ya pertenece a otra palabra. Esto nos permitirá obtener las agrupaciones de caracteres para formar palabras.

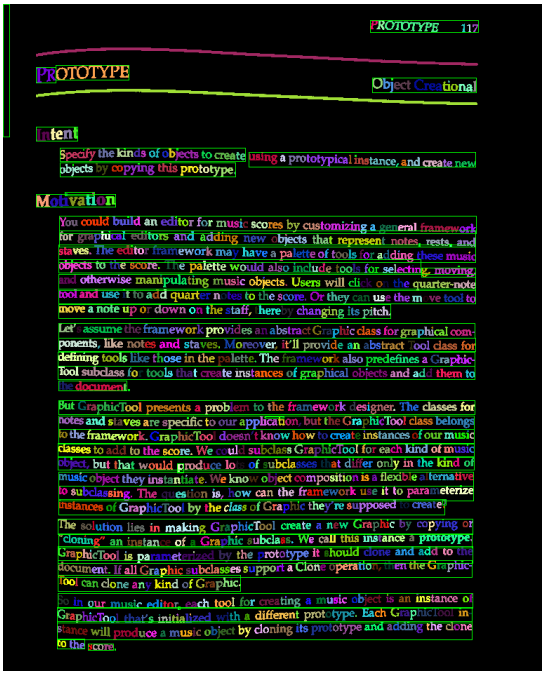


Fig. 7. Clustering de objetos para formar líneas.

Cabe mencionar que esto no esta preparado para cuando se ponen imágenes con mucho ruido o que estén rotadas, ya que en la etapa de preprocesamiento no está diseñada para poder aceptar y corregir imágenes en las que el texto no esté en la orientación adecuada, es decir, que el texto no esté escrito horizontalmente y de izquierda a derecha, ya que si se introducen imágenes con el texto rotado $180°$, en vertical, diagonal o curvo, la etapa de clustering no podrá desempeñar su función correctamente, además de que la red neuronal que se explicará más adelante tampoco podrá decodificar correctamente los caracteres, como es mostrado en las Fig. 8 y Fig. 9.

IV-D. Efectos Visuales

Finalmente, se añaden efectos visuales a la imagen en los que se muestran los distintos objetos encontrados,

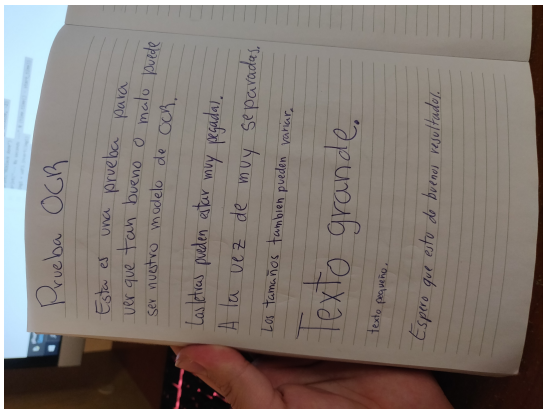


Fig. 8. Ejemplo de imagen no adecuada al sistema.

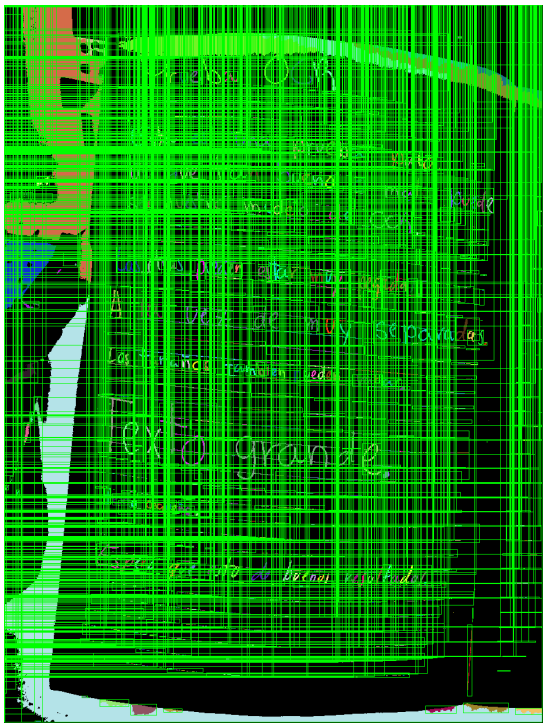


Fig. 9. Clustering fallido.

así como las cajas que delimitan cada uno de estos objetos. Esto se hace principalmente para poder tener una ayuda visual y evaluar la efectividad de cada uno de los algoritmos con las imágenes de prueba. Gracias a estos efectos visuales es que se notaron las limitaciones que tiene la solución implementada y que se mencionaron en la sección previa, pues al mandar diferentes imágenes de prueba se obtenían resultados distintos a los esperados al introducir diferentes tipos de imágenes. Contrastando las imágenes de la Fig.2 contra la imagen de la Fig.7 se pueden notar claramente los efectos visuales que se han añadido a nuestro procesamiento para poder identificar con mayor facilidad las fallas o limitaciones que se tienen en la solución implementada.

IV-E. Red Neuronal

Como último paso se tiene que decodificar y clasificar correctamente los caracteres u objetos que se hayan encontrado en la imagen. Para esto se propusieron varios métodos que involucran el aprendizaje por computadora. Se buscó implementar usando APIs como Keras y Tensorflow, como fue previamente mencionado. Cabe destacar que el entrenamiento de la red neuronal implementada fue hecho en los servicios de Google Colab, pues el entrenamiento requiere de mucho tiempo y exige mucho a los procesadores.

IV-E.1. Dataset: **Previo a empezar a implementar una red neuronal, es importante hacer una recopilación de los datos que se usarán para entrenarla. Estas recopilaciones de datos se conocen como datasets y pueden estar construidos de muchas maneras distintas. Algunos de los datasets más populares son los de NIST, ampliamente usado en muchos ejemplos usando Keras; IAM Database, que es una recopilación de datasets escritos a mano; y MNIST, un subset del dataset general de NIST para caracteres numéricos. Además se trata de una de las etapas más críticas en el buen funcionamiento de una red neuronal pues de la calidad, cantidad y especialización del dataset depende la precisión que pueda tener dicho algoritmo, pues trata de buscar la base de datos que sea más adecuada o parecida al tipo de letra que se intenta reconocer, pues de aquí se irán extrayendo las características en las que la red se debe fijar para discernir entre caracteres.**

MNIST y IAM Database contienen los archivos en cierta organización, así como información del Ground Truth, es decir, lo que dice cada una de las imágenes. Mientras MNIST está hecho para crear un reconocimiento de un solo caracter numérico al momento, IAM Database tiene datasets con letras, palabras o frases enteras. Esto es algo importante a notar, ya que lo que queremos reconocer son palabras escritas. Es cierto que se forman a partir de caracteres, sin embargo, si se

adecúa la red para que identifique palabras en lugar de únicamente reconocer caracteres, las operaciones realizadas para inferir lo que haya escrito en una imagen pueden reducirse considerablemente, de forma que no se invoca tantas veces como si se implementa con una base de datos de únicamente caracteres.

El dataset de NIST es un dataset en el que no se encontró la estructura adecuada y pese a que tiene un archivo md5 en donde se habla sobre la organización de las clases, tuvimos bastantes complicaciones en tan solo el procesamiento de imágenes, estas siendo mas de un millón 500 mil. Todo lo anterior mencionado ocasiona que este dataset no contemos con una Ground Truth, por lo que marca una diferencia fundamental con los que se mencionaron previamente. Al usar un dataset que contiene un archivo de Ground Truth, es mucho más sencillo cotejar el resultado que entrega la red neuronal y se pueden ajustar mucho mejor los parámetros internos de la red, de forma que aprenda con mayor rapidez. Este aprendizaje se le conoce como aprendizaje supervisado.

Se escogió el dataset de IAM Database para esta implementación ya que se puede realizar un aprendizaje supervisado y puede reducir el número de inferencias que se hacen para obtener el resultado final de la decodificación del texto en la imagen. Es importante destacar que las imágenes del dataset tienen un tamaño similar entre todos, pero no está homogeneizado, por lo que al momento de mandar estas imágenes a la red neuronal deben de enviarse con el mismo formato todos, así como con el mismo preprocesamiento que se aplique a la imagen que se vaya a decodificar. En la Fig. 10 se muestra un ejemplo de las imágenes que se encuentran dentro de este dataset.

Fig. 10. Imagen extraída del dataset seleccionado.

IV-E.2. Modelo: **Una red neuronal computacional tiene la capacidad de imitar el funcionamiento de las redes neuronales de los organismos vivos: un conjunto de neuronas conectadas entre sí y que trabajan en conjunto, sin que haya una tarea concreta para cada una. Con la experiencia, las neuronas van creando y reforzando ciertas conexiones para "aprender" algo que se queda fijo en el tejido.**

Existen tres métodos de activación principales en los cuales son relu, softmax y sigmoid, estos métodos engloban distintos tipos de gráficas que permiten tener control sobre la activación de las neuronas individualmente, lo importante a tomar en cuenta de estos métodos de activación es que delimitan en cierta forma el parámetro que puede dar como resultado una neuronas dichos límites tradicionalmente son llevados entre -1 y 1. El método de activación más popularmente usado es relu, que por sus siglas en inglés significa Rectified Linear Unit (Unidad lineal rectificada), ya que la función sigmoid no presenta los mismos resultados que relu, que ha demostrado tener mayor precisión en las predicciones. El método de Softmax es empleado sobre todo en las últimas capas de una red neuronal pues su principal característica es potenciar un solo resultado haciendo que los demás resultados sean disminuidos en magnitud con el fin de disminuir la incertidumbre por valores cercanos en neuronas cercanas, es decir que fomenta las diferencias entre las neuronas dando un solo resultado con probabilidad diferenciada a los demás resultados.

Se decide implementar un modelo basado en una Red Neuronal Convolucional en Tensorflow, ya que estas tienen un gran desempeño en interpretación semántica de imágenes [1], lo cual conlleva un fuerte proceso de segmentación. La red buscará extraer las características de cada uno de los objetos y clasificarlos correctamente, es decir, aprender cuáles son las características de ciertos caracteres en específico. Esto es importante ya que el desempeño que se pueda esperar de esta red depende directamente de la topología que sea elegida, claro que no en todos los casos una misma topología funciona en las mejores condiciones, sin embargo se busca generalizar una red que pueda obtener buenos resultados en la mayoría de los escenarios posibles.

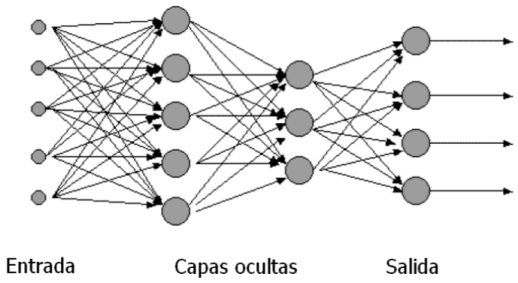


Fig. 11. Modelo simple de una red neuronal.

El principal problema con el texto escrito a mano es que existe una cantidad infinita de estilos de escritura y tipos de fuentes por lo que hacer una red lo suficientemente robusta se presenta como un desafío grande por resolver. Con la implementación con APIs es posible manipular de una manera relativamente sencilla la topología que se aplica en la construcción y diseño de un modelo confiable, se puede modificar el tamaño del modelo, el número de capas, los métodos de propagación, los métodos de activación, el número de neuronas por capas y sobre todo se puede tener implementaciones de distintas configuraciones en un mismo modelo a fin de mejorar el sistema de reconocimiento, recordando siempre que os cambios que se apliquen a la topología de la red afectan de manera directamente proporcional al tiempo de entrenamiento de la red y al tiempo de ejecución para la correcta identificación y clasificación de los caracteres.

Para mejorar el sistema de reconocimiento se debe pasar los elementos a reconocer a un formato de one-hot en donde se busca que la diferencia entre cada elemento sea de un solo dígito, además de que sea lo suficientemente excluyente que no tenga similitud y que no se pueda confundir el resultado, esto se logra utilizando un paradigma de unos anidados con un código especial para cada caracter.

Otro término importante es el método de propagación de la información entre neuronas, y se trata principalmente de como se interconectan unas con otras las neuronas y las capas en las que están organizadas. Esto representa distintos métodos de agrupación y propagación pues se puede desde solo conectar salida con entrada entre etapas, o hacer una conexión completa entre capas y neuronas de manera que sea bidireccional la información.

Las redes neuronales se basan el el concepto del descenso del gradiente y es por ello que el paso de búsqueda de mínimo local o global es importante cuando se habla de implementar una red sobre todo funcional, es decir que se debe elegir un tamaño de paso que sea lo suficientemente grande para que nuestro algoritmo no se quede estancado en un mínimo local que pudiera no ser el mínimo global, pero que tampoco sea tan grande que no pueda alcanzar el mínimo global debido a que estaría brincando al rededor de dicho mínimo sin poder alcanzarlo exitosamente, lo cual implica no solo más poder computacional o mayor tiempo de computo, sino la integridad del resultado pues podría arrogar resultados completamente erróneos.

IV-E.3. Modelo empleado: **Las imágenes que se utilizarán dentro de la red neuronal tienen dimensión de 128×32, por lo que es necesario adecuar la topología de la red a este dato y tomarlo en cuenta para cuando se quiera inferir lo que haya escrito en alguna imagen. Así mismo, la red debe ser entrenada con la misma binarización que se ocupa en el preprocesamiento, ya que de no hacerlo así, es casi seguro que la red falle y no desempeñe correctamente su papel, ya que nunca habrá visto imágenes como las que se le está pidiendo que infiera y pueda que su ejecución se vea afectada.**

Como se ha mencionado, la red está integrada principalmente por una Red Neuronal Convolucional (CNN) de 5 capas con la topología presentada en la Tabla IV-E.3. En cada etapa se tendrá una activación por medio de relu y se hará un pool de máximos, que es lo más usado en las redes neuronales modernas.

Capa	Kernel	Pool
1	5×5	2×2
2	5×5	2×2
3	3×3	1×2
4	3×3	1×2
5	3×3	1×2

TABLE II

TABLA DE EMPRESAS CON SOLUCIÓN OCR.

Al finalizar la quinta etapa de la CNN, se tienen

un total de 256 canales que serán la alimentación a una Red Neuronal Recursiva (RNN) de dos capas. Esta RNN es bidireccional, de forma que se obtienen dos secuencias en la salida que tendrán un tamaño de 32×256 y se concatenan para finalmente ser mapeada a la secuencia de salida con tamaño de 32×80 y que será con lo que se alimente la siguiente etapa. La siguiente línea de código ejecuta lo descrito anteriormente, justo antes de la concatenación. `((fw, bw), _) = tf.nn.bidirectional_dynamic_rnn(cell_fw= stacked, cell_bw=stacked, inputs=rnnIn3d, dtype=rnnIn3d.dtype)`

En esta etapa llamada Clasificación Temporal de Conexiones (CTC) se calcula la pérdida de la matriz de salida de la RNN respecto al Ground Truth que hay de los textos en las imágenes. En la siguiente línea de código se hace este cálculo. `self.loss = tf.reduce_mean(tf.nn.ctc_loss(labels =self.gtTexts, inputs=self.ctcIn3dTBC, sequence_length = self.seqLen, ctc_merge_repeated=True))`. Este valor calculado es importante para el entrenamiento, ya que a partir de este es que se ajustan los parámetros internos del modelo, es decir, los weights and biases. Asimismo, la decodificación del contenido de la imagen es dado por la línea de código que se presenta a continuación. `self.decoder = tf.nn.ctc_greedy_decoder(inputs= self.ctcIn3dTBC, sequence_length= self.seqLen)`

Finalmente, se calcula el promedio de la pérdida en cada uno de los elementos dentro del batch que se usa para entrenar y se manda como parámetro cuando se ejecuta la siguiente línea `self.optimizer = tf.train.RMSPropOptimizer (self.learningRate).minimize(self.loss)`. Gracias a esta línea es que se actualizan los parámetros del modelo y al ir minimizando la pérdida es que el sistema aprende.

Para verificar que el modelo esté mejorando y para entrenarlo, se hicieron pruebas con 25000 datos y 75000 datos en batches de 50 datos. En ambos casos se llegó a un error en el reconocimiento de caracteres de alrededor de 11 %, pero la diferencia de épocas en las que se converge a este resultado es distinta. En el primer caso tardaba alrededor de 40 épocas en converger y no mejorar más, mientras que en el segundo caso bastaron 19 épocas para converger al mismo resultado. Del total de los datos, se usaron 95 % de estos para entrenar y el restante para validar los datos del entrenamiento. Esta validación es precisamente la que nos permite calcular los resultados previamente mencionados.Finalmente, si el modelo tiene mejoras, se guarda el modelo para cuando se busque inferir a partir del modelo entrenado. El modelo que se ha implementado es capaz de decodificar hasta 32 caracteres dentro de una sola imagen.

IV-F. Procesamiento final

Ya en la etapa final, por cada predicción de cada palabra se observa cual es la calificación de exactitud que nos da la red neuronal y si detectamos que este numero esta por debajo del 70 % la palabra la procesamos con la librería de 'SpellChecker' y hacemos las correcciones pertinentes.

Una vez ya procesadas todas las palabras, eliminamos las probabilidades de exactitud (esperando que estas ya sean lo mas altas posibles) e imprimimos las palabras para desplegar cual fue la predicción final.

V. Resultados Experimentales y análisis

Lo primero en la agenda era mejorar la calidad de la imagen de entrada, para ello se dispuso de un sistema de diferentes tamaños de máscaras que luego de haberse ejecutado sobre la imagen original se calcula el promedio de dichas umbralizaciones para poder así tener un mejor acercamiento del texto.

Además a modo de contraste se realizó una prueba con una sola umbralización con una máscara de 257 aquí el resultado en donde podemos observar como claramente el método del promedio de cinco umbralizaciones es mejor.

Los mejores resultados que se han obtenido del umbralizado han sido usando el método de Wolf, ocupando una ventana de 127 píxeles y una k de 0.2; sin embargo, no se ha mostrado 100 por ciento efectiva en los escenarios en donde hay ruido de fondo.

Los objetos se encuentran al poner etiquetas sobre los pixeles que no son parte del fondo. Además de las etiquetas usadas para la creación de objetos, se les otorgan atributos. Estos atributos son una x y y mínimas y máximas, lo cual permitirá la creación de las cajas y permitirá mantener uniformidad en el código y un mejor manejo de los objetos de interés, lo cual será crítico al momento de enviarlos a la red neuronal que se implemente. Posterior a esto se añaden los efectos visuales para poder evaluar los resultados de los algoritmos implementados.

Finalmente se procede a hacer una agrupación o clustering de los objetos encontrados. Esto nos permite crear objetos nuevos a partir de los que ya se tienen. En estos objetos se pueden identificar líneas enteras de texto como si fueran un solo objeto. Esto se obtiene al analizar únicamente la componente y de los objetos que se encuentran

Al tomar en cuenta no solo la componente y de los objetos, sino también la componente x y se define un umbral por el que deben estar separados, de manera similar a como se separan las líneas, se pueden separar ahora por palabras e incluso por columnas. La importancia de esto es crítica, ya que al mandar la información obtenida a las redes neuronales a implementar, se puede mandar en un orden y no se debe de andar reconstruyendo posteriormente.

Una vez entrenado el modelo, se puede mandar a llamar siempre que se quiera inferir lo que tenga una caja que contenga una palabras. Se hicieron pruebas con las Figs. 12-14.

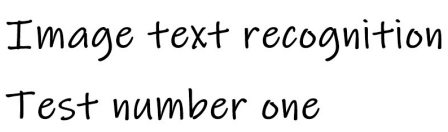


Fig. 12. Imagen de prueba con texto fácil de reconocer.

Estas imágenes son procesadas con todo el tratamiento previamente descrito y se obtienen resultados como los

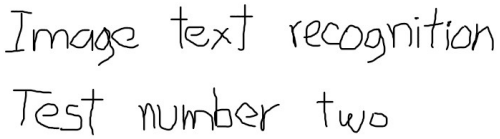


Fig. 13. Imagen de prueba con texto recto.

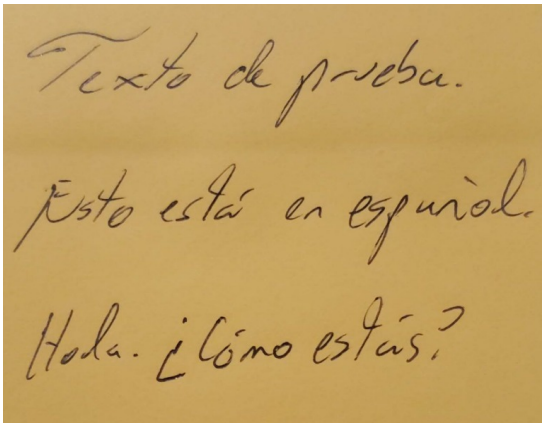


Fig. 14. Imagen de prueba con texto en español e inclinado.

que se presentan en la Fig. 15. Como se puede observar, el texto que está en la consola, es decir, lo que la red neuronal cree que está escrito es correcto en su mayoría, pero comete errores ortográficos, incluso siendo la imagen con el texto más comprensivo de los que se están utilizando para probar el sistema. Se equivoca al predecir "recoonition" en lugar de "recognition".

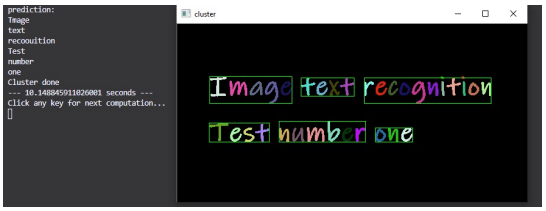


Fig. 15. Reconocimiento de caracteres del sistema en la consola.

Para corregir los errores ortográficos se usa la librería SpellChecker para poder corregir la entrega en el texto. En este caso, se puede observar que el texto corregido por la librería es certero, a pesar de que sustituye las letras mayúsculas por letras minúsculas. En la Fig. 16 se observa el texto ya corregido por la librería.

Se hizo una prueba extra con el mismo tipo de fuente

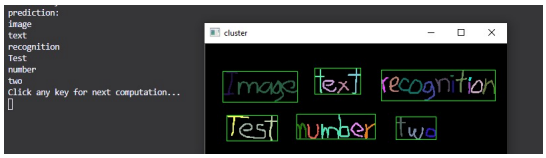


Fig. 16. Corrección de reconocimiento usando SpellChecker.

que la de la Fig. 12 y se puede observar el resultado en la Fig. 17. Como es de esperar, al introducir el uso de la librería se cambia el resultado de algunas palabras gracias a que la librería está diseñada para corregir palabras al idioma inglés, pero el texto con el eque se está trabajando está en español.

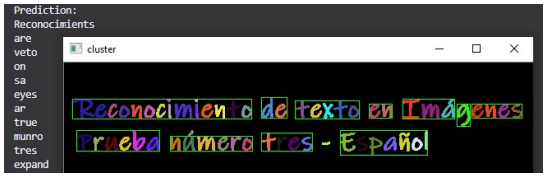


Fig. 17. Predicción errónea por el uso de SpellChecker.

Finalmente, con intención de probar el sistema en un caso extremo, se probó el sistema con el texto en la Fig. 14. En esta imagen, el texto está en español, inclinado e incluso hay personas que tienen problema leyendo lo que está escrito. En la Fig. 18 se puede observar que ni siquiera el proceso de clustering fue hecho de manera correcta, identificando diferentes caracteres como si fueran palabras distintas o incluso no reconociendo un caracter como uno solo, como lo es el caso de la "T" mayúscula que hay al inicio del texto o no reconociendo las tildes como parte del caracter. Esto es sumamente importante, ya que los datos que lleguen a la red neuronal no serán los correctos ni adecuados, por lo que se demuestran las limitaciones que tiene esta solución. Incluso las palabras que alcanzó a englobar correctamente fueron cambiadas por el SpellChecker, por lo que se nota que el sistema no está adecuado para decodificar imágenes en un idioma distinto al que se planeó.

Para validar nuestros resultados y medir nuestro desempeño, se alimentó un Jupyter Notebook en un Colab con la herramienta de OCR de Google, Tesseract. En las Figs. 19-21 podemos observar el desempeño que tuvo esta solución.

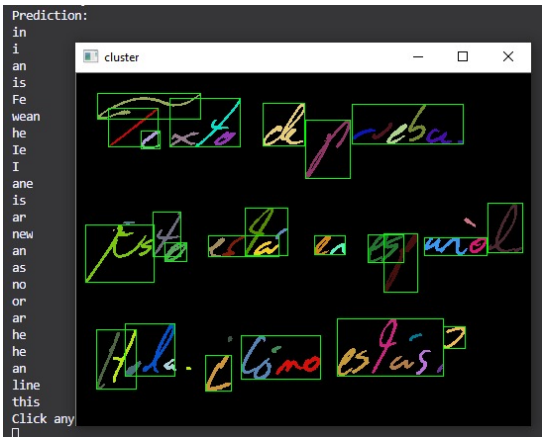


Fig. 18. Predicción errónea por imagen inadecuada

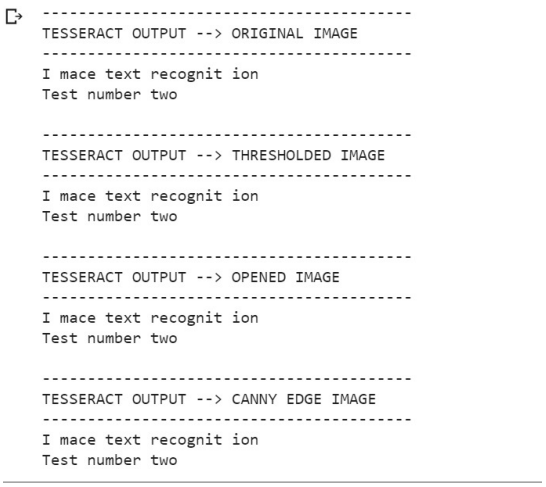


Fig. 19. Predicción del texto en Tesseract.

Se hicieron las pruebas con las figuras propuestas y, como se puede observar, en la Fig. 19 casi logra hacer una decodificación correcta, sin el uso de un SpellChecker como se implementó en nuestro solución. En la Fig. 20 se puede observar que, al tener una caligrafía más pulcra, la decodificación es completamente certera para la imagen de prueba.

Sin embargo, al hacer la prueba con la Fig. 14 se puede observar que el sistema tiene serios problemas para decodificar el texto, mostrado en la salida del sistema dado por Tesseract en la Fig. 21, pues no reconoce ni clasifica correctamente prácticamente ninguno de los caracteres que están escritos, probando que también está limitado ante casos extremos y de alta complejidad como es el que se presenta en esta figura.

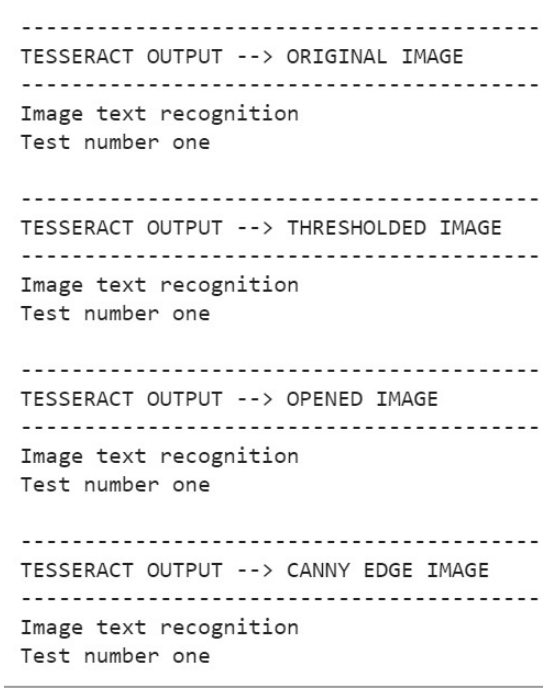


Fig. 20. Tesseract usando la tipografía más clara.

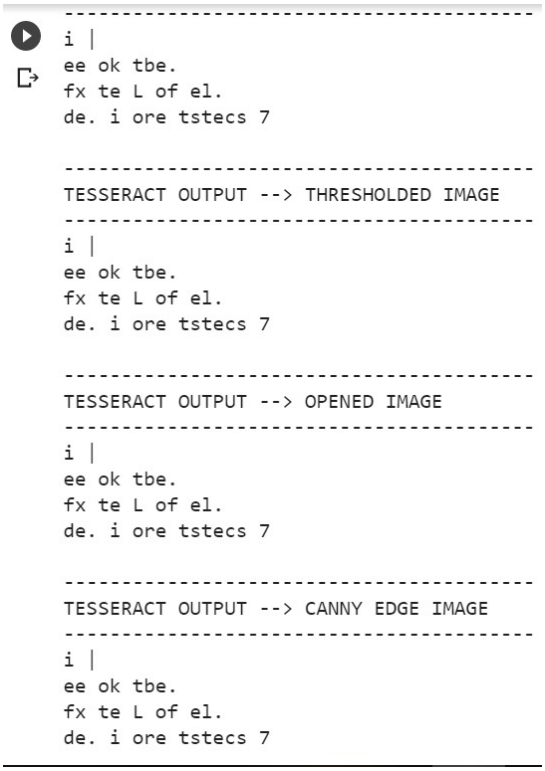


Fig. 21. Fallo completo de Tesseract.

VI. Conclusiones

Luego de haber intentado distintos métodos a través de este reporte, llegamos a la conclusión de que ningún algoritmo por sí sólo funciona de una manera en la que el texto sea legible y esto depende en gran medida a que no tenemos control sobre la entrada al sistema, por lo que debemos generar un algoritmo que sea lo suficientemente robusto, para ello se desarrolló el mecanismo de promedio de cinco umbralizaciones. Luego de probar distintos valores, máscaras y métodos decidimos que el algoritmo de Wolf sería el indicado para el trabajo, pues resultó tener mejor contraste en texto, a pesar del inconveniente que presento en la segunda etapa, de reconocimiento de caracteres, pues es un problema que se puede resolver escogiendo los parámetros correctos del reconocedor.

Hasta aquí logramos tener cierta tolerancia al error derivado del descontrol de las imágenes de entrada, ahora se probó también distintos valores para primero la separación por caracteres y distintos valores para las agrupaciones, teniendo un resultado similar que en la umbralización, es por ello que esta etapa se encuentra avanzada pero podría mejorar, con lo que tenemos algunas palabras aún no se completan correctamente, pero la mayoría, con las imágenes de prueba, muestran cierta certidumbre en la asignación. Con este avance logramos tener una segmentación medianamente positiva para continuar con la siguiente etapa de entrenamiento con la red neuronal.

Posteriormente en la etapa de Clustering creemos que el trabajo realizado fue el óptimo, logramos juntar las áreas ya encontradas en pequeños clusters para su posterior análisis en la red neuronal, pero cabe destacar que creemos que esta etapa puede ser mejorada haciendo uso de redes neuronales no supervisadas que logren encontrar estos clusters en base a la distancia entre cajas de una manera autónoma y no depende de cálculos y estimaciones matemáticas de las cajas encontradas.

El trabajar con la red neuronal por mucho fue lo mas retador del proyecto, pero se logro entender el funcionamiento y creación de esta; el resultado que la red nos brinda es el deseado y este en conjunto con el corrector de ortografía nos brinda con predicciones bastante óptimas. Creemos que esta experiencia trabajando con redes neuronales fue muy enriquecedora debido

a que hoy en día múltiples avances y mejoras que se encuentran en el campo de la Visión para Robots están directamente ligados al uso de estas redes. Finalmente, cabe decir que este proyecto fue construido en base a muchos conocimientos y la suma y unión de estos, creemos que hay mucho campo de mejora y esperamos proseguir con el avance de este proyecto en el futuro, en orden de crear una herramienta que pueda ayudar a muchos.

Trabajo Futuro

El resultado final del proyecto creemos que es bastante óptimo y ha sido probado en múltiples escenarios controlados y en estos ha logrado resultados bastante buenos. Para trabajo futuro creemos que lo primordial seria enfocarnos completamente en generalizar el modelo para que este pueda servir para casos mucho mas generales, un caso muy puntual seria el lograr crear una metodología que nos permita crear las cajas de palabras y de lineas de texto en base a redes neuronales no supervisados que ejecuten el clustering de estas.

Como ya se mencionó, el proyecto actualmente se encuentra limitado a ciertas restricciones, tales: que solo se pueda detectar texto que se encuentre centrado, alineado, que sea en ingles y que este tenga un tipo de fuente claro y no inclinado; a futuro podríamos trabajar en estas limitantes para ir las mitigando poco a poco. De las anteriores mencionadas, el volver el modelo apto para múltiples lenguajes creemos que seria lo mas simple y la primera limitante que resolveríamos, esto debido a que solo seria entrenar la red neuronal con palabras en otro idioma y cambiar al final en la etapa de Procesamiento final que también se haga un SpellChecker en diferentes idiomas.

Otro ejemplo de algo que creemos primordial para trabajo a futuro, sería el lograr que este proyecto crezca mucho mas y hasta empiece a ser de utilidad para la comunidad, y para esto creemos que lo mejor seria el crear una interfaz (tal vez una aplicación de celular) en donde se permita usar todo lo creado de una manera fácil y efectiva, a la par de ir recolectando toda la información de la aplicación para mejorar los modelos de esta.

REFERENCES

[1] F. Lateef and Y. Ruichek, "Survey on semantic segmentation using deep learning techniques," *Neurocomputing*, vol. 338, pp. 321–348, Apr. 2019.

[2] A. Mexico, "ABB muestra conceptualmente un robot móvil de laboratorio para el Hospital del Futuro," 2019.

[3] B. Gerkey, W. Smart, and M. Quigley, *Programming Robots with ROS : A Practical Introduction to the Robot Operating System*. Sebastopol: O'Reilly Media, Inc, USA, 2015.

[4] L. Shanghai Slamtec.Co., "RPLIDAR A2 Low Cost 360 Degree Laser Range Scanner Introduction and Datasheet," *Slamtec*, pp. 1–19, 2017.

[5] J. Martínek, L. Lenc, and P. Král, "Building an efficient ocr system for historical documents with little training data.," *Neural Computing and Applications*, p. 1, 2020.

[6] C. Clausner, A. Antonacopoulos, and S. Pletschacher, "Efficient and effective ocr engine training.," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 23, no. 1, p. 73, 2020.

[7] S. Omri, E. Avshalom, and Z.-G. Maayan, "Toward the optimized crowdsourcing strategy for ocr post-correction.," *Aslib Journal of Information Management*, vol. 72, no. 2, pp. 179 – 197, 2019.

[8] J. V. Urbas, "Neural networks.," *Salem Press Encyclopedia of Science*, 2019.

[9] L. Guoming, X. Yan, Z. Yang, D. Qian, and H. Yanbo, "Evaluating convolutional neural networks for cage-free floor egg detection.," *Sensors (14248220)*, vol. 20, no. 2, pp. 1 – 17, 2020.

[10] A. Fandango, *Mastering TensorFlow 1.x : Advanced Machine Learning and Deep Learning Concepts Using TensorFlow 1.x and Keras*. Packt Publishing, 2018.

[11] R. Orus Perez, "Using tensorflow-based neural network to estimate gnss single frequency ionospheric delay (iononet).," *Advances in Space Research*, vol. 63, no. 5, pp. 1607 – 1618, 2019.

[12] B. Pang, E. Nijkamp, and Y. N. Wu, "Deep learning with tensorflow: A review.," *Journal of Educational & Behavioral Statistics*, vol. 45, no. 2, pp. 227 – 248, 2020.

[13] Y. Xia, J. Zhou, T. Xu, and W. Gao, "An improved deep convolutional neural network model with kernel loss function in image classification.," *Mathematical Foundations of Computing*, vol. 3, no. 1, pp. 51 – 64, 2020.

APPENDIX

Liga al Código fuente