TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE LA PAZ
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

Informe Final de Año Sabático

Proyecto de Investigación

DETECCIÓN Y RECONOCIMIENTO DE TEXTO EN IMÁGENES NO RESTRINGIDAS

Jorge Enrique Luna Taylor

Enero de 2020

Resumen

En este documento presentamos el informe de año sabático realizado con el proyecto de investigación: *detección y reconocimiento de texto en Imágenes no restringidas*. De las diferentes estrategias que se han desarrollado para el reconocimiento de texto en imágenes no restringidas, consideramos de interés aquellas que dividen el problema en una primera fase de *detección* de las regiones que contienen texto, seguida por una fase de *reconocimiento* del texto contenido en las regiones detectadas.

La estrategia que proponemos en este trabajo incluye estas dos fases de detección y reconocimiento del texto. La fase de detección inicia con la identificación de los bordes de la imagen, seguido por el agrupamiento de bordes afines de acuerdo a la magnitud y ángulo de sus gradientes, para posteriormente generar áreas rectangulares que abarcan conjuntos de contornos que posiblemente corresponden a un mismo objeto.

La siguiente fase incluye el entrenamiento de una Red Neuronal Convolucional para el reconocimiento individual de los caracteres contenidos en las áreas rectangulares. Después se agrupan los caracteres en líneas de texto y finalmente se realiza el reconocimiento de palabras con base a un diccionario preestablecido, a través de una técnica de programación dinámica utilizada para el alineamiento de cadenas de ADN y proteínas.

Una diferencia que consideramos importante con respecto a otras propuestas, es que en este trabajo el entrenamiento de la red y el posterior reconocimiento, se realiza utilizando únicamente el contorno o silueta de los caracteres. De esta forma se disminuye el espacio de búsqueda del problema, al no requerir entrenar y reconocer con las imágenes completas del texto, el cual varía frecuentemente en color, iluminación y patrones gráficos del interior de los caracteres.

Las pruebas de esta estrategia se realizaron con tres conjuntos de datos reconocidos en la literatura. Los resultados se compararon con los trabajos del estado del arte, que al igual que nuestra propuesta, basan el entrenamiento de la red en el reconocimiento individual de caracteres. Con los tres conjuntos de datos se lograron resultados competitivos, alcanzando tasas de reconocimiento con uno o dos puntos porcentuales por arriba del mejor método evaluado.

Tabla de Contenido

Resumen 2

Tabla de Contenido 3

Tabla de Figuras 5

Tabla de Algoritmos 6

1 Introducción 7

1.1 Antecedentes 9

1.1.1 Reading Text in the Wild with Convolutional Neural Networks [1] 10

1.1.2 End-to-End Interpretation of French Street Name Signs Dataset [2] 11

1.1.3 Adaptive Ensemble of Deep Neural Networks for Scene Text Recognition [3] 12

1.2 Hipótesis 13

1.3 Objetivos 13

1.4 Justificación 14

2 Análisis de Métodos Previos 15

2.1 Edge Boxes: Locating Object Proposals from Edges [12] 15

2.1.1 Revisión del método 16

2.1.2 Análisis del método 21

2.2 A Computational Approach to Edge Detection [20] 23

2.2.1 Revisión del método 23

2.2.2 Análisis del método 26

2.3 Deep Learning for Text Spotting [13] 27

2.3.1 Revisión del método 27

2.3.2 Experimentos 34

2.3.3 Análisis del Método 35

3 Métodos Propuestos 37

3.1 Detección de texto 37

3.1.1 Identificación de bordes 37

3.1.2 Identificación de segmentos de contornos 45

3.1.3 Generación de recuadros 48

3.1.4 Filtrado de recuadros 55

3.2 Reconocimiento de texto 68
3.2.1 Reconocimiento de caracteres 68
3.2.2 Filtrado de caracteres que se traslapan 80
3.2.3 Identificación de líneas de texto 82
3.2.4 Separación de líneas de texto 84
3.2.5 Identificación de caracteres unidos 85
3.2.6 Reconocimiento individual de caracteres unidos 87
3.2.7 Generación de combinaciones de palabras candidatas 88
3.2.8 Reconocimiento de palabras 90
4 Experimentos y Resultados 94
4.1 Conjuntos de datos 94
4.2 Experimentos 95
4.3 Resultados 95
5 Conclusiones y Trabajo Futuro 99
Bibliografía 101

Tabla de Figuras

Figura 2.1. Detección de bordes. 17
Figura 2.2. Formación de grupos de bordes. 18
Figura 2.3. Generación de recuadros. 20
Figura 2.4. Máscaras de Convolución para filtros gaussianos. 24
Figura 2.5. Aplicación del algoritmo de Canny. 26
Figura 2.6. Arquitectura de la Red Neuronal Convolucional utilizada. 29
Figura 2.7. Ejemplo de la generación de un mapa de prominencia por pixel. 30
Figura 2.8. Ejemplo de una matriz de respuesta P para el caso no-sensitivo. 33
Figura 3.1. Trazado manual de bordes. 38
Figura 3.2. Identificación de bordes a través de Redes Neuronales Convolucionales. 40
Figura 3.3. Imagen con zonas de texto con diferente contraste de color 42
Figura 3.4. Identificación de bordes a través del algoritmo de Canny modificado. 44
Figura 3.5. Ejemplo de segmentos de contornos identificados. 47
Figura 3.6. Generación de recuadros. 53
Figura 3.7. Aplicación del método de generación de recuadros. 54
Figura 3.8 Ejemplo de contornos de letras generadas en un solo archivo 56
Figura 3.9. Ejemplos de letras generadas en archivos individuales. 57
Figura 3.10. Ejemplos de letras normalizadas. 59
Figura 3.11. Ejemplos de letras rotadas. 61
Figura 3.12. Muestras de letras finales rotadas y escaladas. 62
Figura 3.13. Muestras de no-letras generadas. 64
Figura 3.14. Recuadros después de la etapa de filtrado. 67
Figura 3.15. Ejemplo de estiramiento a la izquierda, derecha, arriba y abajo de una letra. 71
Figura 3.16. Ejemplos de letras generadas con borde sencillo y doble. 73
Figura 3.17. Ejemplo de imágenes de no-letras. 74
Figura 3.18. Estructura de Red Neuronal propuesta para el reconocimiento de caracteres. 75
Figura 3.19. Características de 5x5 pixeles de la letra M agrupadas en un mapa de 4x4. 76
Figura 3.20. Ejemplo de caracteres reconocidos en una imagen de prueba. 79
Figura 3.21. Ejemplos de recuadros generados, Izq. Algoritmo 3.13, der. Algoritmo 3.17. 80
Figura 3.22. Ejemplo de una imagen antes y después del filtrado de recuadros traslapados. 81
Figura 3.23. Ejemplo de la identificación de líneas de texto. 83
Figura 3.24. Ejemplo de separación en sub-líneas de texto. 85
Figura 3.25. Ejemplo de recuadros con caracteres unidos. 86
Figura 3.26. Ejemplos de ventanas deslizantes para reconocimiento de caracteres individuales. 87
Figura 3.27. Ejemplo de palabra con caracteres independientes y unidos. 90
Figura 3.28. Ejemplo de alineamientos entre dos cadenas de caracteres. 91
Figura 3.29. Matriz de puntaje propuesta para el emparejamiento entre caracteres. 91
Figura 4.1. Ejemplos de casos de reconocimiento correcto. 97
Figura 4.2. Ejemplos de casos de incorrecto reconocimiento. 98

Tabla de Algoritmos

Algoritmo 3.1. Análisis local para la identificación de bordes. 43
Algoritmo 3.2. Identificación de segmentos de contornos. 46
Algoritmo 3.3. Generación de recuadros. 49
Algoritmo 3.4. Recuadros afines. 50
Algoritmo 3.5. Fusionar recuadros. 51
Algoritmo 3.6. Validar recuadro. 51
Algoritmo 3.7. Identificación de la posición de las letras. 56
Algoritmo 3.8. Normalización de letras. 57
Algoritmo 3.9. Rotación de letras. 60
Algoritmo 3.10. Escalamiento de letras. 61
Algoritmo 3.11. Generación de no-letras. 63
Algoritmo 3.12. Pre-carga de las muestras de letras. 64
Algoritmo 3.13. Filtrado de recuadros. 66
Algoritmo 3.14. Estiramiento de Letras. 69
Algoritmo 3.15. Escalamiento de letras. 71
Algoritmo 3.16. Aumento del grosor del borde de letras. 71
Algoritmo 3.17. Reconocimiento de caracteres. 77
Algoritmo 3.18. Filtrado de caracteres que se traslapan. 81
Algoritmo 3.19. Identificación de caracteres cercanos. 82
Algoritmo 3.20. Identificación de líneas de texto. 83
Algoritmo 3.21. Separación de líneas de texto. 84
Algoritmo 3.22. Identificación de recuadros con caracteres unidos. 86
Algoritmo 3.23. Reconocimiento individual de caracteres unidos. 87
Algoritmo 3.24. Generación de palabras candidatas. 89
Algoritmo 3.25. Alineamiento de cadenas de caracteres (Smith-Waterman). 92

1 Introducción

El texto es una herramienta básica para preservar y comunicar información, gran parte del mundo moderno está diseñado para ser interpretado mediante el uso de etiquetas y otras señales textuales. La detección y el reconocimiento automático de texto en imágenes naturales es un desafío importante para la comprensión visual. Mediante el reconocimiento automático de texto, una parte importante del contenido semántico de los medios visuales puede decodificarse y utilizarse, por ejemplo, para comprender, anotar y recuperar los miles de millones de fotos de consumo que se producen todos los días. Las aplicaciones de la detección y reconocimiento de texto es muy amplio, incluye desde el desarrollo de sistemas automáticos de traducción de textos para el apoyo a personas con discapacidad visual en la interpretación de nombres de calles, etiquetas de medicamentos, denominación de billetes o uso de cajeros automáticos; hasta el reconocimiento automático de numeración de viviendas y texto en mapas geográficos para alimentar Sistemas de Geo-codificación.

Actualmente, las técnicas de *Reconocimiento Óptico de Caracteres* (*OCR*, por sus siglas en inglés) son adecuadas para digitalizar documentos en papel. Sin embargo, cuando se aplican a imágenes de escenas naturales estas técnicas fallan ya que están sintonizadas con el entorno de documentos planos impresos normalmente en blanco y negro. El texto que se produce en las imágenes naturales se extrae de diversas fuentes y es variado en apariencia, diseño, estilo e iluminación; así como suele presentar oclusiones, distintas orientaciones, ruido y presencia de objetos de fondo [1].

Desde hace algunos años, las Redes Neuronales Profundas (*DNN*, por sus siglas en inglés) han dominado el estado del arte en la detección y reconocimiento de texto [1, 2, 3]. Sin embargo, aún existe un amplio espacio para la investigación en este campo, como lo sugieren las bajas tasas de detección (a menudo menos del 80 por ciento) y las bajas tasas

de reconocimiento (a menudo menos del 60 por ciento) de los enfoques del estado del arte [4].

Considerando lo anterior, en este proyecto se propone el diseño e implementación de nuevos métodos para la detección y reconocimiento de texto contenido en imágenes naturales no restringidas. El método propuesto para la fase de detección consiste en la adición de un paso de análisis local para la detección de bordes (3.1.1.2), seguido por nuevos algoritmos para la identificación de segmentos de contornos de los posibles objetos de la imagen (3.1.2), y la generación y filtrado de recuadros candidatos a contener una letra (3.1.3, 3.1.4). Para la fase de reconocimiento se propone un método que consiste en el reconocimiento de caracteres a través de una red neuronal convolucional (3.2.1), seguido por el filtrado de caracteres que se traslapan (3.2.2), identificación y separación de líneas de texto (3.2.3, 3.2.4), identificación y reconocimiento de caracteres unidos (3.2.5, 3.2.6), generación de combinación de palabras candidatas (3.2.7) y finalmente un algoritmo de alineamiento de cadenas para el reconocimiento de palabras de un diccionario (3.2.8). Con esta propuesta, se espera lograr resultados competitivos con respecto al estado del arte.

Este documento está estructurado de la siguiente forma. En el Capítulo 1 se expone brevemente la motivación que condujo al desarrollo de este trabajo, y se muestra un resumen de los antecedentes o estado del arte de la problemática (1.1), para posteriormente definir la hipótesis, objetivos y justificación de este proyecto (1.2, 1.3, 1.4). En el Capítulo 2 se presenta un análisis detallado de 3 métodos que resaltan en el estado del arte, para la detección de bordes (2.2), detección de objetos (2.1) y reconocimiento de texto (2.3), de los cuales hemos tomado algunas ideas generales para la propuesta que realizamos en este trabajo. En el Capítulo 3 se presentan y explican los nuevos métodos que proponemos para la detección de texto (3.1) y para el reconocimiento del texto detectado (3.2). En el Capítulo 4 se muestra el conjunto de datos utilizado para evaluar el método propuesto (4.1), se explica en que consistieron los experimentos realizados (4.2), y se muestra una tabla comparativa de los resultados

obtenidos por este y otros métodos sobresalientes (4.3). En el Capítulo 5 se concluye este proyecto con una breve descripción de la problemática atendida, el trabajo realizado y los resultados obtenidos. Finalmente, se presentan y explican algunas ideas de mejora de este proyecto, para ser consideradas como trabajo a futuro.

1.1 Antecedentes

Actualmente existe un amplio y diverso conjunto de métodos aplicados a la detección y reconocimiento de texto en imágenes no restringidas. Sin embargo, se pueden distinguir dos metodologías generales: 1) *paso a paso* y 2) *integrada*.

La metodología paso a paso tiene cuatro etapas principales: localización, verificación, segmentación y reconocimiento. La etapa de localización aplica una clasificación de *grano grueso* de los componentes de una imagen y los agrupa en regiones de texto candidatas, las cuales se analizan y clasifican posteriormente como regiones de texto o sin texto durante la verificación. El paso de segmentación separa las regiones de texto identificadas, para que sus contornos exclusivos y precisos de bloques de imagen permanezcan para el paso de reconocimiento. Finalmente, el paso de reconocimiento convierte bloques de imagen en caracteres. Algunos enfoques ignoran el paso de verificación y/o segmentación o incluyen pasos adicionales. Propuestas sobresalientes que utilizan esta metodología son [5, 6, 7].

La metodología integrada agrupa la localización y clasificación de caracteres con el reconocimiento de palabras a través de un proceso iterativo. La clasificación de caracteres, como característica central de esta metodología, requiere la discriminación de éstos respecto al fondo de la imagen así como del resto de caracteres, lo cual es un problema multi-clase complejo. Las soluciones requieren de modelos robustos de reconocimiento de caracteres, así como de estrategias apropiadas de integración de éstos, como *word spotting*. El enfoque *word spotting* busca hacer coincidir los conjuntos de

caracteres identificados en la imagen con palabras específicas de un determinado léxico. Algunos trabajos reconocidos que utilizan este enfoque son [8, 9].

Algunos de los métodos que en los últimos años han establecido el estado del arte en el reconocimiento de texto en ambientes no restringidos son: *Reading Text in the Wild with Convolutional Neural Networks* [1], *End-to-End Interpretation of the French Street Name Signs Dataset* [2] y *AdaDNNs: Adaptive Ensemble of Deep Neural Networks for Scene Text Recognition* [3]. A continuación se presenta una breve descripción de cada uno.

1.1.1 Reading Text in the Wild with Convolutional Neural Networks [1]

En este trabajo proponen un sistema de reconocimiento *End-to-End* (detección y reconocimiento de forma integrada) basado en un mecanismo de *regiones candidatas* para la detección de posibles zonas que contengan texto, seguido de un sistema de Redes Neuronales Convolucionales (*CNN*, por sus siglas en inglés) para el reconocimiento a partir de las regiones identificadas.

La etapa de detección inicia con métodos simples pero rápidos para generar propuestas de recuadros delimitadores de palabras. Posteriormente, se incorporan de forma gradual modelos más complejos para mejorar la precisión de los recuadros delimitadores, mediante la identificación y rechazo de falsos positivos, hasta obtener un número manejable de propuestas.

En la segunda etapa reconocen las palabras dentro de los recuadros delimitadores propuestos. Para este fin utilizan una *CNN* para realizar la clasificación a través de un diccionario predefinido. La imagen de cada uno de los cuadros delimitadores se toma como entrada a la *CNN* y ésta calcula una distribución de probabilidad sobre todas las palabras en el diccionario. La palabra con la máxima probabilidad se toma como el resultado del reconocimiento.

El método lo evaluaron con diferentes conjuntos de datos reconocidos, mejorando de forma significativa los resultados reportados en el estado del arte:

- Conjunto de datos ICDAR 2003[1], con una precisión del 87%
- Conjunto de datos SVT[2], con una precisión del 82%
- Conjunto de datos ICDAR 2011, con una precisión del 77% y
- Conjunto de datos ICDAR 2013, con una precisión del 77%

1.1.2 End-to-End Interpretation of French Street Name Signs Dataset [2]

En este trabajo proponen una solución *End-to-End* basada en una red conformada por diferentes componentes. La red fue diseñada específicamente para trabajar con el conjunto de datos *French Street Name Signs* (*FSNS*), que consiste en más de un millón de imágenes de señalamientos con nombres de calles de Francia. Cada imagen tiene cuatro mosaicos de 150 × 150 píxeles distribuidos horizontalmente, cada uno contiene el nombre de la calle y se le ha añadido ruido aleatorio. Cada señalamiento tiene un máximo de 3 líneas de texto. Cada uno de los cuatro mosaicos pretende ser una vista diferente del mismo nombre, tomada desde una posición diferente y/o en un tiempo diferente.

El primer componente del sistema está formado por dos capas Convolucionales combinadas con capas de *Max Pooling*, con el objetivo de que éstas identifiquen bordes y estos se combinen en características. Este componente recibe y procesa cada uno de los mosaicos que forman la imagen de forma separada.

El siguiente componente es un *LSTM* (*Long Short-Term Memory*) de sumarización vertical, que realiza un escaneo vertical con el objetivo de identificar líneas de texto. En cada posición en el eje X realizan tres sumarizaciones verticales de forma independiente:

1. En sentido ascendente para buscar la línea de texto superior.

[1] Conjunto de datos IC03-Full [22] utilizado en los experimentos y comparativas en nuestro proyecto.
[2] Conjunto de datos SVT-50 [23] utilizado en los experimentos y comparativas en nuestro proyecto.

2. Dos recorridos separados ascendente y descendente para identificar la línea de texto intermedia.
3. En sentido descendente para identificar la línea de texto inferior.

Considerando que cada una de las cuatro imágenes puede tener el texto posicionado de forma diferente debido a las diferentes perspectivas, el tercer componente tiene como objetivo reacomodar los datos a lo largo del eje X. Para esto, se utilizan dos capas *LSTM*, una para escanear de izquierda a derecha y la otra de derecha a izquierda, con el objetivo de alinear los caracteres de cada una de las cuatro vistas.

El último componente de la red está compuesto por una capa *LSTM* unidireccional cuyo propósito es combinar las cuatro vistas para producir un resultado más confiable y por una capa final *Sofmax* la cual realiza el trabajo de clasificación.

Este método lo evaluaron utilizando el conjunto de datos *French Street Name Signs* (*FSNS*), logrando un 90% de correcto reconocimiento del texto de más de un millón de imágenes del cual consta el conjunto de pruebas. Estos resultados superaron de forma significativa el 21% de correcto reconocimiento alcanzado con la ibrería de *Tesseract* de Google, sobre este mismo conjunto de datos.

1.1.3 Adaptive Ensemble of Deep Neural Networks for Scene Text Recognition [3]

Proponen un método automatizado para buscar la mejor combinación de un conjunto clasificador *DNN* para el reconocimiento de texto. Para lo obtención del conjunto consideran y explotan diferentes modelos, de acuerdo sus decisiones individuales y la correlación con las hipótesis de los demás. Para esto, utilizan un marco basado en probabilidad Bayesiana para combinar los clasificadores.

Para obtener la mejor combinación posible utilizan un algoritmo genético, el cual genera una población inicial de forma aleatoria de vectores de pesos binarios, donde el valor 1

indica que el clasificador es mantenido. Posteriormente, la población evoluciona de forma iterativa, donde la aptitud de cada vector es medida con base en la tasa de error de reconocimiento sobre un conjunto de validación. Finalmente, la combinación elegida corresponde al vector de pesos que evolucionó con la mejor aptitud.

Con este método participaron en la *Robust Reading Competition* (*ICDAR*, 2017), logrando los mejores resultados de reconocimiento *End-To-End* sobre el conjunto de datos *COCO-Text*, con un 43.58% de precisión, seguido de lejos por el método *Foo & Bar* (basado también en *DNN*) con el 27.01% de precisión y en tercer puesto el método *WPS* con el 18.82%.

1.2 Hipótesis

Es posible detectar y reconocer texto en imágenes no restringidas, con resultados competitivos en relación a los métodos del estado del arte, a través del diseño e implementación de nuevas técnicas de segmentación de caracteres, y la aplicación de redes neuronales artificiales entrenadas para el reconocimiento individual de caracteres, basadas exclusivamente en el contorno de estos.

1.3 Objetivos

Objetivo General

Desarrollar un algoritmo para la detección y reconocimiento de texto en imágenes no restringidas basado en nuevas técnicas de segmentación y en redes neuronales artificiales.

Objetivos Particulares

1. Diseñar e implementar un método para la detección de texto en imágenes naturales basado en el desarrollo y aplicación de nuevas técnicas de segmentación.

2. Diseñar e implementar un método para el reconocimiento de texto en imágenes naturales basado en redes neuronales artificiales, entrenadas a partir del contorno de los caracteres.
3. Mejorar los resultados respecto al estado del arte, al evaluar los métodos desarrollados sobre diferentes conjuntos de datos reconocidos en la literatura.

1.4 Justificación

De acuerdo a la Organización Mundial de la Salud, se estima que existen 285 millones de personas en el mundo con algún grado de discapacidad visual, entre los cuales 39 millones sufren la pérdida total de la visión. Aproximadamente, 19 millones son niños menores de 15 años, de los cuales 1.4 millones tienen ceguera irreversible por el resto de sus vidas y requieren de un apoyo integral para su desarrollo personal y psicológico [10]. Estos datos sugieren la importancia del uso y desarrollo de nueva tecnología para la asistencia a personas con discapacidad visual. Actualmente, existen diversas aplicaciones capaces de leer texto impreso y reproducirlo a través de audio. Sin embargo, estas aplicaciones están limitadas a ciertos contextos específicos, restando un gran número de situaciones donde las personas invidentes pueden requerir apoyo para lectura de texto.

A pesar de los avances en esta área aún existen grandes retos por superar, como lo muestran los resultados de la reciente *Robust Reading Competition* (ICDAR, 2017), donde los métodos de reconocimiento *End-To-End* sobre el conjunto de datos *COCO-Text*, obtuvieron tasas de precisión extremadamente bajas. El conjunto de datos *COCO-Text* [11] consiste en 63,686 imágenes de escenas naturales que contienen texto. Los mejores resultados en esta competición se lograron con el método *AdaDNNs: Adaptive Ensemble of Deep Neural Networks for Scene Text Recognition* [3] basados en *DNN*, con el 43.58% de precisión, seguido de lejos por el método *Foo & Bar* (basado también en *DNN*) con el 27.01% de precisión y en tercer puesto el método *WPS* con el 18.82%.

2 Análisis de Métodos Previos

Basados en la revisión de los métodos del estado del arte, la estrategia general que proponemos para la lectura de texto en imágenes naturales no restringidas, se divide en dos pasos principales. El primero es la *detección* de las zonas donde posiblemente se encuentre texto dentro de la imagen, seguido por una fase de *reconocimiento* del mismo, a través del análisis de las zonas seleccionadas en el primer paso. Un método reciente para la detección de objetos que ha obtenido resultados sobresalientes, es el presentado en [12]. Dado que nuestra propuesta de detección de texto se basa en la idea general de éste método, a continuación presentamos una revisión de las etapas más sobresalientes del mismo y posteriormente un análisis de sus fortalezas y debilidades. Por otro lado, el método en el que nos basamos para el diseño de la etapa de reconocimiento de texto es el presentado en [13], cuyo resumen y análisis de sus ventajas y desventajas queda pendiente para el informe final.

2.1 Edge Boxes: Locating Object Proposals from Edges [12]

En este trabajo proponen un método para detección de objetos en imágenes naturales. La idea central es generar *recuadros* que encierran a los posibles objetos presentes en las imágenes, a partir de la información de los pixeles clasificados como *bordes*. Realizan la observación de que el número de *contornos* (conjuntos de pixeles de bordes que presentan afinidad), que están completamente contenidos en un recuadro, es indicativo de la probabilidad de que éste contenga un objeto. Proponen un puntaje para evaluar la posibilidad de que un recuadro contenga a un objeto completo, que considera el número de bordes que hay dentro del recuadro y pertenecen a contornos que no se traslapan con los límites del recuadro. Inician evaluando millones de recuadros candidatos, quedándose

con unos pocos miles que presentan el mejor puntaje. De acuerdo a sus pruebas, obtienen resultados muy competitivos en relación al estado del arte.

2.1.1 Revisión del método

El método inicia identificando el conjunto de pixeles considerados como bordes de la imagen. Para esto, se utiliza el llamado *Detector Estructurado de Bordes* [14] [15]. El detector estructurado de bordes se basa a su vez en árboles de decisión. Un árbol de decisión $f_t(x)$ clasifica una muestra $\{x \in X\}$ recorriendo recursivamente las ramas izquierda y derecha del árbol hasta alcanzar un nodo hoja. En nuestro problema, x corresponde a un pixel con la información asociada de intensidad, magnitud y orientación del gradiente, de cada componente de color. Cada nodo j del árbol se asocia con una función de decisión con parámetro θ_j (Ecuación (2.1)). Si $h(x, \theta_j) = 0$ el nodo j envía a x a la izquierda, en caso contrario a la derecha. La salida del árbol para una muestra x es la clase $y \in Y$ asociada a la hoja alcanzada por x. En este problema, y es simplemente la etiqueta de ser borde o no.

$$h(x, \theta_j) \in \{0,1\} \qquad \textbf{(2.1)}$$

En su forma más simple, cada componente de x es comparado con un umbral, específicamente, $\theta_j = (k, \tau)$ y $h(x, \theta_j) = [x(k) < \tau]$, donde $[\cdot]$ denota la función indicadora. De esta manera, para un nodo j y un conjunto de entrenamiento $S_j \subset X \times Y$, el objetivo es encontrar los parámetros θ_j de la función de separación $h(x, \theta_j)$, que resultan en una óptima separación de los datos. En la Figura 2.1 se muestran ejemplos de la aplicación de este método de detección de bordes. Primera fila, imágenes originales. Segunda fila, imágenes después de aplicar el detector estructurado de bordes. Posteriormente, se ejecuta la técnica de *Supresión No-Máxima* (*NMS*) ortogonal y se obtiene la magnitud (m_p) y orientación (θ_p) de cada pixel, reconociendo como bordes aquellos pixeles con $m_p > 0.1$. Se define como contorno a un conjunto de bordes que

forman algún segmento coherente, ya sea una línea o curva que pueda ser parte de un objeto.

Figura 2.1. Detección de bordes.

2.1.1.1 Grupos de bordes y afinidades

En la siguiente fase del método, se identifican los contornos que se traslapan con los límites del recuadro, y que por lo tanto no es posible que pertenezcan a un objeto contenido completamente dentro del mismo. Dado un recuadro b, para cada borde dentro de éste (esto es, $p \in b$ con $m_p > 0.1$), se calcula su máxima afinidad con un borde que se encuentra en los límites del recuadro. Se considera que los bordes conectados por contornos rectos deben tener alta afinidad y aquellos no conectados o conectados por un contorno con gran curvatura deben tener baja afinidad. Para efectos de eficiencia computacional se agrupan los bordes que tienen alta afinidad y posteriormente se calculan las afinidades entre grupos de bordes. Los grupos de bordes se forman utilizando un método codicioso sencillo que combina los 8-bordes conectados a uno central, hasta que la suma de las diferencias de sus orientaciones está por arriba de un cierto umbral (en la práctica utilizan $\pi/2$). En la Figura 2.2 se muestran ejemplos de la formación de grupos de bordes. Los pixeles vecinos con un mismo color forman parte del mismo grupo de bordes s_i.

Figura 2.2. Formación de grupos de bordes.

Posteriormente, se mezcla cada grupo con sus grupos vecinos, para esto, dado un conjunto de grupos de bordes $s_i \in S$, se calcula la afinidad entre cada par de grupos vecinos s_i y s_j, basado en sus posiciones medias x_i y x_j, y en sus orientaciones medias θ_i y θ_j. Se considera que dos grupos de bordes tienen alta afinidad si el ángulo entre las posiciones medias de los grupos es similar a las orientaciones de los grupos. El cálculo de la afinidad entre dos grupos de bordes está dado por la Ecuación (2.2).

$$a(s_i, s_j) = \left|\cos(\theta_i - \theta_{ij})\cos(\theta_j - \theta_{ij})\right|^{\gamma} \qquad \textbf{(2.2)}$$

Donde θ_{ij} es el ángulo entre x_i y x_j, y el valor de γ se utiliza para ajustar la sensibilidad de la afinidad a los cambios de orientación, con un valor de $\gamma = 2$ utilizado en la práctica. Si dos grupos de bordes están separados por más de 2 pixeles, su afinidad se asigna como cero. Para incrementar la eficiencia computaciones solo se consideran afinidades arriba de un umbral de 0.05.

2.1.1.2 Puntaje de recuadros delimitadores

Dado un conjunto de grupos de bordes S y sus afinidades, se calcula un puntaje para los recuadros delimitadores b, considerados como candidatos. Para esto, primero se calcula la suma m_i de las magnitudes m_p de todos los bordes p que pertenecen al grupo s_i. Posteriormente, se toma la posición $\overline{x_i}$ de un pixel p que pertenecen al grupo s_i, seleccionado de forma arbitraria. Y para cada grupo s_i se calcula un valor continuo $w_b(s_i) \in [0,1]$, que indica si s_i está totalmente contenido en b, $w_b(s_i) = 1$, o no

$w_b(s_i) = 0$. Se nombra como S_b al conjunto de grupos de bordes que se traslapan con los límites del recuadro b, y para todo $s_i \in S_b$, $w_b(s_i)$ se asigna a cero. De forma similar $w_b(s_i) = 0$ para toda s_i para la cual $\overline{x_i} \notin b$. Para los grupos de bordes restantes, para los cuales $\overline{x_i} \in b$ y $s_i \notin S_b$, se calcula su $w_b(s_i)$ con la Ecuación (2.3).

$$w_b(s_i) = 1 - \max_T \prod_j^{|T|-1} a(t_j, t_{j+1}) \qquad \textbf{(2.3)}$$

Donde T es un camino ordenado de grupos de bordes con una longitud $|T|$, que inicia con algún $t_1 \in S_b$ y termina en $t_{|T|} = s_i$. Si no existe tal camino, se define $w_b(s_i) = 1$. De esta forma se encuentra el camino con más alta afinidad entre el grupo s_i y un grupo que se traslapa con los límites del recuadro. Utilizando los valores de w_b, se define el puntaje final h_b para cada recuadro b, de acuerdo a la Ecuación (2.4).

$$h_b = \frac{\sum_i w_b(s_i) m_i}{2(b_w + b_h)^k} \qquad \textbf{(2.4)}$$

Donde b_w y b_h son el ancho y alto del recuadro y k=1.5 es un parámetro utilizado como bias, para establecer que recuadros más grandes contengan más bordes en promedio.

2.1.1.3 Estrategia de búsqueda

La precisión de los recuadros que se obtienen en este tipo de métodos, es comúnmente medida calculando el área de la intersección del recuadro candidato y el recuadro real de un objeto sobre el área de la Unión de éstos (IoU). Como parte de la estrategia de búsqueda en este trabajo proponen un parámetro δ que representa el valor deseado IoU de los recuadros resultantes. Para valores altos de δ resulta una mayor concentración de recuadros, lo cual representa una mayor probabilidad de contener a un objeto. Para valores bajos de δ los recuadros presentan mayor diversidad. Dado un valor propuesto de

δ, se inicia la búsqueda de recuadros candidatos utilizando una ventana deslizante tanto en posición, escala y proporción. El cambio en la traslación escala o proporción, se determina de tal forma que en cada paso resulten recuadros vecinos con un IoU igual a un valor α. El rango de valores de escala son de un mínimo de σ=1000 pixeles hasta la imagen completa. La proporción varía de $1/\tau$ a τ, donde τ =3. Se utiliza un α =0.65 para la mayoría de los valores de δ. Sin embargo, si se requiere mayor precisión de δ > 0.9, α puede ser incrementada a 0.85.

Después de que el proceso de ventana deslizante concluye, todos los recuadros con puntaje h_b arriba de un cierto umbral son refinados. El refinado se realiza utilizando un algoritmo codicioso que trata de maximizar h_b sobre posición, escala y proporción. Después de cada iteración, el paso de búsqueda es reducido a la mitad. La búsqueda es detenida cuando el paso de traslación es menor a 2 pixeles. Una vez que los recuadros candidatos son refinados, los puntajes máximos son guardados y ordenados. La etapa final ejecuta una *Supresión No-Máxima* (*NMS*) de los recuadros ordenados. Un recuadro es removido si su IoU es mayor que β respecto a un recuadro con un puntaje mayor. Se encontró en la práctica que $\beta = \delta + 0.05$ logra los mejores resultados para todos los valores δ. En la Figura 2.3 se muestran ejemplos de los recuadros obtenidos. Primera fila, recuadros que encierran objetos de forma correcta. Segunda fila, recuadros incorrectos.

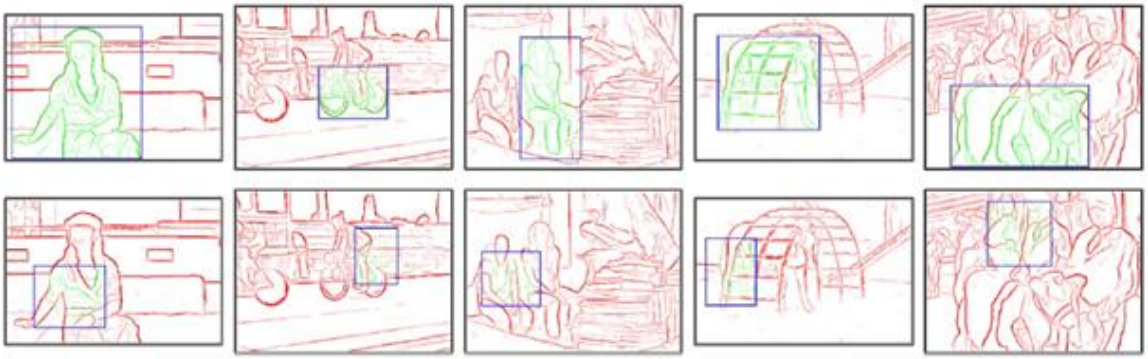

Figura 2.3. Generación de recuadros.

2.1.2 Análisis del método

Consideramos que la fortaleza principal de este método se encuentra en la idea central en la que está sustentado, en cuanto a que el número de contornos que están completamente contenidos en un recuadro es indicativo de la probabilidad de que éste contenga un objeto. Los resultados competitivos alcanzados por este método sobre diferentes bases de datos dan soporte a la efectividad de esta idea. La Tabla 1 muestra el porcentaje de objetos identificados por este y otros métodos sobresalientes, sobre un conjunto de 9963 imágenes de prueba. La segunda y tercera columna corresponden al número de recuadros requeridos por los métodos para recuperar el 25 y 50% de los objetos. La cuarta columna muestra el porcentaje de objetos recuperados cuando los métodos generan 5000 recuadros.

Tabla 1. Porcentaje de objetos recuperados por diferentes métodos de identificación.

Método	25%	50%	Ident.
Rantalankila, et. al. [16]	184	584	68%
Manen, et. al. [17]	42	349	80%
Rahtu, et. al. [18]	29	307	70%
Uijlings, et. al. [19]	28	199	87%
Método Analizado [12]	12	108	87%

Sin embargo, observamos que la propuesta presenta algunas desventajas en los algoritmos específicos que lo implementan. Por un lado, requiere la generación y evaluación de millones de recuadros para obtener finalmente algunos pocos miles como candidatos. Además, a pesar de la gran cantidad de recuadros generados, aún existe la posibilidad de que algún contorno que forma parte de un objeto, quede parcialmente fuera del recuadro y no logre pasar a la fase de reconocimiento de texto. Por otro lado, observamos como desventaja importante, la elevada cantidad de parámetros que se deben sintonizar adecuadamente para el correcto funcionamiento del método. Entre los principales parámetros se encuentran:

- Umbral de la magnitud que debe tener un pixel para ser considerado como borde.
- Umbral de la suma de las diferencias de las orientaciones de los pixeles para pertenecer a un mismo grupo de bordes.
- γ: para ajusta la sensibilidad en el cálculo de afinidad de los cambios de orientación entre cada par de grupos vecinos.
- Un umbral de afinidades entre grupos.
- k : utilizado como *bias* para establecer que recuadros más grandes contengan más bordes en promedio.
- δ : corresponde el valor deseado IoU de los recuadros resultantes.
- α: establece el cambio en la traslación escala o proporción de los recuadros candidatos.
- σ : establece el mínimo de pixeles para el rango de valores de escala.
- τ : determina el rango de valores de la proporción de los recuadros.
- β : establece el valor de IoU para que un recuadro sea removido al compararlo con un recuadro con un puntaje mayor

Considerando las fortalezas y debilidades observadas en este trabajo, la idea general en el diseño de un nuevo método, es mantener la propuesta central de generar recuadros que contengan la mayor cantidad de contornos, que posiblemente correspondan a un mismo objeto (en nuestro caso, a una misma letra), pero sin generar estos recuadros de forma arbitraria en cualquier posición, escala y proporción. Consideramos que los recuadros iniciales se pueden generar a partir de las posiciones de los contornos previamente identificados, evitando un número excesivo de éstos ubicados en zonas donde no existen bordes de objetos. Creemos de suma importancia reducir el número de parámetros requeridos por el método de detección. Si bien, es complejo diseñar un método que no requiera de parámetros, la idea es tratar de reducir al mínimo tanto la cantidad, como la sensibilidad de los resultados del método a pequeños cambios de éstos.

Por otro lado, el *detector estructurado de bordes* [14, 15] utilizado por [12], está basado en un árbol de decisión entrenado con un conjunto de imágenes que no contienen texto. Aunque esto no es necesariamente una limitante para la detección de bordes de texto, en nuestro proyecto proponemos el uso de nuevos métodos para la identificación de los bordes. Una primera opción es diseñar un algoritmo basado en Redes Neuronales Convolucionales entrenado a partir de los bordes identificados de forma manual en imágenes que contienen texto. Una segunda propuesta es aplicar una versión modificada del algoritmo de Canny, el cual es ampliamente reconocido en la literatura y presenta entre sus ventajas que es determinístico y no requiere de entrenamiento previo. La implementación y pruebas de ambas propuestas se presentan en el Capítulo 3.

2.2 A Computational Approach to Edge Detection [20]

En el área de procesamiento de imágenes, la detección de los bordes es de suma importancia, pues facilita las tareas de reconocimiento de objetos y la segmentación de regiones, entre otras. El algoritmo de Canny ha sido utilizado ampliamente para detectar los bordes existentes en una imagen. Este algoritmo está considerado como uno de los mejores métodos de detección de contornos, mediante el empleo de máscaras de Convolución y basado en la primera derivada. Los puntos de contorno son zonas de pixeles en las que existe un cambio brusco de nivel de gris. En el tratamiento de imágenes se trabaja con pixeles en un ambiente discreto, por lo que el algoritmo de Canny utiliza máscaras que representan aproximaciones en diferencias finitas.

2.2.1 Revisión del método

El algoritmo de Canny consiste en tres grandes pasos:

- Obtención del gradiente: en este paso se calcula la magnitud y orientación del vector gradiente en cada píxel.
- Supresión no máxima: en este paso se logra el adelgazamiento del ancho de los bordes, obtenidos con el gradiente, hasta lograr bordes de un píxel de ancho.
- Histéresis de umbral: en este paso se aplica una función de histéresis basada en dos umbrales, con este proceso se pretende reducir la posibilidad de aparición de contornos falsos.

2.2.1.1 Obtención del gradiente

Para la obtención del gradiente, se aplica un filtro gaussiano a la imagen original con el objetivo de suavizarla y tratar de eliminar el posible ruido existente. Sin embargo, se debe de tener cuidado de no realizar un suavizado excesivo, pues se pueden perder detalles importantes de la imagen. Este suavizado se obtiene promediando los valores de intensidad de los pixeles en el entorno de vecindad, con una máscara de Convolución de media cero y desviación estándar σ. En la Figura 2.4 se muestran dos ejemplos de máscaras utilizadas para el filtrado gaussiano. Una vez que se suaviza la imagen, para cada píxel se obtiene la magnitud y módulo (orientación) del gradiente.

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

$\frac{1}{115}$

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2

Figura 2.4. Máscaras de Convolución para filtros gaussianos.

2.2.1.2 Supresión no-máxima

El procedimiento es el siguiente: se consideran cuatro direcciones identificadas por las orientaciones de 0º, 45º, 90º y 135º con respecto al eje horizontal. Para cada pixel se encuentra la dirección que mejor se aproxime a la dirección del ángulo de gradiente. Posteriormente, se observa si el valor de la magnitud del gradiente es más pequeño que al menos uno de sus dos vecinos en la dirección del ángulo obtenida en el paso anterior. De ser así, se asigna el valor 0 a dicho pixel, en caso contrario se asigna el valor que tenga la magnitud del gradiente. La salida de este segundo paso es la imagen con los bordes adelgazados.

2.2.1.3 Histéresis de umbral

La imagen obtenida en el paso anterior suele contener máximos locales creados por el ruido. Una solución para eliminar dicho ruido es la histéresis del umbral. El proceso consiste en tomar la orientación de los puntos de borde de la imagen y dos umbrales, el primero más pequeño que el segundo. Para cada punto de la imagen se debe localizar el siguiente punto de borde no explorado que sea mayor al segundo umbral. A partir de dicho punto, se siguen las secuencias de máximos locales conectados en ambas direcciones perpendiculares a la normal del borde, siempre que sean mayores al primer umbral. Así, se marcan todos los puntos explorados y se almacena la lista de todos los puntos en el contorno conectado. Con este paso se logra eliminar las uniones en forma de Y de los segmentos que confluyan en un punto.

La Figura 2.5 muestra un ejemplo de la aplicación del algoritmo de Canny. Primera fila a izquierda, imagen original. Primera fila a la derecha, imagen después de aplicar un filtro gaussiano. Segunda fila a la izquierda, imagen con supresión de no-máximos. Y segunda fila a la derecha, imagen después aplicar el proceso de histéresis con umbrales de 100 y 130.

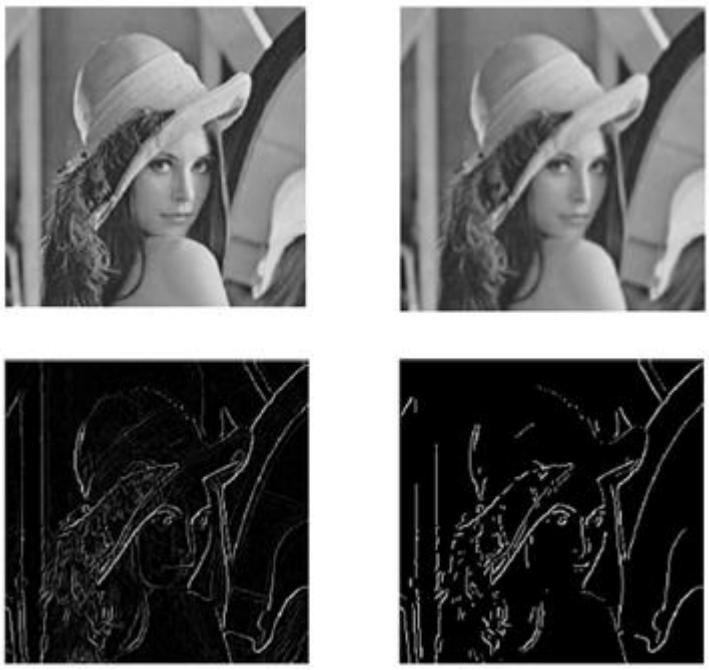

Figura 2.5. Aplicación del algoritmo de Canny.

2.2.2 Análisis del método

A pesar de que el artículo original de este método de detección de bordes [20], no presenta un análisis comparativo de sus resultados en relación a otros trabajos del estado del arte, es un método ampliamente utilizado y reconocido en la literatura, contando a la fecha con 14014 citas de artículos y 408 citas de patentes [21]. Como parte de nuestro análisis, consideramos que presenta el inconveniente de que no realiza una evaluación local para establecer que pixeles del borde se deben de conservar y cuales deben de suprimirse. Esto ocasiona que para algunas imágenes se supriman bordes importantes cuando se manejan umbrales altos, o resulte una gran cantidad de bordes no relevantes si se establecen umbrales bajos. Este problema se presenta principalmente en imágenes donde existen regiones con altos contrastes y otras con contrastes muy bajos. Por ejemplo, en la figura 5D se observa que los bordes de la parte superior del sombrero se pierden, debido a que no rebasan el umbral establecido. Sin embargo, en caso de bajar los

umbrales para que los bordes del sombrero se conserven, se conservaría también demasiado detalle, que para efecto de detección de los objetos de la imagen no son relevantes.

Con base en lo anterior, en el Capítulo 3 se presenta una propuesta de análisis local para establecer los pixeles candidatos a ser bordes.

2.3 Deep Learning for Text Spotting [13]

En este trabajo presentan un sistema de detección y reconocimiento de texto en imágenes naturales, centrado en el manejo de caracteres de forma individual. Esto es, tratan a los caracteres como bloques atómicos de la construcción del texto, modelándolos de forma explícita y construyendo las líneas de texto y las palabras por medio del agrupamiento de los caracteres detectados. Para esto, utilizan una Red Neuronal Convolucional (RNC) para generar un mapa de rasgos sobresalientes de texto/no-texto que dirige la generación de recuadros candidatos a contener texto, y mapas de rasgos sobresalientes de caracteres y lo que nombran como *bigram* (reconocimiento de 2 caracteres contiguos) para asistir en el reconocimiento de palabras dentro de cada recuadro propuesto.

2.3.1 Revisión del método

El elemento central de éste sistema es un clasificador de caracteres basado en una RNC. Las salidas del clasificador se utilizan posteriormente para detectar regiones que contienen texto y para reconocer las palabras que se forman. Para clasificar una región o recuadro x como uno de los posibles caracteres (o como fondo), se extraen un conjunto de características $\Phi(x) = (\phi_1(x), \phi_2(x), .., \phi_K(x))$ con las cuales entrenan el clasificador binario f_c por cada carácter c del alfabeto C. El clasificador es entrenado para calcular una distribución de probabilidad $p(c|x) = f_c(\Phi(x))$ sobre todos los caracteres del alfabeto,

tomando el valor máximo para reconocer el carácter $\bar{c}$ contenido en el recuadro x (Ecuación (2.5)).

$$\bar{c} = \underset{c \epsilon C}{argmax}\ p(c|x) \qquad \textbf{(2.5)}$$

2.3.1.1 RNC para el aprendizaje de características

El entrenamiento para el clasificador de características lo dividen en dos etapas. En la primera etapa, entrenan una RNC como un clasificador de caracteres no sensitivo. En la segunda etapa, los extractores de características resultantes del aprendizaje de la RNC son aplicados a los demás problemas de clasificación requeridos. El producto de estas dos etapas son cuatro clasificadores RNC: un clasificador de texto/no-texto, un clasificador de caracteres no sensitivo, un clasificador de caracteres sensitivo y un clasificador bigram.

2.3.1.1.1 Etapa 1: Clasificador de caracteres no sensitivo

El clasificador no sensitivo utiliza una RNC de cuatro capas, generando como salida una probabilidad $p(c|x)$ sobre el alfabeto C incluyendo las 26 letras, 10 dígitos y ruido/fondo (como no texto), dando un total de 37 clases (Figura 2.6). La entrada $z_1 = x$ de la RNC son imágenes en escala de gris de caracteres recortados de 24 x 24 pixeles, cero-centrados y normalizados por la substracción de la media y divididos por la desviación estándar.

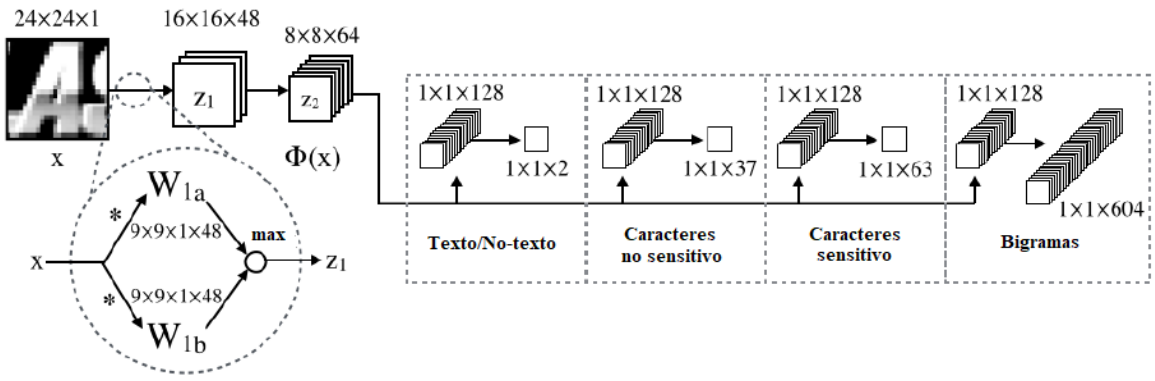

Figura 2.6. Arquitectura de la Red Neuronal Convolucional utilizada.

A diferencia de las arquitecturas convencionales de RNC, no aplican pooling espacial o sub-muestreo. Otra diferencia importante con respecto a los trabajos clásicos con RNCs, es que utilizan $maxout$ como alternativa a las funciones de activación no-lineales comunes como ReLU o Sigmoides. El $maxout$ de dos canales (agrupación de tamaño $g = 2$) de características z_i^1 y z_i^2 es simplemente el valor máximo entre ellos. Esto se puede extender a diferentes tamaños de agrupación g, resultando para una entrada de N_i canales de características, una salida de $N_i' = N_i/g$ nuevos canales.

A la imagen de entrada le aplicaron un convolución con 96 filtros de tamaño 9x9, resultando en un mapa de tamaño 16x16 y 96 características. A las 96 características le aplicaron un maxout de tamaño 2, reduciéndose a 48 canales. En las siguientes capas aplicaron convoluciones con 128, 512, 148 filtros de tamaño 9, 8, 1 y maxout de tamaño 2, 4, 4, resultando en mapas de características de 64, 128 y 37 de tamaño 8x8, 1x1 y 1x1 respectivamente. Las últimas 37 características alimentan un softmax para obtener la probabilidad de cada carácter.

El entrenamiento lo realizaron con el descenso del gradiente y retro-propagación. Para tratar de evitar sobre-entrenamiento aplican dropout en todas las capas excepto la primera capa de convolución. El dropout simplemente deja en cero de forma aleatoria una proporción de los parámetros de la red. La proporción que mantuvieron para cada capa son 1, 0.5, 0.5 y 0.5 respectivamente. Los datos de entrenamiento los aumentaron aplicando rotaciones de forma aleatoria y la inyección de ruido.

Finalmente, como parte de esta primera etapa, generaron lo que nombran como *mapa de prominencia por pixel*, que corresponde a una imagen con la misma resolución y tamaño que la imagen original, donde el color de cada pixel indica la probabilidad de que la región donde éste se encuentra corresponda o no a un carácter. El rango de colores va del rojo variando hasta llegar al color azul. El color rojo indica una mayor probabilidad de la existencia de un carácter, mientras que el azul corresponde a una menor probabilidad (Figura 2.7). El mapa de prominencia lo generaron aplicando directamente sobre la imagen completa los filtros resultantes del entrenamiento de manera convolucional.

Figura 2.7. Ejemplo de la generación de un mapa de prominencia por pixel.

2.3.1.1.2 Etapa 2: Aprendizaje del resto de los clasificadores

El entrenamiento con una gran cantidad de datos, incluyendo texto y no-texto, resultó en que las capas ocultas de la red generaron mapas de características altamente adaptadas para discriminar caracteres. Aprovechando esto, adaptaron estas salidas para las demás tareas de clasificación relacionadas con texto. Para esto, utilizaron la salida de la segunda capa convolucional como el conjunto de características de entrada $\Phi(x) = z_2$. A partir de estas características, entrenaron al clasificador de texto/no-texto (con 2 clases), al

clasificador de caracteres sensitivo (63 clases) y el clasificador bigram (604 clases), cada uno utilizando una RNC de dos capas aplicadas sobre $\Phi(x)$. Las últimas dos capas de las tres RNCs resultaron con mapas de características de 128-2, 128-63 y 128-604 respectivamente, como resultado de aplicar maxout con grupos de tamaño 4. Todas se entrenaron con dropout de 0.5 sobre todas las capas, con una reducción adaptativa de la tasa de aprendizaje.

2.3.1.2 Detección y Reconocimiento

La fase de detección toma una imagen como entrada y genera recuadros candidatos a contener palabras, aplicando el clasificador de texto/no-texto. Posteriormente, las palabras contenidas en los recuadros son reconocidas comparándolas con un diccionario, a través de los clasificadores de caracteres y bigram, en conjunto con otras herramientas de la geometría.

2.3.1.2.1 Detección del Texto

El objetivo de la fase de detección es tomar una imagen como entrada y generar un conjunto de regiones rectangulares, cada una conteniendo la imagen de una palabra. Este proceso inicia aplicando la RNC clasificador de texto/no-texto sobre una ventana que se desliza a través de toda la imagen. Este proceso lo repiten con 16 escalas de tamaños entre 16 y 260 pixeles, redimensionando la imagen de entrada. Este primer paso es para identificar líneas de texto. Para este fin se aplica un umbral de la probabilidad que debe superar cada recuadro. Posteriormente, estas regiones son conectadas en líneas de texto aplicando el *run length smoothing algoritm* (RLSA): para cada fila de pixeles la media μ y la desviación estándar σ de los espacios entre los picos de probabilidad se calculan y las regiones vecinas son conectadas si el espacio entre ellas es menor que $3\mu - 0.5\sigma$.

Encontrando componentes conectados de las regiones enlazadas resultan las líneas de texto candidatas.

El siguiente paso es dividir las líneas en palabras. Para esto, la imagen es recortada de forma ajustada de acuerdo a la línea de texto y se aplica un umbral para separar los posibles caracteres del fondo. Los componentes adyacentes son conectados si su espacio horizontal es menor que la media del espacio horizontal de la línea de texto. Los componentes conectados resultantes dan lo recuadros candidatos para palabras individuales. Finalmente, estos recuadros son filtrados basados en restricciones geométricas (tamaño del recuadro, sus proporciones, etc.) y tras someterse a una supresión no máxima, ordenándolos de forma descendente de acuerdo a la media de su puntaje de prominencia por píxel.

2.3.1.2.2 Reconocimiento de palabras

El objetivo de la fase de reconocimiento de palabras es tomar las imágenes recortadas de las palabras candidatas (de ancho W y altura H), y estimar el texto contenido en ellas. Para esto, a cada hipótesis de palabra se le asigna un puntaje basado en el uso de un diccionario predefinido. La entrada al reconocedor es el mapa de probabilidades de caracteres sensitivos, no sensitivos y del bigram generados por las RNCs. Restringido a las palabras recortadas, el resultado es un mapa de $W \times H$ que corresponde a la hipótesis de cada carácter. Estos mapas los reducen a un tamaño de $W \times 1$ promediando las columnas utilizando un *pesado Gaussiano* centrado en el fila intermedia. Agrupando las matrices de todos los caracteres resultan las *matrices de respuesta* $P \in \mathbb{R}^{37\times W}$, $Q \in \mathbb{R}^{63\times W}$, $R \in \mathbb{R}^{604\times W}$ para los clasificadores no-sensitivo, sensitivo y bigram, respectivamente (Figura 2.8).

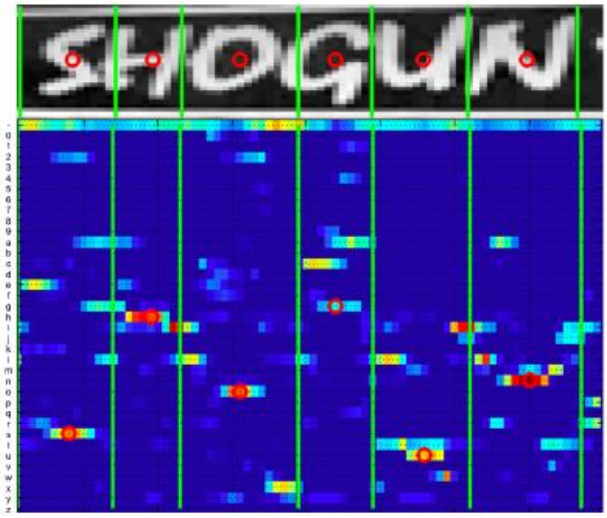

Figura 2.8. Ejemplo de una matriz de respuesta P para el caso no-sensitivo.

Dada las matrices P, Q y R el siguiente paso es obtener un puntaje para cada palabra candidata $w = (c_1, c_2, \dots, c_{L_w})$, donde L_w es el número de caracteres en la palabra. Se asigna $b^w = (b_1^w, b_2^w, \dots, b_{L_w+1}^w)$ como las $L_w + 1$ posiciones que marcan el punto de separación entre los caracteres, $-b_1^w$ corresponde al inicio del primer carácter, $b_{L_w}^w$ al inicio del último carácter y $b_{L_w+1}^w$ al final del último carácter. La hipótesis de punto de separación de la palabra completa (w, b^w) recibe el puntaje dado por la Ecuación (2.6). Este puntaje representa que tan bien la palabra w es explicada por los puntos de separación b^w y las matrices de respuesta P, Q y R. El puntaje es calculado para cada palabra candidata y maximizado a través de la selección de los puntos de separación óptimos utilizando programación dinámica. El resultado de la maximización w^* se toma como la palabra más probable.

$$s(w, b^w, P, Q, R) = \frac{1}{|b^w|}\left(\sum_{i=1}^{|b^w|} m_i(b_i^w, R) + \sum_{i=2}^{|b^w|} \emptyset(b_i^w, b_{i-1}^w, P, Q, R)\right) \quad \textbf{(2.6)}$$

Dada la palabra reconocida, el recuadro se ajusta para coincidir con la estimación de los puntos de separación y se agrega a una lista de regiones de palabras candidatas

reconocidas. El paso final es ejecutar una supresión no-máxima basada en el traslape de este conjunto de recuadros para eliminar las detecciones duplicadas.

2.3.2 Experimentos

2.3.2.1 Conjunto de entrenamiento

El clasificador no-sensitivo lo entrenaron con 186 mil caracteres recortados de 24 x 24 pixeles tomados de los conjuntos de entrenamiento *ICDAR* 2003, 2005, 2011 y 2013, de *KAIST*, de las imágenes naturales de *Chars74k* (no usaron las imágenes sintéticas), de *StanfordSynth* y de un conjunto de caracteres que ellos mismos crearon de manera automática. El clasificador sensitivo lo entrenaron con los mismos datos, excluyendo los que generaron ellos de forma automática, resultando en 107 mil muestras. El clasificador bigram lo entrenaron con 92 mil muestras, tomando los diferentes pares de letras presentes en ICDAR 2003, 2005, 2011 y 2013, KAIST y las imágenes generadas de forma automática.

2.3.2.2 Resultados

El sistema integral de detección y reconocimiento de palabras lo evaluaron sobre 251 imágenes de la ICDAR 2003 [22] y 249 imágenes de la base de datos SVT [23]. Un reconocimiento es considerado correcto si el recuadro que contiene la palabra tiene al menos un 50% de traslape con el recuadro real y si el texto predicho es correcto. El traslape entre dos recuadros b_1 y b_2 lo definen como la relación de su intersección sobre la unión (IoU): $\frac{|b_1 \cap b_2|}{|b_1 \cup b_2|}$. Los resultados de este método se muestran en la Tabla 2 y están expresados en términos del F_valor, el cual relaciona la precisión y la sensibilidad del reconocimiento como un valor único ($F\_valor = 2 \cdot \frac{Presición \cdot Sensibilidad}{Precisión + Sensibilidad}$).

Tabla 2. Resultados de la detección y reconocimiento de palabras (% F_valor).

Método	IC03-50	IC03-Completa	SVT-50
Wang et al., 2011 [23]	68	51	38
Wang et al., 2012 [8]	72	67	46
Alsharif y Pineau, 2014 [24]	77	70	48
Jaderberg, M., 2014 [13] (sin bigram)	76	---	50
Jaderberg, M., 2014 [13]	80	75	56

2.3.3 Análisis del Método

Uno de los aspectos más importantes a destacar de este método, es que logra porcentajes muy competitivos de correcta detección y reconocimiento, lo cual fue de inicio lo que nos motivó a considerar su revisión a detalle. Además, en este trabajo se presentan algunas ideas interesantes, las cuales fueron consideradas como parte de la propuesta de un nuevo método:

- El uso de Redes Neuronales Convolucionales, las cuales están dominando el estado del arte en esta problemática.
- La idea de basar la detección y reconocimiento en la generación de recuadros candidatos a contener texto.
- El entrenamiento y reconocimiento a nivel de caracteres en lugar de palabras completas, lo cual reduce de forma muy significativa la complejidad de las redes neuronales requeridas y los conjuntos de datos de entrenamiento.
- La propuesta de entrenar un clasificador para discernir directamente si una región es candidata a contener texto o no.
- La idea de generar datos de entrenamiento sintéticos a partir transformaciones gráficas de imágenes base.

Por otro lado, este método presenta algunas desventajas, las cuales se requieren tener presente para tratar de evitarlas en el diseño de una nueva propuesta:

- Gran cantidad de tiempo requerido para procesar una imagen, debido principalmente a las múltiples escalas y relaciones de tamaño que deben ser procesadas en las ventanas deslizantes durante la detección con las RNCs.
- La complejidad computacional del proceso para encontrar los puntos de separación óptimos de los caracteres (basado en programación dinámica), crece de forma lineal con el tamaño del diccionario utilizado en el reconocimiento, por lo que se vuelve un problema intratable para diccionarios de miles de palabras.
- El entrenamiento y uso en el reconocimiento de cuatro redes convolucionales distintas. Aunque 3 de las redes aprovechan parte del entrenamiento de una de ellas, debe ser más eficiente utilizar una sola red para todo el proceso.
- Depender de las muestras de las bases de datos de entrenamiento relacionadas con las bases de datos que se utilizarán para las pruebas.
- El uso del bigram requiere de una arquitectura más compleja de la RNC e incrementa significativamente la cantidad de muestras requeridas.

3 Métodos Propuestos

La estrategia general que proponemos en este proyecto para la lectura de texto en imágenes no restringidas, se divide en dos fases principales. La primera fase es la *detección* de las regiones candidatas a contener texto, seguida por la fase de *reconocimiento* del texto contenido en dichas regiones. Esta propuesta está inspirada principalmente en dos trabajos del estado del arte: *Edge Boxes: Locating Object Proposals from Edges* [12] para la fase de detección y *Deep Learning for Text Spotting* [13] para la fase de reconocimiento. Sin embargo, a pesar de que se siguen las ideas generales de estos trabajos, las estrategias y métodos específicos desarrollados, son substancialmente diferentes.

3.1 Detección de texto

La fase de detección de texto propuesto se divide en 4 etapas:

1. Identificación de bordes de las imágenes
2. Identificación de segmentos de contornos
3. Generación de recuadros candidatos
4. Filtrado de los recuadros

3.1.1 Identificación de bordes

La identificación de los bordes de una imagen suele ser una de las primeras fases y a la vez un factor clave del éxito de los métodos de detección de objetos. Por ejemplo, la primera fase del método que presentan en [12], es la identificación de los bordes aplicando el llamado *Detector Estructurado de Bordes*. En esta sección presentamos dos nuevas estrategias para la identificación de los bordes de una imagen. La primera propuesta se

basa en el diseño de Redes Neuronales Convolucionales, entrenadas a partir de contornos de letras generados de forma manual en un conjunto de imágenes. La segunda propuesta es una versión modificación del algoritmo de Canny, al cual se agrega una estrategia de análisis local de los pixeles candidatos, que sustituye la fase de Histéresis del método original.

3.1.1.1 Identificación de bordes a través de Redes Neuronales Convolucionales

Considerando el éxito que han tenido recientemente las Redes Neuronales Convolucionales en diferentes problemas de análisis de imágenes, decidimos probar una nueva estrategia para la detección de los bordes. El primer paso de esta propuesta consiste en trazar de forma manual el borde de un conjunto de letras contenidas en diferentes imágenes, con el objetivo de utilizarlos como ejemplos de entrenamiento de la red. La Figura 3.1 muestra ejemplos del trazado manual de bordes. En color rojo, pixeles que se pasan a la red como ejemplos de bordes. En color verde, pixeles que se pasan como ejemplos de no-borde.

Figura 3.1. Trazado manual de bordes.

Posteriormente, se entrenaron diferentes configuraciones de Redes Convolucionales. Para esto, las imágenes de muestra se dividieron en pequeños fragmentos de 5x5 y 3x3 pixeles. Para cada fragmento se entrenó y posteriormente se clasificó el pixel central, basado en la información de los pixeles que lo rodean. Una vez entrenadas las redes se realizaron pruebas con diferentes imágenes. La Figura 3.2 muestra ejemplos de la identificación de

bordes utilizando las Redes Neuronales Convolucionales. Primera fila, imágenes originales en tonos de gris. Segunda fila, bordes identificados utilizando fragmentos de imagen de 3x3 pixeles, con una primera capa de Convolución de 50 filtros de 3x3 y activación ReLU, una segunda capa de Pooling de 1x1 y una capa final con una neurona con activación Sigmoide. Tercera fila, bordes identificados utilizando fragmentos de imagen de 5x5 pixeles, con una primera capa de convolución de 200 filtros de 5x5 con activación ReLU, una segunda capa de Pooling de 1x1 y una capa final con una neurona con activación Sigmoide. Cuarta fila, bordes identificados utilizando fragmentos de 5x5 pixeles, con una primera capa de Convolución de 50 filtros de 3x3 con activación ReLU, una capa de Pooling de 2x2, una capa totalmente conectada de 40 neuronas con activación Sigmoide y una capa final con una neurona con activación Sigmoide.

Se observa que mientras más complejo es el diseño de la red, se genera menos ruido en las imágenes de bordes resultantes. Sin embargo, de acuerdo a como aumenta la complejidad de la red, también se incrementa significativamente el tiempo de entrenamiento y de las pruebas realizadas. Para evaluar esta propuesta se utilizaron 20 imágenes para el entrenamiento de la red. Sin embargo, consideramos que para obtener resultados aceptables se requiere preparar una cantidad mucho mayor de ejemplos de entrenamientos.

Figura 3.2. Identificación de bordes a través de Redes Neuronales Convolucionales.

Una segunda propuesta para la identificación de bordes es la adaptación del algoritmo de Canny, que se presenta en la siguiente sección.

3.1.1.2 Identificación de bordes a través del algoritmo de Canny modificado

Esta segunda propuesta para la obtención de los bordes de una imagen, está basada en el algoritmo de Canny, con la substitución del paso de histéresis por una propuesta de análisis local de los pixeles candidatos. El método inicia con el suavizado de las imágenes, seguido por el cálculo de la magnitud y orientación del gradiente de cada pixel, continúa con la aplicación de la supresión no-máxima y finalmente el análisis local para suprimir o conservar los pixeles finales de borde. Los primeros pasos de suavizado, gradiente y supresión no-máxima se implementaron de acuerdo al algoritmo original de Canny, tal como se presenta en el capítulo anterior. El paso propuesto de análisis local se presenta a continuación.

3.1.1.2.1 Análisis local para identificación de bordes

La idea general de esta propuesta es conservar los pixeles con mayor contraste de color analizando regiones pequeñas. Esto tiene el propósito de conservar el texto que pueda encontrarse en zonas de poco contraste de color, sin conservar gran cantidad de detalle en zonas donde predomina un alto contraste. Esto es, dos pixeles que tienen un mismo valor bajo de contraste, uno puede conservarse como borde si la zona donde se localiza tienen en general un bajo contraste y el otro pixel puede descartarse si la zona donde está ubicado predomina un alto contraste. La Figura 3.3 muestra un ejemplo de este caso. En la parte inferior de la figura, los pixeles que forman el borde del texto presentan menor contraste en relación a los pixeles que forman bordes en la parte superior del texto.

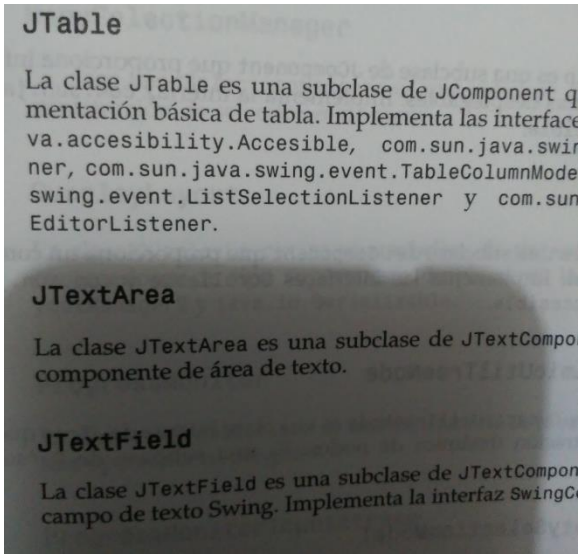

Figura 3.3. Imagen con zonas de texto con diferente contraste de color.

Para implementar esta propuesta, primero se descartan como bordes aquellos pixeles que tienen una magnitud por debajo de cierto umbral. Esto simplifica la cantidad de cálculos a realizar en las siguientes etapas del método. Posteriormente, se compara la magnitud del gradiente de cada pixel candidato con la magnitud de los pixeles vecinos. El tamaño de vecindad se establece previamente como parámetro del método. Si la magnitud del pixel es mayor que un porcentaje establecido de pixeles vecinos, se conserva como borde, en caso contrario se descarta. El Algoritmo 3.1 corresponde a este proceso. La Figura 3.4 muestra ejemplos de la aplicación de las diferentes fases del método de Canny modificado. Primera fila, imágenes de después de aplicar un filtro de suavizado. Segunda fila, imágenes generadas con base en la magnitud del gradiente. Tercera fila, imágenes con supresión no-máxima. Cuarta fila, imágenes con bordes umbralizados con el método propuesto de análisis local.

Algoritmo 3.1. Análisis local para la identificación de bordes

Entrada: Imagen generada después de la Supresión No Máxima
Salida: Imagen con los bordes finales generados

1. Se define DIST_COMP (p.e. 10)
2. Se define MIN_MEJOR (p.e. 0.50)
3. Para cada pixel p de la imagen
4. Si p se conservó como borde después de la fase de supresión
5. contCompara = 0
6. contMejor = 0
7. Para cada pixel v a una distancia no mayor de DIST_COMP de p
8. Se incrementa contCompara
9. Si la magnitud de p es mayor que la magnitud de v
10. Se incrementa contMejor
11. Fin Si
12. Fin Para
*13. Si contMejor es menor que (contCompara * MIN_MEJOR)*
14. Se descarta p como borde
15. Fin Si
16. Fin Si
17. Fin Para

Figura 3.4. Identificación de bordes a través del algoritmo de Canny modificado.

3.1.2 Identificación de segmentos de contornos

Después de identificar los bordes de la imagen, la siguiente etapa del método de detección, es agruparlos de acuerdo a la afinidad de sus vectores gradientes. Esto tiene como objetivo identificar aquellos bordes que forman parte del contorno de un mismo objeto. A pesar de que esta secuencia de etapas corresponde a la idea general del trabajo de [12], los métodos específicos para cada una son substancialmente diferentes. Incluso, los grupos de bordes que se obtienen en este método son de mayor tamaño que en [12], llegando a identificarse el contorno completo de una letra, si ésta se encuentra bien definida en la imagen.

El método propuesto para la identificación de Segmentos de Contornos (Algoritmo 3.2), recorre cada pixel p clasificado como borde de la imagen y evalúa cada uno de sus 8-conectados vecinos v. Si el ángulo y magnitud de p y v son similares, se asocian ambos a un mismo contorno. En el caso de que p ni v estuvieran asociados previamente a contornos, se crea uno nuevo y se asocian ambos a éste. Si v ya estuviera asociado a un contorno y p no, se asocia p al de v. Si estuvieran asociados a contornos diferentes, tanto p como todos los pixeles que pertenecen a su contorno, se asocian al de v. Esto último, corresponde a unir dos contornos contiguos afines. La Figura 3.5 muestra ejemplos de la aplicación de este método. Los pixeles que pertenecen a un mismo contorno están desplegados de un mismo color.

Algoritmo 3.2. Identificación de segmentos de contornos

Entrada: Imagen generada con los bordes identificados
Salida: Imagen con los segmentos de contornos identificados

1. Se define DIF_ANG_MAX (p.e. $\pi/4$, $\pi/12$)
2. Se define DIF_MAGN_MAX (p.e. 0.20, 0.50)
3. Se define MIN_PIXS_CONTO (p.e. 10, 20)
4. Para cada pixel p de la imagen
5. Si p es borde
6. Para cada uno de los 8-conectados vecinos v de p
7. Si v es borde
8. difAng = $Abs(\theta_p - \theta_v)$
*9. dif_mag_max = $Max(m_p, m_v)$ * DIF_MAGN_MAX;*
10. diifMagn = $Abs(m_p - m_v)$
11. Si difAng <= DIF_ANG_MAX y difMagn <= dif_mag_max
12. Si v no pertenece a un contorno
13. Si p no pertenece a un contorno
14. Crea un nuevo contorno
15. Asigna p al nuevo contorno
16. Fin Si
17. Asigna v al contorno de p
18. Caso Contrario
19. Si p pertenece a un contorno diferente de v
20. Para cada pixel k que pertenece al contorno de p
21. Asigna k al contorno de v
22. Fin Para
23. Elimina el contorno anterior de p
24. Caso Contrario, Si p no pertenece a un contorno
25. Asigna p al contorno de v
26. Fin Si
27. Fin Si
28. Fin Si
29. Fin Si
30. Fin Para
31. Fin Si
32. Fin Para
33. Se conservan los contornos con más de MIN_PIXS_CONTO pixeles

Figura 3.5. Ejemplo de segmentos de contornos identificados.

3.1.3 Generación de recuadros

Una vez identificados los segmentos de contornos, la siguiente etapa es generar recuadros que encierran grupos de estos segmentos. Esto tiene como objetivo tratar de encerrar dentro de un mismo recuadro todos los segmentos de contorno que corresponden a una misma letra. Para esto, se generan recuadros que inicialmente abarcan los segmentos individuales de contornos identificados en el paso anterior y posteriormente se generan nuevos recuadros a través de la fusión de dos o más recuadros que se traslapan o son vecinos. Estos nuevos recuadros contendrán dos o más segmentos de contornos, que en su conjunto pueden formar una letra. Lograr que todas las letras de la imagen queden encerradas por un recuadro de forma ajustada y completa, es lo que se considera como detección correcta del texto. Cada recuadro generado se analiza posteriormente por una Red Neuronal Convolucional, en la etapa que nombramos *filtrado de recuadros*, para evaluar la probabilidad de que su contenido sea una letra.

Del Algoritmo 3.3 al Algoritmo 3.6 corresponden al método de generación de recuadros. El algoritmo recibe como entrada la imagen y los segmentos de contorno identificados en el paso anterior e inicia generando recuadros que encierran a un solo contorno de forma ajustada. La Figura 3.6 muestra un ejemplo de la aplicación de este método. En las primeras dos filas se muestra la imagen original y los segmentos de contorno generados, respectivamente. La Figura 3.6 tercera fila, presenta un ejemplo de los primeros recuadros. Se observa que la letra T está encerrada completamente por un solo recuadro, que abarca únicamente los pixeles que la forman. Con esto, se considera que se logró la detección de la letra. Para las demás letras, existen dos o más recuadros que abarcan parcialmente el grupo de contornos que las forman.

Algoritmo 3.3. Generación de recuadros

Entrada: Imagen con los segmentos de contornos identificados
Salida: Imagen con los recuadros generados

1. Para cada segmento de contorno
2. Generar un nuevo recuadro que abarca de forma ajustada al segmento
3. Se establece como activo el recuadro
4. Se calcula la magnitud mínima y máxima de los pixeles del segmento contenido
5. Se calcula su tamaño y área
6. Fin Para
7. Para cada recuadro activo $b1$ generado en el bloque anterior
8. Para cada recuadro activo $b2$ en la lista a partir de $b1$
9. Si $b1$ y $b2$ son afines (Algoritmo 3.4)*, cumpliéndose que:*
a) Se traslapan o son vecinos a una distancia no mayor de 2 pixeles
b) La diferencia de sus áreas tiene una proporción no mayor de 20
c) El recuadro resultante tiene una proporción entre sus lados no mayor de 5
10. Se crea un nuevo recuadro b como resultado de "fusionar" $b1$ y $b2$ (Algoritmo 3.5)
11. Se "valida" que b no se traslape en un 100% con otro recuadro (Algoritmo 3.6)
12. Fin Si
13. Fin Para
14. Fin Para
15. Para cada recuadro activo $b1$ generado en el bloque anterior
16. Para cada recuadro activo $b2$ en la lista a partir de $b1$
17. Si $b1$ y $b2$ son afines, cumpliéndose que:
a) La diferencia de sus áreas tiene una proporción no mayor de 3
b) Se traslapan por lo menos en un 40% del área del mayor de ambos
c) Los intervalos de las magnitudes de sus pixeles se traslapan
d) El recuadro resultante tiene una proporción entre sus lados no mayor de 5
18. Se crea un nuevo recuadro b como resultado de fusionar $b1$ y $b2$
19. Se valida que b no se traslape en un 100% con otro recuadro
20. Fin Si
21. Fin Para
22. Fin Para
23. Para cada recuadro activo $b1$ generado en el bloque anterior
24. Para cada recuadro activo $b2$ en la lista a partir de $b1$
25. Si $b1$ y $b2$ son afines, cumpliéndose que:
a) La diferencia de sus áreas tiene una proporción no mayor de 2
b) Se traslapan por lo menos en un 80% del área del mayor de ambos
c) Los intervalos de las magnitudes de sus pixeles se traslapan
d) El recuadro resultante tiene una proporción entre sus lados no mayor de 5
26. Se crea un nuevo recuadro b como resultado de fusionar $b1$ y $b2$
27. Se valida que b no se traslape en un 100% con otro recuadro
28. Fin Si
29. Fin Para
30. Fin Para

Algoritmo 3.4. Recuadros afines

Entrada: Recuadros a comparar $b1$ y $b2$,
Distancia máxima para considerarse vecinos $distMax$,
Proporción máxima de las áreas $propAreaMax$,
Área mínima de traslape $areaTrasMin$,
Diferencia máxima de las magnitudes $difMagnMax$,
Proporciones máxima del recuadro resultante $propMax$
Salida: Valor booleano que indica si los recuadros $b1$ y $b2$ son afines

// Si los recuadros se encuentran muy separados, no son afines
1. Si $xMax_{b2} < xMin_{b1} - distMax$ *o* $xMin_{b2} > xMax_{b1} + distMax$
$yMax_{b2} < yMin_{b1} - distMax$ *o* $yMin_{b2} > yMax_{b1} + distMax$
2. retorna falso
3. Fin Si
// Si el área de un recuadro es mucho mayor que el otro, no son afines
4. Si $max(area_{b1}, area_{b2})/min(area_{b1}, area_{b2}) > propAreaMax$
5. retorna falso
6. Fin Si
// Si los recuadros no tienen un traslape igual o mayor al requerido, no son afines
7. $sumaTamX = xTam_{b1} + xTam_{b2}$
8. $sumaTamY = yTam_{b1} + yTam_{b2}$
9. $nuevoTamX = max(xMax_{b1}, xMax_{b2}) - min(xMin_{b1}, xMin_{b2})$
10. $nuevoTamY = max(yMax_{b1}, yMax_{b2}) - min(yMin_{b1}, yMin_{b2})$
11. $areaTrasl = (sumaTamX - nuevoTamX) * (sumaTamY - nuevoTamY)$
12. Si $(areaTrasl < max(area_{b1}, area_{b2}) * areaTraslMin)$
13. retorna falso
14. Fin Si
// Si la diferencia entre los rangos de magnitudes de los recuadros es muy grande, no son afines
15. Si $magnMax_{b2} < magnMin_{b1} * (1 - difMagnMax)$ *o*
$magnMin_{b2} * (1 - difMagnMax) > magnMax_{b1}$
16. retorna falso
17. Fin Si
// Si la proporción de lados del recuadro resultante de unir $b1$ *y* $b2$ *es muy grande, no son afines*
18. $propNuevo = max(nuevoTamY, nuevoTamX)/min(nuevoTamY, nuevoTamX)$
19. Si $propNuevo > propMax)$
20. retorna falso
21. Fin Si
// Si no se cumple algo de lo anterior, $b1$ *y* $b2$ *se consideran afines*
22. retorna verdadero

Algoritmo 3.5. Fusionar recuadros

Entrada: Recuadros a fusionar $b1$ y $b2$
Salida: Nuevo recuadro b resultado de la fusión

1. Se crea un nuevo recuadro b
2. $xMin_b = min(xMin_{b1}, xMin_{b2})$
3. $yMin_b = min(yMin_{b1}, yMin_{b2})$
4. $xMax_b = max(xMax_{b1}, xMax_{b2})$
5. $yMax_b = max(yMax_{b1}, yMax_{b2})$
6. $magnMin_b = min(magnMin_{b1}, magnMin_{b2})$
7. $magnMax_b = max(magnMax_{b1}, magnMax_{b2})$
8. $sumaMagn_b = sumaMagn_{b1} + sumaMagn_{b2}$
9. Para cada contorno c en el recuadro $b1$ *y* $b2$
10. Agregar c al recuadro b
11. Fin Para
12. $tamX_b = xMax_b - xMin_b + 1$
12. $tamY_b = yMax_b - yMin_b + 1$
14. $area_b = tamX_b + tamY_b$

Algoritmo 3.6. Validar recuadro

Entrada: Recuadro b a validar
Lista lb de recuadros
Salida: Recuadro de lb posiblemente modificado y
Valor booleano que indica si el recuadro b es válido

1. Para cada recuadro $b2$ *de* lb
2. Si $xMin_b = xMin_{b2}$ *y* $xMax_b = xMax_{b2}$ *y* $yMin_b = yMin_{b2}$ *y* $yMax_b = yMax_{b2}$
3. Si $sumaMagn_b \neq sumaMagn_{b2}$
4. Para cada contorno c de b
5. Si c no pertenece a $b2$
6. $xMin_{b2} = min(xMin_c, xMin_{b2})$
7. $yMin_{b2} = min(yMin_c, yMin_{b2})$
8. $xMax_{b2} = max(xMax_c, xMax_{b2})$
9. $yMax_{b2} = max(yMax_c, yMax_{b2})$
10. $magnMin_{b2} = min(magnMin_c, magnMin_{b2})$
11. $magnMax_{b2} = max(magnMax_c, magnMax_{b2})$
12. $sumaMagn_{b2} += magnMin_c + magnMax_c$
13. Agregar c a $b2$
14. Fin Si
15. Fin Para
16. Fin Si
17. Retorna falso
18. Fin Para
19. Retorna verdadero

Después de la generación inicial de recuadros, se ejecutan 3 bloques de fusión. En el primer bloque se fusionan pares de recuadros que se traslapan o que son vecinos y que presentan afinidad en la magnitud del gradiente de sus pixeles. Esto tiene como objetivo agrupar segmentos de contornos cercanos que posiblemente formen parte de una misma letra. En la Figura 3.6 cuarta fila, se observa que las letras O, R e Y, se logran encerrar por completo por un solo recuadro como resultado de esta primera fusión. Sin embargo, los segmentos de pares de letras T-R (recuadro verde) y R-Y (recuadro azul), también se fusionan de forma incorrecta.

En el segundo bloque se fusionan los recuadros generados en el bloque anterior que presentan más del 40% de traslape. En la Figura 3.6 quinta fila, se observa que de los 4 recuadros que se generaron inicialmente para las letras N, en este bloque se redujeron a 2. Por otro lado, las letras T, R y parte de la Y, se unieron en un solo recuadro. En el tercer y último bloque se fusionan los recuadros del bloque anterior que se traslapan más del 80%. En la Figura 3.6 sexta fila, se observa que finalmente se logra encerrar la letra N por un solo recuadro que abarca todos los segmentos de contorno que la forman. De igual manera se logra con las letras R e Y.

De esta forma, para el ejemplo de la Figura 3.6, se logran generar recuadros que encierran de forma ajustada cada una de las letras, con lo que asumimos que se logra una completa detección. Sin embargo, además de estos recuadros que podemos considerar correctos, también se generan varios recuadros que abarcan letras de forma parcial y recuadros que abarcan más de una letra. Todos los recuadros generados, tanto los iniciales como lo generados durante los 3 bloques de fusión, se pasan y evalúan en la siguiente etapa de filtrado.

Figura 3.6. Generación de recuadros.

Después de realizar diferentes pruebas con los algoritmos de generación de recuadros, se observó que mientras más bloques de fusión se ejecutan, aumenta la probabilidad de generar recuadros que encierran de forma correcta cada letra. Sin embargo, también se incrementa de forma considerable la cantidad de recuadros incorrectos. La desventaja de generar un mayor número de recuadros, es el aumento importante del tiempo de ejecución del método, así como el incremento de falsos positivos en la detección del texto en la imagen. Por experimentación con diferentes imágenes que presentan condiciones variadas en el texto, se determinó que 3 bloques de fusión mantienen un equilibrio entre un buen porcentaje de recuadros correctos y una cantidad aceptable (pocos miles) de recuadros generados. La Figura 3.7 muestra ejemplos de recuadros generados.

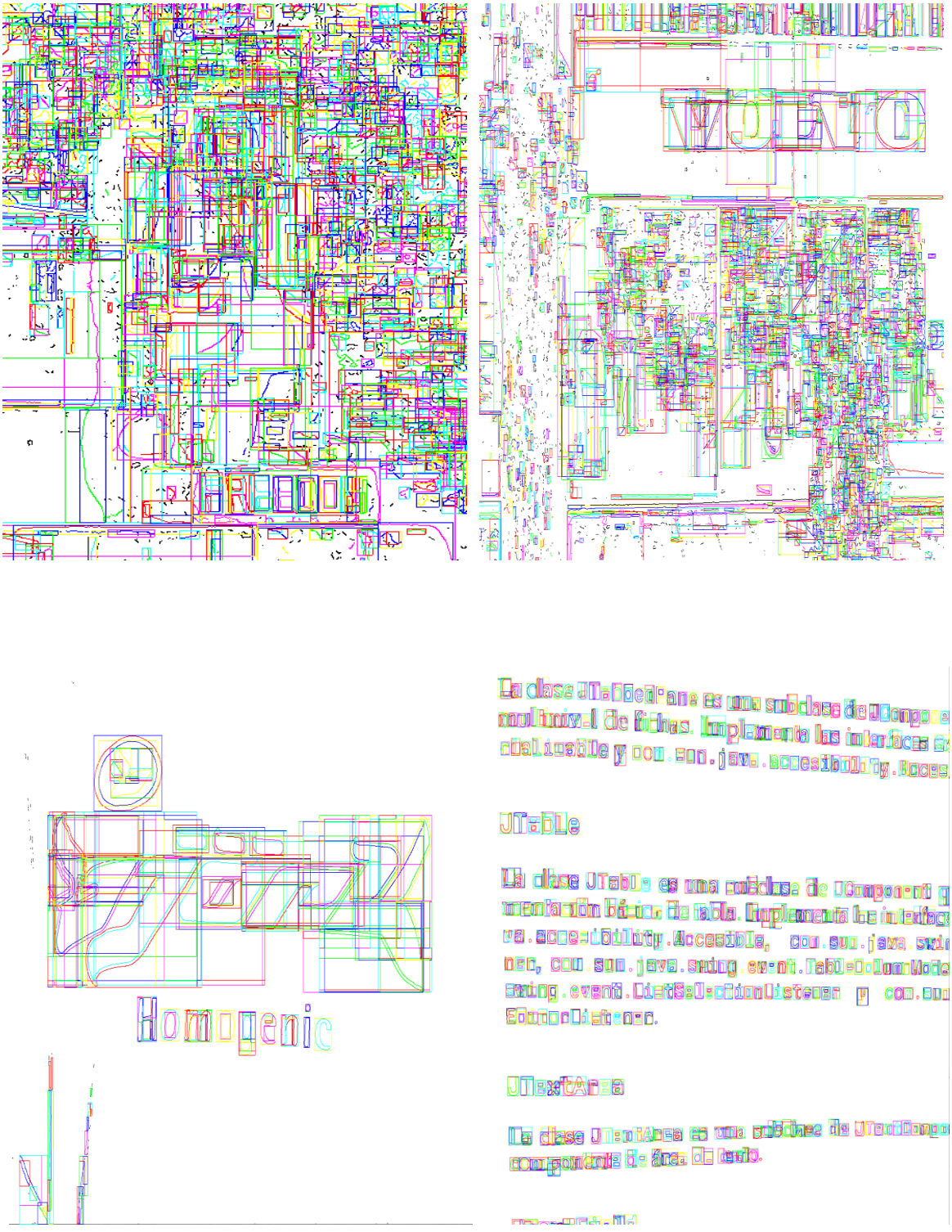

Figura 3.7. Aplicación del método de generación de recuadros.

3.1.4 Filtrado de recuadros

Una vez generados los recuadros candidatos, el siguiente paso es analizar cada uno de éstos para evaluar la probabilidad de que su contenido sea una letra. Para este propósito, se utilizó una Red Neuronal Convolucional, entrenada a partir de un conjunto de imágenes de letras y otro conjunto de imágenes de no-letras. Ambos conjuntos se generaron como parte de este proyecto.

3.1.4.1 Generación del conjunto de imágenes de letras

Para la generación de las imágenes de las letras se aplicaron diversos pasos. Se inició generando una imagen con los contornos de las letras y dígitos (referidos en el resto del documento únicamente como “letras”), de las fuentes más comunes de los editores de texto. La Figura 3.8 muestra ejemplos de los contornos de algunas fuentes de letras generadas. Posteriormente, se crearon y almacenaron imágenes individuales de las letras. Para esto, se desarrolló un método recursivo para identificar la posición de cada una dentro de la imagen general (Algoritmo 3.7). Una vez recibida la posición de cada letra, se recortó y se almacenó en un archivo independiente. La Figura 3.9 muestra ejemplos de las letras generadas como imágenes individuales.

Figura 3.8 Ejemplo de contornos de letras generadas en un solo archivo

Algoritmo 3.7. Identificación de la posición de las letras

Entrada: Imagen con letras generadas
Arreglo de coordenadas visitadas
Posición de un pixel que pertenece al borde de la letra
Coordenada inferior-izquierda candidata de la letra
Coordenada superior-derecha candidata de la letra
Salida: Coordenada candidata actualizada de la letra

1. *Se asigna el estatus visitada a la posición del pixel recibido*
2. *Si el pixel se encuentra abajo o a la izquierda de la coordenada inferior-izquierda candidata*
3. *Se actualiza la posición candidata con la coordenada del pixel recibido*
4. *Fin Si*
5. *Si el pixel se encuentra arriba o a la derecha de la coordenada superior-derecha candidata*
6. *Se actualiza la posición candidata con la coordenada del pixel recibido*
7. *Fin Si*
8. *Si un pixel de los 8-vecinos del pixel recibido no ha sido visitado y es borde de letra*
9. *Llamada recursiva, enviando la posición del pixel vecino*
10. *Fin Si*

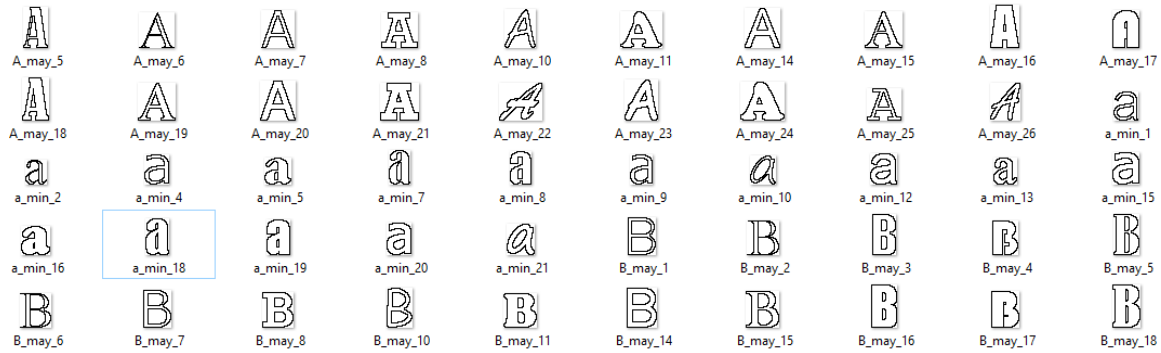

Figura 3.9. Ejemplos de letras generadas en archivos individuales.

3.1.4.1.1 Normalización de Letras

A continuación, con el fin de obtener un mejor entrenamiento de la Red Neuronal Convolucional, se generó una mayor variedad de letras, a través de aplicar rotaciones y escalados verticales y horizontales a cada una. Para esto, se realizó lo que nombramos *normalización* de cada letra, que consiste en homogenizar sus tamaños a 200 x 200 pixeles, afinar sus bordes a un pixel de grosor y almacenarlas en archivos de imagen de 350 x 350 pixeles. El Algoritmo 3.8 corresponde a este proceso de normalización. El objetivo de la normalización es homogenizar el tamaño y grosor de todas las letras, así como incrementar su tamaño y calidad, para no perder definición al momento de aplicar las rotaciones. La Figura 3.10 muestra ejemplos de las letras normalizadas.

Algoritmo 3.8. Normalización de letras

Entrada: Imagen de contorno de una letra
Salida: Imagen del contorno escalado y adelgazado de la letra

1. *Asignar* $TAM = 350,\ TAM\_ESC = 200$ *y* $N = 4$
2. *Generar una imagen en memoria de tamaño* TAM_ESC *con fondo blanco*
3. *Identificar el punto medio de la nueva imagen* (pfN)
4. *Identificar el punto medio de la imagen original* (pfY, pfX)
5. *Calcular el factor de escalamiento entre la imagen original y la nueva (s)*
6. *Para cada pixel (x, y) de la nueva imagen*
7. $yp = pfY + (y - pfN) * s$
   $xp = pfX + (x - pfN) * s$

8. Si el pixel con coordenadas (xp, yp) de la imagen original es color negro
9. Establecer como color negro el pixel (x, y) de la nueva imagen
10. Fin Si
11. Fin Para
12. Generar una imagen final en memoria de tamaño TAM con fondo blanco
13. Insertar de forma centrada, la imagen escalada dentro de la imagen final
14. Repetir N veces
15. Crear e inicializar a falso la matriz booleana $adelgazar$ de tamaño $TAMxTAM$
16. Para cada punto (x, y) de la imagen final
17. Si el pixel en (x, y) es color negro y
(sus 3 vecinos de arriba son blancos y su vecino inferior es negro, o
sus 3 vecinos de la izquierda son blancos y su vecino derecho es negro)
18. Establecer el estatus $adelgazar$ de (x, y) como verdadero
19. Fin Si
20. Fin Para
21. Para cada punto (x, y) de la imagen final
22. Si el estatus $adelgazar$ en (x, y) es verdadero
23. Asignar color blanco al pixel (x, y)
24. Fin Si
25. Fin Para
26. Para cada punto (x, y) de la imagen final
27. Si el pixel en (x, y) es color negro y
(sus 3 vecinos de abajo son blancos y su vecino superior es negro, o
sus 3 vecinos de la derecha son blancos y su vecino izquierdo es negro)
28. Establecer el estatus $adelgazar$ de (x, y) como verdadero
29. Fin Si
30. Fin Para
31. Para cada punto (x, y) de la imagen final
32. Si el estatus $adelgazar$ en (x, y) es verdadero
33. Asignar color blanco al pixel (x, y)
34. Fin Si
35. Fin Para
36. Crear la matriz booleana $escalón$ de tamaño $TAMxTAM$ inicializada en falso
37. Para cada punto (x, y) de la imagen final
38. Si el color del pixel (x, y) es negro y
sus 3 vecinos de una esquina son blancos y los 3 de la esquina opuesta son negros
39. Establecer el estatus $escalón$ de (x, y) como verdadero
40. Fin Si
41. Fina Para
42. Para cada punto (x, y) de la imagen final
43. Si el estatus $escalón$ en (x, y) es verdadero y se mantienen las condiciones de la línea 38
44. Asignar color blanco al pixel (x, y)
45. Fin Si
46. Fin Para
47. Fin Repetir
48. Guardar en archivo la imagen final generada

Figura 3.10. Ejemplos de letras normalizadas.

3.1.4.1.2 Rotación de letras

Una vez normalizadas las letras, se aplicó a cada una rotaciones de -10, -5, 5 y 10 grados. La rotación tiene como objetivo considerar que las imágenes naturales pueden contener texto que no siempre se encuentra alineado perfectamente con la horizontal. La intención es que el algoritmo sea capaz de detectar y reconocer el texto, aunque éste se encuentre ligeramente inclinado hacia la izquierda o a la derecha. Durante este proceso de rotación se eliminó el espacio en blanco alrededor de las letras que se incluyó durante el paso de normalización. Este espacio se incluyó para evitar que durante el cálculo de la rotación algún pixel, que forma parte del contorno de las letras, quede fuera del espacio rectangular definido para almacenar la imagen. El Algoritmo 3.9 corresponde al proceso de rotación. La Figura 3.11 muestra ejemplos de letras generadas después de la rotación.

Algoritmo 3.9. Rotación de letras

Entrada: Imagen de letra normalizada
Salida: Imágenes de la letra rotada -10, -5, 5 y 10 grados

1. *Asignar* $TAM =$ *tamaño actual de la imagen de la letra normalizada*
2. *Crear nueva imagen de tamaño* TAM *x* TAM *con fondo blanco*
3. *Obtener el punto medio de la imagen* (pf)
4. *Para* $a = -10$ *hasta* 10 *con incrementos de* 5
5. *Asignar* -1 *a* $xMin, yMin, xMax$ *y* $yMax$
6. *Para cada pixel* (x, y) *de la imagen de entrada*
7. *Si el pixel* (x, y) *es color negro*
8. $xp = pf + (x - pf)\ Cos\ a - (y - pf)\ Sen\ a$
9. $yp = pf + (x - pf)\ Sen\ a + (y - pf)\ Cos\ a$
10. *Asignar color negro al pixel* (xp, yp) *de la nueva imagen*
11. *Si* xp *es menor que* $xMin$ *asignar* xp *a* $xMin$
12. *Si* xp *es mayor que* $xMax$ *asignar* xp *a* $xMax$
13. *Si* yp *es menor que* $yMin$ *asignar* yp *a* $yMin$
14. *Si* yp *es mayor que* $yMax$ *asignar* yp *a* $yMax$
15. *Fin Si*
16. *Fin Para*
17. $difY = yMax - yMin$
18. $difX = xMax - xMin$
19. *Si* $difX > difY$
20. $TAM\_NUE = difX + 1$
21. $incX = xMin$
22. $incY = yMin - (difX - difY)/2$
23. *Caso contrario*
24. $TAM\_NUE = difY + 1$
25. $incY = yMin$
26. $incX = xMin - (difY - difX)/2$
27. *Fin Si*
28. *Crear imagen final con fondo blanco de tamaño* TAM_NUE *x* TAM_NUE
29. *Para cada pixel* (x, y) *de la imagen final*
30. *Si el color del pixel* $(x + incX, y + incY)$ *de la imagen rotada es negro*
31. *Asignar color negro al pixel* (x, y) *de la imagen final*
32. *Fin Si*
33. *Fin Para*
34. *Guardar la imagen final con un nombre asociado al ángulo de rotación aplicado*
35. *Fin Para*

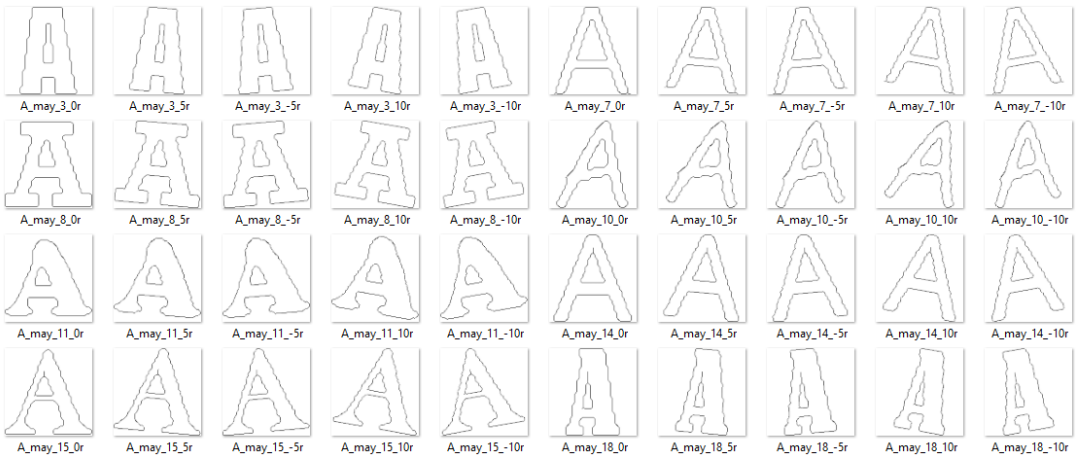

Figura 3.11. Ejemplos de letras rotadas.

3.1.4.1.3 Escalamiento de Letras

A partir de cada letra rotada, se obtuvieron 5 variantes de nuevas letras aplicando escalamiento horizontal y vertical. Para esto, primero se escalaron las letras rotadas manteniendo su tamaño proporcional a un espacio de 28 x 28 pixeles y después se aplicó un escalamiento a 24 x 28, 20 x 28, 28 x 24 y 28 x 20 pixeles. En todos los casos las letras se almacenaron centradas en un archivo de 28 x 28, el cual, tomando como ejemplo la base de datos de dígitos MNIST, es el tamaño elegido para el entrenamiento y posterior clasificación con la Red Neuronal Convolucional. El Algoritmo 3.10 corresponde a este proceso de escalamiento. De cada fuente de letra original, se generaron 25 versiones diferentes. Esto es, de la original se generaron 5 versiones rotadas y de cada una de las letras rotadas se crearon 5 versiones escaladas. Finalmente, se generaron 20,600 letras para utilizarlas como muestras de entrenamiento de la Red. La Figura 3.12 muestra ejemplos de las letras escaladas.

Algoritmo 3.10. Escalamiento de letras

Entrada: Imagen de letra rotada
Salida: Imágenes de la letra escaladas a 28x28, 24x28, 20x28, 28x24 y 28x20

1. *Asignar TAM_ORI = tamaño actual de la imagen de la letra rotada*

2. $TAM = 28$
3. $pfO = (TAM\_ORI - 1)/2$
4. $pfN = (TAM - 1)/2$
5. $sy = TAM/TAM\_ORI$
6. *Repetir para* $TAM\_X = 28$ *hasta* 20 *con decrementos de* 4
7. *Crear nueva imagen en memoria con fondo blanco de tamaño* TAM *x* TAM
8. $sx = TAM\_X/TAM\_ORI$
9. *Para cada pixel* (x, y) *de la imagen de entrada*
10. *Si color del pixel* (x, y) *de la imagen de entrada es color negro*
11. $yp = pfN + (y - pfO) * sy$
12. $xp = pfN + (x - pfO) * sx$
13. *Asignar color negro al pixel* (xp, yp) *de la nueva imagen*
14. *Fin Si*
15. *Fin Para*
16. *Guardar la imagen asociando su nombre al escalamiento en el eje* X
17. *Fin Repetir*
18. $sx = TAM/TAM\_ORI$
19. *Repetir para* $TAM\_Y = 24$ *hasta* 20 *con decremento de* 4
20. *Crear nueva imagen en memoria con fondo blanco de tamaño* TAM *x* TAM
21. $sy = TAM\_Y/TAM\_ORI$
22. *Para cada pixel* (x, y) *de la imagen de entrada*
23. *Si color del pixel* (x, y) *de la imagen de entrada es color negro*
24. $yp = pfN + (y - pfO) * sy$
25. $xp = pfN + (x - pfO) * sx$
26. *Asignar color negro al pixel* (xp, yp) *de la nueva imagen*
27. *Fin Si*
28. *Fin Para*
29. *Guardar la imagen asociando su nombre al escalamiento en el eje* Y
30. *Fin Repetir*

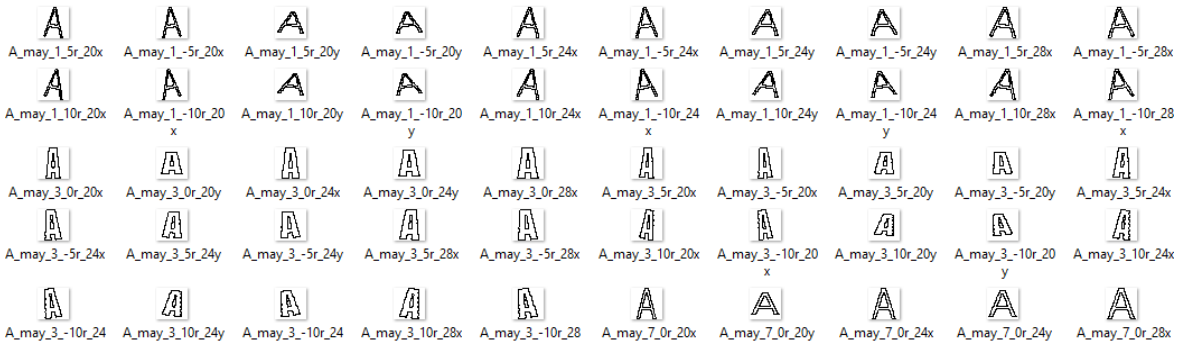

Figura 3.12. Muestras de letras finales rotadas y escaladas.

3.1.4.2 Generación del conjunto de imágenes de no-letras

Una vez generadas las muestras de letras, se generó el conjunto de no-letras. Esto es, los ejemplos para la red de las imágenes que no corresponden a letras o dígitos. Para esto, se probaron diferentes opciones, optando finalmente, hasta este punto del proyecto, por la generación de los ejemplos de no-letras a partir de las propias letras creadas anteriormente. Cada no-letra es también un archivo de imagen de 28 x 28 pixeles, en el cual se insertaron de forma aleatoria fragmentos de 14 x 14 pixeles de las imágenes de letras. El Algoritmo 3.11 corresponde a la generación de los ejemplos de no-letras. Al igual que las letras, se generaron un total de 20,600 no-letras, resultando un total de 41,200 muestras para el entrenamiento de la red. La Figura 3.13 muestra algunos ejemplos de no-letras generadas.

Algoritmo 3.11. Generación de no-letras

Entrada: Imágenes de las letras
Salida: Imágenes de no-letras generadas

1. $TAM = 28$
2. $TAM\_BLQ = 14$
3. $BLOQUES = TAM/TAM\_BLQ$
4. *Repetir el número de veces de letras de entrada*
5. *Genera una nueva imagen de tamaño* TAM *x* TAM *con fondo blanco*
6. *Repetir de* $i = 1, j = 1$ *hasta* $BLOQUES$
7. *Asigna a* $LETRA$ *la imagen de una letra de entrada de forma aleatoria*
8. $BLQ\_X =$ *valor aleatorio de* 1 *a* $BLOQUES$
9. $BLQ\_Y =$ *valor aleatorio de* 1 *a* $BLOQUES$
10. *Repetir de* $x = 1, y = 1$ *hasta* TAM_BLOQUE
11. *Si el pixel* $(BLQ\_X * TAM\_BLQ + x,\ BLQ\_Y * TAM\_BLQ + y)$ *de* $LETRA$ *es negro*
12. *Asigna color negro al pixel* $(BLQ\_X * TAM\_BLQ + i,\ BLQ\_Y * TAM\_BLQ + i)$ *de la nueva imagen*
13. *Fin Si*
14. *Fin Repetir*
15. *Fin Repetir*
16. *Guardar la nueva imagen como no-letra*
17. *Fin Repetir*

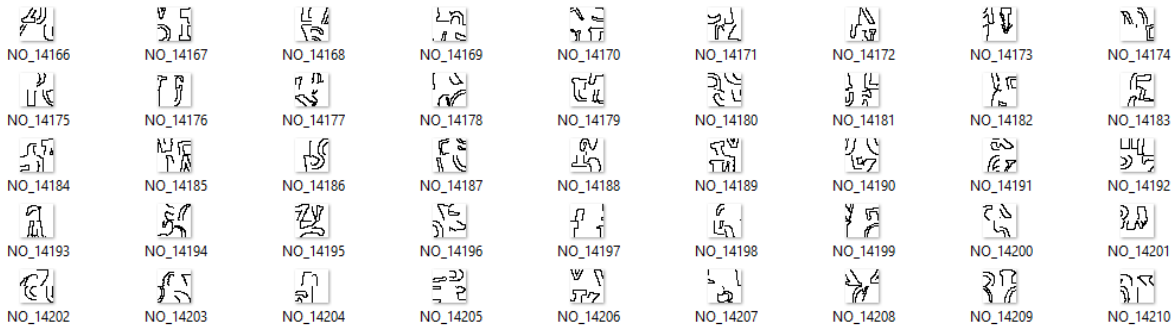

Figura 3.13. Muestras de no-letras generadas.

3.1.4.3 Entrenamiento de la Red Neuronal Convolucional para la clasificación de letras/no-letras

Después de generar los conjuntos de muestras de letras y no-letras, se llevó a cabo el entrenamiento de la red. Para esto, se realizó un paso previo de pre-carga de ambos conjuntos de imágenes, para agilizar el desarrollo de los experimentos en busca de obtener la mejor estructura y parámetros posibles de la red. La pre-carga consiste en crear un archivo de texto con la información de todas las imágenes. Esto permite trabajar con un solo archivo de mayor tamaño, en lugar de decenas de miles de archivos pequeños equivalentes, lo cual resulta bastante más eficiente. El Algoritmo 3.12 corresponde al proceso de pre-carga para las letras. El algoritmo para la pre-carga de las no-letras es similar al de las letras.

Algoritmo 3.12. Pre-carga de las muestras de letras

Entrada: Imágenes de las letras generadas
Salida: Archivo de texto con la información de todas las imágenes de letras

1. *Crea y abre un archivo de texto*
2. *Para cada archivo de imagen de letra*
3. *Abre el archivo de imagen*
4. *Escribe en el archivo de texto el primer carácter del nombre del archivo de la imagen*
5. *Para cada pixel* (x, y) *de la imagen abierta*
6. *Si el color del pixel es negro*
7. *Escribe en el archivo de texto un* 1 *seguido de un tabulador*

8. Caso Contrario
9. Escribe en el archivo de texto un 0 *seguido de un tabulador*
10. Fin Si
11. Fin Para
12. Fin Para
13. Cierra el archivo de texto

Finalmente, el entrenamiento se realizó utilizando una Red Neuronal con una Capa Convolucional de 50 Filtros de 5x5 con función de activación ReLU, una Capa de Pooling de 2x2, una Capa totalmente conectada de 20 Neuronas con activación ReLU y una Capa final de una sola Neurona con función de activación Sigmoide. Se aplicó un factor de aprendizaje de 0.05, un tamaño de mini-lotes de 50 y se realizaron 50 iteraciones.

Con la red entrenada, el siguiente paso es el filtrado de los recuadros candidatos. La idea es analizar con la red cada recuadro generado después de los pasos de fusión y determinar si su contenido es, o no, una letra. Esto tiene como objetivo reducir de forma significativa la cantidad de recuadros que pasarán a la fase de reconocimiento final del texto. Idealmente, lo que se desea es conservar todos los recuadros que contienen de forma completa y compacta una letra y descartar el resto.

3.1.4.4 Proceso de filtrado de recuadros

El proceso de filtrado (Algoritmo 3.13), recorre cada recuadro y genera una imagen con los pixeles que pertenecen a los contornos agrupados por el recuadro. Posteriormente, se escala la imagen a un espacio de 28x28 pixeles para evaluarse por la red neuronal. La red devuelve un valor que corresponde a la probabilidad calculada de que la imagen corresponda a una letra. Los recuadros con probabilidad mayor a 0.50 se conservan y el resto se descartan. La Figura 3.14 muestra ejemplos después de esta etapa de filtrado. Se observa que la cantidad de recuadros disminuye significativamente en comparación con los recuadros generados en los pasos previos (Figura 3.7).

Algoritmo 3.13. Filtrado de recuadros

Entrada: Recuadros generados
Imagen en análisis
Red Neuronal Convolucional entrenada
Salida: Recuadros conservados

1. $TAM = 28$
2. *Para cada recuadro* b *recibido*
3. *Asignar a* $xMin$ *y* $yMin$ *la coordenada de la esquina inferior-izquierda de* b
4. *Asignar a* $TAMx$ *y* $TAMy$ *las dimensiones de* b
5. *Crea una nueva imagen de tamaño* $TAMx$ *x* $TAMy$ *con fondo blanco*
6. *Para cada contorno* c *que perteneces a* b
7. *Para cada pixel* p *que pertenece a* c
8. *Asignar color negro al pixel* $(p.x - xMin, p.y - yMin)$ *de la nueva imagen*
9. *Fin Para*
10. *Fin Para*
11. *Crear una muestra de* TAM *x* TAM *pixeles, inicializados a cero*
12. $pfN = TAM/2$
13. $pfY = TAMy \ / \ 2$
14. $pfX = TAMx \ / \ 2$
15. *Si* $TAMy$ *y* $TAMx$ *son ambos menores que* TAM
16. $s = max(TAMy, TAMx) \ / \ TAM$
17. *Para cada pixel* (x, y) *de la muestra*
18. $yp = pfY + (y - pfN) * s$
19. $xp = pfX + (x - pfN) * s$
20. *Si el color del pixel* (xp, yp) *de la nueva imagen es negro*
21. *Asignar un* 1 *al pixel* (x, y) *de la muestra*
22. *Fin Si*
23. *Fin Para*
24. *Caso Contrario*
25. $s = TAM \ / \ max(TAMy, TAMx)$
26. *Para cada pixel* (x, y) *de la nueva imagen*
27. *Si el color del pixel* (x, y) *de la nueva imagen es negro*
28. $yp = pfN + (y - pfY) * s$
29. $xp = pfN + (x - pfX) * s$
30. *Asignar un* 1 *al pixel* (xp, yp) *de la muestra*
31. *Fin Si*
32. *Fin Para*
33. *Fin Si*
34. *Enviar y clasificar la muestra con la red*
35. *Si la probabilidad es mayor que 0.50 de ser letra*
36. *Conservar* b
37. *Caso Contrario*
38. *Descartar* b
39. *Fin Si*
40. *Fin Para*

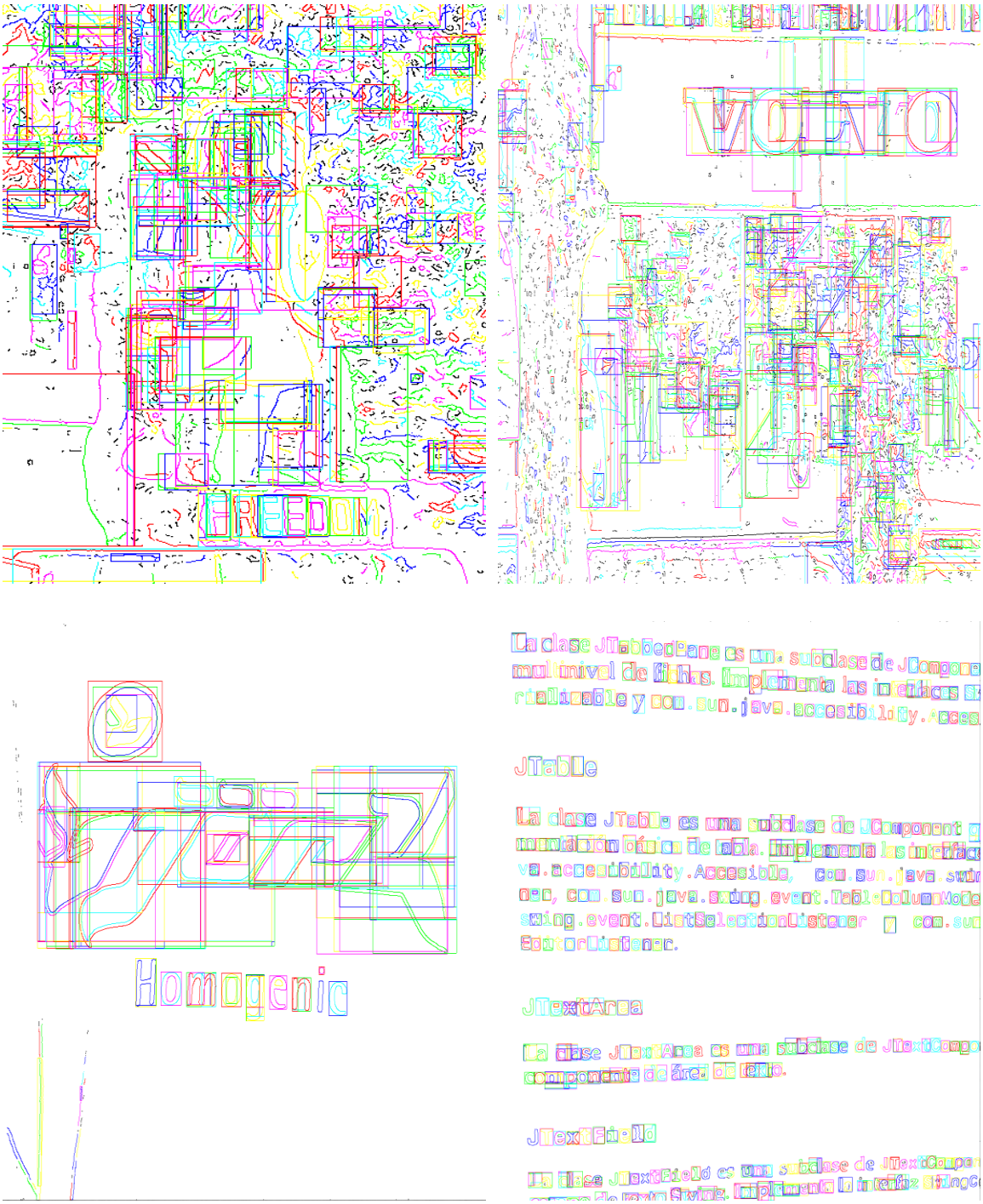

Figura 3.14. Recuadros después de la etapa de filtrado.

3.2 Reconocimiento de texto

Una vez generados los recuadros candidatos a contener texto, la siguiente fase del método es el reconocimiento individual de los caracteres contenidos en los recuadros. Como resultado de este proceso se asigna un valor de probabilidad de la letra a la que corresponde cada carácter. Posteriormente, se descartan los caracteres que se traslapan entre sí en una proporción mayor a un umbral predeterminado. En el caso de dos caracteres que se traslapan, se conserva el que tiene una mayor probabilidad de ser una letra en particular. Finalmente, se agrupan las letras reconocidas y se predicen las palabras que forman realizando una evaluación contra un diccionario determinado. A continuación se explica cada uno de estos procesos.

3.2.1 Reconocimiento de caracteres

Para el reconocimiento individual de los caracteres contenidos en los recuadros, se diseñó y entrenó una Red Neuronal Convolucional utilizando nuevos conjuntos de muestras de imágenes de letras y no-letras. Se requirió preparar estos nuevos conjuntos de muestras, debido a que las pruebas de reconocimiento realizadas con los conjuntos generados previamente y utilizados en la fase de detección, específicamente para el filtrado de recuadros (Sección 3.1.4), no arrojaron los resultados esperados al momento de clasificar la letra específica a la que corresponde cada carácter. Con el diseño y entrenamiento de esta nueva red, no solamente se logró una clasificación aceptable de las letras, sino que además se mejoró el filtrado de los recuadros presentado en la sección anterior.

3.2.1.1 Generación del nuevo conjunto de letras

A continuación se explican las diferencias en la creación del nuevo conjunto de letras con respecto al anterior:

1. Para generar el contorno de las letras, en lugar de utilizar un editor gráfico convencional, se aplicó el algoritmo de *identificación de segmentos de contorno* desarrollado en este mismo proyecto (Algoritmo 3.2).
2. Trabajar con el algoritmo para la generación de contornos, facilitó generar letras de un tamaño mayor, con lo cual se evitó utilizar el *algoritmo de normalización* requerido anteriormente para aumentar el tamaño y refinar las letras.
3. En lugar de las rotación de -10, -5, 5 y 10 grados, se aplicaron rotaciones de -4 a 4 grados con intervalos de un grado.
4. Para incrementar la variación del conjunto de letras, se agregó una nueva transformación que consiste en *estirar* las letras hacia la izquierda, derecha, arriba y abajo, con diferentes factores de *fuerza* (Algoritmo 3.14). La Figura 3.15 muestra un ejemplo del resultado de estirar una letra en las cuatro direcciones.
5. A diferencia de la fase de detección, ahora se generó un solo escalamiento ajustando el tamaño de las letras a un área de 28x28 pixeles (Algoritmo 3.15).
6. Finalmente, se generaron letras con borde de dos pixeles de grosor, tanto en el eje X, en el eje Y, y en ambos ejes de forma simultánea. Para esto, las letras primero se escalaron a espacios de 28x14, 14x28 y 14x14, y después se regresaron a su tamaño de 28x28 (Algoritmo 3.16). Al escalar una letra de 14x14 a 28x28, además de duplicar su tamaño, también se duplica su grosor. La Figura 3.16 muestra algunos ejemplos de letras con borde sencillo y doble.

Algoritmo 3.14. Estiramiento de Letras

Entrada: Imagen del contorno de una letra.
Salida: Imágenes del contorno de la letra estirada en 4 direcciones con 4 factores de fuerza en cada dirección.

1. *Asignar tamaño de la imagen a* $TAMy$ *y* $TAMx$
2. *Asignar* $yMax = 0, xMax = 0, yMin = TAMy - 1, xMin = TAMx - 1$
3. *Para cada pixel (x, y) de la imagen de entrada*
4. *Si pixel (x, y) es color Negro*
5. *Si* $y > yMax$ $yMax = y$ *Fin Si*
6. *Si* $y < yMin$ $yMin = y$ *Fin Si*
7. *Si* $x > xMax$ $xMax = x$ *Fin Si*
8. *Si* $x > xMin$ $xMin = x$ *Fin Si*

9. Fin Si
10. Fin Para
// Estiramiento a la derecha
11. Para $cont = 1$ *hasta* 4
12. Generar una imagen en memoria de tamaño $TAMy, TAMx$ *con fondo blanco*
13. Asignar $sx = cont/40$
14. Para cada pixel (x, y) *de la imagen de entrada*
15. $xp = x - (yMax - y) \cdot sx$
16. Si pixel (xp, y) *es color negro*
17. Asignar color negro al pixel (x, y) *de la nueva imagen*
18. Fin Si
19. Fin Para
20. Guardar en archivo la nueva imagen generada
21. Fin Para
// Estiramiento a la izquierda
22. Para $cont = 1$ *hasta* 4
23. Generar una imagen en memoria de tamaño $TAMy, TAMx$ *con fondo blanco*
24. Asignar $sx = cont/40$
25. Para cada pixel (x, y) *de la imagen de entrada*
26. $xp = x + (yMax - y) \cdot sx$
27. Si pixel (xp, y) *es color negro*
28. Asignar color negro al pixel (x, y) *de la nueva imagen*
29. Fin Si
30. Fin Para
31. Guardar en archivo la nueva imagen generada
32. Fin Para
// Estiramiento hacia abajo
33. Para $cont = 1$ *hasta* 4
34. Generar una imagen en memoria de tamaño $TAMy, TAMx$ *con fondo blanco*
35. Asignar $sy = cont/40$
36. Para cada pixel (x, y) *de la imagen de entrada*
37. $yp = y - (xMax - x) \cdot sy$
38. Si pixel (x, yp) *es color negro*
39. Asignar color negro al pixel (x, y) *de la nueva imagen*
40. Fin Si
41. Fin Para
42. Guardar en archivo la nueva imagen generada
43. Fin Para
// Estiramiento hacia arriba
44. Para $cont = 1$ *hasta* 4
45. Generar una imagen en memoria de tamaño $TAMy, TAMx$ *con fondo blanco*
46. Asignar $sy = cont/40$
47. Para cada pixel (x, y) *de la imagen de entrada*
48. $yp = y + (xMax - x) \cdot sy$
49. Si pixel (x, yp) *es color negro*
50. Asignar color negro al pixel (x, y) *de la nueva imagen*
51. Fin Si
52. Fin Para

53. Guardar en archivo la nueva imagen generada
54. Fin Para

Figura 3.15. Ejemplo de estiramiento a la izquierda, derecha, arriba y abajo de una letra.

Algoritmo 3.15. Escalamiento de letras

Entrada: Imagen de letras rotadas y de letras estiradas
Salida: Imagen de la letra, escalada de forma ajustada a 28x28 pixeles

1. *Asignar a* $TAMy$ *y* $TAMx$ *el tamaño de la imagen de entrada*
2. $TAM = 28$
3. *Crear nueva imagen con fondo blanco de tamaño* TAM *x* TAM
4. $pfN = (TAM - 1)/2$
5. $pfY = (TAMy - 1)/2$
6. $pfX = (TAMx - 1)/2$
7. $sy = TAM/TAMy$
8. $sx = TAM/TAMx$
9. *Para cada pixel* (x, y) *de la imagen de entrada*
10. *Si el pixel* (x, y) *es color negro*
11. $yp = pfN + (y - pfY) * sy$
12. $xp = pfN + (x - pfX) * sx$
13. *Asignar color negro al pixel* (xp, yp) *de la nueva imagen*
14. *Fin Si*
15. *Fin Para*
16. *Guardar la imagen generada*

Algoritmo 3.16. Aumento del grosor del borde de letras

Entrada: Imagen de letra de tamaño 28x28
Salida: Imágenes de la letra, con grosor del borde duplicado

1. $TAM = 28$

2. *Repetir para las combinaciones (14,28), (28,14) y (14,14) de* $nTAMy$ *y* $nTAMx$
3. *Asignar a* $TAMy$ *y* $TAMx$ *el tamaño de la imagen de entrada*
4. $pfY = (TAMy - 1)/2$
5. $pfX = (TAMx - 1)/2$
6. $sy = nTAMy/TAMy$
7. $sx = nTAMx/TAMx$
8. $pfNY = (nTAMy - 1)/2$
9. $pfNX = (nTAMx - 1)/2$
10. *Crear nueva imagen (img_esc) con fondo blanco de tamaño* $nTAMx \times nTAMy$
11. *Para cada pixel* (x, y) *de la imagen de entrada*
12. *Si el pixel* (x, y) *es color negro*
13. $yp = pfNY + (y - pfY) * sy$
14. $xp = pfNX + (x - pfX) * sx$
15. *Asignar color negro al pixel* (xp, yp) *de img_esc*
16. *Fin Si*
17. *Fin Para*
18. *Crear nueva imagen (img_fin) con fondo blanco de tamaño* $TAM \times TAM$
19. $TAMy = nTAMy$
20. $TAMx = nTAMx$
21. $pfN = (TAM - 1)/2$
22. $pfY = (TAMy - 1)/2$
23. $pfX = (TAMx - 1)/2$
24. *Si* $TAMy < TAM$ *y* $TAMx < TAM$
25. $sy = TAMy/TAM$
26. $sx = TAMx/TAM$
27. *Para cada pixel* (x, y) *de img_fin*
28. $yp = pfY + (y - pfN) * sy$
29. $xp = pfX + (x - pfN) * sx$
30. *Si el pixel* (xp, yp) *de img_esc es color negro*
31. *Asignar color negro al pixel* (x, y) *de img_fin*
32. *Fin Si*
33. *Fin Para*
34. *Fin Si*
35. *Si* $TAMy \geq TAM$ *y* $TAMx \geq TAM$
36. $sy = TAM/TAMy$
37. $sx = TAM/TAMx$
38. *Para cada pixel* (x, y) *de img_esc*
39. *Si el pixel* (x, y) *de img_esc es color negro*
40. $yp = pfN + (y - pfY) * sy$
41. $xp = pfN + (x - pfX) * sx$
42. *Asignar color negro al pixel* (xp, yp) *de img_fin*
43. *Fin Si*
44. *Fin Para*
45. *Fin Si*
46. *Si* $TAMy \geq TAM$ *y* $TAMx < TAM$
47. $sy = TAM/TAMy$
48. $sx = TAMx/TAM$
49. *Para* $y = 1$ *hasta* $TAMy$

Para $x = 1$ hasta TAM
  $xp = pfX + (x - pfN) * sx$
  Si el pixel (xp, y) de img_esc es color negro
    $yp = pfN + (y - pfY) * sy$
    Asignar color negro al pixel (x, yp) de img_fin
  Fin Si
 Fin Para
Fin Para
Fin Si
Si $TAMy < TAM$ y $TAMx \geq TAM$
 $sy = TAMy/TAM$
 $sx = TAM/TAMx$
 Para $y = 1$ hasta TAM
  Para $x = 1$ hasta $TAMx$
    $yp = pfY + (y - pfN) * sy$
    Si el pixel (x, yp) de img_esc es color negro
      $xp = pfN + (x - pfX) * sx$
      Asignar color negro al pixel (xp, y) de img_fin
    Fin Si
  Fin Para
 Fin Para
Fin Si
Guardar en archivo img_fin
Fin Repetir

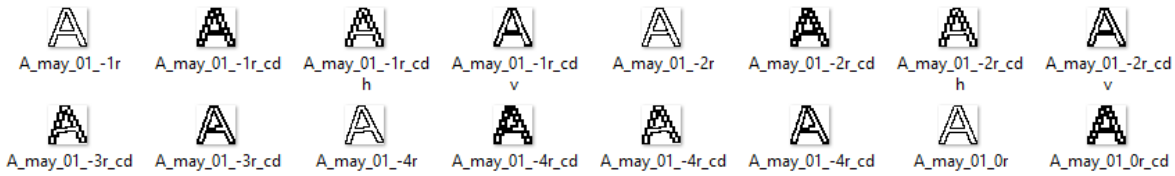

Figura 3.16. Ejemplos de letras generadas con borde sencillo y doble.

En total se generaron 124,604 letras como parte del conjunto de entrenamiento de la nueva Red Convolucional. Además, se generó un nuevo conjunto de no-letras. A diferencia de cómo se crearon las imágenes de no-letras en la fase de detección, en esta etapa se generaron a partir de un conjunto de 254 imágenes arbitrarias que no contienen texto. A estas imágenes se les aplicó el método de generación de contornos y recuadros de éste proyecto, con el agregado de un escalamiento a un área de 28x28 pixeles. En la Figura 3.17 se muestran ejemplos de las no-letras generadas. De esta forma se generaron 60,090

imágenes de no-letras, resultando un total de 184,694 imágenes de muestras para el entrenamiento de la red.

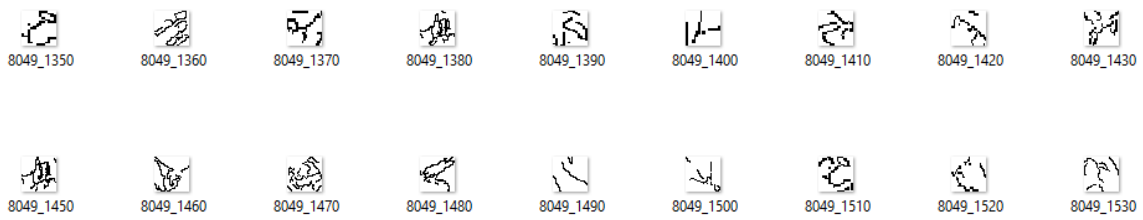

Figura 3.17. Ejemplo de imágenes de no-letras.

3.2.1.2 Diseño de la Red Neuronal Convolucional

Para el reconocimiento de los caracteres contenidos en los recuadros, se diseñaron y probaron diferentes estructuras de redes. Las características deseables que se buscaron en cada prueba son: 1) mejor filtrado de los recuadros que no contienen texto, 2) mayor porcentaje de correcta clasificación de las letras que pasaron el filtro y 3) estructura más sencilla de la red.

La Figura 3.18 muestra la estructura de una de las redes que arrojaron mejores resultados. La red está compuesta por una capa Convolucional de 800 Filtros de 5x5 con función de activación ReLU, una capa de Pooling de 6x6, otra capa Convolucional de 300 Filtros de 4x4 con activación Sigmoide, y una capa final totalmente conectada tipo Softmax de 41 neuronas.

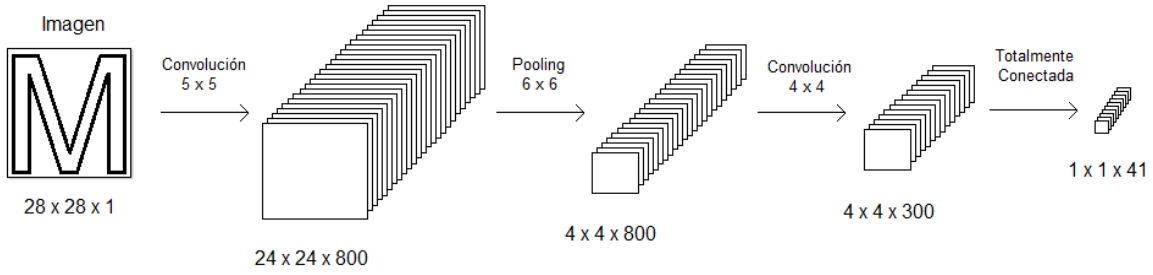

Figura 3.18. Estructura de Red Neuronal propuesta para el reconocimiento de caracteres.

En la capa de salida de la red, las primeras 40 neuronas contienen el valor de probabilidad de que la imagen de entrada corresponda a una letra específica. La neurona 41 contiene la probabilidad de que la imagen sea no-letra. Como se muestra en la Tabla 3, las letras mayúsculas y sus correspondientes minúsculas que son similares se clasificaron como una misma letra, el resto se clasificaron de forma independiente. Las letras *i* y *l*, son un caso particular de letras diferentes que se reconocieron como una misma.

Tabla 3. Letras asociadas a las neuronas de la capa de salida de la red.

Neurona	Letra
1	A
2	a
3	B
4	b
5	C/c
6	D
7	d
8	E
9	e
10	F

Neurona	Letra
11	f
12	G
13	g
14	H
15	h
16	I
17	i/l
18	J/j
19	K/k
20	L

Neurona	Letra
21	M
22	m
23	N
24	n
25	O/o
26	P/p
27	Q
28	q
29	R
30	r

Neurona	Letra
31	S/s
32	T
33	t
34	U/u
35	V/v
36	W/w
37	X/x
38	Y
39	y
40	Z/z
41	No Letra

Con este diseño de red se busca que la primera capa de Convolución, apoyada con la capa de Pooling, identifique las características sobresalientes (de tamaño 5x5 pixeles), de la imagen de entrada. Y en la segunda capa de Convolución se agrupen estas características en un mapa de 4x4, donde cada una de las 16 neuronas de este mapa se active con una

característica particular de una misma letra. Y finalmente, al relacionar todos los mapas de 4x4 con todas las neuronas de la capa de salida, se espera que cada neurona de salida se active con el mapa que, a su vez, se activó con las características de la letra asociada a la neurona de salida. En la Figura 3.19 se muestra un ejemplo hipotético de las características relevantes identificadas de la letra M de la Figura 3.18, que posteriormente se agrupan en un mapa de 4x4.

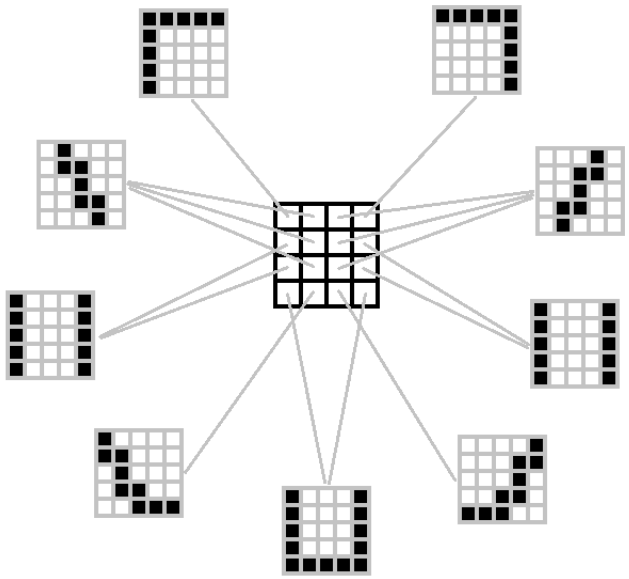

Figura 3.19. Características de 5x5 pixeles de la letra M agrupadas en un mapa de 4x4.

3.2.1.3 Entrenamiento y pruebas preliminares de la red

Una vez generados los conjuntos de imágenes de letras y no-letras, y diseñada la red, se llevó a cabo el entrenamiento de la misma, aplicando un factor de aprendizaje de 0.50, un tamaño de mini-lotes de 50 y se ejecutaron 25 épocas. Con la red entrenada se realizaron pruebas básicas con un conjunto pequeño de imágenes. Las pruebas consistieron en recibir el conjunto de recuadros generados con el Algoritmo 3.3, escalar las imágenes contenidas en los recuadros a un tamaño de 28x28 pixeles y enviarlas a la red con el fin de reconocer y clasificar las posibles letras presentes (Algoritmo 3.17).

Algoritmo 3.17. Reconocimiento de caracteres

Entrada: Recuadros generados
Red Neuronal Convolucional entrenada
Salida: Recuadros conservados
Letras clasificadas (etiquetadas)

1. $TAM = 28$
2. *Para cada recuadro* b *recibido*
3. *Asignar a* $xMin$ *y* $yMin$ *la coordenada de la esquina inferior-izquierda de* b
4. *Asignar a* $TAMx$ *y* $TAMy$ *las dimensiones de* b
5. *Crea una nueva imagen de tamaño* $TAMx$ *x* $TAMy$ *con fondo blanco*
6. *Para cada contorno* c *que perteneces a* b
7. *Para cada pixel* p *que pertenece a* c
8. *Asignar color negro al pixel* $(p.x - xMin, p.y - yMin)$ *de la nueva imagen*
9. *Fin Para*
10. *Fin Para*
11. *Crear una Muestra de* TAM *x* TAM *pixeles, inicializados a cero*
12. $pfN = (TAM - 1)/2$
13. $pfY = (TAMy - 1) / 2$
14. $pfX = (TAMx - 1) / 2$
15. *Si* $TAMy < TAM$ *y* $TAMx < TAM$
16. $sy = TAMy / TAM$
17. $sx = TAMx / TAM$
18. *Para cada pixel* (x, y) *de la muestra*
19. $yp = pfY + (y - pfN) * sy$
20. $xp = pfX + (x - pfN) * sx$
21. *Si el color del pixel* (xp, yp) *de la nueva imagen es negro*
22. *Asignar un* 1 *al pixel* (x, y) *de l a muestra*
23. *Fin Si*
24. *Fin Para*
25. *Fin Si*
26. *Si* $TAMy \geq TAM$ *y* $TAMx \geq TAM$
27. $sy = TAM / TAMy$
28. $sx = TAM / TAMx$
29. *Para cada pixel* (x, y) *de la nueva imagen*
30. *Si el color del pixel* (x, y) *de la nueva imagen es negro*
31. $yp = pfN + (y - pfY) * sy$
32. $xp = pfN + (x - pfX) * sx$
33. *Asignar un* 1 *al pixel* (xp, yp) *de la muestra*
34. *Fin Si*
35. *Fin Para*
36. *Fin Si*
37. *Si* $TAMy \geq TAM$ *y* $TAMx < TAM$
38. $sy = TAM/TAMy$
39. $sx = TAMx/TAM$
40. *Para* $y = 1$ *hasta* $TAMy$
41. *Para* $x = 1$ *hasta* TAM

42. $xp = pfX + (x - pfN) * sx$
43. *Si el pixel* (xp, y) *de la nueva imagen es color negro*
44. $yp = pfN + (y - pfY) * sy$
45. *Asignar un* 1 *al pixel* (x, yp) *de la muestra*
46. *Fin Si*
47. *Fin Para*
48. *Fin Para*
49. *Fin Si*
50. *Si* $TAMy < TAM$ *y* $TAMx \geq TAM$
51. $sy = TAMy/TAM$
52. $sx = TAM/TAMx$
53. *Para* $y = 1$ *hasta* TAM
54. *Para* $x = 1$ *hasta* $TAMx$
55. $yp = pfY + (y - pfN) * sy$
56. *Si el pixel* (x, yp) *de la nueva imagen es color negro*
57. $xp = pfN + (x - pfX) * sx$
58. *Asignar un* 1 *al pixel* (xp, y) *de la muestra*
59. *Fin Si*
60. *Fin Para*
61. *Fin Para*
62. *Fin Si*
63. *Enviar y clasificar la muestra con la red*
64. *Si el valor de probabilidad guardado en la neurona 41 es menor que 0.99*
65. *Descartar b*
66. *Caso Contrario*
67. *Identifica la neurona de la 1 a la 40 con valor máximo*
68. *Asigna la etiqueta de la letra de acuerdo a la* Tabla 3
69. *Fin Si*
70. *Fin Para*

Consideramos que los resultados de las pruebas preliminares fueron aceptables, ya que la red fue capaz de reconocer y etiquetar correctamente más del 80% de letras contenidas en las imágenes utilizadas. En la Figura 3.20 se muestra un ejemplo de caracteres reconocidos, arriba a la izquierda se muestra la imagen original, arriba a la derecha la imagen con los contornos y recuadros generados, y en la parte inferior se muestran los caracteres reconocidos y etiquetados. La última letra del nombre asignado a cada carácter corresponde a la letra con la que éste se etiquetó. Encerradas en un círculo seleccionamos las letras con las cuales se pueden reconstruir las palabras originales (o partes de éstas). En este caso se pueden reconstruir las palabras: “PROFESSIONAL”, “Java”, “SOAP”, “Heny” y “Beque”. También se observa que además de letras correctas, se reconocen letras

parciales, pares de letras y algunos gráficos que no son parte del texto. En las siguientes etapas del proyecto se intenta reducir la cantidad de falsos positivos, así como reconstruir las palabras completas.

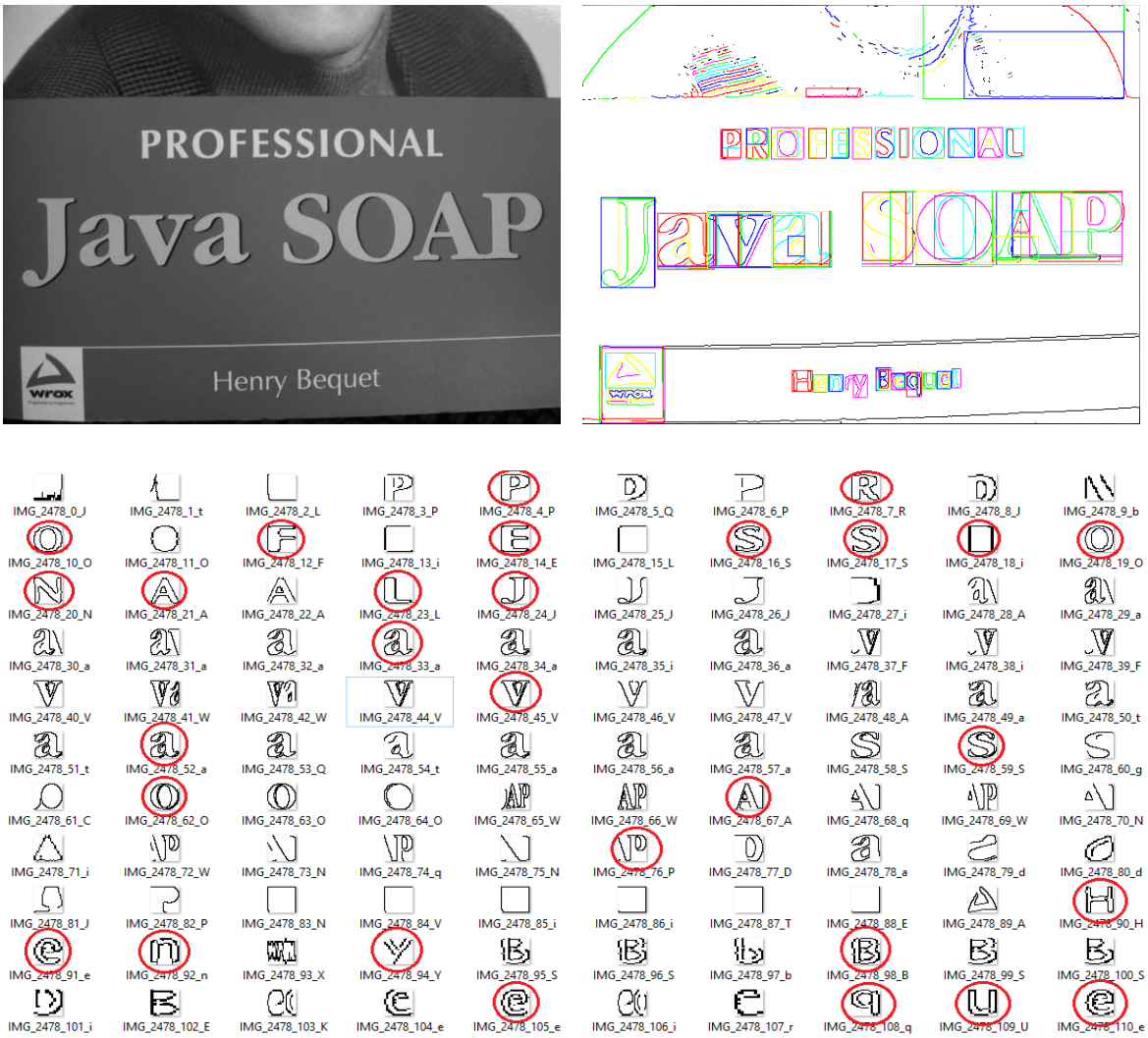

Figura 3.20. Ejemplo de caracteres reconocidos en una imagen de prueba.

Además del reconocimiento de caracteres por la nueva red, también se redujo significativamente la cantidad de recuadros que no contienen texto. En la Figura 3.21 a la izquierda se muestran los recuadros que resultan después de aplicar el filtro de la fase de detección (Algoritmo 3.13), y a la derecha los recuadros que resultan con el Algoritmo

3.17 de esta sección. Se observa que la cantidad de recuadros que no contienen texto disminuye de forma significativa, e incluso se generan nuevos recuadros en zonas de texto.

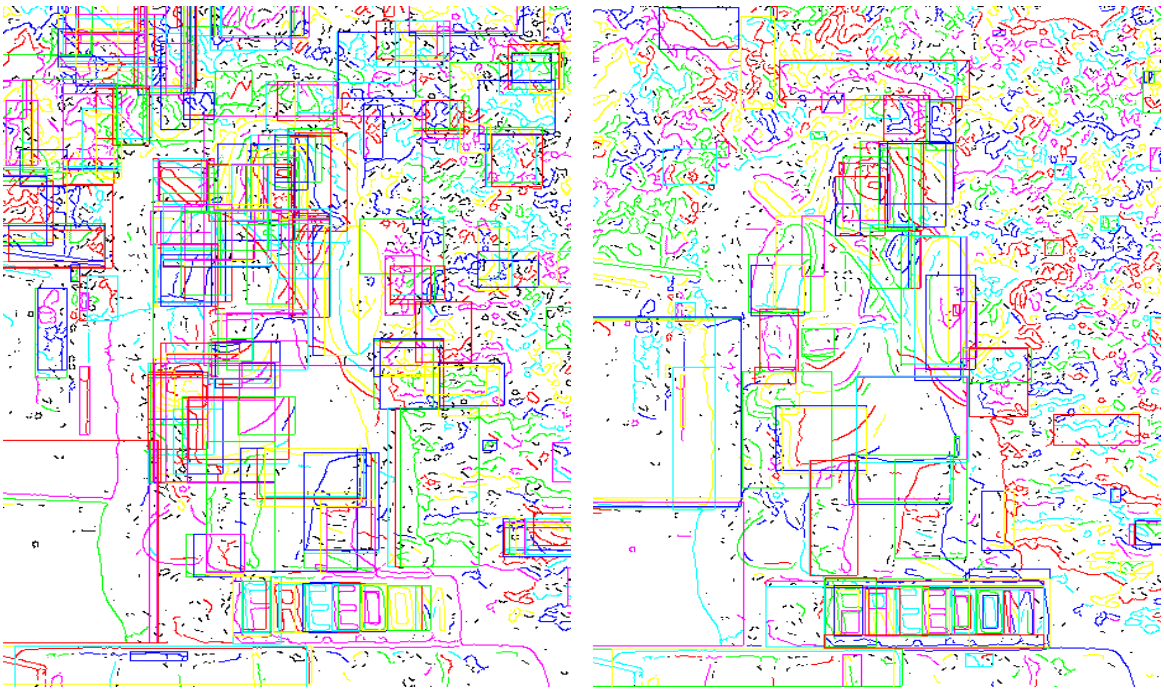

Figura 3.21. Ejemplos de recuadros generados, Izq. Algoritmo 3.13, der. Algoritmo 3.17.

3.2.2 Filtrado de caracteres que se traslapan

Una vez reconocidos los posibles caracteres, el siguiente paso es filtrar aquellos cuyos recuadros que los contienen se traslapan más de un 30% (Algoritmo 3.18). El objetivo es descartar los caracteres que se detectan más de una vez, así como aquellos que se forman como sub-letras dentro de otros. La Figura 3.22 muestra un ejemplo de una imagen antes de la eliminación de los recuadros traslapados (fila superior) y después de la ejecución de este proceso (fila inferior). Como resultado de éste último paso, se reduce aún más el

número de recuadros generados, quedando únicamente aquellos que serán considerados en la etapa de identificación de líneas de texto.

Algoritmo 3.18. Filtrado de caracteres que se traslapan

Entrada: Recuadros con caracteres reconocidos por la Red Convolucional
Salida: Recuadros conservados sin traslape

Para cada recuadro b recibido
  Para cada recuadro b2 ≠ b
    Si ambos recuadros tienen estatus "activo"
      Si el recuadro b está totalmente contenido en b2
        Si b corresponde a una posible sub-letra de b2 (p.e. "o" es sub-letra de "p")
          asignar estatus "desactivado" a b
        Fin Si
      Fin Si
      Si b y b2 se traslapan en más del 30%
        Si b tiene mayor probabilidad que b2 de corresponder a una letra
          asignar estatus "desactivado" a b2
        Caso Contrario
          asignar estatus "desactivado" a b
        Fin Si
      Fin Si
    Fin Si
  Fin Para
Fin Para

Figura 3.22. Ejemplo de una imagen antes y después del filtrado de recuadros traslapados.

3.2.3 Identificación de líneas de texto

El siguiente paso del método es la identificación de las posibles líneas de texto. Para esto, se agrupan los caracteres que se encuentran contiguos y que presentan un tamaño y posición vertical que se considere *compatible*. Para cada carácter c se identifica el carácter más cercano a su izquierda y el más cercano a su derecha, que tengan un tamaño vertical no mayor a dos veces el tamaño de c, y no menor a la mitad del tamaño de c. Además, debe de cumplirse que el centro de estos caracteres se encuentre dentro de los límites verticales de c, o viceversa (Algoritmo 3.19). Con los caracteres cercanos identificados se forman las líneas de texto. Se inicia con un carácter de forma arbitraria y se recorre a un siguiente mientras exista un carácter cercano a la izquierda. Este proceso se repite hacia la derecha (Algoritmo 3.20). En la Figura 3.23 se presenta un ejemplo del resultado de este proceso. De forma ilustrativa se muestra en color rojo las líneas de texto identificadas.

Algoritmo 3.19. Identificación de caracteres cercanos

Entrada: Recuadros con caracteres reconocidos sin traslape
Salida: Recuadros con identificación de sus recuadros cercanos a la izquierda y derecha

Para cada recuadro b
  distMinIzq=-1
  bCercanoIzq=-1
  $bAlto = b.yMax - b.yMin$
  Para cada recuadro $b2 \neq b$
    Si $b.yCentro >= b2.yMin, b.yCentro <= b2.yMax$ *o*
      $b2.yCentro >= b.yMin, b2.yCentro <= b.yMax$
      $distIzq = b.xCentro - b2.xCentro$
      $b2Alto = b2.yMax - b2.yMin$
      Si $distIzq > 0$ *y* $max(bAlto/b2Alto, b2Alto/bAlto) < 3$ *y* $distIzq < distMinIzq$
        $distMinIzq = distIzq$
        $bCercanoIzq = b2$
      Fin Si
    Fin Si
  Fin Para
  Si $bCercanoIzq \neq -1$
    $b.izq = bCercanoIzq$
  Fin Si
  //Se repite el proceso para la derecha
Fin Para

Algoritmo 3.20. Identificación de líneas de texto

Entrada: Recuadros con identificación de sus cercanos izquierda y derecha
Salida: Lista de líneas de recuadros

1. *Crea lista de líneas vacía:* $líneas$
2. $líneaActual = -1$
3. *Para cada recuadro* b
4. *Si no se ha asignado número de línea a* b
5. $aux\_b = b$
6. $líneaActual = líneaActual + 1$
7. $b.línea = líneaActual$
8. *Crea lista de boxes vacía:* $línea$
9. *agrega* b *a* $línea$
10. *agrega* $línea$ *a* $líneas$
11. *Mientras* $b.izq \neq null$ *y* $box.izq.der = b$
12. $b.izq.linea = b.linea$
13. *agrega* $b.izq$ *al inicio de* $linea$
14. $b = b.izq$
15. *Fin mientras*
16. *b=aux_b*
17. *Mientras* $b.der \neq null$ *y* $box.der.izq = b$
18. $b.der.linea = b.linea$
19. *agrega* $b.der$ *al final de* $linea$
20. $b = b.der$
21. *Fin mientras*
22. *Fin Para*

Figura 3.23. Ejemplo de la identificación de líneas de texto.

3.2.4 Separación de líneas de texto

Las líneas de texto se separan en las palabras que las componen. Este proceso se realiza de forma iterativa, utilizando la media de la separación entre cada par de recuadro contiguo. Se dividen en sub-líneas los recuadros cuya separación supera a la media en una proporción establecida previamente y que además es mayor que una tercera parte de la altura promedio los recuadros de la línea. Finalmente, se filtran las sub-líneas que tienen menos de tres recuadros (Algoritmo 3.21). La Figura 3.23 muestra un ejemplo de separación de líneas. De forma ilustrativa se muestra las separaciones con líneas de color. Se observa que la primera línea de texto no se logró dividir en las palabras que la componen, debido a que estas no presentan un espaciado suficiente. La separación de estas palabras se logra posteriormente durante la fase de reconocimiento de palabras del diccionario. Se observa también que la tercera línea de texto, a pesar de que el espacio entre las palabras también es angosto, si logran separarse.

Algoritmo 3.21. Separación de líneas de texto

Entrada: Lista de líneas de recuadros
Salida: Sub-listas de líneas de recuadros

1. $sw\_cambio = verdadero$
2. *Mientras* sw_cambio *sea* $verdadero$
3. $sw\_cambio = falso$
4. *Crea lista de líneas vacía:* $líneasNuevas$
5. *Para cada línea de recuadros recibida*
6. *obtiene la media de la separación de sus recuadros*
7. *obtiene la altura promedio de sus recuadros*
8. *Crea lista de recuadros vacía* $líneaNueva$ *y agrega el primer recuadro* b *de la línea*
9. *Para cada siguiente recuadro* $b2$ *de la línea*
10. *obtiene la separación con el recuadro anterior*
11. *Si* $separación > altura\ promedio/3\ y\ separación > 1.7 * media$
12. *agrega* $líneaNueva$ *a* $líneasNuevas$
13. *inicializa lista de recuadros* $líneaNueva$
14. $sw\_cambio = verdadero$
15. *Fin si*
16. *agrega* $b2$ *a* $líneaNueva$

17. Fin Para
18. agrega $líneaNueva$ a $líneasNuevas$
19. Fin Mientras
20. Filtra líneas con menos de 3 recuadros
21. Devuelve $líneasNuevas$

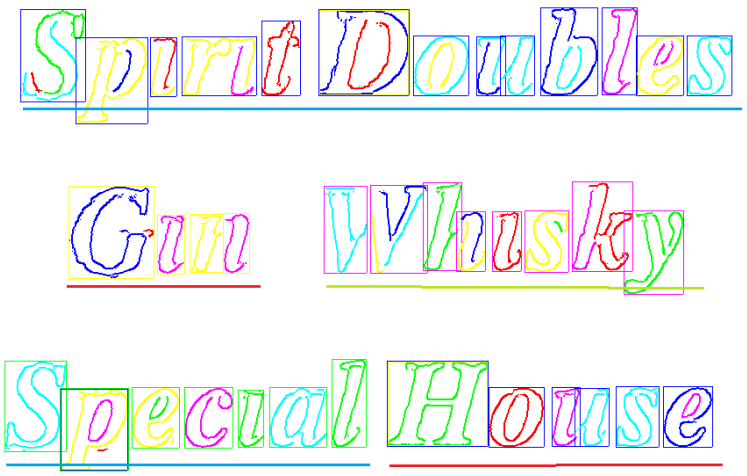

Figura 3.24. Ejemplo de separación en sub-líneas de texto.

3.2.5 Identificación de caracteres unidos

El siguiente paso es identificar los recuadros que contienen dos o más caracteres unidos, y que por lo tanto no fue posible reconocerlos correctamente por la Red Neuronal. Estos son casos que ocurren debido a que en algunas imágenes los caracteres no tienen separación entre ellos. Es decir, no existe ni un pixel de separación entre ellos, por lo que el algoritmo de identificación de contornos los considera como un solo objeto. La Figura 3.25 muestra ejemplos de recuadros con caracteres unidos. La estrategia para este paso se basa en identificar los recuadros que tienen un ancho significativamente mayor que su altura o mayor que la media del ancho de los recuadros de la línea a la que pertenece (Algoritmo 3.22). Los parámetros para considerar dicha significancia se obtuvieron a partir de observaciones y pruebas con diferentes casos de caracteres unidos.

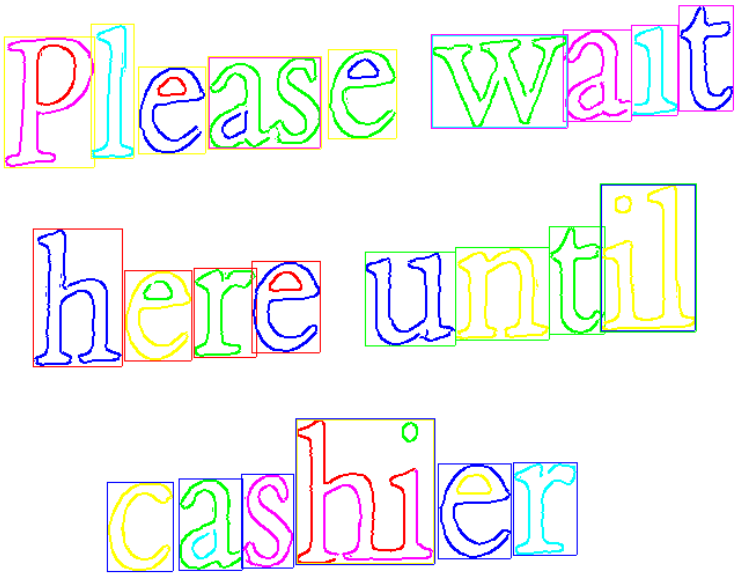

Figura 3.25. Ejemplo de recuadros con caracteres unidos.

Algoritmo 3.22. Identificación de recuadros con caracteres unidos

Entrada: Sub-líneas de recuadros
Salida: Lista de recuadros con posibles caracteres unidos

1. *Para cada sub-lista de recuadros*
2. $cont = 0$
3. $suma = 0$
4. *Para cada recuadro de la sub-lista*
5. $prop = tamX/tamY$
6. *Si* $prop > 0.5$ *y* $prop < 1.3$
7. $suma += tamX$
8. $cont++$
9. *Fin si*
10. *Fin Para*
11. $media = suma/cont$
12. *Crea lista de recuadros vacía:* $listaAux$
13. *Para cada recuadro de la sub-lista*
14. $prop = tamX/tamY$
15. *Si* $prop > 1.4$ *o* $tamX/media > 1.2$
16. *agrega una copia del recuadro a* $listaAux$
17. *Fin si*
18. *Fin Para*
19. *Fin Para*
20. *Devuelve* $listaAux$

3.2.6 Reconocimiento individual de caracteres unidos

Con los recuadros que contienen caracteres unidos identificados, el siguiente paso es reconocer los caracteres de forma individual, a través de la Red Neuronal. Para esto, se generan y deslizan múltiples ventanas de diferente ancho sobre el área del recuadro, recortando y enviando a reconocer a la red los contornos contenidos en cada ventana (Algoritmo 3.23). La Figura 3.26 muestra ejemplos de ventanas de diferentes tamaños que se deslizan sobre un recuadro que contiene a los caracteres *h* e *i* unidos. La ventana de la izquierda es probable que reconozca la letra *i*, la ventana del medio posiblemente reconozca una *u* y la ventana de la derecha la letra *h*.

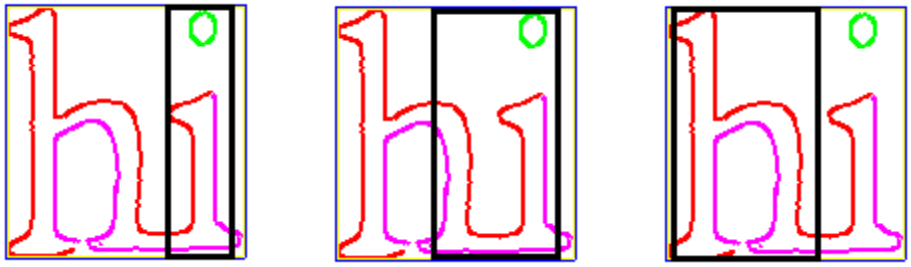

Figura 3.26. Ejemplos de ventanas deslizantes para reconocimiento de caracteres individuales.

Algoritmo 3.23. Reconocimiento individual de caracteres unidos

Entrada: Lista de recuadros con posibles caracteres unidos
Salida: Lista de recuadros con los caracteres individuales reconocidos

1. *Para cada recuadro b de la lista recibida*
2. $incX = b.tamY = 0.1$
3. $iniX = b.tamY * 0.2$
4. $finX = b.tamY * 1.5$
5. *Repite para* $tX = iniX$ *hasta* $finX$ *con incremento* $incX$
6. $posX = b.xMin$
7. $tamX = tX$

Repite mientras $posX + tamX < b.xMax$
  $xMin = -1, yMin = -1, xMax = -1, yMax = -1$
  Para cada contorno c *del recuadro* b
    Para cada pixel p *del contorno* c
      Si $xMin = -1$ *o* $p.x < xMin$ *entonces* $xMin = p.x$
      Si $yMin = -1$ *o* $p.y < yMin$ *entonces* $yMin = p.y$
      Si $xMax = -1$ *o* $p.x < xMax$ *entonces* $xMax = p.x$
      Si $yMax = -1$ *o* $p.y < yMax$ *entonces* $yMax = p.y$
    Fin Para
  Fin Para
  $TAMx = xMax - xMin + 1$
  $TAMy = yMax - yMin + 1$
  Crea nueva imagen de tamaño $TAMx, TAMy$
  Para cada contorno c *del recuadro* b
    Para cada pixel p *del contorno* c
      $xs = p.x - xMin$
      $ys = p.y - yMin$
      agrega un pixel en xs, ys *de la nueva imagen*
    Fin Para
  Fin Para
  Envía la nueva imagen a reconocer por la Red Convolucional
  Si la imagen tiene probabilidad de ser una letra
    agrega el recuadro a la lista de reconocidos
  Fin Si
 Fin Repite
Fin Repite
Fin Para
Devuelve lista de recuadros con caracteres reconocidos

3.2.7 Generación de combinaciones de palabras candidatas

Una vez reconocidas las letras independientes, así como reconocido y separado las letras unidas, se generan de forma recursiva todas las combinaciones posibles de palabras candidatas que se forman con las letras identificadas (Algoritmo 3.24). En la Figura 3.27 se muestra un ejemplo donde se reconocen los caracteres independientes *c, a, s, e* y *r*; y a partir de las letras unidas *hi* se reconocen las letras *h, u* e *i* (como se explicó en la sección anterior). Debido a que la letra *u* se traslapa con las letras *h* e *i*, las combinaciones de palabras que se forman son: *cashier* y *casuer*.

Algoritmo 3.24. Generación de palabras candidatas

Entrada: Lista de recuadros que pertenecen a una línea de texto
Posición pos de la primera letra de la línea
Salida: Lista de listas de combinaciones de recuadros de palabras candidatas

1. *Si* $pos = tamaño$ *de la lista recibida*
2. *crea lista de listas de recuadros:* $combLíneas$
3. *crea lista de recuadros:* $nuevaLínea$
4. *agrega el recuadro de la posición* pos *de la lista a* $nuevaLínea$
5. *agrega* $nuevaLínea$ *a* $combLíneas$
6. *devuelve* $combLíneas$
7. *Caso contrario*
8. *recibe* $combLíneas$ *de llamada recursiva al método con* $pos + 1$
9. *crea* $box = recuadro\ de\ la\ posición\ pos\ de\ la\ línea\ recibida$
10. *Para* $l = 0$ *hasta tamaño de* $combLíneas$
11. *Crea lista de recuadros:* $líneaComb = lista\ l\ de\ combLíneas$
12. *crea* $box2 = primer\ recuadro\ de\ líneaComb$
13. *Si recuadro* box *no se traslapa con* $box2$
14. *agrega recuadro* box *al inicio de lista* $lïneaComb$
15. *Caso contrario*
16. *crea lista de recuadros vacía:* $nuevaLínea$
17. *agrega* box *a* $nuevaLínea$
18. $posBox = 1$
19. *Mientras* $posBox$ *sea menor que tamaño de* $líneaComb$
20. $box2$*=recuadro en la posición* $posBox$ *de* $líneaComb$
21. *Si* box *no se traslapa con* $box2$
22. *aborta Mientras*
23. *Fin Si*
24. $posBox = posBox + 1$
25. *Fin Mientras*
26. *Para* $p = posBox$ *hasta tamaño de* $líneaComb$
27. *agrega recuadro en la posición* p *de* $líneaComb$ *a* $nuevaLínea$
28. *Fin Para*
29. *Si* $nuevaLínea$ *no existe en* $combLíneas$
30. *agrega* $nuevaLínea$ *a* $combLíneas$
31. *Fin Si*
32. *Fin Si*
33. *Fin Para*
34. *Fin Si*
35. *devuelve* $combLíneas$

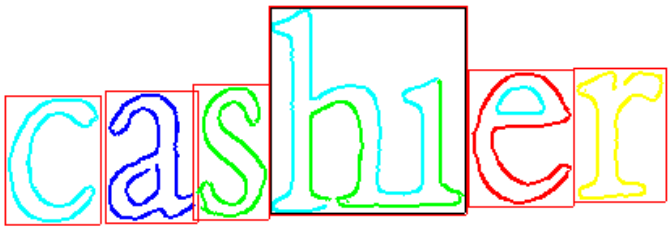

Figura 3.27. Ejemplo de palabra con caracteres independientes y unidos.

3.2.8 Reconocimiento de palabras

Finalmente, se analiza cada palabra candidata para evaluar su similitud con las palabras de un diccionario preestablecido. Para esto, se implementó y aplicó el método Smith-Waterman, el cual es utilizado en bioinformática para el alineamiento de cadenas de ADN y de proteínas. Este método encuentra el mejor emparejamiento entre los caracteres de dos cadenas, buscando todas las combinaciones posibles de corrimientos y espaciados de los caracteres, a través de una técnica de programación dinámica (Algoritmo 3.25). En este trabajo, al emparejamiento del mismo carácter en ambas cadenas se le asignó un puntaje positivo (de 0.7 a 1), a caracteres que no coinciden, pero que son similares gráficamente un puntaje también positivo pero más bajo (0.3 o 0.5) y a caracteres muy diferentes o al emparejamiento de un carácter con un espacio en blanco un puntaje de cero. En la Figura 3.28 se muestra un ejemplo de dos alineamientos para las palabras “Tecnológico” y “Teenlómico”, con sus correspondientes puntajes. El primero es el alineamiento que resulta de la comparación directa sin corrimientos de caracteres y el segundo corresponde al mejor alineamiento posible entre estas palabras. En la Figura 3.29 se muestra la matriz de puntajes entre caracteres utilizada en este proyecto. El valor de cada celda corresponde al puntaje asignado a la coincidencia dentro del alineamiento del carácter de la fila con el carácter de la columna de la matriz. La palabra que se reporta como identificada de una imagen, es aquella que logra el mejor puntaje con alguna palabra en particular del diccionario y que además rebase un puntaje mínimo establecido previamente.

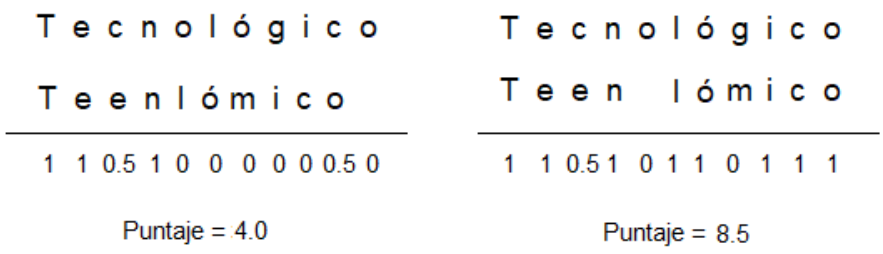

Figura 3.28. Ejemplo de alineamientos entre dos cadenas de caracteres.

	A	a	B	b	Cc	D	d	E	e	F	f	G	g	H	h	I	il	Jj	Kk	L	M	m	N	n	Oo	Pp	Q	q	R	r	Ss	T	t	Uu	Vv	Ww	Xx	Y	y	Zz
Aa	10	10	0	0	0	0	5	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	3	3	0	0	5	5	0	0	0	0	0	0	0	3	0	0	0
Bb	0	0	10	10	0	0	0	5	0	0	0	0	0	3	3	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	3	0	0	0	0	0	0	0	0
Cc	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Dd	0	5	0	0	0	10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ee	0	0	5	0	0	0	0	10	10	5	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ff	0	0	0	0	0	0	0	0	0	10	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	5	0	0	0	0	0	0	0	0
Gg	0	3	0	0	0	0	0	0	0	0	0	10	10	0	0	0	0	0	0	0	0	0	0	0	3	0	0	5	0	0	0	0	0	0	0	0	0	3	0	0	0
Hh	0	0	0	5	0	0	0	0	0	0	0	0	0	10	10	0	0	0	3	0	0	0	5	5	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
Ii	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Jj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Kk	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	3	0	0	0	0
Ll	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	7	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
Mm	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	8	5	5	0	0	0	0	0	0	0	0	0	5	0	3	3	0			
Nn	0	0	0	0	0	0	0	0	0	0	0	0	0	3	5	0	0	0	0	0	3	0	10	8	0	0	0	0	0	3	0	0	0	0	5	0	0	0	0	0	
Oo	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	
Pp	0	0	0	0	0	0	0	0	0	5	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	3	0	0	0	0	0	0	0	0	
Qq	0	5	0	0	0	0	0	0	0	0	0	3	5	0	0	0	0	0	0	0	0	0	0	0	3	0	10	10	0	0	0	0	0	0	0	0	0	0	0	0	
Rr	3	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	5	0	0	0	0	0	0	3	0	0	10	8	0	0	5	0	0	0	0	0	3	0	
Ss	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	
Tt	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	8	0	0	0	0	0	3	
Uu	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	
Vv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	10	0	0	0	0	0	
Ww	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	3	3	0	0	0	0	0	0	0	0	0	0	5	8	0	3	3	0		
Xx	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	10	3	3	0		
Yy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	3	5	0	0	10	10	0		
Zz	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	10	

Figura 3.29. Matriz de puntaje propuesta para el emparejamiento entre caracteres.

Algoritmo 3.25. Alineamiento de cadenas de caracteres (Smith-Waterman)

Entrada: Cadenas $cad1$ y $cad2$
Matriz de puntaje entre caracteres: $puntaje\_car$
Salida: Mejor alineamiento entre las cadenas y su puntaje correspondiente

1. *crea matriz* $puntaje$ *de longitud* $cad1 + 1$, *longitud de* $cad2 + 1$
2. *crea matriz* $celdasPrevias$ *de longitud* $cad1 + 1$, *longitud de* $cad2 + 1$
3. *inicializa en cero ambas matrices*
4. *define* $PUNTAJE\_ESPACIO = -0.75$
5. *define* $IZQ = 1 \quad // 0001$
6. *define* $ARR = 2 \quad // 0010$
7. *define* $DIAG = 4 \quad // 0100$
8. *define* $CERO = 8 \quad // 1000$
9. *Para* $y = 1$ *hasta longitud de* $cad1$
10. *Para* $x = 1$ *hasta longitud de* $cad2$
11. $car1 = carácter\ en\ la\ posición\ y\ de\ cad1$
12. $car2 = carácter\ en\ la\ posición\ x\ de\ cad2$
13. $puntajeDiagonal = puntaje\_car[posición\ de\ car1][posición\ de\ car2]$
14. $puntajeArriba = puntaje[y-1][x] + PUNTAJE\_ESPACIO$
15. $puntajeIzquierda = puntaje[y][x-1] + PUNTAJE\_ESPACIO$
16. $puntaje[y][x] = max(puntajeDiagonal, max(puntajeArriba, puntajeIzquierda))$
17. *Si* $puntaje[y][x] = puntajeDiagonal$
18. $celdasPrevias[y][x] = DIAG$
19. *Fin Si*
20. *Si* $puntaje[y][x] = puntajeArriba$
21. $celdasPrevias[y][x] = ARR$
22. *Fin Si*
23. *Si* $puntaje[y][x] = puntajeIzquierda$
24. $celdasPrevias[y][x] = IZQ$
25. *Fin Si*
26. *Si* $puntaje[y][x] = 0$
27. $celdasPrevias[y][x] = CERO$
28. *Fin Si*
29. *Fin Para*
30. *Fin Para*
31. $puntajeMáximo = 0$
32. *Para* $y = 1$ *hasta longitud* $cad1$
33. *Para* $x = 1$ *hasta longitud* $cad2$
34. *Si* $puntaje[y][x] > puntajeMáximo$
35. $puntajeMáximo = puntaje[y][x]$
36. $yFin = y$
37. $xFin = x$
38. *Fin Si*
39. *Fin Para*
40. *Fin Para*
41. $y = yFin$
42. $x = xFin$

43. *Si* $y > 0$ *y* $x > 0$
44. *Repite*
45. *Si* $celdasPrevias[y][x] = ARR$
46. *Si* $puntaje[y-1][x] > 0$
47. $y = y - 1$
48. *Caso contrario*
49. *aborta Repite*
50. *Fin Si*
51. *Fin Si*
52. *Si* $celdasPrevias[y][x] = IZQ$
53. *Si* $puntaje[y][x-1] > 0$
54. $x = x - 1$
55. *Caso contrario*
56. *aborta Repite*
57. *Fin Si*
58. *Fin Si*
59. *Si* $celdasPrevias[y][x] = DIAG$
60. *Si* $puntaje[y-1][x-1] > 0$
61. $y = y - 1$
62. $x = x - 1$
63. *Caso contrario*
64. *aborta Repite*
65. *Fin Si*
66. *Fin Si*
67. *Fin Repite*
68. *Fin Si*
69. $yIni = y$
70. $xIni = x$
71. *devuelve* $puntajeMáximo, yIni, yFin, xIni, xFin$

4 Experimentos y Resultados

4.1 Conjuntos de datos

Los conjuntos de imágenes utilizadas para las prueba finales son *ICDAR 2003* [22], en su versión completa y una versión simplificada de 50 palabras que forman el diccionario por imagen evaluada, y la *Street View Text dataset 2011 (SVT)* [23]. Estos conjuntos de datos han sido utilizados en diferentes trabajos [23, 24, 8, 1, 13] como parte de los experimentos de reconocimiento de texto en escenas naturales. La *ICDAR 2003* contiene un conjunto de 181 imágenes de entrenamiento y 251 imágenes para pruebas finales, con 860 palabras contenidas dentro del conjunto completo. Siguiendo el protocolo de evaluación propuesto por [23], en este proyecto utilizamos palabras con al menos tres caracteres alfanuméricos. Al conjunto completo de estas imágenes y considerando el total de las 860 palabras para formar el diccionario, se le refiere como *IC03-Full*. Esto es, al analizar una imagen se debe de reconocer y elegir dentro de este conjunto de 860 palabras, el texto real contenido en la imagen. Al mismo conjunto de imágenes pero utilizando un diccionario de únicamente 50 palabras por cada imagen evaluada, se le identifica como *IC03-50*.

El conjunto de imágenes *SVT* consiste en 349 imágenes de alta resolución, de un tamaño promedio de 1260 x 860 pixeles, descargadas del Google Street View de escenas tomadas a nivel de calle. El conjunto de entrenamiento consta de 100 imágenes y 249 para la evaluación final de reconocimiento. Este conjunto de pruebas presenta la complejidad de que un gran número de palabras contenidas en las imágenes no han sido anotadas para su reconocimiento. Además, la *SVT* es un conjunto que representa un gran reto debido a una gran cantidad de ruido, además de tener un amplio rango de fuentes y estilos gráficos de caracteres. Para cada imagen se provee un diccionario de 50 palabras (*SVT-50*).

4.2 Experimentos

Previo a las pruebas finales, se entrenó la Red Neuronal Convolucional para el reconocimiento de caracteres de forma individual. Para esto, se utilizaron los conjuntos de imágenes de contornos de letras generadas a partir de transformaciones gráficas de diferentes fuentes seleccionadas, obtenidas de un editor de texto convencional (Sección 3.2.1.1). Una vez entrenada la red, se realizaron pruebas preliminares de reconocimiento de texto, utilizando únicamente el conjunto de entrenamiento de la *ICDAR 03*. Las pruebas preliminares (al igual que los experimentos finales), consistieron en aplicar de forma automática y continua todo el conjunto de métodos de Identificación (Sección 3.1) y de Reconocimiento de Texto (Sección 3.2), como un solo paso, lo cual es reconocido en la literatura como métodos *End-To-End*. Esto es, el algoritmo recibe el conjunto de imágenes como entrada, las procesa, y devuelve directamente el texto identificado en cada una, sin ninguna intervención intermedia por parte del usuario. Las pruebas preliminares tuvieron como objetivo realizar los ajustes necesarios de los algoritmos, así como establecer los valores más convenientes de los distintos parámetros requeridos. Una vez obtenidos resultados que consideramos satisfactorios con los conjuntos de entrenamiento, se realizaron las pruebas finales con los tres conjuntos de datos.

4.3 Resultados

Como parte de las pruebas realizadas con los 3 conjuntos de datos, se obtuvo el F_valor para los resultados de cada una. El F_valor relaciona la precisión y la sensibilidad del reconocimiento como un valor único. La precisión corresponde al total de palabras correctas encontradas por el método, dividido entre el total de palabras reportadas por el mismo. La sensibilidad corresponde al total de palabras correctas encontradas, dividido entre el total de palabras presentes y anotadas del conjunto de imágenes. Dados estos valores, el $F\_valor = 2 \cdot \frac{Presición \cdot Sensibilidad}{Precisión + Sensibilidad}$.

La Tabla 4 muestra los resultados obtenidos en este proyecto y por los métodos del estado del arte. Las columnas 2 a 4 corresponden al *F_valor* de reconocimiento para cada conjunto de datos. La columna 5 muestra el tiempo promedio que requieren los métodos para procesar una imagen de 1200 x 800 pixeles. Los resultados muestran que el método propuesto obtuvo resultados competitivos con los tres conjuntos de datos. Comparando los resultados con el trabajo de [13], quienes reportan los mejores resultados de los métodos basados en reconocimiento de caracteres individuales, en este proyecto se obtuvieron 1 o 2 puntos porcentuales por encima cuando en [13] utilizan bigram, y 5 o 7 puntos porcentuales por encima, sin uso del bigram. Como se explica en la Sección 2.3, en la propuesta de [13] le nombran bigram al diseño y entrenamiento de una Red Convolucional para el reconocimiento de pares de caracteres unidos (604 neuronas de salida), lo cual incrementa sensiblemente la complejidad de la red. Consideramos importante mencionar que existen métodos que entrenan la Red Convolucional a través de la imagen de palabras completas, que han obtenido los mejores resultados para estos conjuntos de datos. Por ejemplo, en [1] logran un F_valor de 91% para la *IC03-50*, 87% para *ICD03-Full* y 82% para la *SVT-50*. Sin embargo, estas propuestas requieren de un número de neuronas de salida equivalente al tamaño del diccionario del lenguaje que se pretende reconocer (decenas de miles de palabras), lo que implica un esfuerzo computacional extremadamente grande, tanto para el entrenamiento de la red, como para el procesado y reconocimiento de una imagen. Por tal motivo, consideramos que los métodos que trabajan a este nivel de reconocimiento de palabra completa se encuentran en otra categoría de análisis y estadística de resultados.

Tabla 4. Resultados de la detección y reconocimiento de texto (% F_valor).

Método	IC03-50	IC03-Full	SVT-50	Tiempo Prom.
Wang et al., 2011 [23]	68	51	38	15 s
Wang et al., 2012 [8]	72	67	46	---
Alsharif and Pineau, 2014 [24]	77	70	48	---
Jaderberg, M., 2014 [13] (sin bigram)	76	---	50	---
Jaderberg, M., 2014 [13]	80	75	56	90 s
Este Proyecto	81	77	57	28 s

En la Figura 4.1 se muestran ejemplos de imágenes de la *ICDAR 2003*, de casos de reconocimiento exitoso. A la izquierda se muestran las imágenes originales y del lado derecho la imagen procesada, previo al último paso de reconocimiento del texto.

Figura 4.1. Ejemplos de casos de reconocimiento correcto.

En la Figura 4.2 se muestran ejemplos de imágenes de la *ICDAR 2003* de casos que no se logró el reconocimiento. Los casos en los que el algoritmo desarrollado no logró un reconocimiento del texto, se pueden resumir en 4 tipos: 1) fuentes no consideradas en el entrenamiento, 2) ruido excesivo, 3) texto cuyo borde no se distingue del fondo y 4) texto muy pequeño.

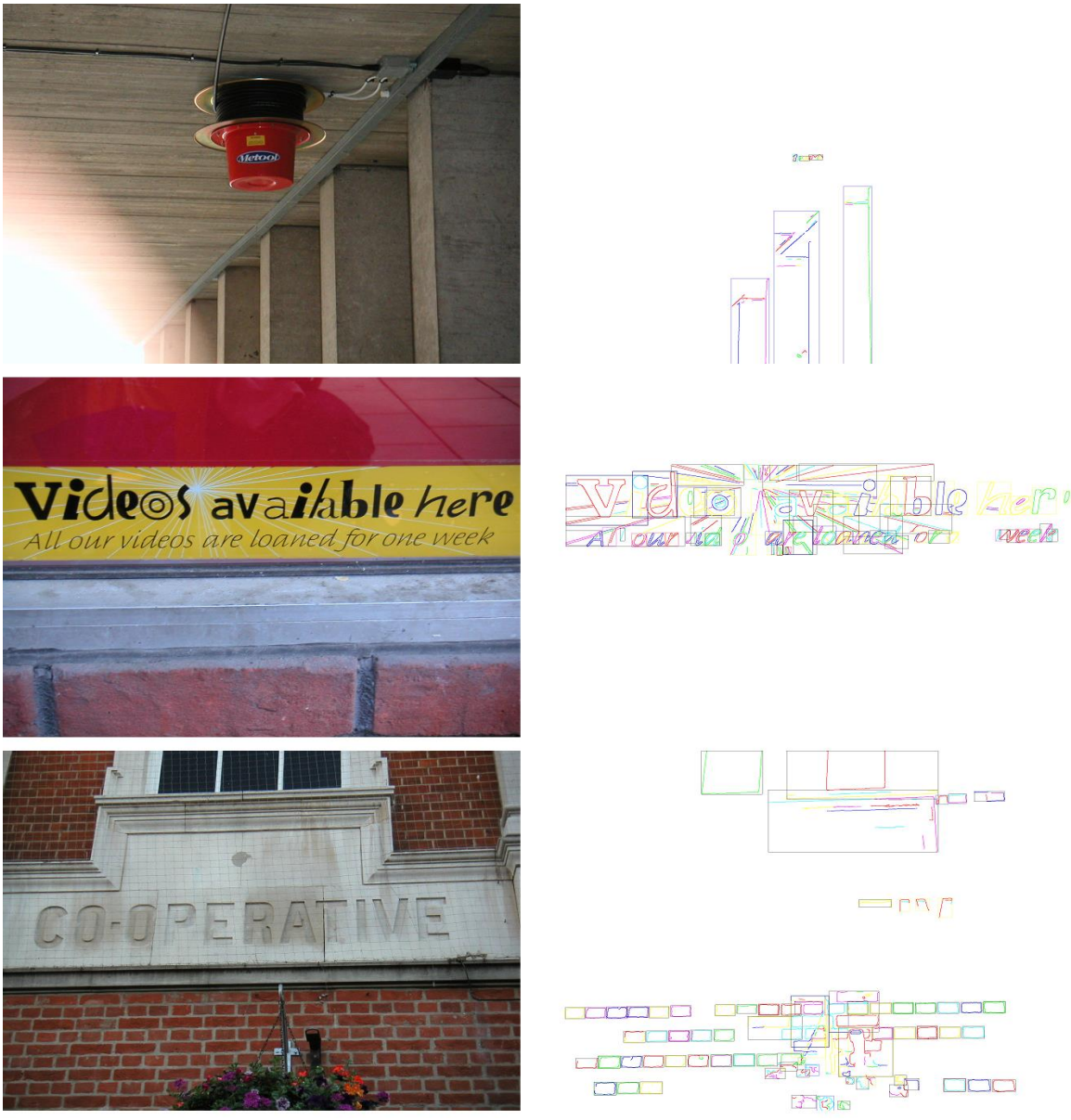

Figura 4.2. Ejemplos de casos de incorrecto reconocimiento.

5 Conclusiones y Trabajo Futuro

En este proyecto se diseñó e implementó una nueva estrategia para el problema de la *Detección y Reconocimiento de texto en Imágenes no restringidas*. Esta propuesta, a diferencia de los trabajos revisados del estado del arte, se basa en el entrenamiento de una Red Neuronal Convolucional utilizando únicamente el contorno o silueta de los caracteres para su posterior reconocimiento. Es decir, no se requiere procesar la parte interior o relleno de los caracteres. Con esto, se reduce significativamente el espacio de búsqueda, tanto en la fase de entrenamiento de la red como en el reconocimiento del texto, al no considerar los múltiples colores, tipos de iluminación o patrones gráficos, que se encuentran regularmente en el texto contenido en imágenes no restringidas. Esto también permite que el conjunto de imágenes de caracteres utilizado para el entrenamiento de la red, se pueda obtener directamente de las fuentes de un editor de texto convencional. Esto es, los datos de entrenamiento no dependen de los conjuntos de imágenes disponibles como casos de estudio.

La estrategia propuesta inicia identificando el borde de los objetos dentro de las imágenes, seguida por la generación de recuadros que encierran el contorno de los objetos que potencialmente corresponden a caracteres. Posteriormente, a través de una Red Neuronal Convolucional, se reconocen los caracteres contenidos en los recuadros, para después agruparlos en líneas de texto. Finalmente, se reconocen las palabras que se forman, comparando con las palabras de un diccionario preestablecido, a través de un algoritmo que aplica programación dinámica, desarrollado originalmente para el alineamiento de cadenas ADN y de proteínas.

Las pruebas finales de reconocimiento se realizaron con los conjuntos de datos *IC03-50*, *IC03-Full* y *SVT-50*, los cuales han sido ampliamente utilizados y documentados como casos de prueba de detección y reconocimiento de texto en imágenes no restringidas. Los resultados muestran que la estrategia propuesta es competitiva con los trabajos

encontrados en el estado del arte que también se basan en el reconocimiento individual de caracteres. Sin embargo, consideramos que estos resultados aún se pueden mejorar, implementando diferentes ideas que quedaron pendientes de aplicar por cuestión de tiempo. Entre otras, quedan pendiente: 1) ampliar las fuentes de caracteres para el entrenamiento de la red. En este proyecto se utilizaron únicamente 30 fuentes comunes, sin embargo, en las imágenes de prueba observamos diversas fuentes no consideradas. 2) Analizar e implementar estrategias más eficientes para la eliminación de ruido. En algunos de los casos de prueba no exitosos, observamos que no se logró identificar correctamente el contornos de algunos los caracteres, debido a la presencia de objetos de fondo o superpuestos más tenues. Estos objetos posiblemente se puedan tratar con técnicas más robustas de eliminación de ruido. 3) Cambiar el uso de rectángulos para encerrar los caracteres, al manejo de paralelogramos u otras formas más flexibles. En algunos casos, por ejemplo, con letras itálicas muy inclinadas, observamos que un rectángulo no puede encerrar completamente un carácter sin abarcar parcialmente a los caracteres que se encuentran a sus lados. Esto regularmente ocasiona que la red no reconozca correctamente al carácter.

Como conclusión final del trabajo de año sabático, consideramos que este proyecto fue una incursión exitosa a la problemática de identificación y reconocimiento de texto en imágenes no restringidas, siendo el primer trabajo que realizamos en esta área de investigación. Los resultados no solo fueron competitivos, sino que lograron mejorar ligeramente los mejores resultados del estado del arte. Creemos que en próximos proyectos, donde se implementen las mejoras comentadas e incorporen nuevas ideas, se pueden alcanzar o mejorar los resultados de los métodos que basan su reconocimiento en el entrenamiento de la red a través de imágenes de palabras completas.

Bibliografía

[1] M. Jaderberg, K. Simonyan, A. Vedaldi y A. Zisserman, «Reading text in the wild with convolutional neural networks,» *Int J Comput Vis, doi: 10.1007/s11263-015-0823-z,* vol. 116, pp. 1-20, 2016.

[2] S. Arnoud, C. Gu, H. Hu, J. Ibarz, D. Lee, S. Lin, R. Smith y R. Unnikrishnan, «End-to-End Interpretation of the French Street Name Signs Dataset,» *ECCV Workshops, doi: 10.1007/978-3-319-46604-0_30,* 2016.

[3] C. Guo, Z. Li, J. Wu, H. Wang, L. Xiao, C. Yang y X. Yin, «AdaDNNs: Adaptive Ensemble of Deep Neural Networks for Scene Text Recognition,» *CoRR, abs/1710.03425,* 2017.

[4] Q. Ye y D. Doermann, «Text Detection and Recognition in Imagery: A Survey,» *IEEE Transactions on pattern analysis and machine intelligence,* vol. 37, nº 7, pp. 1480-1500, 2015.

[5] L. Neumann y J. Matas, «Real-time scene text location and recognition,» *In Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.,* p. 3538–3545, 2012.

[6] J. Weinman, Z. Butler, D. Knoll y J. Feild, «Toward integrated scene text Reading,» *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 3, nº 2, p. 375–387, 2014.

[7] J. L. Feild, «Improving text recognition in images of natural scenes,» *Ph.D. dissertation, Computer Science, Univ. Massachusetts Amherst, Amherst, MA, USA,* 2014.

[8] T. Wang, D. Wu, A. Coates y A. Ng, «End-to-end text recognition with convolution neural networks,» *In Proc. IEEE Int. Conf. Pattern Recognit.,* p. 3304–3308, 2012.

[9] L. Neumann y J. Matas, «Scene text localization and recognition with oriented stroke detection,» *In Proc. IEEE Int. Conf. Comput. Vis.,* p. 97–104, 2013.

[10] «Visual Impairment and Blindness,» World Health Organization, [En línea]. Available: http://www.who.int/mediacentre/factsheets/fs282/en/. [Último acceso: 6 9 2017].

[11] S. Belongie, T. Matera, J. Matas, L. Neumann y A. Veit, «COCO-Text: Dataset and Benchmark for Text Detection and Recognition in Natural Images,» de *CoRR, abs/1601.07140*, 2016.

[12] C. L. Zitnick y P. Dollár, «Edge Boxes: Locating Object Proposals from Edges,» *Microsoft Research,* pp. 1-15, 2014.

[13] M. Jaderberg, «Deep Learning for Text Spotting,» D.Phil Thesis, Robotics Research Group, Department of Engineering Science, University of Oxford, 2014.

[14] P. Dollár y C. L. Zitnick, «Structured forests for fast edge detection,» *In: ICCV,* 2013.

[15] P. Dollár y C. L. Zitnick, «Fast edge detection using structured forests,» *CoRR,* 2014.

[16] P. Rantalankila, J. Kannala y E. Rahtu, «Generating object segmentation proposals using global and local search,» de *CVPR*, 2014.

[17] S. Manen, M. Guillaumin, L. Van Gool y K. Leuven, «Prime object proposals with randomized prims algorithm.,» de *ICCV*, 2013.

[18] E. Rahtu, J. Kannala y M. Blaschko, «Learning a category independent object detection cascade,» de *ICCV*, 2011.

[19] J. Uijlings, K. Van de Sande y T. Gevers, «Selective search for object recognition,» de *IJCV*, 2013.

[20] J. Canny, «A Computational Approach to Edge Detection,» *IEEE Transactions on Pattern Analysis and Machine Intelligence,* pp. 679 - 698, 1986.

[21] «IEEE Xplore Digital Library,» IEEE, 13 Enero 2020. [En línea]. Available: https://ieeexplore.ieee.org/abstract/document/4767851. [Último acceso: 13 Enero 2020].

[22] S. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong y R. Young, «ICDAR 2003 robust reading competitions,» 2003.

[23] K. Wang, B. Babenko y S. Belongie, «End-to-end scene text recognition,» *In: Proceeding of the International Conference on COmputer Vision. IEEE,* pp. 1457-1464, 2011.

[24] O. Alsharif y J. Pineau, «End-to-End Text Recognition with Hybrid HMM Maxout Models,» *In: International Conference on Learning Representations. ,* arXiv:1310.1811..