UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

Diseño e implementación de un Sistema de Reconocimiento de Textos Manuscritos basado en el uso de la Memoria Asociativa Entrópica

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

P R E S E N T A:

VÍCTOR DANIEL CRUZ GONZÁLEZ

Director de Tesis:
Dr. Luis Alberto Pineda Cortés
iimas, unam

Coasesor:
Dr. Carlos Ricardo Cruz Mendoza
iimas, unam

Ciudad Universitaria, CD. MX. Noviembre, 2022

Agradecimientos

Agradezco a mi familia por su compañía en cada paso de esta tesis y en ayudarme a alcanzar una nueva meta en mi vida académica. Específicamente, a mis padres Eloisa y Víctor y mis hermanas Romina y Priscila. Sus consejos, sus palabras de aliento y su tiempo son elementos preciados para que este documento sea posible.

Agradezco a los maestros incorporados al Posgrado y al Departamento de Computación por la calidad de sus clases y su excelente formación académica. Particularmente, a mis asesores Dr. Luis Alberto Pineda Cortés y Dr. Carlos Ricardo Cruz Mendoza por las constantes sesiones que tuvimos sobre este trabajo. Espero que esta tesis sea de gran relevancia para futuros estudiantes e investigadores.

De igual forma, gracias a los sinodales Dra. Wendy Elizabeth Aguilar Martínez, Dr. Rafael Morales Gamboa y Dr. Gibran Fuentes Pineda por formar parte del jurado y tomarse el tiempo de leer y comprender esta tesis. Sus observaciones e ideas enriquecieron su desarrollo.

Finalmente, agradezco a la UNAM y a su Posgrado en Ciencia e Ingeniería de la Computación por su gran nivel académico y la excelencia de sus profesores. Asimismo, al Consejo Nacional de Ciencia y Tecnología (CONACyT) por la beca otorgada durante mis estudios de maestría.

Resumen

En esta tesis se presenta un Sistema de Reconocimiento de Textos Manuscritos (SRTM) que modela el proceso de la lectura humana. El sistema implementa una Memoria Asociativa Entrópica (MAE), la cual almacena representaciones distribuidas y declarativas de caracteres alfanuméricos. Aunque el reconocimiento de textos se ha abordado desde hace mucho tiempo y se han desarrollado distintas soluciones, principalmente en el campo de la Inteligencia Artificial.

El problema se divide en dos grandes áreas: tipográfico y manuscrito. El primero se abordó desde 1914 con la máquina estadística telegráfica de Emanuel Goldberg y, en 1970, Ray Kurzweil inventó un sistema para procesar cualquier tipo de fuente tipográfica[1]. Este último se abordó mucho más tarde cuando IBM fabricó un *scanner* para reconocer los números escritos a mano en 1966. Actualmente, se desarrollan sistemas similares con Redes Neuronales, Deep Learning o Modelos de Markov[2]. El software más popular es el Optical Character Recognition (OCR) y puede reconocer textos tipográficos (OCR-T) o textos manuscritos (OCR-M).

Esta tecnología introduce documentos impresos y manuscritos a una computadora, tales como formularios, documentos gubernamentales e incluso notas escritas en un *post-it*. Los procesos de (1) inscripción y verificación de datos personales, (2) abastecimiento de medicamentos en una farmacia y (3) asistencia a personas invidentes se pueden automatizar. Sin embargo, todavía no existe un hardware o software que resuelva el problema de forma infalible, ya que es computacionalmente complejo de modelar y tecnológicamente abierto. Las soluciones se limitan a reconocer los patrones almacenados en los documentosimagen y regresar su valor aproximado.

El sistema que aquí se presenta se compone de tres partes. La primera consta de algoritmos de visión por computadora para detectar y filtrar objetos sobre una imagen para obtener su representación digital. La segunda modela propiamente la percepción con Redes Neuronales Convolucionales y la memoria con las MAE. La tercera consiste en corregir sus errores ortográficos y gramaticales. El sistema se evaluó exhaustivamente en dos dimensiones y se comparó contra tecnologías alternativas, con resultados muy satisfactorios.

Índice general

Índice de figuras

Capítulo 1

Reconocimiento de textos manuscritos

Este primer capítulo aborda el proceso y el objetivo de la tesis. Se introducen los conceptos básicos relacionados a una Memoria Asociativa Entrópica (MAE)[5] y a un Sistema de Reconocimiento de Textos Manuscritos (SRTM). Asimismo, se presenta la metodología empleada y una introducción a los siguientes capítulos.

1.1. Reconocimiento óptico de caracteres

El *Reconocimiento Óptico de Caracteres* (OCR por sus siglas en ingles) es un proceso que analiza una imagen para regresar el texto que contiene. En su forma más básica, clasifica una imagen con píxeles que forman una letra o un número. Con algunas modificaciones, se transforma en un constructor de palabras y enunciados. Dependiendo de los requerimientos, existen diferentes tipo de OCRs que identifican automáticamente el texto[6]. Su primera división se basa en el tipo de caracteres: si son tipográficos, se conocen como *Printed Character Recognition (PCR)*, mientras que si son escritos a mano, se conocen como *Handwritten Character Recognition (HCR)*. Este último se subdivide en: (1) el uso exclusivo de las propiedades de la imagen y sus píxeles (*Offline Handwritten*) y (2) el uso de *metadata*[1] que almacena los trazos de cada letra (*Online Handwritten*) por medio de una *stylus* o el tacto[7]. El tipo de OCR que se implementa en el sistema propuesto es el primero: *Offline Handwritten Character Recognition (OHCR)* y tiene tres etapas generales: (1) pre-procesamiento, (2) procesamiento y (3) post-procesamiento, las cuales son similares a las del SRTM.

Existe una conferencia internacional que se basa en el análisis y reconocimiento de documentos llamada *ICDAR (International Conference on Document Analysis and Recognition)*. Ahí se realizan diversas competencias entre sistemas de reconocimiento de documentos. Una de ellas es sobre la extracción texto de caracteres manuscritos de tipo *offline* por medio de OCRs. Hay dos métricas base para declarar al ganador: *Character Error Rate (CER)* y *Word Error Rate (WER)*, las cuales calculan la diferencia entre dos cadenas

[1] Esto puede ser la posición del stylus, la velocidad con la que se genero el trazo y, en los más novedosos, la respectiva fuerza con la que fue creado.

de texto. Específicamente se compara un resultado de un OCR y su valor esperado. Evidentemente, si ambos fueran idénticos, la diferencia sería cero, mientras que la tasa crece conforme más caracteres existan o falten en el resultado con base en el esperado. CER cuenta las diferencias entre palabras al analizar caracter por caracter. Por otro lado, WER cuenta las diferencias entre líneas al analizar palabra por palabra, pero si alguna no es idéntica, no se hace un subcálculo por CER. Así, el OCR con el mínimo CER y WER es el ganador.

1.2. Planteamiento inicial sobre el reconocimiento de caracteres

Como se mencionó anteriormente, los símbolos manuscritos individuales se trazan con diversos estilos. En comparación con los tipográficos, estos usan un molde y sin importar la cantidad de veces que se introduzcan, se verán idénticas entre sí. Para el ser humano es imposible aprender todos los que existen para escribir una letra o un número. Hace parecer a la lectura como una acción imposible. Sin embargo, la forma que se ha encontrado para realizarla es por el análisis de los trazos. El cerebro se adaptó a leer caracteres del alfabeto latino en un tiempo corto (comparado con su fecha de origen), un desarrollo verdaderamente rápido, puesto que se utilizó una característica previa: la asociación mental[8]. De esta forma, se generó una similitud entre las letras y las formas encontradas en la naturaleza. El cerebro ha logrado asociar significados con letras, así como objetos con siluetas.

Esta acción cotidiana es un problema tecnológicamente abierto y complejo de modelar en una computadora digital estándar, ya que no posee ninguno de los sistemas visuales (los ojos), ni de procesamiento (el cerebro) del ser humano. Así, surgen dos preguntas: ¿es una computadora capaz de reconocer los caracteres manuscritos? y, de ser así, ¿cómo lo puede lograr? Inicialmente, utilizar análisis de imágenes, visión por computadora, RNA y MAE es una combinación que las puede resolver.

1.3. Proceso de lectura de textos manuscritos

El proceso de lectura en el ser humano es una adaptación a las necesidades de la comunicación no-verbal hasta convertirse en una acción inmediata y natural como hablar o caminar. Normalmente, los detalles no se analizan a gran profundidad para entender el significado de una oración, simplemente se realiza. En comparación, una computadora digital estándar requiere de un módulo lo suficientemente específico para lograr un proceso similar.

A continuación se ilustra informalmente una analogía sobre lo que realiza un SRTM para obtener un texto manuscrito, ejemplificado por un ser humano. Cabe destacar que el proceso de lectura es complicado y no se pretende realizar una copia fiel, sino un modelo simplificado y cualitativo.

Un filósofo llamado Funes terminó de escribir un nuevo fragmento para su libro en una hoja blanca de papel con su pluma fuente de tinta negra (figura 1.1). Antes de publicarlo, decide leerlo para verificar su

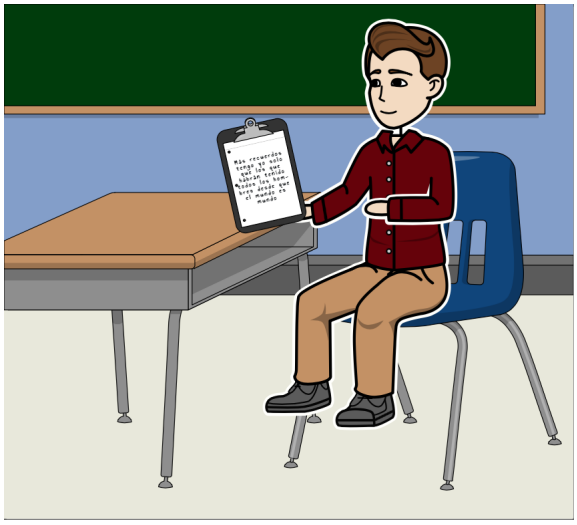


Figura 1.1: Funes leyendo una hoja con texto manuscrito en su escritorio

Figura 1.2: Acercamiento a la hoja de papel sujetada por Funes

coherencia y corregir errores ortográficos o gramaticales (figura 1.2). Hay errores en el texto[9] y Funes los debe notar.

Al principio, Funes sitúa su vista en el extremo superior izquierdo de la hoja, ya que el texto está escrito en español. Antes de comprenderlo, realiza una lectura directa con el fin de ubicar las líneas y sus palabras. Así, sus ojos barren cada línea de izquierda a derecha, sin extraer ningún símbolo aún solamenete ubicando la distancia de los objetos (figura 1.3). Inmediatamente después, se sitúa en la primer que encontró, para iniciar un análisis más detallado de sus letras. Filtra los datos que le parecen obvios o sin sentido, ya sea la transición entre letras o conectores simples.

Su ojo enfoca su atención a una palabra a la vez para transformar sus símbolos manuscritos en significados. Las demás quedan desenfocadas conforme más alejadas se encuentren (figura 1.4). Para que la información sea recibida en el cerebro, la retina transforma las características de cada símbolo en señales eléctricas. Hay

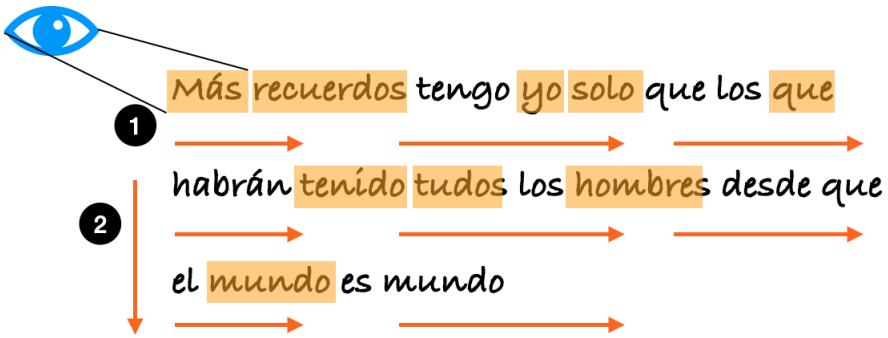


Figura 1.3: Proceso general de lectura visual. Como ejemplo, se resaltan algunas palabras por cada línea para establecer el tipo de movimiento ocular de la lectura. Aunque, el proceso que aquí se presenta analiza todas.

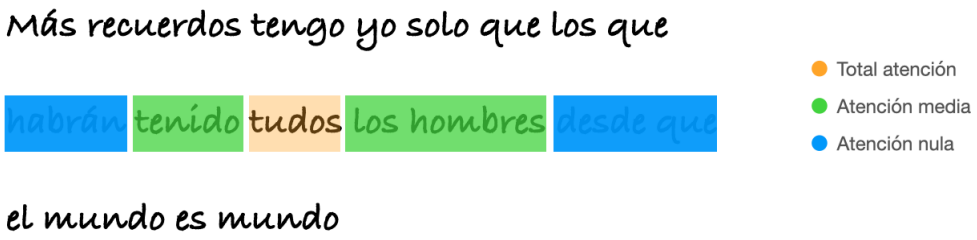


Figura 1.4: Funes presta atención a una palabra en particular, ignorando las que se encuentran en la periferia ocular

una palabra en particular en la que Funes se enfoca con más cuidado: “tudos”.

Internamente, su cerebro usa dos acciones que relacionan los símbolos con significados: la percepción y la memoria. La primera es la forma en la que interpretamos las señales, mientras que la segunda es el lugar en donde se guardan y se filtran esas interpretaciones. Si se tiene una señal por letra, el cerebro genera una pareja de abstracciones (la de percepción y la de memoria) y debe decidir cuál es la predominante para generar la palabra. Los resultados de la memoria pertenecen a una jerarquía superior por su capacidad de discernimiento de datos y son prioritarios. La figura 1.5 muestra el proceso mental de construcción de una palabra. Después de recibir una señal, se envía a la percepción y a la memoria para obtener un resultado colaborativo. La memoria ignora las transiciones, mientras que la percepción no las detecta, tal es el caso para la segunda y cuarta señal.

Al terminar de construir la palabra, se rectifica su sentido gramatical con base en su vocabulario, aunque o si se requiere ajustar, la sustituye (figura 1.6) por una similar o lee nuevamente ese fragmento de texto hasta encontrar la que mejor se ajuste y conozca.

Al final, Funes se da cuenta que la palabra está mal escrita y decide cambiarla por “todos”. Este proceso lo realiza para todas las palabras por línea. Al terminar con una, su cerebro verifica la ortografía y la gramática. El resultado final debe ser una abstracción del texto manuscrito original.

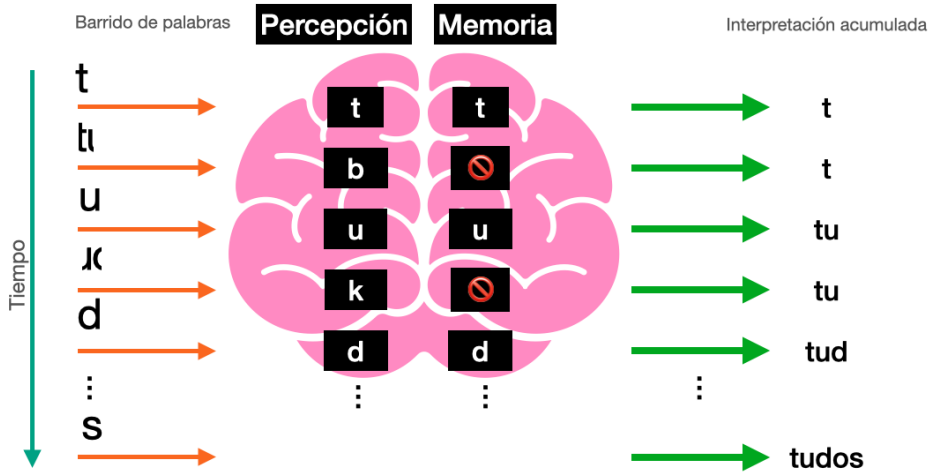


Figura 1.5: Análisis del barrido desde el ángulo de la perspectiva y la memoria

Figura 1.6: El cerebro filtra las palabras que conoce por su similitud al objeto concreto que se lee

Funes realizó un procedimiento similar a lo que haría el Sistema de Reconocimiento de Textos Manuscritos (SRTM) con MAE. En la vida real, conforme el cerebro de un ser humano madure, este proceso toma mucho menos tiempo y requiere mucho menos esfuerzo para reconocer cada palabra.

1.4. Etapas del SRTM

Un SRTM es un programa de cómputo que analiza una imagen de un texto manuscrito y produce su representación digital. En esta sección se presentan tanto el modelo conceptual, como su implementación. El primero se compone de tres etapas:

1. Etapa de pre-procesamiento: Ubicar las palabras dentro de una imagen, filtrar y modificar sus dimensiones para mejorar su análisis.

2. Etapa de percepción y memoria: Reconocer el texto original y obtener su representación computacional como una cadena de caracteres ASCII.

3. Etapa de filtrado: Corregir los errores de (2) y obtener las palabras más cercanas al contenido de la imagen.

Se implementa un Optical Character Recognition (OCR) junto con un sistema de memoria MAE[3], la cual permite registrar, reconocer y recuperar caracteres manuscritos. El módulo central es la segunda etapa y se implementa con Redes Neuronales para modular la percepción y MAE para modular la memoria. En conjunto, se lleva a cabo el reconocimiento de caracteres basado en la analogía de la lectura.

1.4.1. Pre-procesamiento

La imagen es una matriz tridimensional con valores entre 0 y 255. Su permutación en los píxeles genera patrones que representan un símbolo caligráfico. Esta etapa capta y analiza las propiedades básicas de una imagen[10] por medio de Visión por Computadora. Además, filtra y recorta la entrada para aislar las palabras usando Procesamiento de Imágenes. Se asemeja a la lectura directa y el envío de información de la retina al cerebro de la sección 1.3.

Esta etapa entrega un conjunto de imágenes por cada palabra. Éstas fueron transformadas en su tamaño y sus píxeles para la interpretación de la siguiente etapa. Las letras son mucho más claras y definidas y se reduce el ruido proveniente de la fuente.

1.4.2. Percepción y memoria

La segunda etapa analiza las imágenes y produce un conjunto de hipótesis de interpretación las cuales incluyen las palabras que almacenan. Se requiere alguna técnica de transformación imagen-a-texto. Una de

las más conocidas es una Red Neuronal Artificial (RNA), la cual se entrena para identificar patrones de píxeles que representan las líneas y trazos de un caractér y obtener el símbolo que más se relacione[11].

En la analogía del proceso de lectura, la RNA modera la percepción porque no es capaz de reconocer los límites entre los valores de su conjunto de aprendizaje y los que no contiene[12]. Así, no rechaza las transiciones entre letras, sino que regresa una aún cuando su patrón no esté almacenado. Su comportamiento es pseudo-reproductivo y no resuelve el *open set recognition problem*.

En contraste, el concepto de una Memoria Asociativa (MA) se asemeja a la memoria (sección 1.3). Algunas de sus propiedades son la asociatividad y la negación directa. Mientras que sus funciones son el almacenamiento de características (o *features*[2]) de una imagen y, en algunos casos, su reconstrucción o reproducción. Estas almacenan una gran cantidad de patrones y regresan uno a partir de una llave o *cue*.

Uno de los primeros trabajos fundamentales para la MA fue realizado por Bartlett[13] en 1930 en el campo de la psicología. Su experimento consistía en contar alguna historia con diferentes personajes y una trama o mostrar ilustraciones con varios detalles hacia un grupo de personas, una sola vez. Ellos debían recordarla y narrarla en sus propios términos durante las siguientes semanas, meses e incluso años. Sus reportes señalan que las historias eran recordadas con variaciones y perdían detalles conforme pasaba el tiempo. No obstante, se seguía contando la original de forma cada vez más breve y concisa. Esto significa que los elementos contingentes se iban perdiendo, pero su esencia se conservaba[14].

Un ser humano vive en un entorno con un valor de entropía en descenso, no obstante, se reciben múltiples entradas constante y simultáneamente a través de los órganos de los sentidos. MA es capaz de procesar esta información gracias a la distributividad de la información, similar a lo que realizan las neuronas. Pineda et al.[5] presentan la Memoria Asociativa Entrópica (MAE) que permite el registro, reconocimiento y la recuperación reconstructiva de los objetos almacenados distribuidamente en memoria relacionado a dígitos manuscritos utilizando el dataset MNIST[15]. Se compone de Registros de Memoria Asociativa (AMRs por sus siglas en ingles), los cuales almacenan diferentes representaciones de un mismo símbolo y su entropía se relaciona con el total guardado en cada uno. Su reconstrucción se basa en un balance de mínima entropía y un concepto de relevancia o peso.

En un trabajo posterior, Morales et al.[3] evaluaron las MAE sobre un dataset mas grande y variado de símbolos manuscritos, llamado EMINST[16]. Un colaborador en el equipo fue mi propia persona. Se utilizan 47 clases para representar los números y las letras del alfabeto latino en mayúsculas y minúsculas (figura 1.7). Para ambos datasets, se usaron imágenes como *cue* para que sean reconstruidas por sus features, similar a una memoria natural. Los resultados en ambos artículos fueron satisfactorios.

Las MAEs tienen el potencial de obtener resultados mucho más precisos que las RNAs, porque la primera implementa el concepto de rechazo en sus resultados. Ambas generan un resultado por cada imagen del barrido por palabra(sección 1.3). La etapa termina con dos conjuntos de cadenas de texto sin procesar y con hipótesis verdaderas y falsas, ya que no todas las letras recuperadas coinciden con la palabra original,

[2]Una feature es un elemento geométrico de la imagen, como sus bordes o *blobs*.

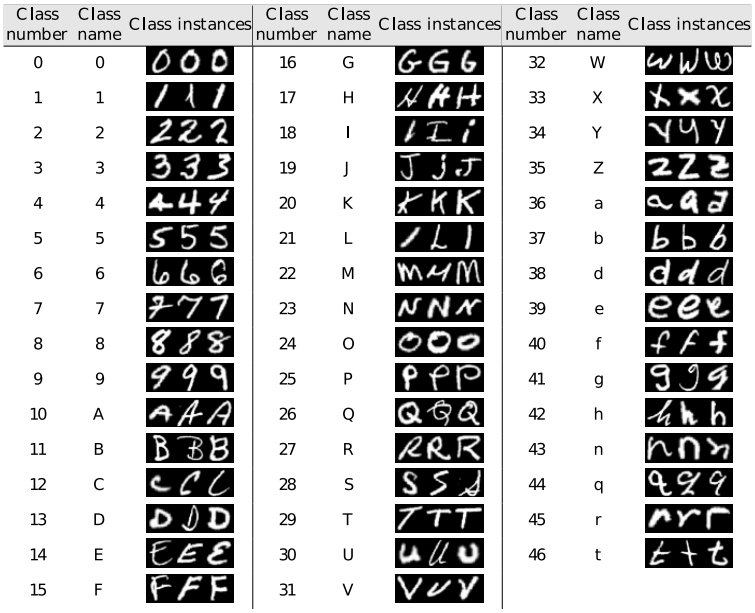


Figura 1.7: Alfabeto EMNIST-47[3]

además de que muchas se reiteran durante el proceso.

1.4.3. Filtrado

La tercera y última etapa corrige las palabras construidas de la percepción y la memoria, ya que ambas carecen de una lógica del lenguaje. La clasificación que esta etapa hace por cada trazo puede llegar a equivocarse por la naturaleza de la escritura humana. Algunos de los caracteres que se confunden con facilidad son: la letra 'o' mayúscula o minúscula con el número 0 (cero), la letra mayúscula 'I' con el número 1 (uno) o la letra minúscula 'l' o la letra 's' mayúscula o minúscula con el número 5 (cinco), entre muchos otros ejemplos. Así por el contexto de los demás caracteres, esta etapa da coherencia a cada palabra para alinear solo números o letras, mas no una combinación.

Asimismo, se filtran las hipótesis falsas sobre lo obtenido en la percepción y la memoria. En caso de que la segunda rechace alguna transición, el resultado de la primera se omite. Con esto, se reduce el tamaño de cada palabra y ambas se combinan por medio de reglas que priorizan los resultados de las MAE y las precisiones de la percepción y la memoria. La palabra resultante debería ser la representada por la imagen.

Se analizan sus errores ortográficos. En caso de modificarse, se cambia por la más similar conforme a un diccionario del idioma y una métrica de distribución. El barrido de las letras puede omitir, añadir o cambiar símbolos. La última acción que se realiza en esta etapa es corregir gramaticalmente las nuevas palabras conforme a su contexto. Se revisa que los sujetos, los verbos y los predicados de una oración concuerden entre sí y con sus tiempos. Así, se traduce una imagen a texto, se obtienen las palabras u oraciones expresadas conforme a la escritura manuscrita.

1.5. Organización de la tesis

Por último, en esta sección se describen los siguientes capítulos y su alcance.

- **Literatura sobre SRTM**: Se describe cronológicamente la literatura que existe actualmente sobre los dos temas principales de la tesis (OCR y MAE).
- **Reconocimiento de caracteres manuscritos**: Para este capítulo, se incluyen una descripción sobre el proceso de lectura en el ser humano, los requerimientos funcionales del SRTM y sus restricciones.
- **Recursos de SRTM**: Se detalla el corpus de imágenes para probar el sistema y el proceso de traducción hacia el corpus de entrenamiento. Además, se definen las medidas de comparación para analizar la eficiencia del sistema.
- **Símbolos manuscritos en la Memoria Asociativa Entrópica**: Este capítulo recolecta el contenido de diferentes versiones de MAE[3, 17] y se realizan experimentos sobre el dataset de datos de entrenamiento. Se tiene un antecedente para utilizar a MAE como parte del sistema OCR.
- **Arquitectura de SRTM**: Se detalla la forma en la que se implemento el OCR utilizando MAE, de forma teórica y técnica.
- **Pruebas sobre texto caligráfico y sus resultados**: Se muestra el proceso para realizar las pruebas, obtener los benchmarks del OCR y discutir sus valores y razones.
- **Conclusiones generales sobre SRTM y MAE**: Se discute sobre el trabajo en general, se compara el OCR con otros trabajos y se mencionan los trabajos futuros.

Capítulo 2

Literatura sobre SRTM

En este capítulo se presentan de manera introductoria los conceptos principales de la presente investigación: MAE y SRTM. Ambos se han desarrollado con gran rapidez en los últimos años, pero con diferentes enfoques. El primero se investiga académicamente, mientras que el segundo se aplica en la industria. Los campos de investigación con mayor interés son Inteligencia Artificial y Visión por Computadora.

2.1. Memorias asociativas

Las memorias asociativas (MA) se han desarrollado desde el siglo pasado, tomando como base ciertos estudios de la psicología y el comportamiento cerebral humano. A continuación, se presentan cronológicamente algunas publicaciones que han permitido su desarrollo. El primer modelo conocido sobre MA es la *Lernmatrix* de Steinbuch[18] en 1961. Utiliza un hardware que representa a un *crossbar*[1] para almacenar y recuperar analógicamente patrones por medio de circuitos ferromagnéticos y sobrelapando sus conectores. Esto se considera como una memoria heteroasociativa, ya que las representaciones están asociadas a una sola clase.

La arquitectura del *Correlograph* de Wilshaw[19] se publicó hasta 1969. Se basa en la *Lernmatrix* y añade el concepto de la holografía como medio de almacenamiento. En lugar de una matriz, se utiliza una red asociativa que es similar a una red neuronal, porque es el primer modelo en proponer una activación neuronal durante su funcionamiento. Asimismo, se caracteriza por una resistencia al ruido.

En 1972, el Asociador Lineal de Kohonen[20] aún toma como base a la *Lernmatrix*. Su diferencia principal es el uso de una matriz de correlación. A diferencia de otras, no se usa una llave escalar, sino una vectorial. Este modelo es "tolerante a fallas" y recupera fácilmente su información, puesto que aprende por medio de subconjuntos de la matriz con datos seleccionados aleatoriamente. Con algunas modificaciones, se puede convertir en una memoria auto-asociativa o en un clasificador.

[1] *Crossbar* es una matriz que almacena valores con pesos

Uno de los modelos más famosos se publicó en 1982 por Hopfield[21]. Se propone un sistema computacional con una memoria direccionable por contenido basado en "organismos biológicos". Esta recupera información total por medio de un argumento parcial. Se conoce como la Red de Hopfield y es un modelo estocástico que aprende patrones discretos y digitales. Además por primera vez, se propone un funcionamiento asíncrono, similar a la actividad neuronal. Este modelo es una de las bases de la Red Neuronal Artificial (RNA) por la adopción del concepto "neurona" en el campo de la inteligencia artificial y la optimización basada en energía. Su almacenamiento se limita a $0.15 \cdot N$, donde N es el número de neuronas especificadas en la red.

Hasta 1988, existían dos tipos de memorias unidireccionales: auto- y hetero- asociativas. Kosko[22] presenta la Memoria Asociativa Bidireccional (MAB). En su aprendizaje, se generan dos relaciones dado una pareja de elementos (A, B): (1) $A \rightarrow B$ y (2) $A \leftarrow B$. Las demás memorias trabajaban con valores binarios $\{0, 1\}$, mientras que MAB utiliza $\{-1, 1\}$. La matriz de correlación y el uso de los pesos se conservaron. La entrada puede ser cualquier elemento de la pareja y la memoria recupera su contra parte.

Hopfield y Krotov[23] rediseñaron su red al introducir la Memoria Asociativa Densa en 2016. Las propiedades de MA y *Deep learning* se unen para aumentar su capacidad de almacenamiento y utilizar una nueva función de energía. Esta última se basa en MA, pero se usa para analizar propiedades de la RNA.

A continuación se mencionan algunas de las aplicaciones de MA. En particular, no existe alguna que combine OCR y MAE, sino técnicas de aprendizaje profundo o heurísticas basadas en modelos de Markov, principalmente. La principal aportación del presente trabajo es una implementación que combinen las primeras dos. Solamente se ejemplifican las aplicaciones en otros contextos.

El reconocimiento facial es una de las primeras aplicaciones publicadas. El sistema de Su y Chou[24] analiza las features de una cara para clasificarla, en caso de que falle, no se da acceso. Utiliza una Memoria Autoasociativa que funciona por medio de operaciones matriciales. Los autores reconocen que no se necesita buscar en RAM, ya que la fotografía que se tome de un rostro nunca será idéntica a la almacenada. Además, se percataron de la variedad de factores que impiden obtener un resultado fiable. Su sistema se compara con una red neuronal utilizando fotos de rostros y globos, dado su gran similitud estructural. La red neuronal considera a los segundos como una cara válida. Esto es una brecha de seguridad que fácilmente se puede explotar para permitir el acceso a cualquiera. Por último, se reportan dos características sobre la memoria: (1) localizar rápidamente un conjunto de caras semejantes a la de la imagen y (2) validar si ese conjunto son semejantes a sus objetos previamente guardados para evitar el problema de la red neuronal.

El agrupamiento es otra aplicación. Yang y Ding[25] proponen un sistema que agrupa imágenes similares para clasificarlas, se conoce como AMOC (*Associative Memory Optimized Method on deep neural networks for Image Classification*, por sus siglas en ingles). Se basa en la teoría de la psicología que menciona que las ideas y la memoria están asociadas por la experiencia. Por ende, para clasificar las imágenes, se debe generar una relación entre ellas bajo una misma categoría. Su sistema aprende sobre los patrones entre una pareja de imágenes (una es la muestra y otra es la imagen central de la categoría) para extraer sus features, pasarlas a una MA y generar nuevas para tener una mejor clasificación.

Namba y Zhang[26] utilizan a la MA para reconocer imágenes en Braille. Su sistema implementa CNN (Cellular Neural Network for associative memory) y se reporta un rendimiento superior a la red de Hopfield. Una particularidad es que la conexión interna de cada una de sus células no es total. Varios patrones se almacenan con anterioridad para extraer las features de una imagen y encontrar una similitud. Sin embargo, su asociatividad no es del todo clara, se asemeja a un tipo específico de red neuronal.

2.2. OCR

Los primeros inventos y patentes de OCR se basaban en ayudar a personas invidentes o en experimentar con los descubrimientos de su época. En 1914 se registró la Máquina Estadística de Goldberg[1], la cual transforma caracteres impresos en código telegráfico. El reconocimiento se realizaba sobre celdas fotoeléctricas vistas sobre un proyector de películas.

En los siguientes años, se inventaron múltiples máquinas transformadoras de texto tipográfico. La "Máquina taquigráfica de un solo ojo" (1916) de Flowers[27] lee por un ojo electromecánico de Selenio y escribe texto a través de una máquina de escribir con electroimanes. Codelupi[28] inventó la "Máquina lectora para ciegos" en 1921, la cual transformaba los cambios de luminosidad de un texto en pulsos táctiles. En 1951, GISMO[29] se patentó para leer texto y código Morse en voz alta letra por letra. Linvill[30] desarrolló un dispositivo portátil para que los invidentes leyeran texto llamado Optacon en 1962. En 1974, se inventó el OCR omni-fuente por Kurzweil[1] con el mismo propósito.

Los primeros inventos y aplicaciones dedicados a *leer* texto manuscrito se inventaron después de 1960. La IBM 1287[31] es el primer *scanner* que lee números manuscritos en cheques u hojas de cálculo impresas con diferentes colores de tinta, utiliza un corrector de errores *on-line* y manejaba una política de reintentos antes de desechar el número. Las empresas Adobe y Google[32] decidieron incluir la lectura de textos en archivos PDF sobre Adobe Acrobat y Google Drive en 2015.

En años recientes, los OCRs han tomado gran relevancia en la práctica y la teoría. Se usan en las empresas para automatizar y optimizar procesos que incluyen formularios, documentos o cheques escritos a mano y evitar capturarlos manualmente. Por otro lado, se han generado diferentes técnicas para extraer textos y reducir su tasa de error.

En el estado del arte se encuentran programas computacionales que extraen texto de forma *offline* e inmediata por medio de redes neuronales. De forma masiva, Apple lanzó una herramienta para sus dispositivos móviles en iOS 15 para extraer cualquier tipo de texto de una foto, su cámara o alguna imagen de internet, por ejemplo: números de teléfonos, direcciones y correos electrónicos. Amazon y Microsoft desarrollan API's privadas que regresan el texto en un formato específico y estandarizado, como JSON o XML. Por último, Tesseract es un OCR open-source inventado por HP en 1985. A partir del 2006 hasta 2018 Google añadió varias mejoras[32, 33]. Actualmente, es una de las librerías más populares en múltiples lenguajes de programación.

2.2.1. Etapa de preprocesamiento

Choudhary et al.[34] diseñaron un flujo de programación en Matlab para obtener individualmente los símbolos dentro de una imagen que combinan caracteres tipográficos, manuscritos y figuras. Muchas de las funciones que plantean pertenecen a la paquetería incluida en Matlab, como `bwlabel` y `regionprops`. Su estrategia se basa en obtener las áreas de todos los objetos y descartar aquellos que tengan valores menores a 25 y mayor a 3,000 píxeles^2. Ellos suponen que esos valores son más que suficientes para reconocer entre una letra de molde, cursiva, tipográfica y una figura adicional, como un logo.

Las arquitecturas de Cavalin et al.[35] y Bhende et al.[36] utilizan Modelos Ocultos de Markov (HMM por sus siglas en inglés) para encontrar una cadena de Markov por medio de variables observables. En un OCR[35], las últimas son las features extraídas de una imagen y los estados en la cadena de Markov son sus segmentos en los que se une cada letra. Con esto, se revisan sus patrones y se delimitan sus resultados. Si se desea implementar un HMM, se identifica la cadena de Markov con sus probabilidades. Una palabra manuscrita se segmenta por medio de dos etapas para leer letra a letra, después se clasifican por medio de sus *N-best* hipótesis y, por último, se verifican para obtener la mejor.

El sistema de Bhende et al.[36] detecta las líneas del texto y compara cualquier contraste entre áreas claras y oscuras, pues en cada una de sus fronteras, existe un trazo que representa a una letra tipográfica. Así se detectan los componentes interconectados de la imagen. A diferencia de otros sistemas, se implementa un componente de discretización para transformar el contenido de la imagen inicial en objetos cuadráticos y filtrar la información de cada uno de los trazos.

Los sistemas anteriores son discretos al segmentar cada una de las letras y no consideran su continuidad en el trazo hacia las siguientes. Ambos afectan el módulo de extracción al generar cortes adicionales de los *heads* o *tails*[2] de una letra. Por ejemplo, al analizar la letra 'p' cursiva, se tiene una *tail* prolongada y se malinterpreta como una nueva letra, supongamos 'i'.

2.2.2. Etapa de procesamiento

En esta etapa se desarrollan módulos en artículos de investigación, repositorios *open-source* o software comercial para obtener representaciones directas en ASCII de una imagen. Las empresas dedicadas a desarrollar software en la nube[37] ofrecen tecnologías privadas para cargar una imagen en sus sistemas y obtener el texto asociado. Chen[38] implementó un OCR para EMNIST en la nube de Microsoft (Azure). Se utiliza CNN (Convolutional Neural Network) como su tecnología principal de clasificación ya que esta incluida en CNTK (Microsoft Cognitive Toolkit). Se crea una red neuronal totalmente interconectada y cada imagen de EMNIST se clasifica por letras o números. El sistema se entrena y después se prueba para confirmar que la red neuronal clasifica correctamente los símbolos del dataset. Se obtuvo el 81.56 % de accuracy. No se probó con trazos de un usuario ni con una letra tipográfica.

[2] *Head* y *tail* son la extensión prolongada de un trazo de una letra a su inicio o fin, respectivamente.

En la industria se busca un OCR universal que reconozca símbolos manuscritos y tipográficos. Uno de los mayores retos es reconocer los límites entre ambos. El sistema de Sharma et al.[39] utiliza Deep CNN con 30 o hasta 100 capas de neuronas para analizar formularios. El documento se alinea por medio de constantes en alto y ancho, densidad de píxeles y su alineación con cada una de sus casillas. Se propone una CNN que incluye 512 neuronas en una densa capa oculta, un dropout de 0.25 y 32 filtros en su capa Convolucional 2D. El sistema se entrena con EMNIST y se reporta un accuracy del 95.62 % entre símbolos numéricos y alfabéticos. Se menciona que los formularios se rellenaron con los mismos símbolos del dataset al realizar las pruebas y no con trazos caligráficos de los usuarios.

2.2.3. Etapa de postprocesamiento

BERT[40] es un sistema desarrollado por Google y reconocido para aplicaciones de NLP (Natural Language Processing). Se utiliza para mejorar los resultados del OCR, ya que predice la siguiente palabra o letra. Para que BERT funcione correctamente, se entrena previamente, lo cual es computacionalmente costoso.

Nguyen et al.[41] desarrollaron un sistema similar, al detectar y corregir palabras erroneas del OCR en el contexto del enunciado que se le dio como parte de la imagen de entrada. Su OCR analiza enunciados realizados con símbolos tipográficos. Por lo cual, (1) BERT obtiene el enunciado para detectar los errores, junto con *MLM (Masked Language Model)* y *NSP (Next Sentence Prediction)* y sustituirlos con *tokens*[3]. Posteriormente, (2) BERT predice el reemplazo de los errores para mejorar el significado a la oración.

[3]Elementos vacíos que sirven como indicadores sobre en cuál posición existe un error dentro del enunciado

Capítulo 3

Reconocimiento de caracteres manuscritos

En este capítulo se enlista las funciones que el sistema realiza dado su arquitectura. Como se mencionó en la sección 1.2, la interpretación de caracteres manuscritos no es trivial, ya que se requieren sistemas para reconocer texto en una imagen y obtener una equivalencia computacional como ASCII. La estrategia se basa en la lectura humana. Al leer palabras, se utiliza (1) la intuición (o percepción) para reconocer los caracteres y es asistido por (2) la memoria para filtrar y corregir elementos irrelevantes.

3.1. Evolución de la lectura visual en el ser humano

El cerebro humano se ha desarrollado desde hace más de 100,000 años, mientras que la lectura se inventó hace más de 5,000 años. La brecha temporal entre ambas es significativa, sin embargo esta habilidad se utiliza con la misma frecuencia que las más antiguas, como hablar o ver[42]. Se ha mejorado gracias a la neuroplasticidad y la adaptación de generación y consumo de información en los seres humanos. Actualmente, se conoce una serie de etapas para establecer un proceso visual y cerebral de lectura.

El origen de la lectura visual en el ser humano se origina por una reutilización de una habilidad anterior que relaciona a objetos con los conceptos interiorizados en el cerebro basado en sus siluetas. Este proceso se llama "reciclaje neuronal"[43] y durante la evolución se transformó un patrón cerebral. El origen de la forma de las letras del alfabeto latino y los números se asemejan a las figuras que usualmente se observan. Por ejemplo, las esquinas de un cubo forman las letras 'Y', 'L' o 'I', mientras que la superposición de los objetos forma 'T' y los ojos de un mamífero o una fruta forman 'O'. En el proceso de lectura, en lugar de la activación neuronal por medio de la abstracción de aspectos específicos de un objeto o ser vivo para asociar su concepto, se activan por medio de los trazos de los caracteres.

Existe un ciclo entre dos procesos que permite leer cada palabra de un texto (fig. 3.1)[4]. El ojo humano

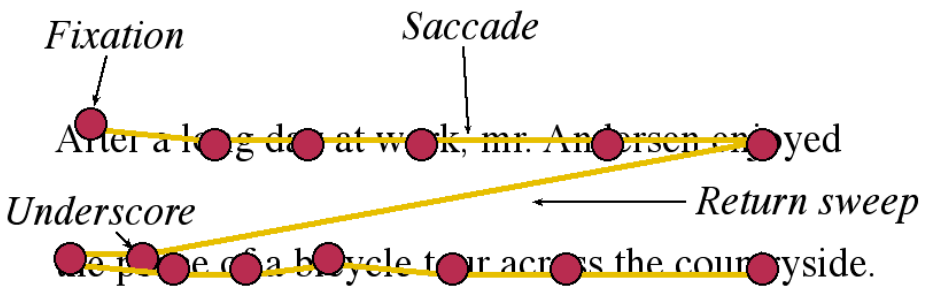


Figura 3.1: Proceso de la lectura visual realizada por la fóvea. Imagen obtenida de [4]

realiza *sacádicos* o pequeños saltos entre la palabra y, en pequeños intervalos de tiempo, pausa para extraer cada letra. A esto se le conoce como *fijación*. Durante la etapa del crecimiento humano, el cerebro capta más frases en una sola fijación y se incrementa la comprensión del texto en un menor tiempo de lectura. El ciclo se rompe momentáneamente para hacer *regresiones* por la falta de comprensión. Se *regresa* a una palabra, se reanuda el ciclo y se corrigen los errores. El aparato visual que ayuda a enfocar la atención en los trazos de un texto es la fóvea[44]. El ser humano adulto puede reconocer palabras completas con apenas ver las primeras letras y su longitud aproximada basado en el tipo de letra y su experiencia.

El SRTM adapta el proceso anterior para obtener las palabras y las oraciones. En lugar de la fóvea que se enfoca a una palabra, se utiliza la etapa de pre-procesamiento para identificar las zonas predominantes de caracteres manuscritos. Posteriormente, la etapa de procesamiento extrae el texto, similar a lo que se realiza con los sacádicos y las fijaciones. Por último, las regresiones y el post-procesamiento confirman o mejoran la información almacenada.

3.2. Requerimientos funcionales del SRTM

Para que un OCR funcione correctamente, es necesario implementar las tres etapas, además de permitir como entrada a una imagen y como salida a una cadena de caracteres. En esta sección se analiza cada requerimiento a mayor detalle para delimitar las funciones que se realizan.

Se debe manipular la imagen de entrada solo con sus elementos básicos, como su tamaño y sus píxeles para que sea *Offline*. Con esto, se usan las técnicas de visión por computadora o procesamiento de imágenes para extraer las partes importantes de la imagen, es decir, las que contengan trazos con texto, emitiendo dibujos o logotipos. Por ende, el sistema se basa en la implementación de un OHCR. Como se especificará en los siguientes capítulos, los filtros o los algoritmos de detección de intensidad son herramientas válidas para el reconocimiento de caracteres.

Se deben implementar las MAE en la etapa de procesamiento del OHCR para extraer el texto. En entrenamientos anteriores[3, 5] se usaron letras o números de forma individual y las memorias no reconocen palabras completas, sino que se construyen conforme se ejecute su proceso de lectura visual en etapas

posteriores. Existe un proceso específico para analizar los resultados generados de las MAE.

El resultado final del sistema es una representación en formato ASCII del conjunto de trazos manuscritos guardados en la imagen de entrada. Con esto se pueden generar *benchmarks* por medio de diferentes datasets que existen para medir la utilidad de un OCR. El sistema se limita a hacer una extracción y análisis de las palabras en ingles y sus oraciones.

3.3. Restricciones específicas del sistema

Hay ciertos límites sobre el SRTM respecto a su entrada y procesamiento. En el código se omitieron varios componentes por no estar planteados dentro del alcance de la tesis. Como es el caso de no procesar video, el tipo de imágenes que puede aceptar, los caracteres manuscritos que serán aceptados por el sistema y el formato que debe tener la imagen (como el tipo de fondo y color de la letra), entre otros.

Los argumentos de entrada válidos en el sistema OHCR son imágenes en formato JPG o PNG, sin un tamaño particular, pero con una paleta de colores monocromática. Específicamente, se espera que el contenido de la imagen sea un fondo claro con trazos en color oscuro, con el fin de reconocer el texto manuscrito. Por lo general, se puede usar un fondo blanco con un color negro como parte del trazo. Las imágenes pueden basarse en la escritura con *stylus*, pero en forma de una captura de pantalla, ya que no se utiliza su *metadata* sobre coordenadas y trazos. De igual forma, la imagen no debe contener figuras o subimágenes que estén fuera de los trazos de un caracter manuscrito.

Dentro de esta tesis se descarta el reconocimiento de caracteres especiales, como acentos, comas y signos de exclamación o admiración. Solamente se consideran las letras mayúsculas (de la A a la Z), los números (del 0 al 9) y algunas letras minúsculas que están en el dataset EMNIST-47[3]. Por otro lado, para realizar los benchmarks, el dataset principal utilizado es IAM[45], ya que contiene un conjunto de palabras y enunciados para que sean probados.

Capítulo 4

Recursos de SRTM

En este trabajo de tesis, se utiliza el dataset IAM[45] para realizar la evaluación del sistema. IAM es una base de datos que contiene imágenes de texto manuscrito en inglés y contiene diferentes categorías de imágenes. Algunas de estas son: (1) líneas de texto (figura 4.1) con 13,353 imágenes y (2) palabras (figura 4.2) con 115,320 imágenes etiquetadas. Los autores realizaron una segmentación sobre documentos transcritos por 657 escritores y añadieron manualmente los resultados. Sin embargo, hay elementos que fueron complejos de interpretar y se utiliza una bandera booleana que indica la legibilidad del texto.

El dataset que se utiliza es EMNIST[46], el cual incluye todas las letras mayúsculas y minúsculas del alfabeto del idioma inglés de forma manuscrita, así como los diez dígitos. En total, solamente se analizan 47 clases, llamado alfabeto EMNIST-47 (figura 4.3). Hay once letras minúsculas que se distinguen de sus contrapartes mayúsculas (las clases 36 a 46). Existen diferentes variantes sobre este dataset, se selecciona el segmento Equilibrado que incluye instancias de 2,800 por cada clase.

4.1. Traducción de IAM a EMNIST-47

El sistema se entrenó sobre el dataset EMNIST-47, pero no se relaciona con las imágenes de IAM por dos razones: (1) el primero es una recopilación de caracteres alfanuméricos, mientras que el segundo utiliza palabras y líneas y (2) se escribieron con diferentes caligrafías. Los símbolos de EMNIST se depuraron para obtener una imagen con el caracter sin ruido y con un fondo plano. Por otro lado, IAM omite la depuración de ruido en sus imágenes. Así, su evaluación no sería consistente con su entrenamiento. Al examinar a IAM,

Figura 4.1: Imagen de una línea manuscrita incluida en el dataset IAM

Figura 4.2: Imagen de una palabra manuscrita incluida en el dataset IAM

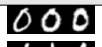
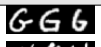
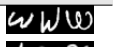
Class number	Class name	Class instances	Class number	Class name	Class instances	Class number	Class name	Class instances
0	0		16	G		32	W	
1	1		17	H		33	X	
2	2		18	I		34	Y	
3	3		19	J		35	Z	
4	4		20	K		36	a	
5	5		21	L		37	b	
6	6		22	M		38	d	
7	7		23	N		39	e	
8	8		24	O		40	f	
9	9		25	P		41	g	
10	A		26	Q		42	h	
11	B		27	R		43	n	
12	C		28	S		44	q	
13	D		29	T		45	r	
14	E		30	U		46	t	
15	F		31	V				

Figura 4.3: EMNIST-47 Alphabet

se añade que varias de las caligrafías son cursivas y no son horizontalmente uniformes. En consecuencia, se opto por realizar una conversión de IAM a EMNIST para reconocer palabras o enunciados. A continuación se detalla el proceso de traducción.

Ambas categorías de IAM contienen una carpeta con imágenes indexadas y un archivo con la relación entre texto manuscrito, palabra base en ASCII y su bandera sobre el estado de su conversión. El traductor utiliza este documento para obtener una nueva imagen equivalente con la concatenación de caracteres de EMNIST.

Para traducir las palabras de IAM, se define un conjunto de palabras $I = \{w_j \mid 0 \leq j < n\} \subset IAM$ con un tamaño n configurable por el usuario, se transforman los caracteres c_j de w_j en símbolos de EMNIST-47. Sea $translator : IAM \rightarrow EMNSIT$ una función que transforma un caracter de IAM a EMNIST-47, $random_e$ una función que selecciona aleatoriamente un caracter de EMNIST basado en un caracter ASCII y $EMNIST$ una matriz que almacena sus imágenes con un índice entero y agrupadas por categoría. Se tiene que

$$translator(c_j^{iam}) = EMNIST[c_j^{iam}][rand] = c_j^{emnist}, \forall c_j^{iam} \in w_j^{iam}$$
$$rand \sim \mathcal{U}(0, |EMNIST[c_j^{iam}]|)$$

El procedimiento para obtener una palabra se especifica con la siguiente serie de pasos:

1. Se extrae una palabra del documento de IAM.
2. Se transforma en mayúsculas, a excepción de los números.
3. Se obtiene un caracter de EMNIST-47 con $translator(c_j^{iam})$.
4. Se almacena en un buffer.
5. Se repite el paso (3) hasta terminar con todas las letras de la palabra IAM.
6. El buffer se exporta a PNG con espacios de píxeles aleatorios d_c entre sus elementos.

En el *codebase* del SRTM se programaron funciones que se muestran en el algoritmo B.2. La distancia en píxeles entre símbolos d_c se elige aleatoriamente para simular el trazo de un ser humano al escribir una palabra y sin sobreponerlas para que el texto sea legible.

$$d_c \sim \mathcal{U}(3, 4)$$

La función de aleatoriedad implementada en Python se modela como una variable aleatoria distribuida uniformemente. La función $random_e$ utiliza el parámetro c_j^{iam} para filtrar a EMNIST, obtener las imágenes que representan ese mismo caracter ASCII con diferentes trazos y regresar aleatoriamente una. Se utiliza un diccionario como estructura de datos para relacionar un caracter ASCII con las imágenes de cada clase de EMNIST-47. Un ejemplo sobre su funcionamiento es la figura 4.4. Las imágenes producidas cumplen con los dos primeros bloques de la etapa de pre-procesamiento.

El proceso es similar para traducir las líneas de IAM. Se utiliza el anterior y se concatenan los resultados con un espaciado aleatorio. Los pasos son:

1. Se extrae una línea del documento de IAM.
2. Se transforma en mayúsculas.
3. La cadena de texto se divide por los espacios entre palabras y se obtiene un arreglo.
4. Se traduce una palabra por el proceso anterior.
5. Su resultado se guarda en un buffer
6. Se repite el paso (4) hasta recorrer cada elemento del arreglo.
7. El buffer se exporta a PNG con espacios de píxeles aleatorios d_w entre sus elementos.

$$d_w \sim \mathcal{U}(9, 12)$$

4.2. Medidas de comparación

Las medidas de comparación utilizadas para evaluar las palabras y líneas de texto se definen como Character Error Rate (CER) y Word Error Rate (WER), respectivamente. Ambos conceptos están definidos por dos ecuaciones (4.1 y 4.2) que utilizan un par de palabras como argumento: la palabra base (b) y la hipótesis (h). Se cuenta el total de elementos que se tienen que sustituir (S), eliminar (D) y añadir (I), similar a la distancia de Levenshtein. Para CER, se realiza sobre las letras en una palabra, mientras que para WER son palabras en un enunciado. En ambos, N significa el total de todos los elementos incluidos.

$$CER(b, h) = \frac{S(b, h) + D(b, h) + I(b, h)}{N} \tag{4.1}$$

$$WER(b, h) = \frac{S_w(b, h) + D_w(b, h) + I_w(b, h)}{N_w} \tag{4.2}$$

Los resultados son directamente proporcionales a la diferencia entre palabras. Un sistema que obtiene un CER del 0.20 (o 20 %) indica proporcionalmente que el caracter en la posición $floor(0.20 \times N)$ no se reconoce correctamente. Las mediciones deben ser muy cercanas a cero para que un OCR se considere eficaz, lo que implica que la mayoría de las palabras y líneas de hipótesis son similares o idénticas a la base. Tener

Figura 4.4: Imagen generada por una de las palabras de IAM utilizando los símbolos guardados en EMNIST. La palabra original es `COUNTING`.

un WER tres o cuatro veces más grande que CER se considera un comportamiento normal. Esto se debe a que varias de las palabras no son idénticas y en su cálculo no desglosa su CER, simplemente se guarda como erróneo y su error aumenta. Por ejemplo, al analizar "Los *gato* tienen *nubes* vidas" sobre su enunciado base "Los gatos tienen nueve vidas", claramente hay dos palabras que se modifican, lo cual genera que $S_w = 2$ y WER de 2/5. El CER de ambas son $CER(\text{“}gatos\text{”}, \text{“}gato\text{”}) = 1/4$ y $CER(\text{“}nueve\text{”}, \text{“}nube\text{”}) = 2/5$. No obstante, WER no pondera S_w con base en CER, sino que su valor es definitivo.

Capítulo 5

Símbolos manuscritos en la Memoria Asociativa Entrópica

En este capítulo se introduce la arquitectura y desempeño de la MAE. Anteriormente, este sistema se utilizó como recuperador de caracteres de EMNIST[3]. El sistema implementa una versión llamada Memoria Asociativa Entrópica con Pesos (MAE-P)[17], la cual añade un nuevo factor para elegir o rechazar una cue: su similitud con el contenido de las memorias. En esta versión se excluyen los parámetros ι y κ y se aligera el análisis realizado sobre los registros y sus operaciones. Los resultados obtenidos se basan en el uso de EMNIST como dataset de aprendizaje y prueba. A continuación, se muestra la variante y su capacidad heteroasociativa al incluir letras mayúsculas y minúsculas dentro de la misma categoría de aprendizaje.

5.1. Descripción Funcional

En MAE-P se presentan ciertos experimentos para comprobar sus operaciones de registro (λ), reconocimiento (η) y recuperación (β) de representaciones de caracteres manuscritos con variantes en mayúsculas y minúsculas con resultados satisfactorios. Al igual que en artículos anteriores[5, 3], cada *AMR (Associative Memory Register)* de tamaño $n \times m$ tiene un registro auxiliar asociado definido como una tabla de la misma dimensión que se utiliza para colocar la cue de memoria para reconocer y registrar.

Este tipo de memoria es asociativo porque se accede a los AMRs a través de sus contenidos codificados como funciones discretas representadas en un formato tabular. Las representaciones concretas de los símbolos manuscritos se colocan en buffers de entrada y salida y se traducen a sus representaciones abstractas, que se ingresan a los AMRs a través del codificador. La salida del AMR es una representación abstracta del mismo tipo que se alimenta al decodificador para *re*construir su representación concreta.

Las representaciones de MAE son distribuidas porque las instancias individuales de los objetos almacenados se representan como funciones que se superponen dentro del AMR correspondiente. Así las celdas

de la tabla de registros pueden contribuir a la representación de más de un objeto y esta comparte celdas de una clase con otras. De esta forma, la relación entre las unidades de memoria y sus contenidos es *muchos-a-muchos*.

Sean los conjuntos $A = \{a_1, ..., a_n\}$ y $V = \{v_1, ..., v_m\}$ con $|A| = n$ y $|V| = m$. Ambos son el dominio y codominio de una relación $r : A \to V$. A diferencia de una función, un elemento de su dominio se puede relacionar con varios del codominio. Un ejemplo básico es una circunferencia en un plano cartesiano con su centro en el origen. Para un valor en x dentro de su diámetro, se obtienen dos valores en y. Mientras que su función representa una mitad inferior o superior. Para identificar que un objeto en A se relaciona con otro en V para cierta r, se define w_{ij} como el peso acumulado del valor v_j en un argumento a_i y una función booleana $R : A \times V \to \mathbb{N}_{\geq 0}$[1], el cual representa el peso $w_{ij} > 0$ de una relación r, tal que:

$$R(a_i, v_j) = \begin{cases} w_{ij} & \text{si el argumento } a_i \text{ se relaciona con el valor } v_j \\ 0 & \text{en otro caso.} \end{cases} \tag{5.1}$$

El contenido de cada columna se considera como la distribución de probabilidad Ψ en la que la masa de probabilidad de cada una de sus celdas es el peso dividido por la suma de todos sus pesos en la columna. Así, Ψ_i es el elemento de un arreglo de distribuciones con una cardinalidad igual a n y $0 < i \leq n$.

Por un lado, los AMRs tienen una entropía asociada, que se define como la medida de indeterminación promedio de su categoría distribuida. Por otro lado, sus columnas representan los argumentos de una relación (a_i), mientras que sus hileras son los valores resultantes (v_j), tal que $v_j \in r(a_i)$. Aunque todos los v_j se relacionan con las cues, no todos se devuelven, sino que se elige un valor v_j aleatoriamente para el argumento a_i, tal que $\{v_j \mid (a_i, v_j) \in r\}$. En otras palabras, no se limita a almacenar una relación *uno-a-uno*, ya que se pueden superponer. Así $R(a_i, v_j) > 0$ si se almacenan en el AMR y $R(a_i, v_j) = 0$ si no se tienen registros. Tales representaciones son *distribuidas*[47].

La figura 5.1 muestra un ejemplo de AMR que almacena los argumentos y valores de una relación arbitraria. Por ejemplo, al calcular $R(a_2, v_2) = 0$, es evidente que no existe registro o una celda marcada en la tabla. Las representaciones concretas introducidas se traducen en representaciones abstractas en forma de funciones y son los objetos registrados, reconocidos y recuperados en los AMR. El conjunto de celdas con $R(a_i, v_j) > 0$ es una abstracción de un objeto en la memoria.

Como un primer acercamiento para calcular la entropía de un AMR básico[5], se necesitan los siguientes argumentos: sea μ_i el número de valores asignados al argumento a_i en la relación r; sea $\nu_i = 1/\mu_i$ y n el número de argumentos en el dominio de la relación. En caso de que haya columnas vacías o una relación parcial, $\nu_i = 1, \forall a_i$ sin valores asignados en r. En conjunto, la entropía computacional básica $e(r)$ se define como la ecuación 5.2. La representación que contiene una sola función está completamente determinada, por lo que su entropía es cero.

[1] La relación se establece con letras minúsculas, mientras que su función booleana en letras mayúscuclas

$$e(r) = -\frac{1}{n}\sum_{i=1}^{n}\log_2(\nu_i) \tag{5.2}$$

En comparación con un modelo pesado, la ecuación de entropía se adapta a los nuevos valores que se almacenan. Cada celda contiene un número entero mayor o igual a cero. Para aglomerar los features y relacionar todas a un argumento, se unen en una mmisma estructura de datos matricial y, por ende, se cambia la indeterminación de la memoria. El registro almacena una gran variedad de valores que generan diferencias en la probabilidad de seleccionar una celda para recuperar una representación de un objeto.

Los AMRs mantienen relaciones con cierta indeterminación. La entropía está relacionada con la cantidad de objetos almacenados en la memoria. La entropía depende de las interacciones entre las funciones, es decir, el número de argumentos que comparten el mismo valor para un número de funciones: cuanto mayor es la interacción, menor es la entropía. Sin embargo, tomar una decisión con solo este elemento no es suficiente. Existen elementos dentro de los AMR que permiten seleccionar un mejor caracter.

A continuación se define una serie de ecuaciones que permiten conocer el peso de un AMR de forma completa[17]. El peso acumulado de una columna i se define como en la ecuación 5.3. La masa de probabilidad p de una celda en una hilera j en la columna i esta definida por la ecuación 5.4. Con estos elementos, se define la fórmula de entropía de Shannon (ecuación 5.5) para una columna i. Es decir, se calcula el promedio de la cantidad de información total contenida en i. La entropía completa del AMR (ecuación 5.6) se define como la entropía promedio de todas las columnas.

$$W_i = \sum_{j=1}^{m} w_{ij} \tag{5.3}$$

$$p_{ij} = \begin{cases} \frac{w_{ij}}{W_i} & \text{si } W_i \neq 0 \\ 0 & \text{en otro caso} \end{cases} \tag{5.4}$$

$$e_i = -\sum_{j=1}^{m} p_{ij}\log_2(p_{ij}) \text{ donde } \log_2(p_{ij}) = 0 \text{ si } p_{ij} = 0 \tag{5.5}$$

V_4	0	0	0
V_3	0	**35**	0
V_2	**10**	0	0
V_1	0	0	**250**
	A_1	A_2	A_3

Figura 5.1: Ejemplo de un AMR con Pesos con registros arbitrarios entre las celdas de argumentos y valores

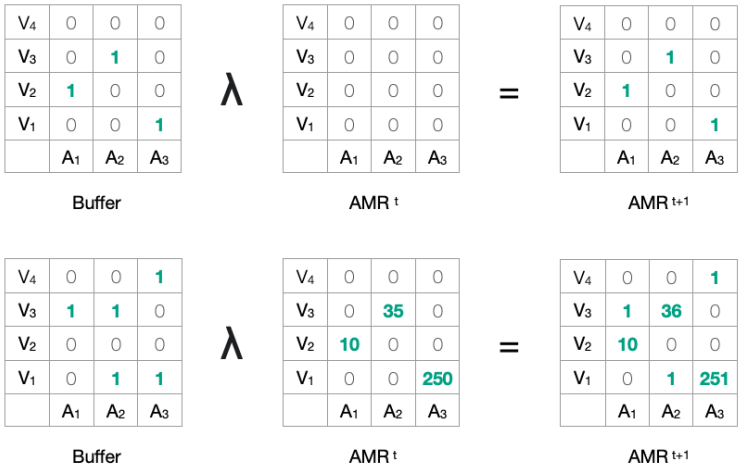


Figura 5.2: Dos ejemplos de la operación de registro sobre MAE-P

$$e_w = \frac{1}{n}\sum_{i=1}^{n} e_i \tag{5.6}$$

5.1.1. Operaciones del AMR

Para efectuar las tres acciones de un AMR, se requiere de una tabla adicional que funcione como registro auxiliar. Esta instancia funciona como un canal I/O para ingresar nueva información al sistema, reconocer una cue y recuperar representaciones del registro principal.

Sean dos relaciones arbitrarias r_f y r_a con dominio igual a A y codominio igual a V. La operación λ o de registro (ecuación 5.7) se define como la suma aritmética ($\oplus$) entre el estado actual de la tabla de un AMR y del registro auxiliar. Los valores resultantes se guardan en el AMR. Se considera como una operación distributiva porque la inserción de cada valor es paralela. La figura 5.2 muestra dos casos: el primero sucede cuando se registran inicialmente los valores y el segundo cuando previamente existían valores y se añaden nuevos. La suma se realiza celda por celda, respecto a los nuevos valores almacenados en el buffer.

$$\lambda(r_f, r_a) = q = r_f \oplus r_a, \text{ tal que } Q(a_i, v_j) = R_f(a_i, v_j) \oplus R_a(a_i, v_j)\ \forall a_i \in A \text{ y } v_j \in V \tag{5.7}$$

La operación η o de reconocimiento (ecuación 5.8) se define como la implicación material lógica (relajada por el parámetro ξ) entre las celdas de la tabla del registro auxiliar (r_a) y las del AMR (r_f). Adicionalmente, MAE-P utiliza una nueva comparación basada en los pesos del registro y de la cue: $\rho \geq \Omega$. Donde, ρ (ecuación 5.9) es el peso de la cue y Ω (ecuación 5.10) es el peso promedio de todas las columnas de la relación r_f con valores mayores a cero. Su resultado es booleano sobre los valores normalizados al elemento máximo por cada columna. La operación regresa `false` si no se cumple la nueva condición y, por ende, si alguna celda de r_a está marcada, pero la del r_f no y dependiendo de al menos $n - \xi$ argumentos. Tal es el único caso `false`

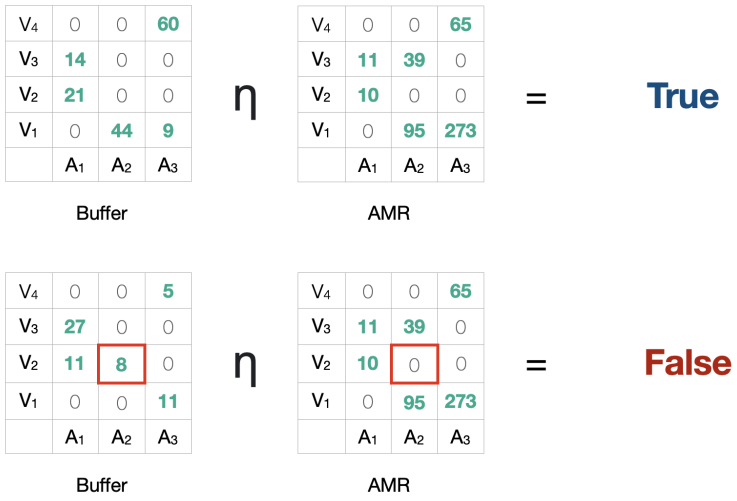


Figura 5.3: Dos ejemplos de la operación de reconocimiento sobre MAE-P

dentro de la implicación material. De lo contrario, se devuelve `true`. En la figura 5.3 se muestra los posibles resultados, dependiendo de la cue.

$$\eta(r_a, r_f, \xi) = \begin{cases} 1 & \forall i,j \text{ si } R_a(a_i, v_j) \rightarrow R_f(a_i, v_j) \text{ , con al menos } n - \xi \text{ argumentos de } r_f \text{ y } \rho \geq \Omega \\ 0 & \text{en otro caso} \end{cases} \tag{5.8}$$

$$\rho = \frac{1}{n} \sum_{i=1}^{n} R_f(a_i, v_{cue}) \text{ donde } v_{cue} \text{ es el valor } v_j \text{ del argumento } a_i \text{ en la cue } R_a(a_i, v_j) \tag{5.9}$$

$$\Omega = \frac{1}{n} \sum_{i=1}^{n} \omega_i \tag{5.10}$$

Donde ω_i (ecuación 5.11) es el peso promedio de un argumento a_i sobre todas las celdas mayores a cero en su columna.

$$\omega_i = \frac{1}{k} \sum_{j=1}^{m} R_f(a_i, v_j) \text{ con } k \text{ como el número de celdas de la columna } i \text{ en } r_f \mid w_{ij} \neq 0 \tag{5.11}$$

La operación β o de recuperación (ecuación 5.12) se define como la recuperación de representaciones de objetos almacenados en un AMR basado en una cue. Se regresa un resultado si y solo si el valor de la operación η es `true`. Un valor de cada atributo es seleccionado aleatoriamente con base en Ψ sobre la columna del argumento correspondiente. En la figura 5.4 se muestra un ejemplo de recuperación. Todas las celdas del Buffer coinciden con al menos una del AMR por cada columna. La recuperación del objeto que se almacena sobre $\texttt{Buffer}^{t+1}$ se construye por la selección aleatoria de un elemento por columna. En este ejemplo, las celdas (A_1, V_3) y (A_3, V_6) del Buffer coinciden con las del AMR, pero sus columnas contienen más de un elemento. Por ende, se generó un resultado con una diferencia en la columna A_1 con el valor V_1.

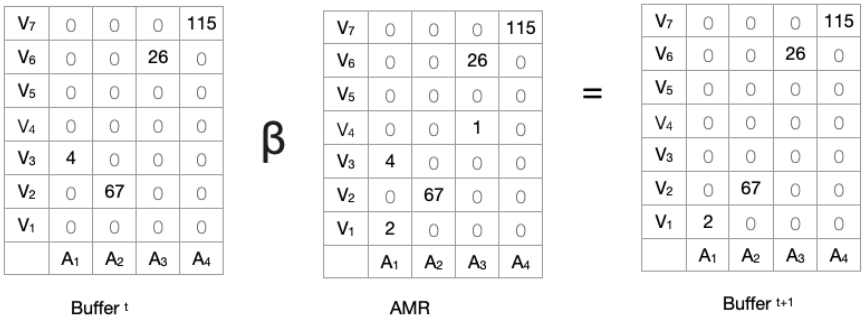


Figura 5.4: Ejemplo de la operación de reconocimiento sobre MAE-P

$$\beta(f_a, r_f, \sigma) = \begin{cases} f_v & \text{si } \eta(r_a, r_f, \xi) = 1 \\ \texttt{null} & \text{en otro caso} \end{cases} \tag{5.12}$$

Con el resultado f_v se obtiene aleatoriamente un valor representado por el renglón j (v_j) relacionado a $r_f(a_i)$ por medio de una variable aleatoria ζ con distribución normal. Su valor central es la cue $f_a(a_i)$ y con desviación estándar σ. Además, el seleccionado se traduce hacia uno de los valores de un arreglo Φ_i definido como el producto escalar entre Ψ_i y ζ_i.

5.2. Memoria visual con pesos para símbolos manuscritos escritos a mano

Las arquitecturas de RNAs funcionales y el valor óptimo del hiperparámetro n de los AMRs se determinaron utilizando el corpus de entrenamiento a través de experimentos preliminares. Se exploraron las potencias enteras de 2 y n se fijó en 32 para todos los resultados. Posteriormente, se entrenaron a las redes neuronales (el codificador, el decodificador y el clasificador) con el mismo corpus para un *10-fold validation* que permita disminuir la varianza de los resultados. Luego, el codificador se usó para procesar los corpus recordados y de prueba para producir los conjuntos correspondientes de representaciones abstractas. Estos fueron a su vez de entrada a los AMR correspondientes.

La arquitectura de MAE-P (figura 5.5) introduce una interacción con EMNIST durante el entrenamiento, prueba y utilización para el sistema. El dataset incluye imágenes de tamaño $p = 32 \times 32$ de un solo canal (blanco y negro) y se traducen en sus features (o cue) de tamaño n por medio del codificador para que las memorias operen. Con el módulo de decodificación se obtiene el desempeño de recuperación de las representaciones de EMNIST-47. El SRTM utiliza tanto el clasificador, como las memorias, ya que ambos regresan el codificador los caracteres reconocidos desde las imágenes.

Se implementa una red neuronal tipo VGG5[48] como codificador, ya que es un modelo de reconocimiento de objetos de "alto rendimiento" que se adapta en las arquitecturas de reconocimiento de imágenes. Una de

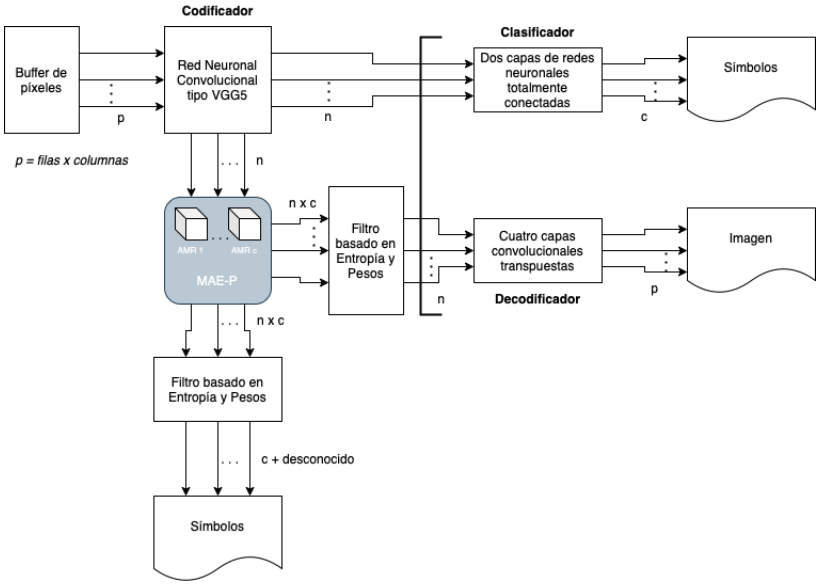


Figura 5.5: Diseño de arquitectura para MAE-P

sus mayores características son sus capas de convolución al tener un tamaño de 3×3. Específicamente, se conforma de cinco capas de este tipo más otras: (1) max-pooling para reducir el tamaño de los filtros, (2) densas y (3) *softmax* para determinar la clase de la imagen. Al final, se tienen tres capas de *neuronas* conectadas de forma total que analizan formas geométricas únicas de cada figura. En la etapa de entrenamiento, los trazos manuscritos de cada una de las letras de EMNIST-47 son clasificadas hacia una en ASCII.

El clasificador analiza las features y regresa el caracter más cercano a su aprendizaje, ya que se comporta como una RNA densa. Las memorias introducen la cue en sus AMRs y verifican si existe una representación similar para regresarlo o, en caso contrario, se rechaza. Se obtienen un arreglo con c features, dependiendo de las AMRs instanciadas en la memoria. El **Filtro basado en Entropía y Pesos** regresa al SRTM un solo caracter al filtrar el índice de la representación por medio de la ecuación 5.13. La división que se minimiza se conoce como *penalty* y se componse del arreglo de entropías e_w y de pesos p_{ij} para identificar aquel con la mínima entropía y su máxima masa de probabilidad.

$$\begin{aligned} index &= argmin_i(\frac{e_i}{p_{ij}}) \ \forall j \\ char &= EMNIST[index] \end{aligned} \tag{5.13}$$

5.3. Desempeño de MAE-P

Se investiga el desempeño de los AMR con características operativas satisfactorias en relación con su entropía. Se calculan las características de AMR de tamaño 32×2^m para $0 \leq m \leq 9$:

1. Se registra la totalidad de *RemCorpus* en su registro correspondiente mediante la operación $Memory\_Register$;

2. Se prueba el rendimiento de reconocimiento de todas las instancias del corpus de prueba a través de la operación $Memory\_Recognize$;

3. Se calcula la precisión, recuperación, exactitud y entropía promedio de las memorias individuales.

4. Para cada instancia del corpus de prueba, se recupera un objeto único mediante la operación $Memory\_Retrieve$ o se rechaza la cue.

5. Se calcula la precisión promedio y la recuperación del sistema integrado.

6. Se selecciona el parámetro m con el mejor equilibrio entre precisión y recuperación.

7. Se determina el rendimiento del sistema con tal tamaño de memoria $n \times 2^m$, para diferentes cantidades de *RemCorpus* y niveles de entropía.

En la Figura 5.6 se muestra la precisión, recuperación, exactitud y entropía promedio de los AMRs a lo largo del experimento con 10-fold validation. La precisión y la recuperación se calculan por AMR de la manera estándar de la siguiente manera:

$$precision = TP/(TP + FP) \tag{5.14}$$

$$recuperacion = TP/(TP + FN) \tag{5.15}$$

donde TP, FP y FN representan verdaderos positivos, falsos positivos y falsos negativos, respectivamente. Además, exactitud se obtiene por medio de

$$exactitud = w^{cue} \times \frac{TP + TN}{TP + FN} \tag{5.16}$$

La precisión es muy baja para valores pequeños de m pero crece conforme aumenta el número de hileras a una tasa 2^x. Por el contrario, la recuperación es muy alta para valores bajos de m, ya que la información se confunde cuando el número de hileras es muy bajo y se aceptan la mayoría de las instancias. Sin embargo, cuando se aumenta el valor de m, la cuadrícula se vuelve más fina, se mezclan las instancias verdaderas con las falsas positivas y se reduce la recuperación. Mientras que la exactitud se mantiene con niveles intermedios con valores bajos de m, ya que la cue pudo coincidir con varias de las celdas marcadas de forma distribuida en los AMR, aún cuando era el caracter incorrecto. De igual forma, su valor aumenta al incrementar el valor de m, hasta estabilizarse después de 32 hileras con un valor muy cercano al 100 %. El comportamiento es muy similar al de la precisión, ya que al tener TP mayores o FP menores, significa que las celdas del caracter correcto coincidieron con la cue ingresada.

El valor óptimo de m es un compromiso entre los gráficos de precisión y recuperación. La figura 5.6 muestra que existe un buen compromiso en $m = 7$ y $m = 8$, es decir, para las hileras de 128 y 256. Sin embargo, al cruzar la información entre la recuperación y la exactitud, su compromiso se ubica entre $m = 4$

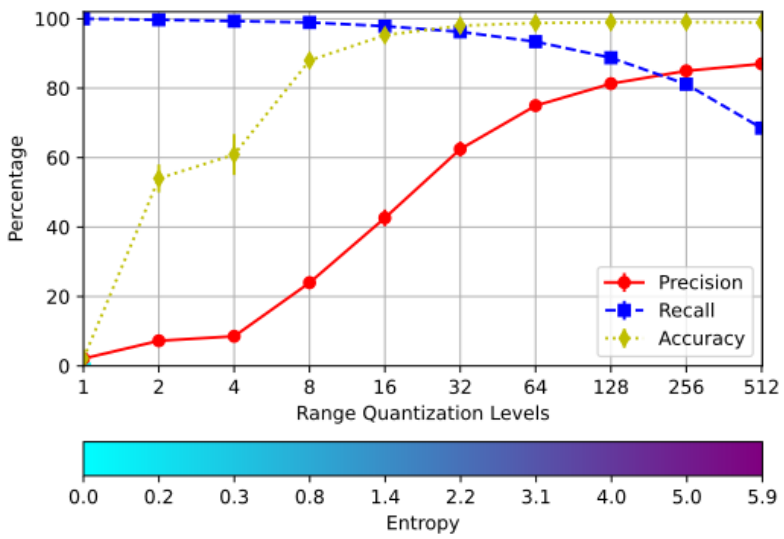


Figura 5.6: Precisión, recuperación y exactitud promedio de los AMRs

y $m = 5$, es decir, para las hileras de 16 y 32. La figura también muestra la entropía en la barra inferior. Como puede verse, su valor aumenta casi linealmente con el tamaño de los AMR, comenzando desde 0 para hileras de 1, y es máximo para hileras de 512.

La figura 5.7 muestra la precisión y la recuperación en función del número de hileras del sistema como un todo. En este caso, si todos los AMR rechazan una instancia, cuenta como un falso negativo para el sistema de memoria y reduce la recuperación total. En caso de que la instancia sea aceptada por al menos un AMR, se elige con base en la ecuación 5.13. Si el sistema entrega una clase incorrecta, entonces es un falso negativo para la clase correcta y un falso positivo para la clase que acepta, y aumenta en uno el recuento de falsos positivos y falsos negativos del sistema en su conjunto. La figura 5.7 muestra que la precisión y la recuperación tienen un patrón muy similar, ya que ambos crecen de un valor muy bajo a uno más alto de acuerdo con el aumento del número de hileras. Sin embargo, cuando la cuadrícula es demasiado fina, las instancias verdaderas se rechazan y la recuperación comienza a disminuir como antes. Las figuras 5.6 y 5.7 son coherentes, y ambas muestran que existe un buen compromiso en las hileras de 128 y 256. Principalmente, la primera indica el punto de .[eq]uilibrio.[en]tre *recall* y *precision* en ese rango.

La figura 5.8 muestra el número promedio de AMR que responden para cada instancia por tamaño de AMR. Como se puede observar, este número va desde 47 para un AMR de una hilera, pasando por 2 para uno de 2 hileras y hasta 1 para 64 o más hileras. Entre menor sea el número de memorias que respondan ante una cue, mayor claridad se tiene sobre el caracter que se debe elegir. Después de 32 hileras, la incertidumbre de elegir un valor con respecto a otro baja drásticamente.

Al unir esto valores con los de la figura 5.9, se observa que el rango entre 8 y 128 se encuentra el mayor

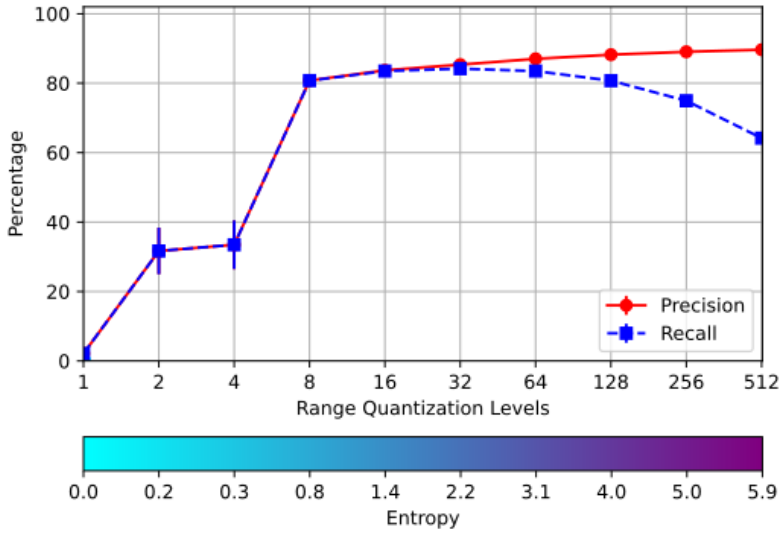


Figura 5.7: Precisión y recuperación en función del número de hileras del sistema como un todo

porcentaje de respuestas elegidas correctamente. Sin embargo, el máximo se ubica en 32 hileras, aunque no es el mínimo con respuestas correctas no elegidas, ni con el porcentaje sin respuesta. En este caso, 16 es la mejor selección. Mientras que 64 hileras genera un mucho mayor porcentaje sin respuesta. Con todo esto, se eligen 32 hileras como el mejor tamaño de memoria.

Si el número de features "recordadas" es muy bajo, la entropía también lo es. En el caso límite, si solo hay una, la entropía es cero, porque el total de opciones que se puede regresar es constante y único. Esto significa que existen muy pocos objetos o uno solo para elegir y, por ende, la incertidumbre de un AMR es baja o nula. Similar a la entropía en una función matemática. Al ejecutar la operación de reconocimiento de memoria, la precisión es muy alta. Sin embargo, incluso las señales que sean muy similares a los objetos almacenados serán rechazadas. Por lo tanto, la recuperación es muy baja.

La precisión decrece ligeramente con el aumento de la información recordada e inherentemente su entropía aumenta, pero la recuperación de las memorias individuales crece con bastante rapidez, hasta que se alcanza un compromiso satisfactorio entre precisión y recuperación. Sin embargo, si la entropía aumenta aún más, la precisión comienza a disminuir, pero la recuperación continúa creciendo. En este caso, los AMR están saturados, la mayoría de las instancias se aceptan y la recuperación es muy alta, mientras que la precisión se reduce significativamente.

La figura 5.10 muestra la tasa de almacenamiento de cada celda de un AMR. La mayoría no se utilizan, mientras que los valores 6 al 8 son los que almacenan la mayor cantidad de información. Por ejemplo, la celda (1, 6) tiene un porcentaje de almacenamiento mayor al 80 % y, por otro lado, se almacena el 0 % en cualquier celda con valor 2.

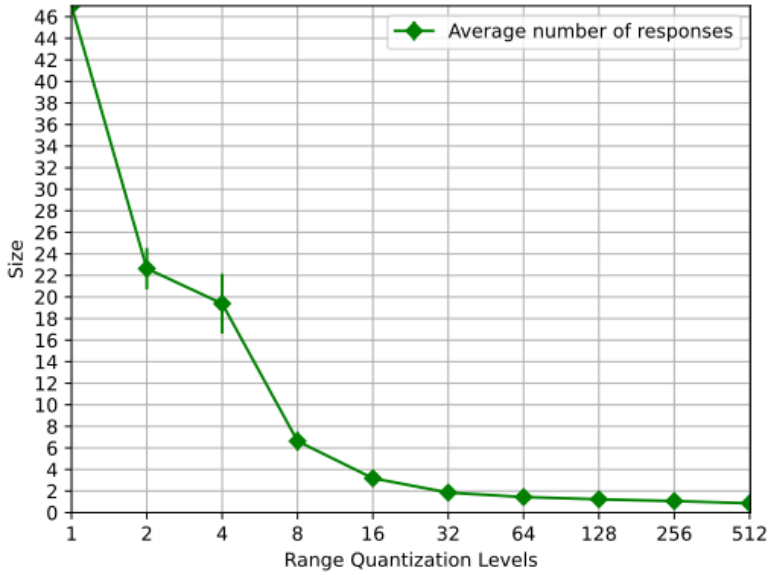


Figura 5.8: Número promedio de registros AMR de 47 a 1, con 32 o más hileras

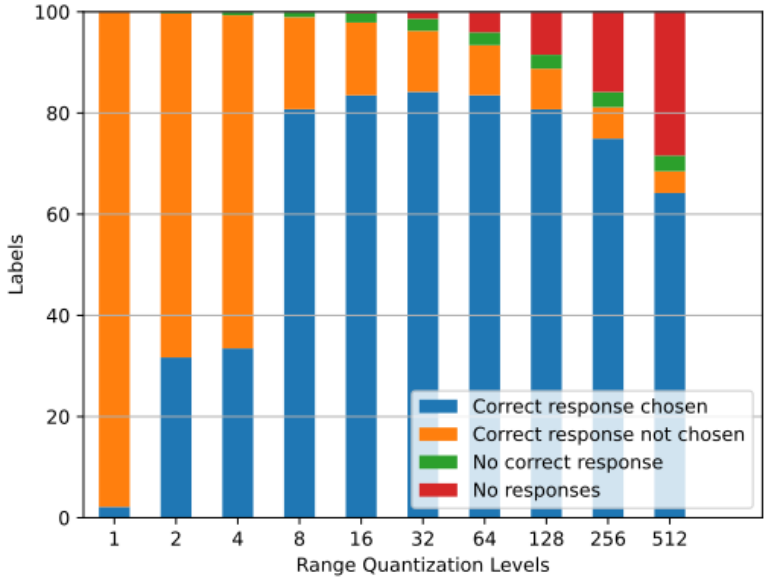


Figura 5.9: Gráfica que muestra los resultados hechos por las memorias a diferentes niveles de cuantización

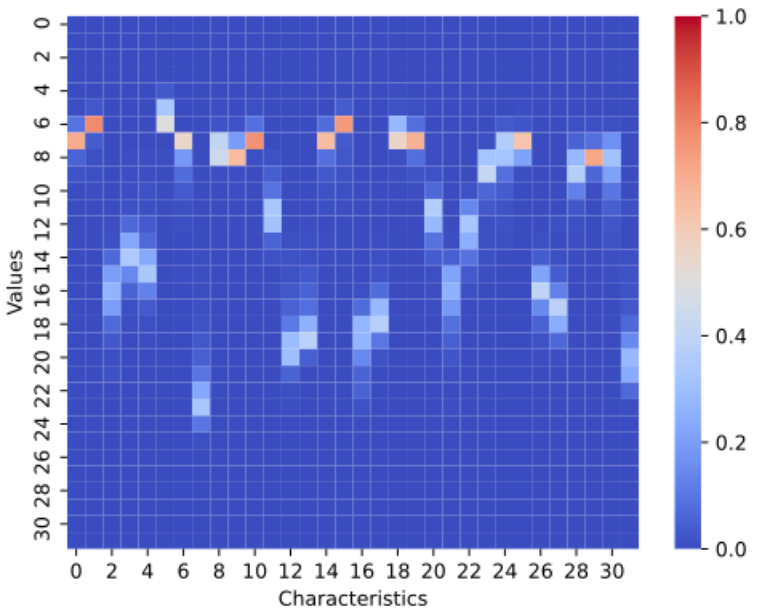


Figura 5.10: Composición de un AMR de 32×32

La figura 5.11 muestra que el mejor balance entre precisión y recuperación para AMR de tamaño 32×32 ocurre cuando el porcentaje del corpus recordado incluido en la memoria es 16 % donde se cruzan las gráficas correspondientes. A lo largo de la gráfica, la exactitud del AMR se mantiene con valores cercanos al 100 % y no es afectada por los comportamientos de las otras dos.

El desempeño del sistema como un todo, mostrado respectivamente en la figura 5.12, muestra que la precisión permanece alta cuando se usa la totalidad del corpus recordado, pero la recuperación es ligeramente menor en comparación a esta, aunque es su valor máximo, y los gráficos no se cruzan. Para evaluar el rendimiento del sistema también se debe considerar el costo del recurso de memoria que se duplica para el registro más grande.

Con el cálculo del peso se obtiene la similitud entre una cue y las representaciones almacenadas en una clase. Esto se realiza de forma directa gracias a la propiedad de distributividad de las representaciones. Entre *más pesado* sea una relación, mayor certeza se tiene que la abstracción del objeto buscado se encuentra en el AMR. La unión de ambos elementos permite una mejor selección de caracteres que en artículos pasados[14, 3]. Esto se observa en una composición más compacta de los AMR. Lo que se traduce en un ahorro en espacio de almacenamiento y la expresividad de las representaciones no es tan explícita, pero se mantiene su recuperación concreta. Los AMRs pueden contener la representación de más de una clase, con la única penalización de un aumento de la entropía, como se muestra en un trabajo anterior[5].

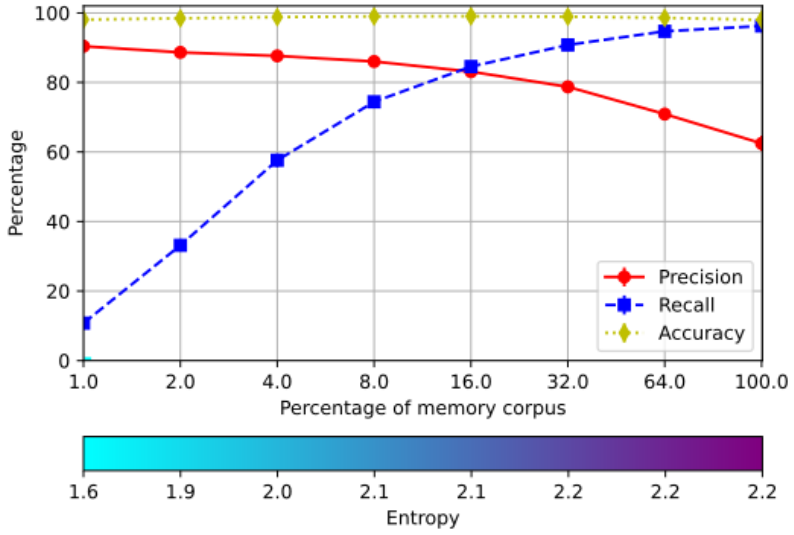


Figura 5.11: Compromiso entre precisión y recuperación para AMR de 32×32

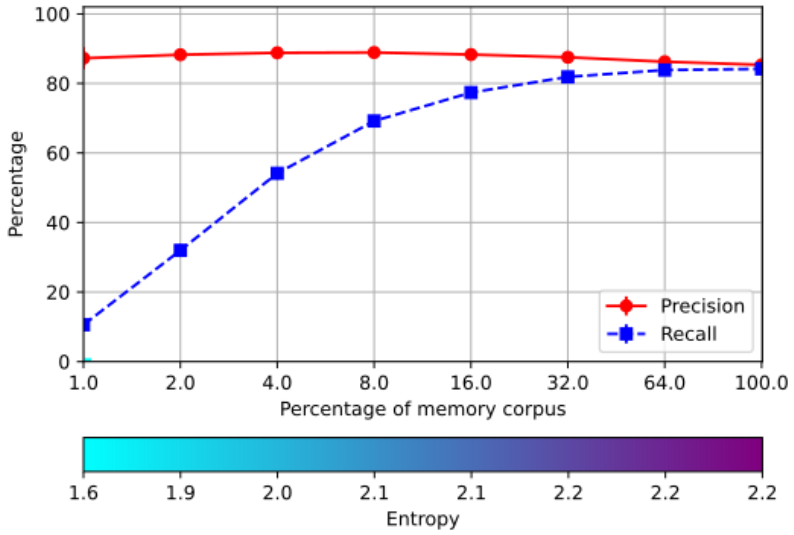


Figura 5.12: Compromiso entre precisión y recuperación para el sistema como un todo con un AMR de 32×32

Capítulo 6

Arquitectura de SRTM

En este capítulo se describen los bloques principales de la arquitectura del SRTM (figura 6.1). Estos son el pre-procesamiento, el procesamiento y el post-procesamiento. En la etapa de pre-procesamiento se procesa la imagen para identificar zonas con texto manuscrito, se recorta la imagen para aislar las palabras y se analizan en la etapa de procesamiento. En esta se realiza un barrido del extremo izquierdo al derecho para extraer sus letras. La etapa de post-procesamiento filtra su resultado y corrige errores ortográficos y gramaticales. En las siguientes secciones se explicará cada etapa, su funcionamiento, algoritmos y estrategias.

6.1. Pre-procesamiento

Se realiza el siguiente proceso a cada imagen:

1. Cambio a un `Gray-scale`: Se promedian los valores RGB de los píxeles por medio de una fórmula basada en su longitud de onda.

2. `Binarización` de la imagen: Se reducen los colores a blanco o negro. Se cambia cada píxel dependiendo de un umbral de valor, si es menor a 128, el color cambia a negro y, si es mayor, cambia a blanco.

3. `Ubicación` de palabras: Se utiliza el método Otsu[49] para obtener los contornos rectangulares en los cuales se encierran a las palabras.

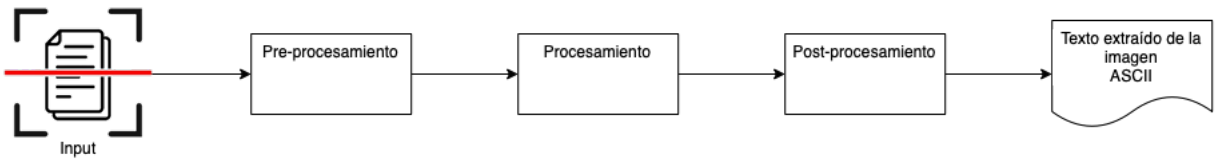


Figura 6.1: Diagrama de la arquitectura del sistema OHCR a alto nivel.

4. `Filtración` de ruido.

5. `Recorte` de contornos rectangulares: Se aíslan las palabras y se generan múltiples chops.

6. `Redimensión` de los chops: Las dimensiones de cada una de las imágenes se transforman a 32x32 píxeles.

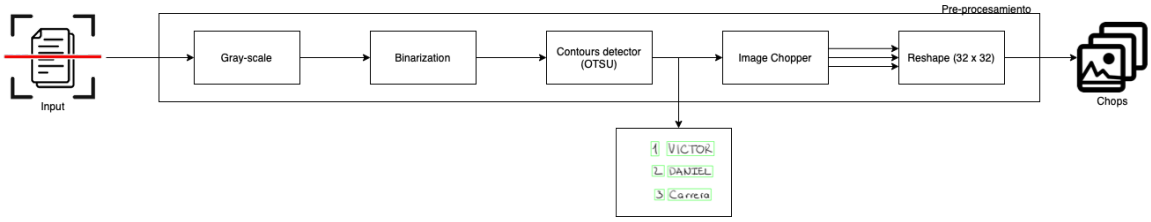


Figura 6.2: Diagrama del flujo correspondiente a la etapa de Pre-procesamiento sobre la figura 6.1.

En esta etapa, la imagen se divide en *chops*/recortes de las palabras encontradas. Para obtenerlos se inicializa una ventana de recorte con ancho configurable y con el mismo alto que la imagen. Se posiciona en el extremo izquierdo de la palabra y genera un chop. Se mueve δ píxeles hasta llegar al extremo derecho.

6.2. Procesamiento

En esta etapa se procesa cada chop/imagen de una palabra hasta transformarla en una cadena de texto (figura 6.3). Se inicializan dos tipos de reconocedores: la MAE y un clasificador/decodificador. El número de instancias por clasificador es configurable con un mínimo de uno y máximo de diez. Los pasos necesarios son:

1. `Entrenamiento`: Este paso se realiza una sola vez. Se crea un *encoder*, el cual transforma el dataset (EMNIST-47) de imágenes a su representación binaria (features). Los features a su vez son utilizados para llenar las MAE y los decodificadores[3].

2. `Extracción` de features: Una vez que se entrena el decodificador, se utiliza el decoder para obtener las features de todos los recortes a identificar.

3. `Reconocimiento` de features: El flujo se bifurca por los dos reconocedores que las asocian con un símbolo o un rechazo.

   a) MAE: Se introduce un arreglo de features en las MAE. Se regresa un arreglo del mismo tamaño en donde por cada feature se obtiene: si la memoria lo reconoció, un caracter de EMNIST-47 o, en otro caso, `null`.

   - `Desenfocador`: Por medio de procesamiento de imágenes se indica si un chop contiene una letra centrada, con un trazo continuo y sin tocar sus bordes o si es una transición de una palabra.

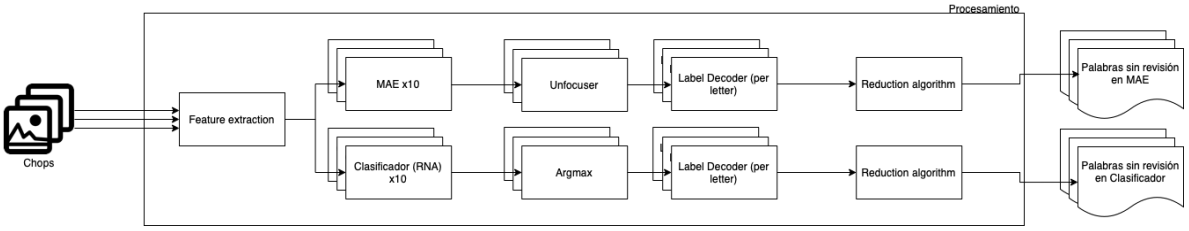


Figura 6.3: Diagrama del flujo correspondiente a la etapa de Procesamiento sobre la figura 6.1.

b) Clasificadores: Se introduce un arreglo de features y el clasificador transforma cada una en un caracter de EMNIST-47. Siempre se devuelve el más parecido. De igual forma, se regresa un arreglo con el mismo número de features.

- `Argmax`: Cada elemento que devuelve es otro arreglo de probabilidades con un tamaño igual al de EMNIST-47, similar a *one-hot*. Se elige el índice con la probabilidad máxima.

4. `Label Decoder`: Se traducen los índices de ambos arreglos hacia los caracteres relacionados a EMNIST-47. Se utiliza la misma numeración que aparece en la figura 1.7.

5. `Reducción`: Cada instancia de ambos clasificadores regresa su arreglo. Sin embargo, en ambos se sintetizan sus resultados por mayoría simple para tener un solo caracter por chop.

Las features de los chops se obtienen del codificador de MAE-P para igualar las condiciones del análisis por memoria y clasificador. Este componente transforma una imagen en blanco y negro hacia un arreglo o vector de tamaño igual al dominio $n = 32$ que contiene su representación normalizada y posteriormente se usará como entrada para la operación β y la RNA.

La estimación del caracter de un chop con MAE se basa en la instanciación de 47 memorias (el total de categorías utilizadas en el entrenamiento) y se llenan sus AMRs antes de realizar cualquier operación con el tamaño seleccionado en el capítulo 5, es decir, 32×32 y $\zeta = 0.25$. La información almacenada en las memorias se reduce a la medida de entropía para reconocer su nivel de incertidumbre sobre el total de representaciones guardadas en cada una.

Así, se puede llegar a conocer con antelación el nivel de precisión con el que se reconocerá un caracter y de cuál AMR provendrá. Es decir, entre mayor entropía exista sobre varios registros, mayor es el total de información almacenada y menor certeza se tiene que una feature exista en un solo registro, ya que varios pueden responder a la misma cue con el mismo número de celdas reconocidas por la operación η. Sin embargo, esto se contrarresta con su valor total de peso, porque se mide el nivel de similitud con la información guardada en un AMR. La categoría resultante es aquella que tenga la menor entropía y el máximo peso. En caso de que la operación η sea `false` ante una chop, entonces automáticamente el sistema la rechaza.

La recuperación de caracteres en los chops realizado por MAE depende de su operación β y del desenfocador basado en procesamiento de imágenes para identificar las transiciones entre caracteres. Los chops que

se deben reconocer son aquellos que contienen una letra centrada en sus dimensiones y sin tocar sus bordes, además que sea un trazo verticalmente continuo, es decir, que no sea una transición entre letras. No obstante, la caligrafía del ser humano puede generar contra-ejemplos. En la figura 6.4 se observa una transición y se puede confundir con una 'J', cuando en realidad es una imagen que captura una parte de las letras 'N' y 'T'.

El componente indica con una bandera booleana si un chop se trata de una transición o no. El proceso de distinción (algoritmo B.3) es el siguiente:

1. La matriz del chop se transforma en un arreglo con la suma de todas sus columnas
2. Se verifica que el arreglo no cumpla con la condición de tener un vacío en el medio de la imagen, tal que
   - Se define un subarreglo alrededor del elemento central del arreglo con el 20 % de su largo por la izquierda y la derecha.
   - Si la suma de los elementos del subarreglo es igual a cero, entonces la condición es `true`. De lo contrario, es `false`
3. Se verifica que el arreglo no cumpla con la condición de ser una transición, tal que
   a) Se cuenta el número de trazos en el arreglo
   b) Se obtienen sus índices en donde inician y terminan
   c) Si existe un subarreglo con dos o más ceros contiguos, entonces la condición es `true`. De lo contrario, es `false`.
4. Se aplica el operador lógico OR sobre ambas condiciones (pasos 2 y 3) y se regresa su resultado.

La segunda condición considera una tolerancia de un píxel por chop para considerarse como `true`, ya que la caligrafía del ser humano puede generar caracteres con trazos mínimamente espaciados para caracteres que son de más de un trazo y requieren separar el lápiz. En caso de que el desenfocador genere `true`, pero MAE obtenga un caracter, entonces se rechaza su resultado. De esta forma, se reduce aún más el número de las hipótesis falsas de la lectura de los chops.

La estimación del caracter de un chop con RNA se basa en instanciar el clasificador y cargar las features. Posteriormente, se ejecuta su operación de *predicción*. Por último, se obtiene el índice que mejor coincidió con los datos que guardo en el entrenamiento.

Al obtener las múltiples hipótesis de todos los chops por todas las instancias de ambos reconocedores, se utiliza un algoritmo de reducción para sintetizar a una sola pareja por imagen (algoritmo B.4). Se contabilizan los resultados de cada instancia, similar a la emisión de un voto. Se conserva el caracter más votado, también conocido como votación por mayoría simple.

El resultado final es un par de arreglos que contiene hipótesis de la palabra que se proceso entre ambos clasificadores. Estas pueden ser verdaderas o falsas.

Figura 6.4: Chop de transición para la palabra “TWENTIETH” entre las letras ‘N’ y ‘T’.

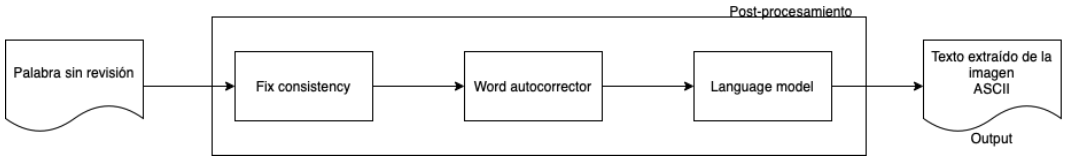


Figura 6.5: Diagrama del flujo correspondiente a la etapa de Post-procesamiento sobre la figura 6.1.

6.3. Postprocesamiento

Se analiza en parejas cada elemento de los arreglos de los clasificadores para filtrar las hipótesis falsas que generan ruido en la extracción (figura 6.5). Sea una palabra w_i una construcción de diferentes hipótesis h_j^i con $0 \leq j < |w_i|$ y $\prod$ una función de concatenación de caracteres:

$$w_i = \prod_j h_j^i$$

Los pasos que se realizan en esta etapa son:

1. `Fix consistency`: Se sintetizan las hipótesis de los clasificadores de una palabra. Si la de MAE es nula, entonces se descarta su pareja de RNA. De lo contrario, se conserva aquella que tenga la máxima mayoría simple en un buffer temporal.

2. `Word autocorrector`: Se arreglan errores ortográficos por medio de un diccionario del idioma inglés y se analiza cada palabra por dos estrategias. En caso de que la nueva palabra cambie radicalmente a la original, entonces se preserva.

   - Teorema de Bayes: se basa en la solución de Norvig[50] que realiza cambios basado en el caracter que maximice el teorema de Bayes (ecuación 6.1). Esto es similar a un modelo de lenguaje con un mono-grama basado en caracteres. El diccionario que usa es big.txt con 32,192 palabras.
   - Distancia de Levenshtein: Se utiliza la distancia de Levenshtain para ordenarlas de menor a mayor, respecto a la palabra original. Se regresa la que obtenga el mínimo valor. El diccionario que usa es english-words con más de 466 mil palabras.

3. `Language model`: Se arreglan errores gramaticales por medio de un modelo de lenguaje.

$$\begin{aligned} edit = argmax_{c \in candidates} P(c \mid w) = argmax_{c \in candidates} P(c)P(w \mid c)/P(w) \\ \Rightarrow edit = argmax_{c \in candidates} P(c)P(w \mid c) \end{aligned} \tag{6.1}$$

En `Fix consistency` existen chops contiguos que se rechazan o se analizan. El buffer temporal que se genera con estos últimos se filtra a un solo caracter para formar una nueva palabra. El contenido del buffer se asocia al comportamiento de una campana gaussiana, ya que en los extremos se encuentran caracteres similares al original y con un bajo porcentaje, mientras que en el medio se tiene el caso contrario.

El sistema reconoce los caracteres repetidos de forma contigua por medio de su conteo (algoritmo B.5) y los resume a uno solo. En caso de existir un empate, se elige el que obtuvo el máximo promediado de la etapa anterior. Contar el total de caracteres repetidos en el buffer se considera como una variable aleatoria distribuida normalmente y, por ende, se calcula su media μ_c y su desviación estándar σ_c. Si existe un número mayor a un umbral mayor a $\mu_c + \sigma_c$, el sistema rompe la subcadena en dos y se obtiene su repetición. Los *outliers* de esta variable se generan por la similitud de los chops entre dos letras iguales, ya que en los resultados de MAE se considera que simplemente la transición del mismo caracter se extendió.

Las letras pueden mezclarse con los números dada su gran similitud. Por ejemplo, el número '1' con la letra 'i' o 'l'. Este mismo bloque analiza su contexto y, en caso de que exista un mayor número de letras en la palabra, todos los números se cambian por una similar y viceversa. En caso de tener una relación[1], se analizan los caracteres vecinos para evitar una tripleta de consonantes o vocales.

6.3.1. Distancia de Levenshtein

Sea $d : w \times w \to N$ una función que toma como argumento dos palabras w_1 y w_2 y representa la distancia de Levenshtein[51]. Se calcula el número de inserciones, eliminaciones y modificaciones para que w_2 sea idéntico a w_1. Entre mayor sea el resultado, ambas son muy diferentes entre sí. Si las dos son idénticas, entonces $d(w_1, w_2) = 0$.

Por ejemplo, la palabra "llanla" tiene un error. Al pasar por este bloque, el resultado es "*llanta*". La distancia d("llanla", "llanta")=1 es la mínima. Solamente se cambia la quinta letra por 'l', no existen inserciones o eliminaciones. En caso de existir un empate, se regresa la primera ordenada alfabéticamente.

El código se basa en un algoritmo de programación dinámica para optimizar tiempo y espacio. El algoritmo B.1 tiene una complejidad $O(|w_1| * |w_2|)$ en tiempo y espacio. Al calcular la mejor palabra entre ambos autocorrectores sobre `Word autocorrector`, se elige la que tenga la menor distancia de Levenshtein. Dado que se utiliza un diccionario del idioma inglés, una palabra con otro origen puede cambiar radicalmente. Para evitar eso, se calcula el CER entre la del bloque anterior y la corregida y, si su valor es mayor a 0.5, se conserva la primera, sino se regresa la segunda.

6.3.2. Modelo de lenguaje

El modelo de lenguaje o *n-grama*[52] en palabras identifica y corrige el sentido gramatical de una oración. Específicamente, se usa un *tri*-grama. Se busca una función $ml : w \times w \times w \to N \mid ml(w_1, w_2, w_3) = P(w_3 \mid$

[1] Una correspondencia de más de un caracter entre letras o números

$[w_1, w_2]$). Por medio de la Regla de la Cadena de la probabilidad, se evita comparar todas las palabras anteriores a w_3 y solo las $n-1$ anteriores. Se utilizan dos datasets para ampliar el reconocimiento de tuplas:

- *Reuters*: Contiene diferentes títulos de artículos periodísticos y son de alto nivel gramatical
- *Punkt*: Contiene diversas palabras obtenidas de diversas fuentes y formatos de texto

Se selecciona un diccionario como estructura de datos para almacenar la tripleta de palabras. La pareja (w_1, w_2) como llave y w_3 como valor principal.

Al tener una oración con más de dos palabras, se analiza cada tripleta que se puede generar dentro de la oración. Se asegura que todas las palabras sigan el mismo contexto de la oración. En caso de que la tupla no exista en el modelo de lenguaje, el sistema vuelve a calcular la distancia de Levenshtein para el siguiente mejor resultado, máximo se repite hasta la décima mejor. De lo contrario, se deja la palabra inicial.

El resultado final del SRTM que se obtiene es el texto extraído de una imagen con el mayor sentido ortográfico y gramatical. El formato resultante es en ASCII.

6.4. Estándares utilizados

Esta sección contiene diferentes elementos que son los estándares utilizados en esta tesis. Hay dos áreas que se pueden describir, el primero es el de código y el segundo el de las partes que usa cada bloque, tanto de entrada, como salida.

Para el primer punto, el sistema se desarrollo con base en los lineamientos de Clean Code[53] y se describe su estructura general, además de mencionar la particularidad del nombramiento de las funciones y variables usadas, solamente las más importantes y con el *scope* más alto de todo el proyecto. Además, se implementa el paradigma Orientado a Objetos para facilitar la lectura, el flujo y reutilización del código.

El código utiliza interfaces y herencia para abstraer la funcionalidad de un mismo objeto, con especificaciones muy diferentes. Por ejemplo, en lugar de usar OTSU, se pudo haber usado EAST o Tesseract y cualquiera de estas opciones hacen la misma función: detectar el texto en la imagen. El patrón de diseño es el *factory method*[54]. Por ende, el bloque que mande llamar esta funcionalidad no debe enfocarse en implementar una más que otra, simplemente se invoca su mismo funcionamiento básico. Además a lo largo del código, se establecieron funciones que hacen una y exclusivamente una responsabilidad, para evitar la fragilidad o rigidez (o ambas), como se establece en SOLID. De esta forma, el proyecto está abierto a recibir modificaciones en sus módulos o expandirse a nuevas características.

Para el segundo punto, se comenta acerca del formato estándar de las imágenes y sus colores que se introducen al OCR. Además, dentro de la etapa de post-procesamiento, es necesario mencionar algunas de las reglas gramaticales que se implementaron y que permiten un resultado más preciso que el de la etapa anterior.

Como se mencionó en la sección 1.4, una imagen JPG o PNG se pueden ver como matrices que contienen una tupla con tres o cuatro números. Ambos, corresponden a su propio estándar, ya que tienen su conjunto de reglas y elementos para poder interpretar correctamente los valores binarios que se almacena en cualquier extensión. En general, las celdas que contienen se les llama píxel y guardan otro estándar llamado RGB (Red, Green, Blue). Si al crear una imagen en algún programa de edición de imágenes se elige JPG, este estándar calcula el valor de los colores de ciertos píxeles en función de los que están a su alrededor, lo cual permite comprimir la imagen original. Mientras que si la elección es PNG, entonces se tiene mayor flexibilidad que el anterior porque permite guardar más información por cada píxel, como su nivel de transparencia. Además de que este formato soporta colores en 8 y 24 bits.

De igual forma, otro estándar que se usa desde la base de la implementación de la computadora es el American Standard Code for Information Interchange (ASCII). Este permite formalizar cierta combinación de números binarios para representar algún símbolo en pantalla o en alguno de los documentos que puede guardar una computadora. En sus inicios, cada combinación estaba constituida por 7 bits, sin embargo se ha ido modificando a lo largo de los años, hasta poder usar 8 o 16 bits. En el OHCR, esto se usa desde el codebase, hasta el entrenamiento de las MAE y el RNA utilizados, ya que dado un *cue* se puede mapear hacia un caracter ASCII basado en EMNIST-47.

Por último, dentro de los estándares que se usa en esta tesis es el idioma con el que se post-procesan las palabras. Para realizar los benchmarks del siguiente capítulo se eligió inglés. Por lo que el dataset del autocorrecto y del modelo de lenguaje están basados en este mismo idioma. Sin embargo, se puede expandir a otros idiomas que usen un alfabeto latino, al cambiar las palabras en los datasets. Si las imágenes que se introducen contuvieran símbolos en otro tipo de alfabeto, entonces se debería de re-entrenar el sistema para reconozca este alfabeto y encontrar nuevos datasets con palabras y enunciados, ya que solo la última etapa es la que toma en cuenta las reglas ortográficas y gramaticales.

Capítulo 7

Pruebas sobre texto caligráfico y sus resultados

Se presentan los resultados sobre MAE utilizando la conversión de las palabras y enunciados IAM. Los parámetros que se utilizaron para correr los experimentos se encuentra en el algoritmo B.6. Se utilizaron los 10 folds del capítulo 5 para validar los valores obtenidos.

El total de elementos en cada uno de los datasets son mayores a los 10 mil. Se calcula una muestra representativa (ecuación 7.1) mucho más pequeña y con resultados muy cercanos. Se utiliza un 95 % de nivel de confianza que representa $z = 1.96$ y un error $e = 1\,\%$. Además se establece una probabilidad base de $p = 0.50$. La variable N representa las muestras totales de cada dataset. Este valor es usual cuando no se tiene una probabilidad de éxito p previamente calculada.

$$n = \frac{\frac{z^2 p(1-p)}{e^2}}{1 + (\frac{z^2 p(1-p)}{e^2 N})} \tag{7.1}$$

Los experimentos se realizaron con los mismos 10 folds que se crearon después del entrenamiento de MAE. En cada uno se analizan el mismo subconjunto del dataset IAM. El total del elementos y los promedios de ambas medidas se encuentran en la tabla 7.1. Se sigue una serie de pasos para estimar la precisión del sistema.

1. Se eligen las palabras o enunciados aleatoriamente y se traducen.
2. Se traducen (sección 4.1).
3. Se analiza el resultado de cada imagen con un fold de MAE.
4. Se calcula el CER o WER (sección 4.2).
5. Se repite el paso (3) hasta realizarlo con los diez folds.
6. Se obtiene el promedio de ambas métricas de todos los folds

	Promedio	Figura	Muestra total	Muestra representativa
Palabras IAM - CER	14.31 %	7.1	115,320	8,866
Enunciados IAM - WER	52.75 %	7.4	13,353	5,587

Cuadro 7.1: Tabla con el resumen de cada benchmark realizado utilizando pesos

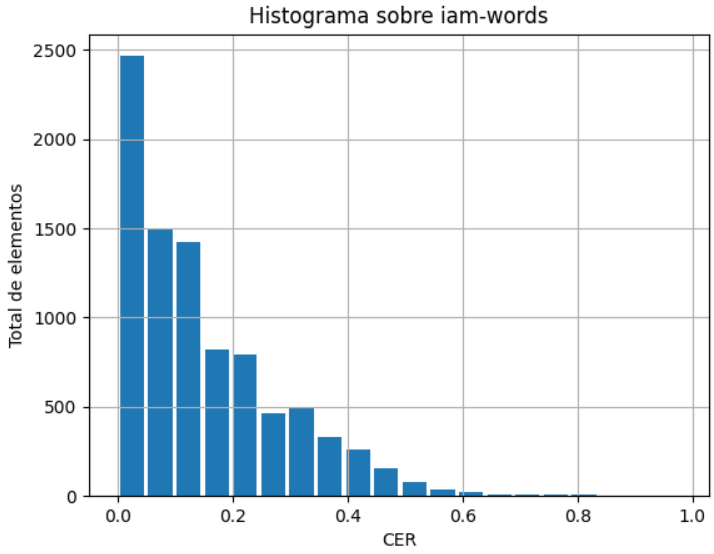


Figura 7.1: Histograma de los resultados de CER realizados sobre el dataset de palabras de IAM usando pesos

Se realizaron las pruebas sobre el corpus de palabras y el histograma promedio de los 10 folds (figura 7.1) muestra una concentración de resultados sobre 0.14 %. La mayoría de los resultados de MAE-P comparados con los valores originales son similares. Alrededor de 2,500 (28 %) palabras se obtuvieron de forma exacta. Mientras que el rango entre 0 y 20 % de CER engloba el 69 % del corpus total. Los valores obtenidos discreparon en hasta dos letras añadidas, eliminadas o sustituidas. El comportamiento de la gráfica es similar a una distribución geométrica.

Se seleccionan aleatoriamente las palabras del corpus. La figura 7.2 muestra que desde el origen de los datos, el 14.86 % se formaron con diferencias ortográficas con respecto a la imagen original de IAM, mientras que el 85.14 % restante se crearon correctamente. Entre ambos grupos, los resultados (figura 7.3) son similares: el sistema reconoció el 82.98 % con un CER menor o igual al 20 % de palabras con errores y el 85.92 % de palabras sin errores.

En el histograma del cálculo de WER de los enunciados (figura 7.4) se muestra un comportamiento similar al de una distribución Gaussiana con una media igual al 52 %. Este valor es cuatro veces más grande que el del CER. Más de la mitad de las palabras no coincidieron con su valor original y se desconoce la distancia de Levenshtein o CER. Sin embargo, con el resumen de CER se estima que la probabilidad de generar una

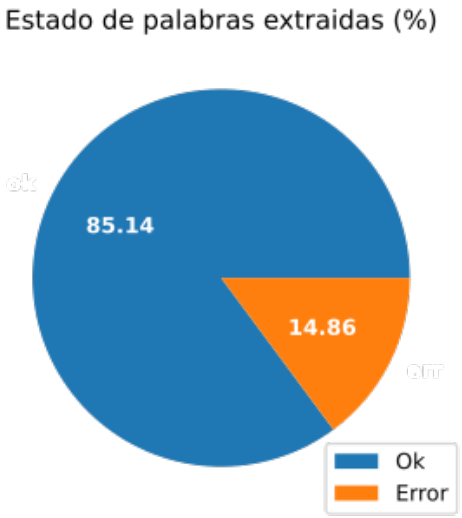


Figura 7.2: Porcentaje del tipo de palabras IAM utilizadas

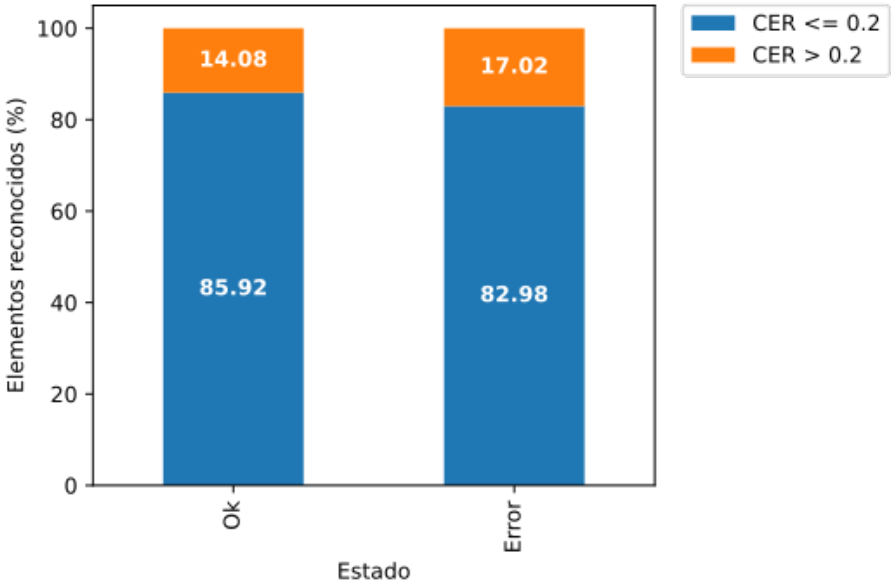


Figura 7.3: Porcentaje de elementos reconocidos por CER agrupados por tipo de palabras IAM

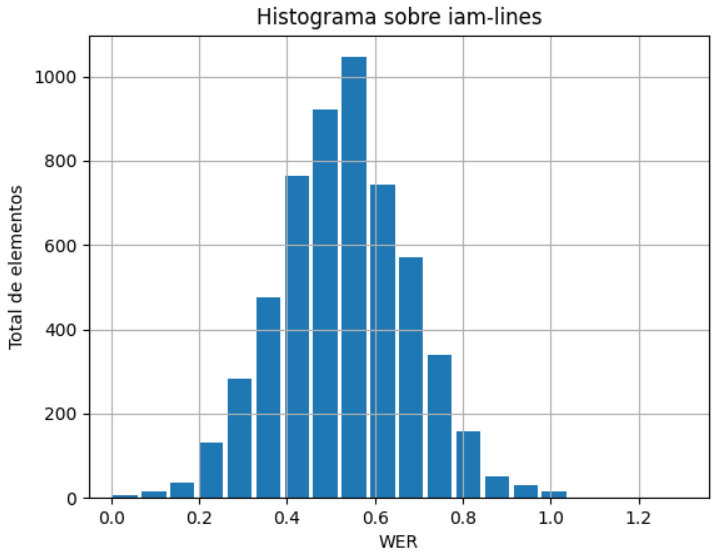


Figura 7.4: Histograma de los resultados de WER realizados sobre el dataset de líneas de IAM usando pesos

palabra con un error mayor al 20 % respecto a la original es alrededor del 15 %. Por lo tanto, el sentido lógico de la línea aún se comprende en gran medida.

Similar a la creación del corpus de las palabras IAM, se obtiene aleatoriamente un subconjunto de elementos sobre su categoría de líneas. La figura 7.5 muestra que desde el origen de los datos, el 12.77 % se formaron con diferencias ortográficas y sintácticas con respecto a la imagen original de IAM, mientras que el 87.23 % restante se crearon correctamente. Entre ambos grupos, los resultados (figura 7.6) son similares: el sistema reconoció el 91.43 % con un WER menor o igual al 20 % de palabras con errores y el 86.88 % de palabras sin errores.

7.1. Análisis de los resultados

Respecto a la medición de CER en la figura 7.1, se obtuvieron resultados que respaldan la efectividad de las MAE, ya que de 8,866 palabras analizadas, casi el 40 % se obtuvieron de forma correcta, es decir, con un CER igual a cero. Las palabras con CER mayor a 20 % y menor a 30 % ocupan un 21 % del total. El rechazo de MAE-P sobre los chops de una palabra permite que la etapa de procesamiento filtre varías de las hipótesis falsas y quedarse con la mayoría que son verdaderas. Dado que la diferencia entre dos chops contiguos es muy pequeña ($d = 0.15\,\%$), se pueden generar repeticiones sobre un mismo símbolo. Esto es útil, ya que aumenta la probabilidad de que alrededor de esos chops, la letra que se obtiene reiteradamente es la que se escribió.

Los casos en los que exisen letras contiguas repetidas se obtuvieron resultados satisfactorios y con mínimo error. Por ejemplo, la palabra “bigger” reitera la letra ‘g’ y su conteo total es igual a ocho, mientras que

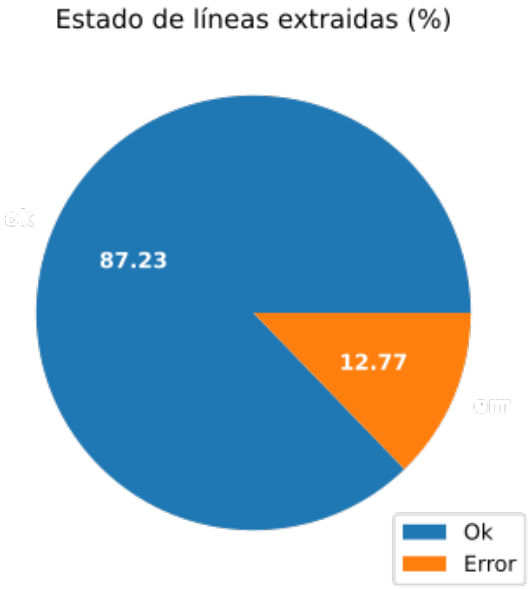


Figura 7.5: Porcentaje del tipo de líneas IAM utilizadas

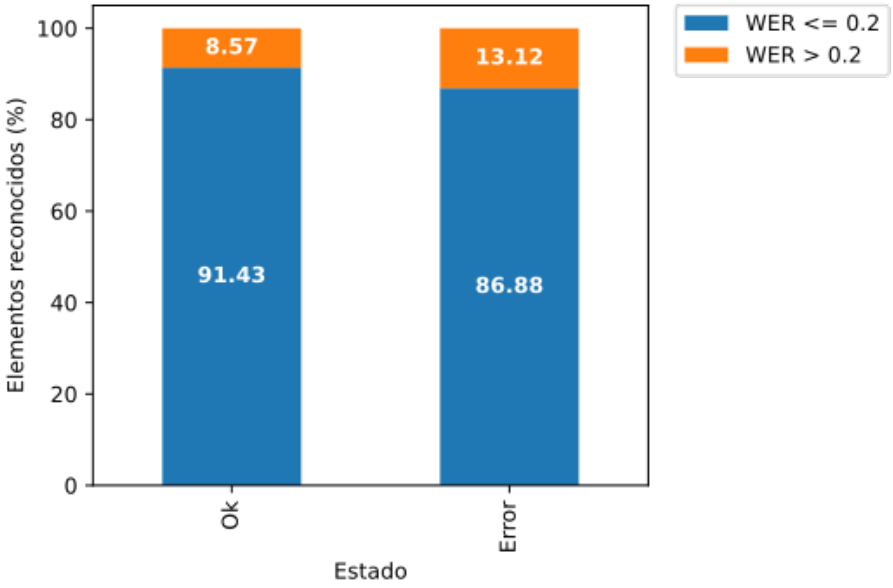


Figura 7.6: Porcentaje de elementos reconocidos por WER agrupados por tipo de líneas IAM

Figura 7.7: Palabra “3ting” del corpus de las palabras de IAM

Figura 7.8: Palabra “favourite” del corpus de las palabras de IAM

$\mu_c = 4$ y $\sigma_c = 2$. Por ende, se considera como outlier y el resultado fue una palabra con doble letra. Al correr la sub-etapa de autocorrección, se reconoció directamente en el diccionario que la palabra obtenida de la etapa de Procesamiento existe.

Al ejecutar la etapa de post-procesamiento, el sistema analiza la palabra sobre sus dos estrategias de corrección, pero si la diferencia entre la corrección y la generada por el sistema es demasiado grande, se conserva la original. De esta forma, varias de las palabras o nombres que tienen un origen etimológico diferente al inglés se conserva. Por ejemplo, “3ting” (figura 7.7) es una de las palabras del dataset IAM, pero no de los diccionarios que se usan en la autocorrección. Posterior al análisis, se mantiene la palabra de la etapa de procesamiento. Así, las memorias junto con el desenfocador son lo suficientemente útiles como para extraer la palabra de una imagen. Los casos en los que se existen inconsistencias son con letras que al ser recortadas y una parte de ellas está centrada son similares a otras, como la letra ‘R’. Asimismo, existen casos en los que el caracter se confunde con otro tipo, tal es el caso de “favourite” (figura 7.8), en el que la palabra de los reconocedores es “favdurite”. Una sola letra cambia, la ‘o’ por la ‘d’, ya que la palabra generada contiene una ‘D’ redondeada y visualmente se confunde con ‘O’.

El promedio total de CER de las palabras analizadas indican que los valores son similares a otros sistemas[55]. Sin embargo, la medición se realiza en el contexto de una traducción de palabras, sin necesariamente utilizar las originales. El corpus es una concatenación de elementos de un dataset atómico a IAM con una caligrafía distinta. Si bien, los resultados son semejantes, el CER se tendría que transformar con algún tipo de función abstracta para igualar sus condiciones.

La MAE-P aumenta los porcentajes de precisión y recuperación, comparado con MAE. El proceso intermedio de rechazo durante la etapa de procesamiento permite descartar transiciones con mayor facilidad y el filtrado del Postprocesamiento genera una palabra idéntica o muy similar a la original. En el peor caso, añade caracteres entre letras. Las imágenes que contienen números tienen un alto nivel de complejidad en la última etapa, ya que no existe un diccionario que los pueda rectificar y los valores obtenidos siguen siendo números. Varios de los resultados obtenidos generaron el número original, pero con caracteres repetidos, ya que no se pudieron descartar o mejorar.

Los resultados del corpus de líneas de IAM (figura 7.4) son una extensión del corpus de palabras. La etapa de pre-procesamiento recorta la imagen dependiendo de lo que se encuentre por el método de Otsu y las

	Entrada		*Salida*		
#	**Original**	**Imagen**	**Procesamiento**	**PostProcesamiento**	**CER**
1	acquaintance		ACGZALNTANEE	acquaintance	0.0
2	connivance		CONNIJANCE	connivance	0.0
3	rivermead		RIRERNEAD	riverhead	0.11
4	legislation		LEGISLATIDV	legislativ	0.18
5	confirmed		CONFJRNED	concerned	0.33

Cuadro 7.2: Tabla que muestra el proceso del sistema para ciertas palabras guardadas en el dataset IAM, con sus respectivos valores de salida provenientes del sistema OCR

nuevas se almacenan en otro buffer. Posteriormente, se procesan de forma individual para extraer su texto. A diferencia del benchmark de las palabras, el sistema analiza el contexto de las palabras por el modelo de lenguaje. Si no se encuentra una relación entre una palabra y su pareja contigua a la izquierda, se mantiene la misma y se prosigue con las restantes.

El modelo de lenguaje cambia las palabras en caso de que su aprendizaje en la etapa de entrenamiento almacene relaciones suficientemente extensas, de lo contrario, los cambios serán nulos. Por ejemplo, la tercera oración de la tabla 7.4 contiene una palabra que no existe en ninguno de los datasets de oraciones y, mucho menos, se le ha relacionado con otras: "czechoslovak" con "the" o la pareja ("kitchen", "again"). Al utilizar el modelo de lenguaje, se regresa la misma, ya que se prioriza el resultado obtenido de MAE y, dado que el porcentaje de error sobre una palabra es del 14 %, su resultado coincide con la esperada. Por otro lado, en la segunda oración de la tabla 7.4, se corrige la tercera palabra generada en la etapa de Procesamiento ("n1j01ng") por "muddling". Dado su pareja anterior, se analiza otra palabra por medio del modelo del lenguaje porque existe una relación entre ambos elementos. Sin embargo, su distancia de Levenshtein es mayor comparado con la generada por los reconocedores del Procesamiento.

Como es de esperarse, el histograma y el promedio de WER es 3.7 veces mayor que CER, porque las palabras se comparan en su totalidad y no por sus partes con respecto a las esperadas. Aún cuando se detecte un WER mayor a cero, no se ejecuta un sub-análisis con CER. Es decir, si al diferencia entre dos palabras es al menos un solo caracter, la medición las consideras como incorrectas y aumenta su tasa de error. El WER promedio es 52 %. En su comparación contra otros sistemas[55], se observa que sus niveles son mayores, aunque se tendría que usar la misma función abstracta del benchmark anterior para traducirlos.

La tabla 7.2 muestra un resumen de los datos de entrada que recibió el sistema sobre las columnas enmarcadas con "Entrada" y lo mismo para "Salida". Como se puede observar, las imágenes se hicieron a partir de la palabra original (primer columna de izquierda a derecha con letras mayúsculas), respecto a lo que se tiene en el dataset IAM. Cada una de las letras de la palabra se formó por medio de EMNIST. Los primeros dos resultados de la tabla muestra una extracción exacta (los resultados en minúsculas) respecto a la original. Esto implica que las memorias para el único fold encontraron las letras correctas en todo el

	Entrada	
#	**Original**	**Imagen**
1	great chance that should be grasped	
2	the czechoslovak kitchen again is	
3	fatherinlaw after muddling up some sheep	
4	difficult though that would be she became	
5	small back yard newly dec near shops and tube	

Cuadro 7.3: Tabla que muestra la entrada del sistema al analizar el dataset de líneas de IAM

barrido "visual" que hizo el sistema, además de rechazar las transiciones entre letras. Después de hacer el filtrado de la serie de todo el barrido, otro componente que ayudo a encontrar la palabra exacta fue el autocorrector, ya que aproximo lo encontrado a la etapa de procesamiento a una del diccionario de habla inglesa.

En las dos tablas se muestran imágenes con letras mayúsculas y minúsculas según EMNIST-47. El SRTM reconoce ambas representaciones hacia el mismo caracter registrado en MAE. En condiciones ideales, la *percepción* del sistema también capta su diferencia. Esto reafirma su propiedad de heteroasociatividad[3].

	Entrada	***Salida***		
#	**Original**	**Procesamiento**	**PostProcesamiento**	**WER**
1	great chance that should be grasped	great chance that shoulid be grasped	great chance that should be grasped	0.0
2	the czechoslovak kitchen again is	the czechoslovak kitchien again is	the czechoslovak kitchen again is	0.0
3	fatherinlaw after muddling up some sheep	atherinidae after n1j01ng up some sheep	atherinidae after muddling up some sheep	0.16
4	difficult though that would be she became	difficult though that would de she becane	difficult though that nould de she became	0.28
5	small back yard newly dec near shops and tube	nhili rack yard fedia bec wear shaps ana tube	mail rack yard fedia bec wear shaps ana tube	0.77

Cuadro 7.4: Tabla que muestra el proceso del sistema para ciertas líneas guardadas en el dataset líneas de IAM, con sus respectivos valores de salida provenientes del sistema OCR

Capítulo 8

Conclusiones generales sobre SRTM y MAE

El sistema se compone de tres etapas: Preprocesamiento, Procesamiento y Postprocesamiento. Se ingresa una imagen con texto manuscrito para obtener su representación en ASCII. El primero permite obtener chops estandarizadas provenientes de las palabras encontradas por el método de Otsu. En el paso intermedio se implementa cualquier tecnología que reconozca sus características a un caracter específico, por lo general se utiliza alguna RNA. Específicamente, se usan las MAE-P como traductor *chop-a-caracter* por medio de sus tres operaciones. Si la imagen se reconoce por alguno de sus registros, el resultado consta de la recuperación del objeto abstraído en formato ASCII con el máximo peso y la mínima entropía, de lo contrario, se regresa un valor nulo.

La concatenación de las traducciones de los chops genera un buffer con hipótesis verdaderas y falsas respecto a una palabra recortada de la imagen original. En comparación con las tecnologías de SRTM del estado del arte[55], el rechazo de MAE ante ciertas cues establece límites entre las transiciones de una letra a otra, similar a la separación de elementos en documentos (como CSV o TSV) y estructuras de datos (como los índices de los arreglos o diccionarios). Para fomentar el rechazo entre transiciones, se aprovecha la ausencia de color en un chop para identificar sus patrones por medio del desenfocador. Sin embargo, se genera ruido o caracteres diferentes a los esperados en la frontera de cada letra manuscrita. Por lo que, se realiza un resumen de todas las hipótesis no nulas y contiguas entre sí para simplificar el buffer y obtener una sola palabra. Una de las principales estrategias es la votación por mayoría simple, en el que el valor resultante es aquel que tiene el máximo número de repeticiones.

Por último, se analiza la palabra y, en caso de que existan errores ortográficos, se corrige a otra almacenada en dos diccionarios y con su mínima distancia de Leveshtein. No obstante, la estrategia del autocorrector permite conservar la palabra del Procesamiento, si y solo si su distancia sobrepasa un umbral pre-establecido, con el fin de mantener palabras con un origen etimológico diferente al inglés.

En el capítulo de Resultados se destacan las ventajas de esta arquitectura. Se muestra en los benchmarks que se puede sustituir el core usual de un OCR por las memorias. En el caso de la extracción de palabras, se obtuvo un CER del 14 %, mientras que para las líneas, un WER del 52 %. Aún cuando más del 10 % de los elementos seleccionados en ambas pruebas fueron traducidos con errores. Un SRTM con RNN, memorias a corto-largo plazo y dropout[56, 55] obtuvo un CER similar (13.92 %), pero un WER inferior (31.48 %). Esto se debe a la formación de palabras concatenadas y su similitud con las originales. En el caso de MAE-P, varias de las palabras que no fueron exactas a las medidas con WER son muy similares y con un CER menor al 20 %. Sin embargo, por la rigidez de la primera, su porcentaje de error aumenta.

Cabe aclarar que la traducción *IAM-EMNIST* distorsiona los resultados de ambas métricas, por lo que se requiere de alguna función abstracta que los convierta. Esta transformación se realizó para obtener un mismo tipo de caligrafía entre todas las pruebas, además de que el entrenamiento se hizo con ese mismo dataset, EMNIST-47. Dado que la variedad caligráfica es demasiado grande, si se omitía este paso antes de iniciar los benchmarks, el SRTM no reconocería varias de las letras.

Se utiliza una tecnología innovadora y con la suficiente flexibilidad de recuperar más elementos que los que puede aprender una RNA, gracias a la operación de β de MAE. Además, se implementa un SRTM con una estrategia similar (mas no cercana) a la que se emplea en el ser humano para leer de forma visual.

Hay diferentes aspectos del SRTM que se pueden mejorar y se dejan como *trabajo futuro* en cada una de sus etapas. La traducción entre datasets es la principal limitante para poder comparar el sistema con otras tecnologías en igualdad de circunstancias. Por lo que se puede adaptar un módulo que analice y extraiga individualmente las letras de cada palabra de cualquier dataset con palabras manuscritas y las filtre y almacene de forma similar a EMNIST. Antes de correr cualquier benchmark, se construyen los elementos con el corpus de caracteres originales, sin alterar su caligrafía. Para incrementar la capacidad de reconocimiento de MAE-P, los nuevos recortes de caracteres se retroalimentan en un dataset global y almacenado en disco.

Uno de los elementos que se descartaron es el uso de los símbolos no alfanuméricos, como signos de puntuación o exclamación. Estos permiten generar una pauta y administrar diferentes ideas a ciertos tiempos en la lectura, con el fin de propagar la idea de forma clara. Sin embargo, no se consideraron, ya que no se considera en el dataset EMNIST-47. Se pueden extraer con la nueva característica del párrafo anterior y se pueden realizar pruebas para verificar su funcionamiento con un mayor número de categorías que el dataset actual. Se tendría que adaptar la etapa de pre-procesamiento para reconocerlos e incluirlos en la palabra más cercana y la etapa de Procesamiento los analice como otro caracter más.

Como se especificó en el capítulo 5, la implementación de MAE-P omitió el uso de dos parámetros utilizados en una versión reciente de MAE[17]. Al realizar las adaptaciones necesarias, se espera tener un incremento natural en la capacidad de reconocimiento y recuperación de las categorías registradas en los AMRs. Los valores de los dos tipos de métricas en el SRTM deberían mejorar y acercarse a los valores de los estados del arte, ya que su tolerancia ante el peso de una cue y el peso promedio de un argumento a_i serían más específicos, rechazarían mejor las transiciones y filtrarían las hipótesis falsas. El resultado final después

de la etapa de Procesamiento sería una palabra mucho más similar a la contenida en una imagen.

El SRTM está limitado a procesar imágenes. Sin embargo, se puede implementar una lógica más extensa para generar un sistema que reconozca documentos en PDF o video. Se puede tomar una captura de pantalla directamente del vídeo o el documento y ejecutar todos los pasos de la arquitectura actual. El mayor reto para desarrollarlo es capturar la mejor imagen, el cual puede ser resuelto con visión por computadora.

Actualmente, existen implementaciones que son superiores a los resultados del modelo de lenguaje, pero su consumo de recursos es mucho mayor. El cambio permite perfeccionar el sentido gramatical de la línea obtenida por la etapa de Procesamiento y reducir su WER. Una de las alternativas es el uso de APIs (*Application Programming Interface* por sus siglas en inglés) de empresas dedicadas a la corrección de textos en tiempo real, como Ginger. Se envía el texto propuesto y su interfaz regresa una nueva con las palabras corregidas.

Asimismo, se puede escalar a utilizar una mezcla de tecnologías en las tres etapas para generar resultados más precisos. Por ejemplo, en la etapa de pre-procesamiento, se puede incluir Tesseract para identificar las ubicaciones de las palabras con diferentes fondos, tipos de tinta y caligrafía o tipografía. No obstante, los demás procesos se conservan y no se utiliza su componente nativo para incluir texto. Similarmente, se podría añadir un subproceso que reconozca símbolos de puntuación y discernir entre mayúsculas y minúsculas gracias a la heteroasociatividad del sistema.

Apéndice A

Especificación de software

Se utilizó Python como principal lenguaje de programación y Tensorflow como plataforma de aprendizaje automático. Se instalaron librerías especializadas en algoritmos sobre visión por computadora y procesamiento de imágenes, CER y WER, además de un modelo de lenguaje. El código se encuentra en Github. Las instrucciones para replicar el funcionamiento del sistema se encuentran en un archivo nombrado `README.md`.

El sistema corre por medio de una consola de comandos tipo `sh`. Se realiza un pase por argumentos para obtener resultados. Las especificaciones de cada uno se encuentra en `main.py` con la bandera `--help`. Si las dependencias especificadas en `requirements.txt` se instalaron correctamente, se despliega el banner del sistema (figura A.1).

El sistema y los benchmarks se ejecutaron sobre una máquina ubicada dentro del IIMAS, UNAM y es un modelo Alienware Aurora R5. Gracias a las GPUs que se tienen instalada en la máquina, las operaciones matriciales se realizaron de forma casi instantánea y mucho más rápido que en una computadora convencional. Las especificaciones más relevantes son:

- Sistema operativo: Linux 4.15
- RAM: 64 GiB
- Espacio en disco: 2 TB
- Dos GPUs: Nvidia GeForce GTX Titan-x y Nvidia GeForce GTX 1080

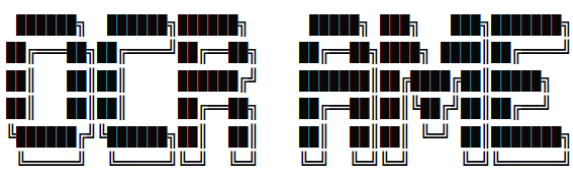


Figura A.1: Banner inicial del sistema al correr `main.py`.

- Python: 3.8

El `codebase` se programó sobre un entorno local. En lugar de utilizar los componentes nativos, se instaló una red de contenedores de Python especificados en un `Dockerfile`. Se emula las características básicas de la computadora principal y evitar problemas de compatibilidad en estructura e instalación de dependencias.

La manipulación de imágenes se realizó a través de OpenCV[1], el cual es un proyecto open-source. Se incluyen operaciones básicas, como recortar, redimensionar, cambiar las capas de colores o añadir objetos a una imagen. Por otro lado, la librería `NLTK`[2] incluye diferentes datasets para Natural Language Processing (NLP), como `reuters` o `punkt`, además la implementación directa de bigramas o trigramas.

Los tipos de imágenes más utilizados es JPG y PNG. Ambos son aceptados por el sistema y se implementan como una matriz de dos dimensiones. Sus celdas guardan una tripleta de números entre 0 y 255 que representan la composición de un color en RGB. Con PNG, se usa una tupla de cuatro elementos, ya que se añade un canal extra del nivel de transparencia entre cero y uno. En el capitulo 7 se usa PNG para mantener la fidelidad de la imagen.

El modelo de lenguaje se implementa con una librería de Python llamada TextBlob. Esta es una API especializada en Natural Language Processing (NLP) y tiene algoritmos de clasificación, traducción y corrección. Dentro de esta última, se almacena una función de corrección gramatical sobre enunciados. Se instancia un objeto `TextBlob` con la oración como cadena de texto y se llama a su método `correct`. Se regresa una nueva que cambia .

[1] Open Source Computer Vision Library
[2] Natural Language Toolkit

Apéndice B

Códigos

```python
def minimum_edit_distance(s1, s2):
    if len(s1) > len(s2):
        s1, s2 = s2, s1
    distances = range(len(s1) + 1)
    for index2, char2 in enumerate(s2):
        new_distances = [index2 + 1]
        for index1, char1 in enumerate(s1):
            if char1 == char2:
                new_distances.append(distances[index1])
            else:
                new_distances.append(1 + min((distances[index1],
                                              distances[index1 + 1],
                                              new_distances[-1])))
        distances = new_distances
    return distances[-1]
```

Algoritmo B.1: Algoritmo que calcula la distancia de Levenshtain entre dos palabras basado en RosettaCode

```python
class WordGenerator:
    def choose_random_index(self, label):
        logic_output = self.corpus_labels == label  # Boolean array
        indexes = np.where(logic_output)[0]  # Got a unique-element tuple
        index: int = np.random.choice(indexes)
        return index

    def get_character(self, c: str):
        label = get_index(c)
        index = self.choose_random_index(label)
        return self.corpus_images[index]

    def append_letter(self, image_array, c):
```

```
        img_c = self.get_character(c)
        if image_array is None:
            return img_c
        else:
            return np.concatenate((image_array, img_c), axis=1)
```

Algoritmo B.2: Clase que traduce las palabras de IAM en EMNIST-47

```
class Unfocuser:
    def select_unfocused(self, images):
        are_medium_empty = np.array([self.is_medium_empty(i) for i in images])
        are_transitioned = np.array([self.is_transitioned(i) for i in images])
        return are_medium_empty | are_transitioned

    def is_medium_empty(self, image, percentage=0.20):
        added_columns = image.sum(axis=0)
        middle = img_columns // 2
        low_bound = int(middle * (1 - percentage))
        high_bound = int(middle * (1 + percentage))
        return added_columns[low_bound:high_bound].sum() == 0

    def is_transitioned(self, image):
        added_columns = image.sum(axis=0)
        strokes_indexes = self.get_strokes_indexes(added_columns)
        return bool(len(strokes_indexes) > 1 and (
                self.is_previous_stroke_predominant(strokes_indexes[0], len(added_columns))
    or self.has_big_separation(
            strokes_indexes)))

    def get_strokes_indexes(self, added_columns):
        strokes_indexes = []
        start_counting = False
        for i, col in enumerate(added_columns):
            if col > 0 and not start_counting:
                min_index = i
                start_counting = True
            elif col == 0 and start_counting:
                max_index = i - 1
                start_counting = False
                strokes_indexes.append((min_index, max_index))
        if start_counting:
            max_index = len(added_columns) - 1
            strokes_indexes.append((min_index, max_index))
        return strokes_indexes

    def is_previous_stroke_predominant(self, first_stroke, total):
```

```
        return first_stroke[0] == 0 and first_stroke[1] - first_stroke[0] > 0.50 * total

    def has_big_separation(self, strokes_indexes):
        last_pixel_first_stroke = strokes_indexes[0][1]
        first_pixel_second_stroke = strokes_indexes[1][0]
        if first_pixel_second_stroke - last_pixel_first_stroke > LETTER_SEPARATION:
            return True
```

Algoritmo B.3: Algoritmo del componente de Desenfocador

```
def choose_symbols(letters_memory, letters_network):
    symbols_network = [utils.get_max_probable_symbol(l) for l in letters_network]
    symbols_memory = [utils.get_max_probable_symbol(l) for l in letters_memory]
    return symbols_network, symbols_memory

def get_max_probable_symbol(arr):
    arr = [a.upper() if a is not None else None for a in arr]
    counts = initialize_dictionary_count(arr)
    max_count = 0
    max_symbol = ""
    for symbol in arr:
        counts[symbol] = counts[symbol] + 1
        max_count = max(max_count, counts[symbol])
        if max_count == counts[symbol]:
            max_symbol = symbol
    probability = max_count / len(arr) * 100
    return max_symbol, probability
```

Algoritmo B.4: Algoritmo de reducción de reconocedores en la etapa de Procesamiento

```
def remove_nplets(word, n):
    if len(word) < n:
        return word
    reviewing = word[0]
    count = count_repeated_character(word, reviewing)
    j = count if count >= n else 1
    return reviewing + remove_nplets(word[j:], n)

def count_repeated_character(word, reviewing):
    count = 0
    while count < len(word) and word[count] == reviewing:
        count += 1
    return count
```

Algoritmo B.5: Algoritmo que genera una palabra consistente por un buffer de caracteres de la etapa de Procesamiento

```
python main.py
    -l info -d 0.15 --rectangle_width 0.85 -t 2
    --dataset iam --word_file test-objects.txt --specific_fold $i
```

Algoritmo B.6: Comando para configurar un benchmark con alguno de los 10 folds de MAE

- `--dataset`: el dataset a procesar
- `--word_file`: el archivo que incluye las palabras a procesar
- `--rectangle_width`: el tamaño de la ventana configurable
- `-d`: el movimiento de la ventana configurable
- `-t`: la tolerancia de MAE
- `--specific_fold`: el número de fold con el que se realiza el benchmark

Acrónimos

ASCII American Standard Code for Information Interchange. 10, 19, 21, 24, 46, 47, 57

CER Character Error Rate. 4–6, 25, 26, 45, 49, 51, 53, 54, 58, 60

HCR Handwritten Character Recognition. 5

MA Memoria Asociativa. 11, 14–16

MAE Memoria Asociativa Entrópica. 1, 5, 6, 10–15, 20, 21, 27, 41–45, 47, 48, 51, 53–55, 57, 58, 65

MAE-P Memoria Asociativa Entrópica con Pesos. 3, 27, 30, 32, 33, 42, 49, 51, 53, 57, 58

NLP Natural Language Processing. 61

OCR Reconocimiento Óptico de Caracteres. 5

OCR Optical Character Recognition. 1, 5, 6, 10, 13, 15–18, 21, 25, 54, 56, 58

OHCR Offline Handwritten Character Recognition. 4, 5, 20, 21, 40, 47

PCR Printed Character Recognition. 5

RNA Red Neuronal Artificial. 6, 11, 15, 32, 33, 42–44, 47, 57, 58

SRTM Sistema de Reconocimiento de Textos Manuscritos. 1, 5, 6, 10, 13, 14, 20, 21, 24, 32, 33, 40, 46, 55, 57–59

WER Word Error Rate. 4–6, 25, 26, 49, 51, 54, 58–60

Bibliografía

[1] M. Edmonds, *A brief history of Optical Character Recognition (OCR).* Pitney Bowes Business Manager, United Kingdom and Ireland, 2020. [Online]. Available: https://www.pitneybowes.com/content/dam/pitneybowes/uk/en/shipping-and-mailing/e-invoicing/Blog_E-invoicing-The_Brief_History_of_OCR.pdf

[2] R. G. Casey and E. Lecolinet, "A survey of methods and strategies in character segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 18, no. 7, pp. 690–706, 1996.

[3] R. Morales, N. Hernández, R. Cruz, V. D. Cruz, and L. A. Pineda, "Entropic associative memory for manuscript symbols," *arXiv preprint arXiv:2202.08413*, 2021.

[4] C. Wartenberg and K. Holmqvist, "Daily newspaper layout–designerspredictions of readersvisual behaviour–a case study," *Lund University Cognitive Studies*, vol. 126, pp. 1101–8453, 2005.

[5] L. A. Pineda, G. Fuentes, and R. Morales, "An entropic associative memory," *Scientific Reports*, vol. 11, no. 1, pp. 1–15, 2021.

[6] K. Karthick, M. Premkumar, R. Manikandan, and R. Cristin, "Study on diverse automatic identification techniques," *International Journal of Engineering & Technology*, vol. 7, no. 4, pp. 2895–2898, 2018.

[7] M. Kumar, M. Jindal, and R. Sharma, "Review on ocr for handwritten indian scripts character recognition," in *International Conference on Digital Image Processing and Information Technology.* Springer, 2011, pp. 268–276.

[8] S. Dehaene and G. Dehaene-Lambertz, "Is the brain prewired for letters?" *Nature neuroscience*, vol. 19, no. 9, pp. 1192–1193, 2016.

[9] J. L. Borges, "Funes, el memorioso," *Petrotecnia*, vol. 1, p. 95, 2004.

[10] R. Szeliski, *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.

[11] N. V. Rao, A. Sastry, A. Chakravarthy, and P. Kalyanchakravarthi, "Optical character recognition technique algorithms." *Journal of Theoretical & Applied Information Technology*, vol. 83, no. 2, 2016.

[12] T. G. Dietterich, "Steps toward robust artificial intelligence," *AI Magazine*, vol. 38, no. 3, pp. 3–24, 2017.

[13] F. Bartlett, "Remembering: A study in experimental and social psychology," 1932.

[14] L. A. Pineda, *Racionalidad Computacional*, 1st ed. AMEXCOMP, 2021.

[15] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database. 2010," *URL http://yann. lecun. com/exdb/mnist*, vol. 7, no. 23, p. 6, 2010.

[16] G. Cohen, S. Afshar, J. Tapson, and A. Schaik, "The emnist dataset," *NIST*, 2016.

[17] L. A. Pineda and R. Morales, "Weighted entropic associative memory: A case study on phonetic representation and learning," *Enviado*, 2022.

[18] K. Steinbuch, "Die lernmatrix," *Kybernetik*, vol. 1, no. 1, pp. 36–45, 1961.

[19] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "Non-holographic associative memory," *Nature*, vol. 222, pp. 960–962, 1969.

[20] T. Kohonen, "Correlation matrix memories," *Computers, IEEE Transactions on*, vol. C-21, pp. 353 – 359, 05 1972.

[21] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the USA*, vol. 79, no. 8, pp. 2554–2558, 1982.

[22] B. Kosko, "Bidirectional associative memories," *IEEE Transactions on Systems, man, and Cybernetics*, vol. 18, no. 1, pp. 49–60, 1988.

[23] D. Krotov and J. J. Hopfield, "Dense associative memory for pattern recognition," in *Advances in Neural Information Processing Systems*, vol. 29, 2016, pp. 1172–1180.

[24] M.-C. Su and C.-H. Chou, "Application of associative memory in human face detection," in *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, vol. 5. IEEE, 1999, pp. 3194–3197.

[25] G. Yang and F. Ding, "Associative memory optimized method on deep neural networks for image classification," *Information Sciences*, vol. 533, pp. 108–119, 2020.

[26] M. Namba and Z. Zhang, "Cellular neural network for associative memory and its application to braille image recognition," in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. IEEE, 2006, pp. 2409–2414.

[27] J. B. Flowers, "La macchina che legge e che scrive," *La scienza per tutti XXIII*, vol. 11, pp. 166–167, 1916.

[28] C. Codelupi, "Macchina per leggere pei ciechi," *La scienza per tutti XXVIII*, vol. 2, p. 20, 1921.

[29] D. Romuald and T. Mens, "Gismo: a domain-specific modelling language for executable prototyping of gestural interaction," in *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2015, pp. 34–43.

[30] J. G. Linvill, *Research and development of tactile facsimile reading aid for the blind: The optacon.* US Department of Health, Education and Welfare, Office of Education, Bureau . . . , 1973.

[31] J. Memon, M. Sami, R. A. Khan, and M. Uddin, "Handwritten optical character recognition (ocr): A comprehensive systematic literature review (slr)," *IEEE Access*, vol. 8, pp. 142 642–142 668, 2020.

[32] R. Smith, "An overview of the tesseract ocr engine," in *Ninth international conference on document analysis and recognition (ICDAR 2007)*, vol. 2. IEEE, 2007, pp. 629–633.

[33] T. Hegghammer, "Ocr with tesseract, amazon textract, and google document ai: a benchmarking experiment," *Journal of Computational Social Science*, pp. 1–22, 2021.

[34] A. Choudhary, R. Rishi, and S. Ahlawat, "A new approach to detect and extract characters from off-line printed images and text," *Procedia Computer Science*, vol. 17, pp. 434–440, 2013.

[35] P. R. Cavalin, A. de Souza Britto Jr, F. Bortolozzi, R. Sabourin, and L. E. S. Oliveira, "An implicit segmentation-based method for recognition of handwritten strings of characters," in *Proceedings of the 2006 ACM symposium on Applied computing*, 2006, pp. 836–840.

[36] S. Bhende, K. Thakur, J. Teseng, M. L. Ali, and N. Wang, "Character recognition using hidden markov models," *Int. J. Recent Technol. Eng*, vol. 7, no. 4S2, pp. 105–110, 2018.

[37] B. Hayes, "Cloud computing," 2008.

[38] "Bored of mnist? let's build your own ocr deep learning computer vision ai using microsoft cntk with emnist (step by step guide)," Nov 2017. [Online]. Available: https://techcommunity.microsoft.com/t5/educator-developer-blog/bored-of-mnist-let-8217-s-build-your-own-ocr-deep-learning/ba-p/379968

[39] A. S. Sharma, M. A. Mridul, M.-E. Jannat, and M. S. Islam, "A deep cnn model for student learning pedagogy detection data collection using ocr," in *2018 International Conference on Bangla Speech and Language Processing (ICBSLP)*. IEEE, 2018, pp. 1–6.

[40] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[41] T. T. H. Nguyen, A. Jatowt, N.-V. Nguyen, M. Coustaty, and A. Doucet, "Neural machine translation with bert for post-ocr error detection and correction," in *Proceedings of the ACM/IEEE joint conference on digital libraries in 2020*, 2020, pp. 333–336.

[42] M. Wolf and C. J. Stoodley, *Proust and the squid: The story and science of the reading brain.* Harper Perennial New York, 2008.

[43] S. Dehaene and L. Cohen, "Cultural recycling of cortical maps," *Neuron*, vol. 56, no. 2, pp. 384–398, 2007.

[44] E. R. Schotter, B. Angele, and K. Rayner, "Parafoveal processing in reading," *Attention, Perception, & Psychophysics*, vol. 74, no. 1, pp. 5–35, 2012.

[45] U.-V. Marti and H. Bunke, "The iam-database: an english sentence database for offline handwriting recognition," *International Journal on Document Analysis and Recognition*, vol. 5, no. 1, pp. 39–46, 2002.

[46] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: an extension of MNIST to handwritten letters," *arXiv preprint arXiv:1702.05373*, 2017.

[47] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart, "Distributed representations (chapter 3)," in *Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Vol.1: Foundations*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, Mass.: The MIT Press, 1986.

[48] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[49] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[50] P. Norvig, "Natural language corpus data," *Beautiful data*, pp. 219–242, 2009.

[51] A. A. Z. Galil *et al.*, *Pattern matching algorithms.* Oxford University Press on Demand, 1997.

[52] D. Jurafsky and J. H. Martin, "N-gram language models," *Speech and language processing*, vol. 23, 2018.

[53] R. C. Martin, *Clean code: a handbook of agile software craftsmanship.* Pearson Education, 2009.

[54] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and D. Patterns, "Elements of reusable object-oriented software," *Design Patterns*, 1995.

[55] L. Kang, "Robust handwritten text recognition in scarce labeling scenarios: Disentanglement, adaptation and generation," Ph.D. dissertation, Universitat Autònoma de Barcelona, 2020.

[56] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout improves recurrent neural networks for handwriting recognition," in *2014 14th international conference on frontiers in handwriting recognition.* IEEE, 2014, pp. 285–290.