UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Una aplicación para móviles de apoyo al aprendizaje del idioma japonés usando reconocimiento de kanjis basado en trazos

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Licenciada en Ciencias de la Computación

PRESENTA:

Rosa Victoria Villa Padilla

Director de Tesis:

Dr. Gustavo De la Cruz Martínez

Ciudad Universitaria, Cd. Mx., 2017

Dedico esta Tesis a mis padres que me dieron la vida, su amor, apoyo y confianza. A ustedes que me enseñaron que la única forma de alcanzar mis sueños es trabajando muy duro por ellos. A ustedes que contra viento y marea lucharon por hacer de mí una persona de provecho.

Y a mis tías por su apoyo, amistad, cariño y compresión que me brindaron en los momentos que más lo necesite, en forma incondicional y desinteresada. Les agradezco los consejos y la forma sutil de saberme guiar por el sendero del éxito.

A todos ustedes a quien debo todo lo que soy, nunca podre pagarles todo lo que me han dado. Por esto y más, ***¡Gracias!***

Índice general

1. Introducción. Los kanjis en el idioma japonés. 1

2. Reconocimiento óptico de caracteres (Optical Character Recognition) 8

2.1. Introducción 8
2.2. Desarrollo del OCR 8
2.2.1. Los inicios del OCR 9
2.2.2. Edad de corta e intenta 10
2.2.2.1. Método de Comparación de Plantillas (Template Matching) 10
2.2.2.2. Métodos de Análisis de Estructura (Structure Analysis Method) 13
2.3. Resumen 17

3. Métodos de reconocimiento de kanjis 18

3.1. Introducción 18
3.2. Reconocimiento de escritura china 18
3.2.1. Introducción 18
3.2.2. Función de extracción y clasificación 19
3.2.3. Las tasas de reconocimiento obtenidos en diversas bases de datos 21
3.2.4. Coincidencia Estructural(Structural matching) 21
3.2.5. Método híbrido 24
3.3. Reconocimiento de escritura Coreana 24
3.3.1. Introducción 24
3.3.2. Metodologías 25
3.4. Reconocimiento de escritura Japonesa 28
3.4.1. Introducción 28
3.4.2. Preprocesamiento (normalización) 29
3.4.3. Grupos de Investigación 29
3.4.3.1. El trabajo de Wakahara 29
3.4.3.2. Trabajo del grupo de Miyake-Kimura 30
3.5. Métodos de reconocimiento de kanjis en diccionarios actuales . 32
3.6. Resumen 35

4. Reconocimiento de kanjis basado en trazos. 36
4.1. Mapas autoorganizados para el reconocimiento de trazos 37
4.1.1. Redes Neuronales 37
4.1.2. Redes no supervisadas 39
4.1.3. Mapas autoorganizados 39
4.1.3.1. Normalización de la entrada 41
4.1.3.2. Cálculo del rendimiento de cada neurona . . . 42
4.1.3.3. Eligiendo la neurona ganadora 42
4.1.3.4. Cómo aprende un mapa autoorganizado . . . 42
4.1.3.5. Proceso de entrenamiento 43
4.1.3.6. Tasa de aprendizaje 44
4.1.3.7. Ajuste de pesos 45
4.1.3.8. Calculo del error 46
4.2. Árboles de decisión n-arios 46
4.2.1. Generalidades y partes de un árbol 47
4.2.2. Árboles de decisión 48
4.2.2.1. Funcionamiento visto con ejemplos 48
4.2.2.2. Decisiones 50
4.2.2.3. Complejidad de la Decisión 51
4.2.2.4. Rendimiento de los árboles de decisión 53
4.3. Metodología desarrollada 53
4.3.1. Reconocimiento de trazos básicos usando mapas auto-organizados 54
4.3.1.1. Binarización del trazo 55
4.3.2. Almacenamiento de trazos básicos en el orden que se fueron trazados 58
4.3.3. Selección de árbol de decisiones a partir del número de trazos 59
4.3.3.1. Los kanjis vistos como una secuencia de trazos 59
4.3.3.2. Construyendo los árboles n-arios 63
4.3.3.2.1. Podando el árbol 64
4.3.3.3. Propiedades de los árboles de decisión 65
4.3.3.4. Selección de árbol de decisiones a partir del número de trazos 67
4.3.3.5. Obteniendo un kanji en el árbol de decisión . 67
4.4. Resumen 68

5. Diseño e implementación del reconocimiento de kanjis en dispositivos móviles 70
5.1. Implementación de referencia 70
5.1.1. Mapas Autoorganizados 70
5.1.1.1. Clase de normalización del SOM 71
5.1.1.1.1. Cálculando los factores 71
5.1.1.1.2. Creación de la matriz de entrada . . 72
5.1.1.2. Clase de implementación del SOM 72
5.1.1.3. Clase de entrenamiento del SOM 72
5.1.1.3.1. Iteración de entrenamiento 73
5.1.1.3.2. Evaluación de Errores 73
5.1.1.3.3. Forzando una ganadora 73
5.1.1.3.4. Ajustando los pesos 73
5.1.2. Reconocimiento de caracteres con mapas autoorganizados 74
5.1.2.1. Trazado de imágenes 74
5.1.2.1.1. Tamaño y posición 75
5.1.2.1.2. Realización de la reducción de la resolución 75
5.1.2.1.3. Visualización de la versión binarizada 76
5.1.2.1.4. Uso del mapa de autoorganización . 76
5.1.2.1.5. Entrenamiento de la Red Neural . . 77
5.2. Implementación del reconocimiento de trazos 78
5.2.1. Aplicación para entrenamiento del mapa autoorganizado 79
5.2.1.1. Entrenamiento 79
5.2.1.2. Guardar y cargar un conjunto de entrenamiento 82
5.2.1.3. Verificación del entrenamiento 84
5.2.1.4. Descripcion de otros elementos de la aplicación 85
5.2.2. Pruebas con la división del lienzo y selección del actual 86
5.2.3. Pruebas con los trazos 88
5.2.4. Parámetros finales 90
5.3. Implementación de la identificación del kanji 91
5.3.1. Codificación de árboles 91
5.3.2. Identificación del kanji 93
5.4. Aplicación para móviles de apoyo al aprendizaje del idioma japonés 94
5.5. Pruebas de reconocimiento de kanjis trazados por usuarios reales 101
5.5.1. Usuario con completo desconocimiento del idioma . . . 102
5.5.2. Usuarios con conocimientos adquiridos sin instrucción formal 103
5.5.3. Usuario con conocimientos adquiridos en clases con instrucción formal 106

5.6. Resumen 106

6. Conclusiones **108**
6.1. Resumen 108
6.2. Conclusiones 111
6.3. Trabajo a Futuro 112

Bibliografía **113**

Índice de figuras

1.1. Realización de kanji en un cuadro 2
1.2. Traductor agrupado por trazos y radicales[12] 5

2.1. Dispositivo de Gustav Tauschek [1] 10
2.2. Ilustración de la reducción de 2-D a 1-D por una ranura [2]. (a)El número 4 es escaneado por una ranura de izquierda a derecha. (b)La proyección del área negra sobre el eje x, el escaneo es en dirección de la ranura. 11
2.3. Ilustración del método de mirilla [2]. 12
2.4. Extensión del método de mirilla al método de análisis de estructura [2] 14
2.5. Ilustración del método de sonda [3] 15

3.1. Ilustración de la extracción de características [4] 19
3.2. Extracción de una característica de malla para un enfoque holístico [5] 26
3.3. Construcción de múltiples modelos de grafemas: (a) diversas formas estructurales de un grafema y (b) corresponde al modelo del grafema [6] 27
3.4. Un ejemplo de secuencia de trazo [7]: (a)Después del preprocesamiento; (b)resultado de la secuencia de trazo 28
3.5. Posiciones de los radicales 33
3.6. Ejemplo de uso de los radicales 33

4.1. Estructura básica de una red neuronal 38
4.2. Mapa autoorganizado [8] 40
4.3. Representación de un árbol de decisión n-ario 47
4.4. Arbol de decisión 49
4.5. Árbol de decisión con decisión realizada 50
4.6. Ancho de un árbol de decisión 51
4.7. Árbol de decisión binario profundo 52
4.8. Árbol de decisión con cuatro ramas 52
4.9. Trazo t_3 en el lienzo 56
4.10. División del lienzo en una malla de 10x10 56
4.11. Rellenado de la representación 57
4.12. Árbol de kanjis 63
4.13. Árbol podado 64
4.14. Toma de decisiones 66

4.15. Búsqueda en el árbol de tres trazos 68

5.1. Inicio de la aplicación de Entrenamiento 79
5.2. Agregar una muestra al conjunto de entrenamiento 80
5.3. Imagen binarizada de un trazo muestra 80
5.4. Entrenamiento terminado 81
5.5. Trazo reconocido por el mapa autoorganizado 82
5.6. Archivo "muestra.dat" 82
5.7. Guardar conjunto de entrenamiento 83
5.8. Carga del archivo "muestra.dat" 83
5.9. Borrar muestra 84
5.10. Parte inferior muestra imagen binarizada del trazo realizado . . . 85
5.11. Limpieza del lienzo y la imagen binarizada 86
5.12. UML de las Clases involucradas en la codificación de los árboles 91
5.13. Representación de visual del árbol de tres trazos 93
5.14. Actualización de búsqueda de kanjis 94
5.15. Diagrama del funcionamiento de la aplicación. 94
5.16. Pantalla inicial 95
5.17. Inicialización terminada 95
5.18. Realización de un trazo 96
5.19. Reconocimiento de kanji de un trazo 96
5.20. Realización de un segundo trazo 97
5.21. Árbol de dos trazos 97
5.22. Reconocimiento de kanji de dos trazos 98
5.23. Árbol de tres trazos 98
5.24. Reconocimiento de kanjis conformados por los trazos **b,a,b** . . . 99
5.25. Realización de garabato 99
5.26. Mensaje de fallo en el reconocimiento de kanji 100
5.27. Limpieza del lienzo 100
5.28. Realización de todos los trazos antes de presionar el botón "Reconoce" 101
5.29. Kanji trazado (a)Kanji realizado por el usuario. (b)Kanji trazado correctamente. 102
5.30. Lógica del error al mostrar kanji equivocado. (a)Trazos realizados por el usuario. (bTrazos correndientes del kanji arrojado por la aplicación. 103
5.31. Falta de inclinación del primer trazo. (a)Kanji realizado por el usuario. (b)Kanji trazado correctamente. (c)Kanji de tipografía de computadora 104
5.32. Segundo trazo con curvatura no propia del kanji. (a)Kanji realizado por el usuario. (b)Kanji trazado correctamente. (c)Kanji de tipografía de computadora 104

5.33. Trazado incorrecto del segundo trazo. (a)Kanji realizado por el usuario. (b)Kanji trazado correctamente. (c)Kanji de tipografía de computadora 105
5.34. Omisión del gancho en segundo trazo. (a)Kanji realizado por el usuario. (b)Kanji trazado correctamente. (c)Kanji de tipografía de computadora 105
5.35. Kanji mal trazado. (a)Kanji realizado por el usuario. (b)Kanji trazado correctamente. (c)Kanji de tipografía de computadora . 106

Índice de tablas

4.1. Tipos de datos 50
4.2. Trazos básicos 54
4.3. Tabla de kanjis de tres trazos 60
4.4. Selección de kanjis cuyo primer trazo es t_2 61
4.5. Selección de kanjis cuyo primer trazo es t_2 y segundo trazo t_5 . . 62

5.1. Resultados del reconocimiento con lienzos de 10x10, 15x15,20x20 y 30x30 86
5.2. Número de muestras por trazo realizadas en tres fases de pruebas 89
5.3. Resultados de las pruebas de reconocimiento de trazos básicos . 90
5.4. Kanjis visto como secuencia de caracteres (trazos) 92

1 Introducción. Los kanjis en el idioma japonés.

Los kanjis son la raíz fundamental de la escritura china y japonesa. Cada kanji puede representar una cosa, un animal, un verbo o un adjetivo, es decir, puede representar una o varias ideas, dependiendo de su contexto.

Esta forma de escritura puede llegar a ser conflictiva ya que el número de kanjis existentes es muy grande, y esto hace muy complicado el proceso de aprendizaje. Los kanjis se pueden combinar entre sí para formar otros kanjis y un conjunto de kanjis explican ideas más complejas reduciendo así el número de caracteres necesarios.

Según el Ministerio de educación, cultura, deportes, ciencia y tecnología de Japón [9] un niño japonés al terminar la primaria ha aprendido mil seis kanjis, mientras un joven japonés durante la secundaria aprende otros novecientos treinta y nueve kanjis y el resto de su vida se dedica a aprender más kanjis.

Para cada kanji se tiene que memorizar el número de trazos, el orden de los trazos y su significado. La verdadera problemática es que la mayoría de los kanjis tiene más de una lectura. Esta lectura depende del contexto en el que se encuentre.

Y por si esto fuera poco existen dos silabarios [1], uno es una guía para la escritura fonética de las palabras, de flexionar los verbos y adjetivos, y de definir las funciones sintácticas dentro de las oraciones (esto es lo que le da contexto a los kanjis); el otro hace posible la escritura de vocablos extranjeros, entre otras cosas. Éstos son el hiragana (silabario empleado para cosas propias del país) y el katakana (silabario empleado para referirse a cosas extranjeras).

Los kanjis tienen dos tipos de lectura, la “onyomi” y la “kunyomi”, la primera

[1] Silabario, cualquier forma de escritura en la que los caracteres representan sílabas, como las usadas en las lenguas tamil, cheroqui, inuktitut, china o japonesa

es de origen chino (en los diccionarios la encontramos escrita en katakana) y la segunda es japonesa (en los diccionarios se encuentra escrita en hiragana) por ejemplo el kanji de caballo 馬 se puede leer “uma” o “ba”.

Ahora bien, la caligrafía occidental se basa en obtener el símbolo final esperado sin tener en cuenta el modo de escribir dicho símbolo, es decir, mientras el símbolo se parezca a la letra molde es correcta su escritura. Sin embargo, la caligrafía oriental (china, japonesa y coreana) tiene reglas de escritura estrictas y requiere una disciplina muy superior a la que estamos acostumbrados en occidente. Un kanji debe escribirse en un cuadrado perfecto y de manera centrada como se ve en la figura 1.1.

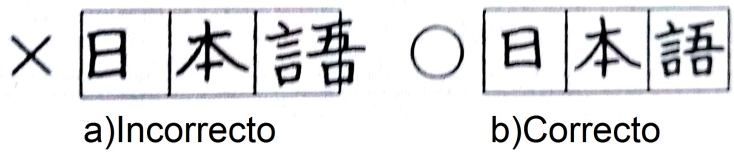


Figura 1.1: Realización de kanji en un cuadro

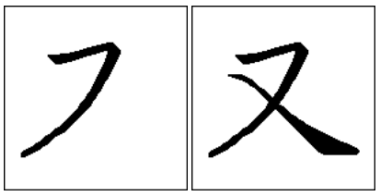
Cada kanji se compone de trazos que deben ser realizados en un orden específico. Aprender a escribir kanjis depende en gran medida de la atención que se preste al correcto orden de sus trazos. Cientos de años de experiencia han dado lugar a una serie de reglas básicas sobre cómo debe escribirse un kanji, pero aun así, existen kanjis que no siguen estas reglas. Pueden ocurrir conflictos entre dos o más reglas, entonces la única manera de conocer las excepciones es sabiendo de antemano cómo se escriben los kanjis. Con las reglas que se dan a continuación la mayoría de los kanjis se escriben sin ningún problema [10].

Primer regla: Arriba antes que abajo.

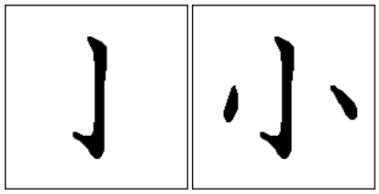
Segunda regla: Izquierda antes que derecha.

Tercera regla: Líneas horizontales antes que una línea que las cruza.

Cuarta regla: Líneas inclinadas hacia la derecha antes que líneas inclinadas hacia la izquierda cruzando las anteriores.



Quinta regla: Parte central antes que elementos simétricos a ella.



Sexta regla: Exterior antes que interior. Con la excepción que la «caja» que contiene a otros trazos no se cierra hasta haber acabado la parte interior.

De poco nos serviría tener una lista de los 7000 kanjis que componen la lengua japonesa si no estuvieran clasificados. Una buena clasificación es vital para un acceso rápido a los kanjis, algo fundamental para una traducción medianamente eficiente. El tiempo medio de búsqueda de un kanji depende en gran medida de la información que tengamos del mismo. Las fuentes que recopilan y clasifican los kanjis se conocen con el nombre de “diccionarios de kanjis”. La mayoría suelen usar la clasificación por número de trazos. La gran ventaja de la clasificación por número de trazos es que en principio siempre podremos contar el número de trazos de un kanji.

La clasificación por número de trazos se basa en que los kanjis tienen un mínimo de un trazo hasta un máximo de treinta trazos. Un trazo es una línea que se hace sin levantar el pincel del papel. El número de trazos de un kanji, en principio, es información que siempre vamos a tener sobre un kanji. Saber cuantos trazos tiene un kanji es útil para poder buscarlo dentro del total de trazos, ya nos permite eliminar una gran cantidad dentro del total reduciendo el tiempo medio de localización de un carácter. Si sabemos que nuestro kanji, por ejemplo, tiene diez trazos, entonces sólo tendremos que buscar entre unos doscientos kanjis, no entre más de dos mil.

Con esta pequeña introducción puede que más de uno se haya preguntado: si una persona, incluso un japonés, se encuentra con un kanji que no entiende, ¿cómo puede buscar su significado? para ello existen métodos de búsqueda en diccionarios por el número de trazos, por sus radicales, por su lectura, etc, sin embargo este proceso de búsqueda en diccionarios de papel toma varios minutos, lo cual hace que el aprendizaje y la comprensión de lectura del idioma japonés sea lento. Aún en un diccionario en línea el proceso de búsqueda es muy lento, ya que, en algunas aplicaciones, la persona debe de saber cómo se lee el kanji para poder indicar el kanji; por ejemplo para escribir el kanji □ en el traductor de google debes saber alguna de sus lecturas (ya sea onyomi o kunyomi) y escribirlas para que te sugiera al kanji (figura **??**) y posteriormente busqué su significado.

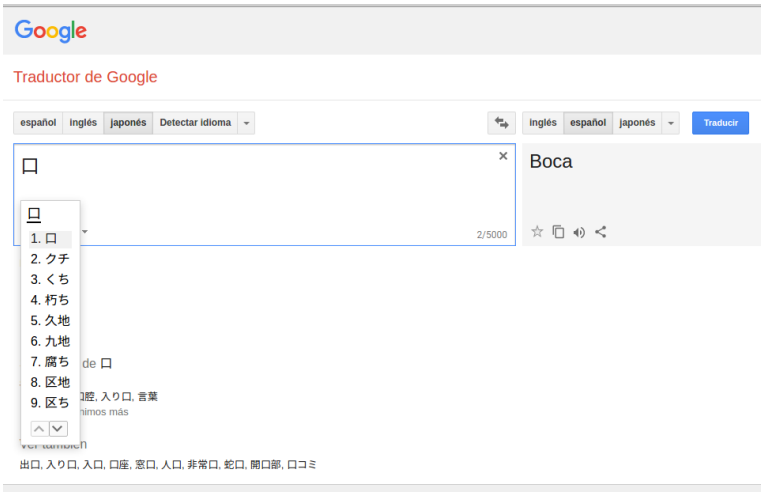


Figura 1.2: Kanji 口 lectura kunyomi: くち (kuchi) [11]

En otras aplicaciones, los kanjis están agrupados por número de trazos o radicales, por lo cual la persona debe de conocer el número de trazos o los radicales que lo componen 1.2.

Figura 1.3: Traductor agrupado por trazos y radicales[12]

Una estrategia que se ha utilizado es diseñar diccionarios para dispositivos móviles. Algunas de estas propuesta plantean que el usuario trace el kanji directamente en el dispositivo y se muestre su significado. Entre las aplicaciones gratuitas encontradas la tienda de aplicaciones de Google para este fin son:

Japanese. Permite al usuario dibujar un kanji, pero en las pruebas realizadas se observó que falla con frecuencia en el reconocimiento[13].

Japanese English dictionary. Plantea reconocer un kanji a partir del trazo,

sin embargo no se pudo probar ya que requiere descargar un archivo adicional, pero al intentar hacer la descarga, siempre falla [14].

Kanji Recognizer. Esta aplicación enumera los trazos que vas realizando y sugiere los kanjis que coinciden con los trazos realizados [15].

Akebi Japanese Dictionary. Esta aplicación permite hacer trazar un kanji después de hacer una búsqueda en texto, lo cual es poco claro [16].

También existen las aplicaciones que usan las estrategias de buscar manualmente el kanji usando el número de trazos o radicales, lo cual sigue siendo un proceso lento. Las aplicaciones que usan esta estrategia son:

Japanese for Android. Ofrece una búsqueda por radicales que se puede refinar agregan más radicales [17].

Kodansha Kanji Learner's Dict. Al igual que la anterior aplicación esta ofrece una búsqueda por radicales sin embargo esta aplicación es de paga [18].

Cabe mencionar que las aplicaciones anteriormente mencionadas son diccionarios Japonés-Inglés, también existen las aplicaciones de japones a otros idiomas como es Filipino [19], Vietnamita [20], Nepal[21], Malasia [22], Alemán[23] entre otros.

Para el caso del español se encontro una aplicacion, **Diccionario Japonés** [24], que permite el reconocimiento de un kanji por sus trazos sin importar el orden en que los vaya realizando.

El objetivo de este trabajo es hacer una aplicación para dispositivos móviles con Windows, en la que el usuario trace (dibuje) un kanji y nos indique el significado.

La hipótesis central de este trabajo es que es posible realizar el reconocimiento de trazos en dispositivos móviles usando mapas autoorganizados, para así realizar la identificación de un kanji trazado por el usuario, lo que sería la base de un proceso de búsqueda para un diccionario básico japonés-español.

Cabe resaltar que aunque existen aplicaciones que reconocen los trazos de un kanji y buscan su significado, no existe documentación disponible sobre su fun-

cionamiento, por lo que no se puede hacer una comparación con su desempeño.

En el capítulo dos se realiza una pequeña recopilación de los métodos desarrollados para el reconocimiento óptico de caracteres alfanuméricos. En este capítulo se puede observar que la creación de estos sistemas no siguió una sola dirección, sino que tuvo varias vertientes con diferentes resultados siendo olvidados los métodos con más pobres resultados.

En el capítulo tres se lleva a cabo el análisis de los métodos desarrollados para el reconocimiento de caracteres de lengua asiática, se presentan trabajos realizados en China, Corea y Japón. Corea presenta los resultados más pobres mientras que Japón fue el más activo y tiene una amplia gama de trabajos realizados por lo que solo se presentan los más destacados.

En el capítulo cuatro se plantea las teorías que serán utilizadas para desarrollar la aplicación propuesta. En este capítulo ofrecemos una breve introducción a las redes neuronales, se describe cómo funcionan los mapas autoorganizados y el cómo funcionan los árboles de decisión n-arios usados para la identificación de los kanjis.

En el capítulo cinco se muestra el diseño y la implementación de las aplicaciones, se describe la implementación de Heaton [8] para el reconocimiento óptico de caracteres alfanuméricos, en la que hacen uso de los mapas autoorganizados, y cómo fue adaptada esta implementación para que reconozca los trazos de un kanji. En este capítulo también se describen el entrenamiento realizado y sus pruebas. Posteriormente se muestra la implementación del proceso de identificación de un kanji usando los árboles de decisión. Una vez presentado el funcionamiento general de la propuesta para el reconocimiento de kanjis, se describe el funcionamiento de la aplicación para el entrenamiento del mapa autoorganizado, así como la aplicación para el apoyo al aprendizaje del idioma japonés. Finalmente, es discutido los resultados de las pruebas con diferentes usuarios.

En el capítulo seis se presentan las conclusiones finales de este trabajo y también es planteado el trabajo a futuro de la aplicación realizada.

2 Reconocimiento óptico de caracteres (Optical Character Recognition)

2.1. Introducción

El camino que ha recorrido la investigación del reconocimiento óptico de caracteres (ROC), generalmente conocido como reconocimiento de caracteres y referido con frecuencia con la sigla OCR (del inglés Optical Character Recognition) y el reconocimiento de voz no ha sido una línea recta, más bien, un camino lleno de baches. Esta investigación tiene sus orígenes en el campo del reconocimiento de patrones y a manera introductoria, en este capítulo se menciona, de manera breve, cómo fue evolucionando esta investigación.

Es importante resaltar que la idea de hacer un sistema reconocedor de kanjis existe desde hace muchos años, ya que este fue uno de los objetivos de diversos grupos de investigadores de Japón durante varias décadas, una buena recopilación de estos trabajos los podemos leer en el artículo “Historical Review of OCR Research and Development” [2].

Estos sistemas no surgieron de una idea única, es decir, se han propuesto diversos métodos y teorías que mencionaremos más adelante a detalle, a través de muchas pruebas y errores se fueron puliendo y mezclando metodologías, que individualmente no eran tan buenas sin embargo en conjunto daban un alto índice de exactitud. Es por eso que este capítulo es importante para entender la relevancia de la idea planteada en esta tesis.

2.2. Desarrollo del OCR

En principio se consideró el tema de reconocimiento de caracteres como un problema fácil de resolver, sin embargo al darse cuenta que no era así se fue

diversificando a otros temas como el reconocimiento de imágenes y el reconocimiento 3-D.

La investigación se bifurcó en comparación de plantillas(template matching), y análisis de estructuras (structure analysis), estos dos enfoques están convergiendo en soluciones que toman lo mejor de cada uno.

La participación de Japón ha sido clave en la investigación de los OCR, ejemplos de los trabajos más reconocidos fueron realizados por Mori [25] e Iijima [26].

2.2.1. Los inicios del OCR

De acuerdo a Mori, Suen, y Yamamoto (1992) [2], los primeros conceptos de OCR se remontan hasta antes de la era de las computadoras, con una patente presentada por Gustav Tauschek en Alemania en 1929[1] y que posteriormente fue presentada por P. W. Handel en los Estados Unidos en el año de 1933[27].

El dispositivo creado por Tauschek refleja la tecnología con la que se contaba en ese momento, basada en la óptica y la mecánica. La luz pasaba a través de una máscara mecánica y esta era capturada por células fotoeléctricas que eran escaneadas mecánicamente.

Cuando se producía una coincidencia, la luz no se detectaba y la máquina reconocía los caracteres impresos en el papel.

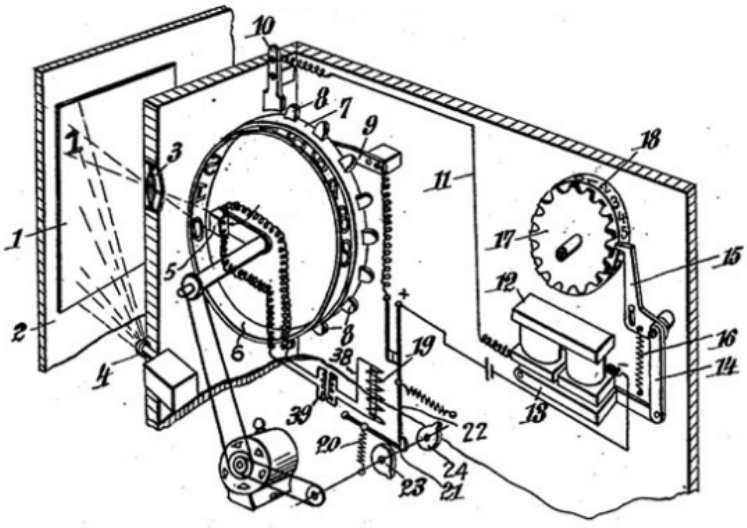


Figura 2.1: Dispositivo de Gustav Tauschek [1]

Este artefacto está basado en el axioma de superposición, descrito por primera vez por Euclides, este principio de superposición se usó en tecnologías más avanzadas como los tubos de rayos catódicos y circuitos electrónicos analógicos. Este trabajo es considerado el origen principal de la tecnología OCR.

2.2.2. Edad de corta e intenta

La primera computadora comercial, UNIVAC I, fue instalada y empezó a trabajar en el año de 1951, con ello empezó la era de las computadoras y con esto se vio como una posible realidad la investigación de OCR; sin embargo había grandes limitantes, como la dificultad de acceso a los equipos y los pocos recursos de hardware disponibles.

A continuación mostraremos los inicios de los dos enfoques antes mencionados.

2.2.2.1. Método de Comparación de Plantillas (Template Matching)

En el año de 1956 Kelner y Glauberman publicaron el artículo titulado “Character Recognition for business machines”[28], en este artículo mencionan como

realizaron un dispositivo basado en histogramas de proyección que proporcionan el número de píxeles negros en las direcciones vertical y horizontal del área especificada de un carácter.

Este funcionaba de la siguiente manera, un carácter era escaneado verticalmente de arriba a abajo a través de una ranura, de la cual se reflejaba la luz en el papel de entrada y esta era transmitida a un fotodetector. Todos los cálculos eran realizados con sumas algebraicas para así obtener un valor proporcional al área que estaba en negro dentro de la ranura de los segmentos de caracteres. Después los valores del muestreo eran transformados de un valor analógico a digital. El dispositivo de Comparación de Plantillas tomaba la suma total de la diferencia entre el valor de muestra y el valor de la plantilla, cada uno de estos se normalizaba. Cabe mencionar que esta máquina nunca se comercializó. El proceso de Comparación de Plantillas podía dividirse en dos procesos: por un lado, la superposición de una entrada en una plantilla; y por otro lado, medir el grado de coincidencia entre el aspecto de la entrada y la plantilla. La proyección puede ser tomada horizontal o verticalmente, lo que hace que la posición de la superposición de proceso sea invariante de la dirección. Como podemos ver en la figura 2.2.

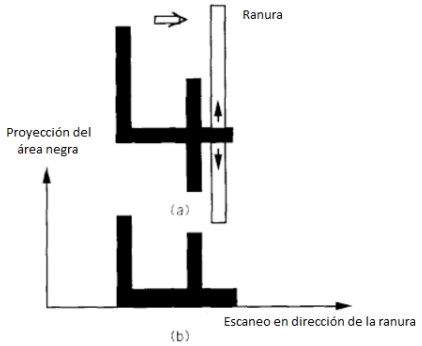


Figura 2.2: Ilustración de la reducción de 2-D a 1-D por una ranura [2].
(a)El número 4 es escaneado por una ranura de izquierda a derecha.
(b)La proyección del área negra sobre el eje x, el escaneo es en dirección de la ranura.

Cuando la ranura es lo suficientemente larga para cubrir los números de entrada, no hay cambio en el valor de área de color negro proyectada sobre el eje x, incluso si los números se mueven verticalmente.

Sin embargo se necesita detectar el comienzo y el fin de puntos de un carácter de entrada para registrarlos en contra de la plantilla correspondiente. Esto es muy fácil de hacer cuando los elementos están conectados y hay suficiente espacio

entre cada uno. Actualmente esta técnica de proyección ha sido ampliamente usada para segmentar cadenas y regiones de imágenes de los documentos, dicho procedimiento se llama preprocesamiento en la terminología OCR.

La llegada de computadoras influyó en el diseño de hardware y los algoritmos de OCR. El más simple de estos fue llamado el método de mirilla. Este asume que el carácter de entrada está binarizado. La binarización es una parte importante del preproceso en la tecnología OCR. Idealmente un carácter de entrada debe estar en dos niveles de densidad, es decir, blanco y negro, comúnmente representados por 0 y 1 respectivamente.

Imaginemos un plano de dos dimensiones en el que se almacena un carácter de entrada binarizado y registrado de acuerdo con alguna regla, este carácter debe estar posicionado en la esquina superior derecha como se muestra en la figura 2.3.

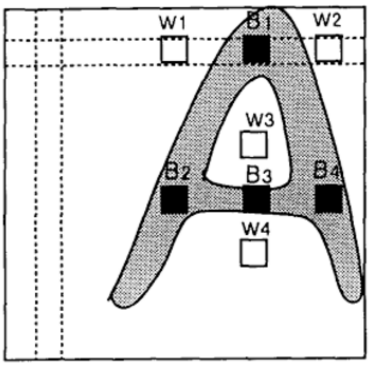


Figura 2.3: Ilustración del método de mirilla [2].

En el mejor de los casos, el carácter debe de tener un trazo de ancho y tamaño constante, porciones negras siempre son negras y lo mismo para el fondo blanco. El plano es dividido en regiones del mismo tamaño que mencionaremos cómo pixeles. Los pixeles son escogidos apropiadamente para ambas regiones de manera que los píxeles seleccionados permitan distinguir el carácter de entrada de otros caracteres de entrada.

El primer OCR basado en el método de mirilla fue anunciado por el Grupo de Electrónica Solatron y este fue llamado ERA (Electric Reading Automation) en 1957[29]. Los caracteres leídos por este OCR eran impresos por una caja registradora.

La velocidad de lectura era de 120 caracteres por segundo (chs), bastante rápido para la época y esto se debía a las simples operaciones lógicas que utilizaba. El

total de mirillas era 100, que sería lo necesario para obtener un reconocimiento estable para datos reales.

En el año de 1958, Iijima crea ETL un OCR diseñado y basado en el mismo esquema [30], sin embargo este diseño era más sistemático que ERA ya que usaba tres niveles de lógica haciéndolo más eficiente. El carácter podía ser reconocido entre 72 caracteres alfa-numéricos, se usaron 10x12 mallas. El número total de mirillas usadas fueron 44 para caracteres de 10 píxeles.

Como la mayoría de métodos, la comparación de plantillas tiene un punto débil, el cual es que si se tiene una ligera desviación en la forma, el tamaño o la orientación del carácter, este se leerá de manera incorrecta, para evitar esto se tendría que tener una plantilla diferente para cada posibilidad.

2.2.2.2. Métodos de Análisis de Estructura (Structure Analysis Method)

El método de la comparación de plantillas es apropiado para el reconocimiento de caracteres impresos. Sin embargo, al considerar los caracteres escritos a mano, tenemos una variación tan grande que es difícil crear plantillas para cada uno de estos.

A raíz de esta problemática se crearon los métodos de análisis de estructura. Los métodos que a continuación se describirán fueron aplicados a fuentes estilizadas(lo que ahora nombramos como “tipo de fuente”).

En el caso de los métodos de análisis de estructura no existe un principio matemático que los fundamente, más bien continúa siendo un problema abierto, por lo que la intuición ha sido la mayor arma para atacar este problema [2].

Dado que una estructura se puede dividir en partes, estas pueden ser descritas por las características de estas partes y la relación que existe entre ellas. El problema radica en elegir las características y las relaciones entre estas, para que tener una descripción clara de cada carácter identificado. Por lo tanto, la extracción de las características se ha convertido en un punto clave en la investigación de reconocimiento de patrones.

Según Mori [2] los métodos más destacados son:

1. Slit/Análisis de trazo: Ya hemos mencionado que el método de mirilla se considera un tipo de método de Comparación de Plantillas. La mirilla se limita a un solo píxel. en lugar de ello, se puede expandir a una ranura o ventana, con lo que se tendría que fijar la posición específica en el plano de dos dimensiones. La relación lógica entre dos píxeles se puede extender a una relación general entre ellos como se muestra en la figura 2.4.

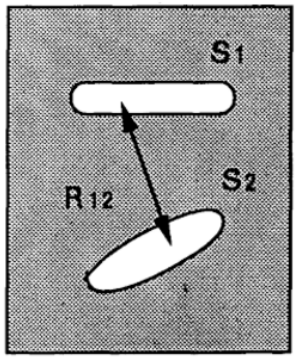


Figura 2.4: Extensión del método de mirilla al método de análisis de estructura [2]

Quizás el ejemplo más simple es la llamada técnica de conteo de cruzamientos (cross counting), donde las líneas escaneadas son consideradas como ranuras. La función de la ranura es identificar el número de regiones en negro en la misma. En el año de 1954 W. S. Rohland, propone esta técnica [31], en la que el escaneo vertical se utilizó principalmente. En 1961, R. W. Weeks [32], utiliza este enfoque pero de una manera más simple. En este método de exploración se toma en cuenta cuatro direcciones: vertical, horizontal y dos diagonales ortogonales, para cada dirección se utilizan seis líneas igualmente espaciadas y paralelas para cubrir un carácter. Un recuento de cruces, se define como el número de veces que cada una de las seis líneas se cruzan en el área negra, sin embargo, si se tiene tres o más cruces se consideran solo tres y así son posibles contar 0, 1, 2 y 3. Por lo tanto, la entrada de un carácter se codifica en 6x6x4=144 bits, que se utilizarían para el reconocimiento usando un método de decisión estadística.

El método de decisión estadística tiene un enfoque teórico muy importante para el reconocimiento de patrones en general, los siguientes tres artículos se consideran fundamentales para entender el tema:

- W. H. Highleyman en el articulo “Linear decision functions with applications to pattern recognition” en 1962 [33].

- R. 0. Duda y P. E. Hart en el articulo “Pattern Recognition and Scene Analysis” en 1973 [34]. y finalmente

- G. Nagy con el articulo “Optical character recognition-Theory and practice,” en 1982 [35].

Las ranuras son generalmente convexas, dentro de cada una de ellas podemos detectar características tanto topológicas y geométricas. Mediante el uso de las ranuras podemos detectar dos características principales: los componente conectados, que son el recuento de cruces y las áreas negras, que están en función de la longitud y ancho de la porción de estas. En los casos anteriores, la ranura está dada por una línea recta debido al mecanismo de exploración simple. No obstante, necesitamos no estar restringidos a este método de escaneo.

En realidad R. B. Johnson en 1956 [3] y T. L. Dimond en 1957 [36] utilizan una sonda como una ranura como se muestra en la figura 2.5. En los números identificaron que se podía poner dos puntos básico alrededor de estos. Al pasar la sonda por los dos puntos centrados se podía contar el número de cruces en cada hendidura, y a partir de esto, distinguir fácilmente el número escaneado. El esquema puede volverse más rígido asignando peso a cada sonda, lo que fue hecho por L. A. Kamentsky [37].

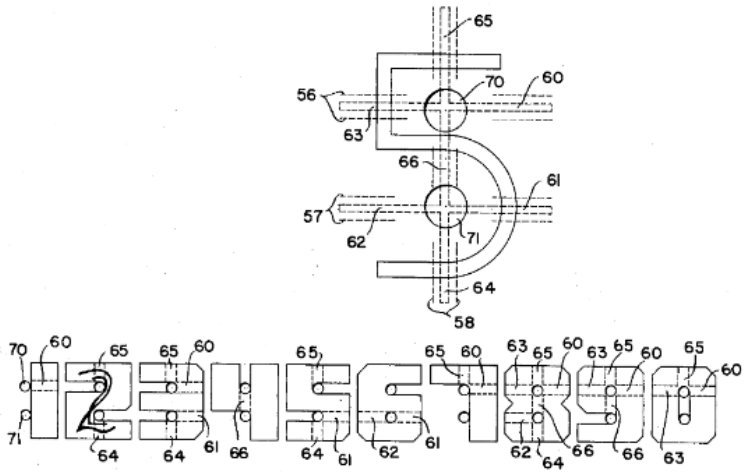


Figura 2.5: Ilustración del método de sonda [3]

Hasta ahora solo se han descrito características topológicas dentro de la

ranura, pero las características geométricas también se han usado ampliamente. En particular, el método que se conoce como codificación de longitud de ejecución (run-length coding) es típicamente usado en el análisis de ranuras.

Supongamos que existe una línea base, tal como una línea horizontal de izquierda a derecha, la característica geométrica es la distancia desde la línea de base hasta el primer pixel negro. Si la ranura escanea al carácter de arriba hacia abajo, entonces tenemos una función de valor único, que es más fácil de analizar. En este caso se trata de una función convexa.

Con la generalización del método de análisis ranura/análisis de trazo, podemos introducir un prototipo basado en un método de lectura asíncrona que es independiente de la velocidad de escaneo.

Este prototipo fue iniciado por T. Sakai, M. Nagao, y Y. Shinmi, en 1963 [38]. Por otro lado, el enfoque de análisis de estructura de ranura es muy eficaz, siempre que el número de categorías de caracteres a leer sea limitada y las formas de los caracteres están diseñados mecánicamente con una fuente E-13B en el MICR (magnetic ink character reader) que se usa para el procesamiento de cheques.

En base a este enfoque del Análisis de Estructura, tiempo después (en 1968), se desarrolla la primera máquina automática clasificación de cartas para los números de código postal desarrollada por Toshiba, este trabajo fue realizado por Genchi, K. Mori, Watanabe, y Katsuragi.

2. Híbrido de Comparación de Plantillas y Análisis de Estructuras: Hasta este punto hemos clasificado los métodos de reconocimiento en dos clases: comparación de plantillas y análisis de estructuras. Ambos métodos tienen sus ventajas, por ejemplo, comparación de plantillas es muy sensible al cambio de posición pero puede ser muy fuerte en el sentido de coincidencia global. Por otro lado, el análisis de estructuras detecta las característica de los trazos de los caracteres.

   Un ejemplo de estas propuestas es el método características por zona (zoned feature), como primer paso de este, se pone un caracter de entrada en un frame que está dividido en subregiones, en cada uno de las cuales detecta algunas características locales. Por lo que la sensibilidad a la posición del método de comparación de plantillas se reduce ya que este se concentra en las características en ciertas subregiones.

Veamos un ejemplo, supongamos que un carácter se representa en una matriz de 15x15 en la que toma 5x5 pixeles como una subregión. Después la matriz es dividida en una matriz de 3x3, y en cada uno de los cuales se detectan en cuatro direcciones; horizontal, vertical y dos direcciones ortogonales. Por lo que un carácter es codificado en 3x3x4=36 bits. Este ejemplo, citado por Mori en [2], está basado en la investigación realizada en el NEC (Nippon Electric Company) [39], una idea similar fue propuesta por J. H. Munson [40], y otra idea similar por E. C. Greanias, P. E. Meagher, R. J. Norman, y P. Essinger [41], a partir de estas ideas fue construido un famoso OCR, el IBM 1287.

2.3. Resumen

En este capítulo abordamos el problema del reconocimiento óptico de caracteres OCR, se muestran los inicios de la investigación sobre el OCR desde el dispositivo de Gustav Tauschek en el 1929, hasta investigaciones más actuales. En este capítulo se observa que el desarrollo de estos sistemas tuvo dos grandes vertientes: comparación de plantillas y análisis de estructuras. Estas vertientes han crecido hasta convertirse en una tecnología concreta de OCR y durante el proceso de investigación, estas se fueron fusionando para convertirse en una amplia corriente que constituye una parte esencial de las tecnologías del reconocimiento de patrones de OCR.

Ahora bien, una vez revisadas las primeras investigaciones de reconocimiento óptico de caracteres, es conveniente ver cómo se atacó el problema del reconocimiento de escritura asiática, que si bien no son los mismos caracteres de China, Corea y Japón, estos otorgaron un reto más elevado dada la complejidad de sus escritura, esto se abordará en el siguiente capítulo.

3 Métodos de reconocimiento de kanjis

3.1. Introducción

En el capítulo anterior hablamos del estudio del reconocimiento de caracteres de uso universal sin embargo para este trabajo es necesario profundizar y hablar sobre las investigaciones realizadas por los asiáticos [4](en específico mencionaremos China, Corea y Japón) para el reconocimiento de los caracteres de su país, ver los diferentes puntos de vista que tuvieron para atacar un problema similar, las metodologías de reconocimiento, las características exploradas, bases de datos utilizadas, así como los esquemas de clasificación investigados.

3.2. Reconocimiento de escritura china

3.2.1. Introducción

Muchos científicos han perseguido la investigación sobre el reconocimiento del carácter chino escrito a mano durante más de 30 años. Una revisión muy completa se publicó en 1984 [42].

Una gran parte del trabajo de investigación se divide en dos grupos principales. El primer grupo de métodos se enfoca en la extracción de vectores de características a partir de muestras de entrenamiento, para calcular las distribuciones estadísticas de estas características y clasificar una entrada desconocida mediante la extracción de su vector de características y la estimación de la clase de caracteres más probable a la que pertenece. El segundo grupo de métodos trata de reconocer un carácter, haciendo coincidir los trazos individuales entre el carácter de entrada y un número de caracteres de la plantilla.

3.2.2. Función de extracción y clasificación

Dado que los caracteres chinos se componen de trazos en cuatro direcciones principales, la mayoría de los métodos tratan de extraer características relacionadas con la posición del trazo y la dirección. Por ejemplo, como se ilustra en la Figura 3.1, la característica periférica corresponde a la distancia AB a partir de un punto de la trama de caracteres para el primer píxel negro, mientras que el CD corresponde a la distancia entre las primeras transiciones de negro a blanco y blanco a negro en la segunda. Todas estas distancias son eficaces para la representación de la forma externa y la estructura interna de un carácter.

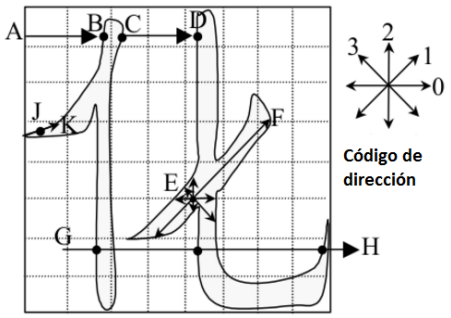


Figura 3.1: Ilustración de la extracción de características [4]

Otra característica útil es la densidad de la dirección del trazo, es decir, si tomamos como ejemplo el área del trazo que abarca el píxel E de la Figura 3.1, las distancias a lo largo de las ocho direcciones de este píxel a los píxeles límites del trazo (un pixel límite es un pixel que corresponde a un extremo del trazo analizado) se utilizan como características. Si la distancia a lo largo de la dirección EF es grande y las distancias a lo largo de otras direcciones son pequeñas, implica que este píxel pertenece a un trazo inclinado.

La técnica conteo de cruzamientos registra el número de transiciones de blanco a negro. Por ejemplo, en el píxel G de la Figura 3.1, el escaneo horizontal a lo largo de la dirección GH produce un conteo de 3 cruzamientos. También se llevan a cabo el escaneo a lo largo de la dirección vertical y las direcciones de las diagonales, los recuentos obtenidos representan la densidad de los trazos a lo largo de varias direcciones, visto desde la posición G.

La dirección tangencial de un píxel límite representa la dirección local en el trazo. Por ejemplo, en el pixel J de la Figura 3.1, la dirección JK puede ser codificada por los códigos de dirección 0 a 3 como se indica en la figura. La

codificación se puede hacer seleccionando el código con la dirección más cercana a JK, o resolver JK en dos componentes con las dos direcciones de código adyacentes más cercanas.

Después de conseguir estas características en diversas posiciones en el plano del kanji, se ensamblan en un vector de características de la siguiente manera. Para las características periféricas, el plano de caracteres se divide en bandas horizontales y verticales, y se suman los valores de característica en cada banda. Por ejemplo, en la Figura 3.1, se muestran ocho bandas horizontales y ocho verticales, resultando en un vector de características de 32 dimensiones para las distancias periféricas del trazo, tales como AB, y otro vector de 32 de dimensiones para las distancias entre trazos, como el CD. Para las otras características (dirección de densidad contributiva, conteo de cruzamientos, dirección del trazo tangencial), el plano del kanji se divide en celdas rectangulares. Dentro de cada celda, los valores de características con los mismos atributos de dirección se suman (con o sin ponderaciones de posición), lo que resulta en un vector de características de unas pocas dimensiones. El vector global de características se obtiene simplemente concatenando los vectores de todas las celdas. La dimensión del vector global de características resultante es generalmente grande, y puede estar dentro del intervalo de unas pocas decenas a más de mil. Hay muchas combinaciones diferentes y variaciones de los métodos anteriores, lo que resulta en función de los vectores que representan las propiedades locales y globales.

El vector de características de un kanji de entrada desconocido se compara con los vectores del conjunto de entrenamiento y algún tipo de medida de distancia para seleccionar la categoría de kanji más probable. Por ejemplo, la distancia puede ser la distancia Mahananobis, la distancia euclidiana, etc. Si la dimensión del vector de características es demasiado grande, se puede reducir por transformaciones tales como la transformación de Karhunen-Loeve y retener esos componentes con grandes valores propios. Debido al gran número de clases de kanjis chinos, la tarea de clasificación puede tardar mucho tiempo, y algunos métodos se utilizan para acelerar este proceso. Por ejemplo, N. Kato. [43], utiliza la distancia entre bloques para seleccionar rápidamente las clases más probables, 30 candidatas de entre las 3000 clases de kanjis, y luego se utiliza una distancia de Mahalanobis asimétrica para una buena clasificación. Por otro lado Y. Y. Tang [44], adopta un procedimiento de agrupamiento para realizar grupos de clases de kanjis similares en grupos no disjuntos, y un proceso de clasificación de 5 etapas, en el que cada que se empareja con diferentes características, para reducir las clases candidatos probables.

3.2.3. Las tasas de reconocimiento obtenidos en diversas bases de datos

Con el fin de comparar el rendimiento de diferentes algoritmos, es necesario probar los algoritmos con la misma base de datos. En Japón, hay bases de datos habituales para estos casos. La base de datos ETL8 consta de 881 clases de kanjis chinos escritos a mano y 75 clases de caracteres Hiragana, con 160 muestras por clase. Por otra parte, la base de datos ETL9 consta de 2,965 clases de kanjis chino y 71 caracteres Hiragana, con 200 muestras por clase. Los caracteres de estas dos bases de datos generalmente no están mal escritos. En Taiwán, la base de datos de ITRI (Industrial Technology Research Institute) consta de 5.401 clases de kanjis chinos escritos a mano con 200 muestras por clase. Además de estas bases de datos, también hay otras bases de datos recogidas por grupos de investigación independientes. Las tasas de reconocimiento reportados para las pruebas sobre la base de datos ETL9 son generalmente altos y alcanzan el 99,4 % en las pruebas realizadas por Kato [43]. Para la base de datos ITRI con 5,401 clases de kanjis, una tasa de reconocimiento del 89,0 % ha sido reportado por Y. Y. Tang [44]; en su reporte indica que es la tasa de reconocimiento más alta de esta base de datos, pero el número de clases de kanjis utilizado fue sólo 2,000.

3.2.4. Coincidencia Estructural(Structural matching)

Aunque los métodos de extracción de características, junto con las técnicas estadísticas dan muy buenos resultados de reconocimiento, todavía sigue existiendo un margen de mejora, especialmente para la escritura altamente cursiva y distorsionada. El enfoque de coincidencia estructural trata de llenar este vacío. Su objetivo es, precisamente, que coincidan los trazos individuales entre el kanji de entrada desconocido y los kanjis de las plantillas almacenadas. El kanji de entrada S se modifica de manera óptima para que coincida con el kanji de referencia R. Las técnicas de coincidencia propuestas son generalmente procesos de optimización iterativa, y para mitigar el problema de estar atrapado en mínimos locales, generalmente se adopta una estrategia de grueso a fino (coarse to fine). En esta estrategia se realiza el emparejamiento de estructuras de gran escala o globales durante las fases iniciales En iteraciones sucesivas, la escala se reduce gradualmente de modo que los detalles más finos se emparejan.

T. Wakahara en el año de 1999 [45], aplica una transformación afín global (global affine transformation GAT) al kanji de entrada S de modo que se reduzca al

mínimo la media de las distancias entre puntos vecinos más próximos entre S y el kanji de referencia R. El GAT requerido se obtiene mediante un procedimiento iterativo. Después de la aplicación del GAT, se aplica una transformación afín local (local affine transformation LAT) a cada pixel de S para que coincida mejor con R. Para evitar distorsiones excesivas, la función objetivo que se está minimizando intenta forzar la continuidad en los LAT aplicados a los píxeles cercanos, es decir, los LAT aplicados a píxeles cercanos dentro del mismo vecindario deben ser aproximadamente iguales. El tamaño de la zona se reduce gradualmente en iteraciones sucesivas de manera que se adaptan cada vez más los detalles más finos. El método ha sido aplicado a la coincidencia de los kanjis chinos escritos a mano. El autor propone dos enfoques diferentes para el reconocimiento: el primero es la normalización de un carácter de entrada con respecto a la plantilla de referencia mediante la distorsión de la entrada con el proceso de coincidencia anterior, los vectores de características se extraen del kanji de entrada y se utilizan para el reconocimiento con las técnicas de reconocimiento de patrones estadísticos convencionales, se espera que las características extraídas después del procedimiento de distorsión de la normalización serán más estables y por lo tanto más fiable para la clasificación; el segundo enfoque es utilizar la distancia residual entre el ajuste GAT/LAT del kanji de entrada y la plantilla de referencia para la clasificación. Ambos enfoques se pueden combinar en un sistema multi experto para obtener un mejor rendimiento.

En 1998, C.H. Leung [46] distorsiona sistemáticamente los caracteres de entrada y los de la plantilla para que coincidan entre sí. Se define una función de energía para guiar las distorsionés iterativamente. La función consta de dos términos, uno mide qué tan cerca están los dos patrones y el otro mide la cantidad de distorsión. Por lo tanto minimizando la suma de estos dos términos, los dos patrones se ven obligados a coincidir entre sí sin distorsiones excesivas. Para disminuir el problema de quedar atrapado en mínimos locales, un cierto rango de valores se utiliza para controlar la vecindad de interacción. Inicialmente, una gran vecindad se utiliza con el efecto de que las estructuras estén alineadas de una manera global. El tamaño de la vecindad se reduce gradualmente en sucesivas iteraciones de modo que los detalles más finos sean alineados. Leung reportó que se obtuvo coincidencia satisfactoria incluso para patrones de caracteres complejos. Después de compararlos, se identifican los segmentos de trazos correspondientes de los dos kanjis, y características tales como conectividad, la curvatura, la dirección, el grado de coincidencia, y la puntuación de preclasificación se utilizan para el reconocimiento. El algoritmo fue probado en una base de datos privada con una tasa de reconocimiento del 96,1 %.

El método de coincidencia de relajación (relaxation matching) se adopta en

el articulo “Multi-stroke relaxation matching method for handwritten Chinese character recognition” de M.H.Cheng [47]. La probabilidad P(i,m) del segmento de trazo i en el kanji de entrada que coincide con el segmento m en el kanji de referencia se actualiza en sucesivas iteraciones. Una función de compatibilidad se utiliza para medir qué tan bueno es el emparejamiento entre el segmento i, y su vecino j en el kanji de entrada con el segmento m y su vecino n en el kanji de la plantilla. Si la coincidencia de i para m es constante (o inconsistente) con la coincidencia de j para n, P(i,m) se fortalece (o debilita). En cada iteración, P (i, m) se actualiza de acuerdo con el apoyo de todos los vecinos j y n. Por lo tanto, coincidencias coherentes dominarán después de algunas iteraciones. La fusión y selección de los procedimientos se utilizan para refinar las coincidencias para que varios segmentos en el kanji de entrada puedan coincidir con un único segmento en la plantilla y viceversa. Los experimentos de reconocimiento se llevaron a cabo con 2000 clases de subconjuntos de la base de datos ITRI. Se obtuvo una tasa de reconocimiento del 93.8 %. Sin embargo el método de coincidencia de relajación generalmente requieren una alta complejidad de cálculo en cada iteración.

En el 2001 C.L. Liu, I.J. Kim y J.H. Kim [48], proponen un algoritmo de coincidencia de trazos (stroke matching). Los trazos de cada carácter de referencia se extraen de las trayectorias de la pluma de un sistema entrada de caracteres en línea(on-line). Una base de datos de los atributos de trazo y las relaciones entre trazos para cada kanji de referencia se construye de forma manual. La información incluye el tipo de trazo, longitud del trazo, la orientación, la tolerancia a las variaciones en la longitud y orientación, y otros doce tipos de relaciones entre trazos.

Los tipos de trazo incluyen horizontal, vertical, punto, raya vertical, gancho, etc. Las relaciones entre trazos incluye información como: el punto de inicio del trazo A esta cerca del punto final del trazo B, el punto de inicio del trazó C se encuentra cerca de la parte media del trazo D, etc. En este reconocimiento, el kanji de entrada desconocido es pre-procesado para dar una gráfica de segmentos de línea. Un algoritmo de búsqueda elaborado se utiliza para obtener de los trazos candidatos del kanji de entrada que corresponden a los trazos en el kanji de referencia. Se define una función objetivo que depende de la distancia entre los trazos de entrada es definida, y el trazo de referencia, así como de la compatibilidad entre este par de trazos y otros pares de trazos relacionados. El algoritmo intenta buscar una solución con un costo mínimo de tal manera que las limitaciones impuestas por los tipos de relaciones están satisfechas. Los experimentos en una base de datos de 783 clases de caracteres con 200 muestras por clase producen una tasa de error del 1,5 % y una tasa de rechazo del 0,6 %.

3.2.5. Método híbrido

Un método híbrido que combina distorsión de coincidencia (distortion matching) y la clasificación estadística fue propuesto por Mizukami [49]. Donde una imagen de entrada es preprocesado para dar cuatro planos de imagen, cada dimensión es de 16 x 16 píxeles y cada uno contiene píxeles trazo en solo una de las cuatro direcciones principales. Se define una funcionalidad energética que incluye una función de distorsión como parámetro. Una función de distorsión se ha de buscar de tal manera que distorsione los planos de la imagen de entrada para que coincidan con las plantillas. La función de energía incluye dos términos: uno para medir el grado de coincidencia y el otro mide la suavidad de la función de distorsión. La idea es buscar una función de distorsión que da buenas coincidencias, pero al mismo tiempo sólo distorsiona los planos de imagen de entrada de una manera suave. Un procedimiento de minimización iterativo se utiliza para obtener la función de distorsión. La función de distorsión se utiliza como una transformación de "normalización" aplicada a los planos de imagen de entrada. Se proponen varias medidas de distancia para la clasificación estadística. Una de las distancias propuestas es la distancia euclídea entre los planos de imagen "normalizada"de entrada y los planos de las plantillas dividido por un término de varianza. La varianza se deriva durante la fase de aprendizaje de las distancias entre los caracteres de capacitación "normalizados" y la plantilla. Los experimentos en la base de datos ETL8B dieron una tasa de reconocimiento del 96,7 %.

3.3. Reconocimiento de escritura Coreana

3.3.1. Introducción

Al igual que el reconocimiento de los kanjis chinos, es un desafío para reconocer caracteres coreanos debido al gran número de clases. Además, el grado de similitud entre clases es relativamente alta debido a que 2,350 (de un máximo de 11,172) caracteres comunes son generados a partir sólo 24 grafemas. Debido a estas circunstancias, la mayoría de los investigadores en Corea creen que el problema de reconocimiento de caracteres de Corea es más difícil que el reconocimiento de kanjis chinos. Una buena revisión de las técnicas de reconocimiento de caracteres Coreanos se puede encontrar en el artículo "A Survey on Korean Character Recognition Researches" de Lee y Park [50], y Kim en su artículo

"A Survey on the Off-line Recognition of Handwritten Korean Characters"[51] que resumen las actividades de investigación antes de 1998.

3.3.2. Metodologías

Los métodos más recientes para el reconocimiento de caracteres Hangul se pueden clasificar en enfoque holístico o basado en la segmentación. En un método holístico, un carácter en sí es una unidad de reconocimiento y, por tanto los grafemas no tiene que ser segmentado para reconocimiento de caracteres. En un método basado en la segmentación, por otra parte, un grafema es la unidad de reconocimiento y todos los grafemas deben ser segmentados a partir de un carácter previo a su reconocimiento. Desafortunadamente, ambos enfoques sufren la paradoja del huevo o la gallina.

Debido a la variación cursiva de manuscritos, es difícil segmentar grafemas, y también es difícil de reconocer un carácter completo sin segmentar grafemas. Aparte de estos dos enfoques principales, reconocedores en línea(on-line) también han sido probados en datos fuera de línea (off-line). Convierten la imagen del carácter fuera de línea en un conjunto de datos temporales en línea mediante la estimación de la secuencia del trazo, y luego aplicar un clasificador de alto rendimiento en línea con los datos convertidos. El éxito de este enfoque depende de la consistencia de la secuenciación del trazo.

1. El enfoque holístico

   La mayoría de los enfoques holísticos extraen una característica global, como se ilustra en la Figura 3.2, y adoptan una estrategia de múltiples etapas para disminuir la dificultad de tener un gran número de clases de caracteres. En esta estrategia, se realiza primero una clasificación aproximada o pre-clasificación, y luego toman una decisión final mediante un patrón de análisis estadísticos tales como redes neuronales, coincidencia de trazos, modelos ocultos de Markov (HMM), coincidencia de patrones no lineal, algoritmo de los k-vecinos más cercano, y así sucesivamente [52],[53],[5]. Hay muchas ventajas con el enfoque holístico: el algoritmo de reconocimiento es bastante simple, puede ser útil para las aplicaciones en las que el conjunto de clases de caracteres puede estar restringido, como la automatización postal.

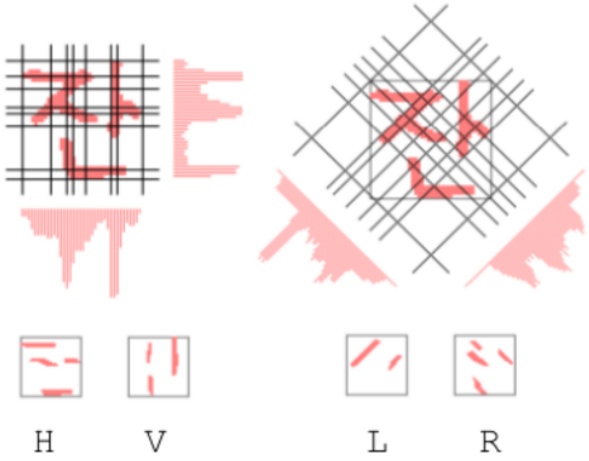


Figura 3.2: Extracción de una característica de malla para un enfoque holístico [5]

Aunque los enfoques holísticos trajeron algo de éxito, tienen inconvenientes graves también. El gran número de clases de caracteres Hangul hace inviable los enfoques estadísticos y de difícil solución. La complejidad computacional para la construcción y reconocimiento del modelo es linealmente proporcional al número de clases de caracteres. Por otra parte, la existencia de muchas clases de caracteres de formas similares hace que los enfoques holísticos sean inviable debido a que los métodos generalmente utilizan las características estadísticas globales de los patrones de los caracteres de entrada. Los enfoques holísticos también tienen una tendencia a confundirse por las distorsiones de formas locales causadas por tocar los grafemas.

La exactitud de reconocimiento de los sistemas de última generación que pertenecen al enfoque holístico varía entre el 80 % y el 86 %, medidos con bases de datos públicas o privadas.

2. Enfoque basado en la segmentación

   Otra parte de la investigación es el enfoque basado en la segmentación. Si los grafemas se segmentan con éxito de un carácter, su reconocimiento sería un mero trabajo debido a que la forma de los grafemas es simple y el número de clases de los grafemas son comparativamente pequeña. Sin embargo, el problema de la segmentación grafema ha demostrado ser otra dificultad, ya que es más difícil que el problema de la segmentación de caracteres en el reconocimiento de palabras Inglés debido a la naturaleza 2-D de la estructura del carácter de Corea. Uno de los métodos para mitigar este problema es obtener retroalimentación de un motor de

reconocimiento durante la segmentación del grafema. Estos enfoques se pueden formular como un problema que encuentra el mejor candidato de segmentación con el puntaje más alto reconocimiento de entre todos los posibles candidatos de segmentación de grafemas. En este enfoque, todos los candidatos para la segmentación del grafema compiten y el mejor es elegido [54],[55],[56] - vea la Figura 3.3 como un ejemplo.

Figura 3.3: Construcción de múltiples modelos de grafemas:
(a) diversas formas estructurales de un grafema y
(b) corresponde al modelo del grafema [6]

El método basado en la segmentación es intuitivo, en el sentido de que refleja el principio de la generación de caracteres de Corea. Además, este método es completo, es decir, puede reconocer caracteres de todas las clases en el conjunto de 11,172 caracteres Coreanos. Una de las desventajas del algoritmo basado en la segmentación es que se hace complejo debido a la segmentación de grafemas.

Este enfoque es más sensible a los ruidos, tales como los ruidos de sal y pimienta (estos ruidos se caracterizan por cubrir de forma dispersa toda la imagen con una serie de pixeles blancos y negros, los ruidos de alta frecuencia son los ruidos de sal mientras que los ruidos de baja frecuencia son los ruidos de pimienta), que distorsionan la estructura de trazos de caracteres. El algoritmo no es flexible para el cambio de objetivo de reconocimiento, ya que trabajaron con un subconjunto de 2,350 clases de caracteres coreanos el cual no puede ser reemplazado por otro subconjunto, es decir, este enfoque es eficaz sólo para el reconocimiento de caracteres coreanos, pero es difícil extenderlo al reconocimiento de caracteres multilingües.

La exactitud de reconocimiento de los sistemas de última generación que pertenecen al enfoque basado en la segmentación varía de 78 % a 86 %, medida con algunas bases de datos públicas o privadas.

3. Aproximación en línea (on-line)

   Motivados por el hecho de que el rendimiento del reconocimiento del clasificador en línea (conforme se va trazando el carácter se hace reconociendo) es generalmente más alta que la de fuera de línea (esta aproximación se hace hasta tener el carácter terminado, normalmente se usan plantillas de estos), algunos métodos han adoptado clasificadores en línea cómo fue Kim en su artículo "Recognition of Off-line Handwritten Korean Characters"[7]. En el que es necesario extraer los trazos y la secuencia de estos para describir un carácter bidimensional de entrada como representación unidimensional. Se define 28 trazos primitivos y se introduce reglas de separación de 300 trazos para recortar a los trazos de caracteres coreanos. Para encontrar una secuencia de trazo usaron códigos de trazo y relaciones entre dos trazos consecutivos. El carácter de entrada se reconoce al buscar en los árboles de reconocimiento de caracteres con la secuencia de trazo. El método fue probado con un conjunto de 1000 caracteres más utilizados escritos por 400 escritores diferentes y mostró la tasa de reconocimiento del 94 %.

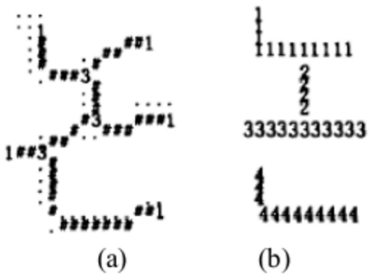


Figura 3.4: Un ejemplo de secuencia de trazo [7]:
(a)Después del preprocesamiento;
(b)resultado de la secuencia de trazo

3.4. Reconocimiento de escritura Japonesa

3.4.1. Introducción

El reconocimiento escritura es un campo muy activo en Japón, haciendo valiosas contribuciones al reconocimiento de escritura japonesa. Sin embargo solo mencionare las investigaciones más reconocidas.

3.4.2. Preprocesamiento (normalización)

El principio de reconocimiento utilizado en OCR se centra en el producto interior en un espacio de características. Por lo tanto, la normalización es esencial en el campo del reconocimiento del kanji para la posición y el tamaño. Sin embargo, este método convencional no es suficiente, por lo que se han desarrollado métodos no lineales llamados "métodos de perturbación" [57],[58], [59],[60].Como fue el caso de Yamada [58], que propone un método de normalización no lineal denominado ecualización de densidad de línea en el que se realiza un nuevo muestreo para equiparar el producto de una densidad de línea local y la inclinación de la muestra. En consecuencia, se homogeniza la densidad de línea en el espacio obteniendo una normalización estable para forma parcialmente irregulares. Sin embargo, estas técnicas pueden tratar con sólo una gama limitada de transformación afín.

3.4.3. Grupos de Investigación

3.4.3.1. El trabajo de Wakahara

Wakahara llevó a cabo experimentos a gran escala para los caracteres utilizados a diario en Japón [61]: Kanji, Kana, y alfanuméricos, basado en sus técnicas de normalización, llamadas Transformación Afín Global (Global Affine Transformation GAT) [62], y Transformación Afín Local (Local Affine Transformation LAT). El punto esencial de GAT es la normalización adaptativa. Es decir, la forma de una imagen 2D de entrada está normalizada de manera adaptativa para dar la mejor coincidencia con las imágenes de referencia 2D de cada candidato. Más específicamente, los esqueletos de las imágenes de entrada y de referencia se usan y se definen como conjuntos de vectores de puntos 2D.

El sistema de reconocimiento global está construida en tres etapas: (i) la clasificación aproximada por el OCR básico, (ii) la normalización de adaptación por el GAT, y (iii) la reclasificación por el OCR básico. Para el OCR básico se utiliza la llamada contribución de dirección periférica extendida (extended peripheral direction contributivity ePDC) [63]. El OCR basado en la función de ePDC logra alrededor una tasa de reconocimiento del 95 % para la escritura a mano de Kanjis de la base de datos ETL9B [64].

Este sistema proporciona K candidatos en la primera etapa de clasificación apro-

ximada, donde la distancia se indica como $D1(k), 1 \leq k \leq K$. Luego, en la segunda etapa, estos K candidatos son adaptativamente normalizado y se obtiene la distancia DNN (k). Finalmente, se alimentan a la ePDC de nuevo y se da la distancia D2 (k). Así, la distancia total D(k) se define como

$$D(k) = [\alpha D1(k) + (1-\alpha)D2(k)] + \beta DNN(k)$$

donde $\alpha \in [0,1]$ y $\beta > 0$. Estos parámetros fueron establecidos para experimentar una parte de la base de datos ETL9B.

El ePDC fue entrenado utilizando una base de datos privada de caracteres japoneses escrito a mano de 3,201 categorías, incluyendo Kanji, Kana, y alfanuméricos, escritos en el estilo cuadrado de 1,350 personas para cada categoría. El grado de variación de la forma o la distorsión por categoría es casi el mismo que el ETL9B. Sin embargo, el número de muestras por categoría es mucho más grande que la de ETL9B, es decir, 1350 vs 200. Como los datos de la prueba, se utilizan los 28, 694 imágenes recolectadas de caracteres japoneses escritos a mano totalmente sin restricciones escritas en papel blanco por 300 personas. Los resultados se analizaron según la complejidad de la forma entre Kanji, kana, y alfanumérico de la siguiente manera:

La tasa de reconocimiento global es del 87,4 %. La tasa de reconocimiento original era 85,8 % en el que no se aplicó la GAT. La normalización no siempre da un mejor resultado para cada categoría. Es decir, tanto efectos "positivas" como "negativos" aparecen. Como era de esperar, GAT es eficaz para Kana y alfanuméricos en particular. El tiempo de reconocer un carácter es 0.12 segundos con GAT y 0.01 segundos sin GAT en un Pentium III de 650 MHz

3.4.3.2. Trabajo del grupo de Miyake-Kimura

El grupo Miyake-Kimura llevó a cabo un experimento de reconocimiento de Kanjis a gran escala utilizando la base de datos ETL9 [65]. Ellos anteriormente habían hecho un sistema, el cual fue mejorado con un preprocesamiento. La motivación de este preprocesamiento es que los Kanjis tienen algunos grupos de formas mutuamente similares como son los radicales. Además para los Kanjis complejos, su preprocesamiento es difícil debido a la drástica reducción de la dimensión con una fuerte borrosidad, aunque es muy efectivo mantener la naturaleza topológica del plano 2D. Utilizaron característica direccional para introducir efecto borroso (blurring).

En la detección de la dirección, se utilizaron tanto el código de cadena y método de gradiente. Por lo tanto, la resolución del nuevo sistema se aumenta a 49 x 49, aunque el tamaño final se mantiene la misma que antes. Las mejoras para estos cambios se resumen de la siguiente manera:

	Código de cadena (%)	Gradiente (%)
5 X 5 GF [1]	99.14	99.29
13 x 13 GF	99.23	99.31
5 x 5 GF + 13 x 13 tamaño de la muestra	99.03	99.21
31 x 31 GF + 49 x 49 tamaño de la muestra	99.26	99.34

Por otra parte, intentaron la mejora en términos de normalización mediante la construcción de una variación de absorción de la matriz de covarianza.

Las mejoras se muestran a continuación:

	Vieja matriz de covarianza	Nueva matriz de covarianza
Código de cadena	99.07	99.22
Gradiente	99.26	99.38
Gradiente + 49 X 49 tamaño de la muestra +31 x 31 GF	99.31	99.41

Para Hiragana, la tasa de reconocimiento es del 98.4 %. La velocidad de procesamiento es bastante alta, 14.2 caracteres por segundo para el nuevo sistema. El viejo sistema tiene 20.8 %. Como el procedimiento de ensayo se utiliza el método de rotación. La base de datos ETL9B consta de 200 muestras por categoría. Así que las muestras se dividen en 10 subconjuntos. Al probar un subconjunto particular, los subconjuntos restantes se utilizan como muestras de aprendizaje. El procedimiento repite 10 veces y el promedio se tomó como los resultados de 10 subgrupos analizados.

[1] Los autores toman como referencia para esta comparación una misma dirección obtenida como se muestra en la figura 3.1 usando método de extracción de características, reportan el desempeño con la dirección GF sin pérdida de generalidad.

3.5. Métodos de reconocimiento de kanjis en diccionarios actuales

Como ya se mencionó en el capítulo 1, existen métodos de búsqueda en diccionarios tales como es por el número de trazos, radicales, lectura, veamos más a detalle estos para tener un mayor entendimiento [66]:

Número de trazos: Un factor a considerar cuando se aprende kanjis es el número de trazos ya que la cantidad de trazos es finita para cualquier kanji, los kanjis pueden tener sólo un trazo como es el caso del número uno, hasta ochenta y cuatro trazos que es el kanji más largo, sin embargo la media es de doce trazos, dado que en un diccionario decente de japonés existen al menos cuatro mil kanjis, si se promedia la media veremos que reducimos la consulta a trescientos treinta y tres kanjis.

Radical principal: Un radical es un sub-elemento común que podemos encontrar en diferentes kanjis. Cada kanji tiene un radical o es un radical por sí mismo. Los radicales expresan la naturaleza del kanji, es decir, nos da una pista de su origen, significado o pronunciación.

Por ejemplo: 貝 significa concha, en la antigüedad la moneda eran las conchas he de ahí que kanjis como 買 (comprar), 貸 (prestar), 費 (gasto), 資 (capital monetario) tenga una concha presente.

La lista oficial de kanjis contiene 214 radicales, lo que convierte la media obtenida al consultar un radical en el diccionario de cuatro mil kanjis a diecinueve aproximadamente (aunque el número de kanjis por radical está muy poco equilibrado). Es difícil reconocer cual el radical ya que en muchas ocasiones, el elemento de más peso puede considerarse el que más espacio ocupa en un cuadrado imaginario que rodea al kanji, el que está en la posición más privilegiada, el que aporta mayor significado, o el que, en competencia con otros radicales, tiene "prioridad" en la lista oficial. Para facilitar esto, los radicales están divididos en 7 grupos, dependiendo de la posición que ocupen en el kanji: hen, tsukuri, kanmuri, ashi, tare, nyou, kamae.

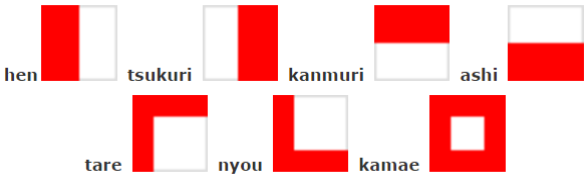


Figura 3.5: Posiciones de los radicales

En la figura 3.6 se puede ver algunos ejemplos:

Figura 3.6: Ejemplo de uso de los radicales

Lectura: Los kanjis japoneses tienen principalmente dos clasificaciones para sus lecturas: las lecturas “kun” y las lecturas “on”. Estos nombres categorizan a las lecturas según sus fonemas procedan del japonés o del chino respectivamente. Cuando la ciencia y la tecnología china entraron en Japón (lo cual incluye a los kanjis), también lo hicieron cientos de palabras de origen chino que en japonés simplemente no existían. Así que lo que hicieron los japoneses fue elegir kanjis para representar sus propias palabras, conservando sus lecturas chinas para palabras que en su idioma no existían.

En la mayoría de los casos, aunque con muchísimas excepciones, las palabras de origen japonés suelen estar representadas por un solo kanji y las de origen chino por una composición de éstos. Por lo general es más fácil aprender las pronunciaciones kun de los kanjis porque los vocablos japoneses son más largos y variados que los chinos. Éstos últimos tienden a ser monosílabos y de sonidos alargados, como “shou”, “ryou” o “sei”., de forma que se repiten mucho entre los kanjis. Mientras tanto, los japoneses tienden a ser de dos sílabas con algún fonema fuerte, como “umi”, “kome” o “mado”.

Ocurre muy comúnmente que leemos un kanji del cual conocemos su lectura “kun”, pero no recordamos su lectura “on”, o no sabemos cuál de sus lecturas on se utiliza en esa palabra en concreto.

Buscar en este tipo de casos un kanji por su pronunciación (sobre todo si es kun) suele ser bastante efectivo, aún teniendo en cuenta que el número de palabras homónimas en japonés enorme.

Otra cosa interesante sobre las lecturas es que, en muchas ocasiones, kanjis compuestos por otros kanjis suelen heredar entera o parcialmente las lecturas “on” de sus radicales.

Por ejemplo: El kanji 銀(plata) lectura “ぎん(gin)” esta sub-compuesto del kanji 金(oro) lectura “きん(kin)”

Este hecho es especialmente útil cuando sobre la marcha tenemos que hacer suposiciones sobre cómo se lee un kanji que no conozcamos.

3.6. Resumen

En este capítulo se abordan los métodos desarrollados por las culturas asiáticas (China, Corea y Japón) para atender la problemática del reconocimiento de su propia escritura.

Podemos ver que en China desarrolla el método de *función de extracción y clasificación de kanjis*, para este método es fundamental la posición, dirección y densidad de los trazos que componen a los kanjis para así identificarlos. También desarrollan el método de *coincidencia estructural*, que si bien el anterior método funcionaba de manera aceptable, este último fue desarrollado para escrituras más distorsionadas. Para este método es importante contar con una base de datos de la información de los tipos de trazos horizontal, vertical, punto, gancho, etc.

Por otro lado en Corea desarrolla el método de *enfoque holístico*, este método está basado en la categorización de clases de caracteres Hangul, y el método *basado en la segmentación*, donde segmentan grafemas, sin embargo estos son muy impreciso por lo que desarrollaron el método de *aproximación en línea*, donde extraen los trazos y la secuencia de cómo fueron escritos.

En Japón se propone por primera vez un preprocesamiento conocido como normalización, los métodos desarrollados por ellos se basan en la obtención de la posición y tamaño de los kanjis, también plantearon el análisis de estos a partir de sus radicales.

Al final de este capítulo, se muestra las técnicas de reconocimiento de kanjis en diccionarios actuales, ya sean digitales o de papel, los cuales están basados en el número de trazos, el radical principal y lectura.

En base a los rasgos más destacados de estas investigaciones; posición, extracción y normalización de los trazos de los kanjis que fueron presentados en este capítulo, en el siguiente capítulo se describe la metodología desarrollada.

4 Reconocimiento de kanjis basado en trazos.

Retomemos el problema inicial por el cual se está proponiendo una aplicación. Un estudiante del idioma japonés puede tener dificultades para aprender este idioma por la gran cantidad de kanjis y los diversos significados que puede tener cada uno de estos. El uso de diccionarios convencionales (de papel) pueden hacer más lento y tedioso el aprendizaje por el tiempo de búsqueda del kanji. En este trabajo se propone una tecnología potencialmente interesante capaz de reconocer kanjis a partir de la identificación de los trazos introducidos por el usuario (se esperaría que fuese estudiante del idioma japonés) para que en un futuro no muy lejano se pueda elaborar un diccionario japonés-español, pretendiendo agilizar la búsqueda de kanjis, de tal manera que el usuario obtenga una respuesta prácticamente inmediata a la petición de este.

Como se pudo observar en los capítulos anteriores el reconocimiento de caracteres, en específico el reconocimiento de kanjis, se ha abordado mediante diversas metodología y se ha obtenido diferentes resultados.

En este trabajo se aborda el reto de reconocimiento de kanjis combinando dos técnicas:

1. El uso de mapas autoorganizados, también llamados redes neuronales auto organizadas, para el reconocimiento de trazos.

2. El uso de árboles de decisión, para la identificación de kanjis.

4.1. Mapas autoorganizados para el reconocimiento de trazos

4.1.1. Redes Neuronales

Las redes neuronales son un modelo artificial y simplificado del cerebro, en este modelo se trata de emular ciertas características propias del ser humano, como la capacidad de memorizar y de asociar hechos.

Es decir, las redes neuronales tratan de asemejar la capacidad del cerebro de acumular reglas de aprendizaje que se van adquiriendo a través del tiempo (lo que comúnmente llamamos “experiencia”).

La unidad de procesamiento de información que es fundamental para la operación de una red neuronal es conocida como “neurona”.

Las redes neuronales emplean una interconexión entre neuronas, las cuales almacenan conocimiento. Según Haykin [67] esta idea se asemeja al cerebro en dos aspectos:

- El conocimiento es adquirido por la red desde su entorno a través de un proceso de aprendizaje.
- Las fuerzas de conexión entre las neuronas, conocidas como pesos sinápticos, se usan para almacenar los conocimientos adquiridos.

El proceso usado para desarrollar el proceso de aprendizaje es llamado algoritmo de aprendizaje, cuya función es modificar los pesos sinápticos de la red de manera ordenada para lograr un objetivo deseado.

La estructura general de una red neuronal consiste de una o más entradas. Existe un primera primer capa, que consiste en grupo de neuronas que generan una salida igual a las entradas presentadas. En esta capa es conocida como capa de entrada y en esta no existe ningún tipo de procesamiento [68]. Además cuenta con una o más neuronas de salida. También conocido como capa de salida, en esta capa cada neurona de entrada, es procesada y finalmente genera una salida.

La figura 4.1 muestra las neuronas de entrada conectadas a las neuronas de sa-

lida a través de una "caja negra", la cual es la encargada de llevar a cabo el proceso de aprendizaje. El modelo exacto de red es determinado por la naturaleza de esta caja negra, la cual puede estar formada por una o más capas de neuronas, que se conocen como capas ocultas o intermedias.

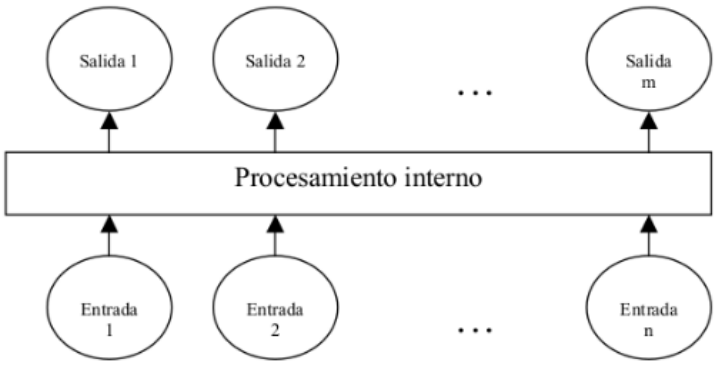


Figura 4.1: Estructura básica de una red neuronal

El proceso mediante el cual la red neuronal será capaz de almacenar conocimiento a partir de las entradas presentadas se conoce como entrenamiento o regla de aprendizaje.

Una red neuronal puede ser entrenada de dos formas; entrenamiento supervisado y entrenamiento no supervisado. En el primero, cada muestra (conjunto de entradas presentadas a la red en un tiempo dado) dentro del grupo de entrenamiento tendrá información tanto de las entradas como de las correspondientes salidas, las cuales se conocen como "salidas deseadas". Para cada muestra, se comparan las salidas calculadas por la red con las presentadas en la muestra de salidas deseadas en un orden aleatorio. Posteriormente se actualizan los pesos sinápticos de manera que reduzca una medida de error en los resultados de la red. Llamamos "época" o "iteración" a cada vez que es presentado a la red un conjunto de muestras de entrenamiento y se actualizan los pesos sinápticos. Las épocas son repetidas varias veces hasta que la red alcance un desempeño satisfactorio.

La segunda forma de entrenamiento se muestra en la siguiente sección.

4.1.2. Redes no supervisadas

La redes no supervisadas son llamadas así por su regla de aprendizaje de tipo no supervisado. En este tipo de regla de aprendizaje se basa en que solo se suministra a la red los datos de entrada (es decir, no se le pasa información de una salida esperada), para así extraer las características esenciales, es decir, debe encontrar caracteristicas, regularidades, correlaciones o categorías que puedan establecer entre los datos de entrada. Dado que no hay una supervisión que le indique a la red la respuesta que debe generar en una entrada concreta, por lo que este puede tener varias posibilidades en cuanto a la interpretación de las salidas, que dependen de su estructura y de la regla de aprendizaje empleado. En algunos casos, la salida representa la similitud entre las entradas y la información que se le han mostrado en el pasado. En otro caso se hace el establecimiento de categorías, indicando a la salida a qué categoría pertenece la información presentada en la entrada. Siendo la propia red quien debe encontrar las categorías apropiadas de correlacionar la información presentada.

Finalmente, algunas redes con aprendizaje no supervisado realizan un mapeo de características, obteniendo en las neuronas de salida una disposición geométrica que representa un mapa topográfico de las características de los datos de entrada, de tal forma que si se presentan a la red informaciones similares, siempre sean afectadas neuronas de salida próximas entre sí, en la misma zona del mapa [67]. En esta categoría es en la que se clasifican los mapas autoorganizados, que se describen con mayor detalle en la siguiente sección.

4.1.3. Mapas autoorganizados

Los mapas autoorganizados [69](Self Organizing Maps, SOM), fueron introducidos en el año de 1982 por Teuvo Kohonen. Esta red emplea un tipo de aprendizaje no supervisado de tipo competitivo.

En el aprendizaje competitivo las neuronas compiten unas con otras con el fin de llevar a cabo una tarea específica. Aquí se pretende que cuando se presente a la red un patrón de entrada de la muestra de entrenamiento, sólo una de las neuronas de salida (o un grupo de vecinas) se active. Esto es, las neuronas compiten por activarse, mientras el resto son forzadas a sus valores de respuesta mínimos.

El modelo SOM está compuesto por solo dos capas de neuronas como pode-

mos ver en la figura 4.2. La capa de entrada (formada por N neuronas, una por cada señal de entrada) encargada de recibir y propagar a la capa de salida las señales de entrada procedentes del exterior. La capa de salida (formada por M neuronas) encargada de procesar la información y formar un mapa de rasgos. Las neuronas de la capa de salida se organizan, generalmente, en forma de mapa bidimensional aunque pueden usarse capas de diferentes dimensiones.

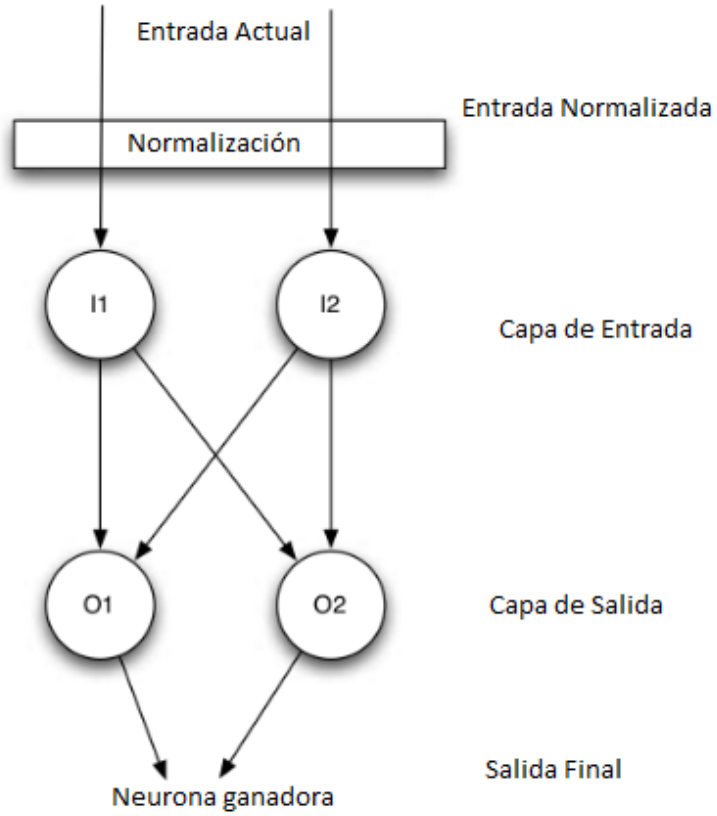


Figura 4.2: Mapa autoorganizado [8]

Para examinar cómo es el procesamiento de información, se debe comprender cuál es el proceso de cálculo que lleva la red neuronal. De acuerdo con Heaton, Estos pasos incluyen: normalización del vector de entradas, cálculo de la neurona de salida, mapeo de números en formato bipolar (opcional), selección de la neurona ganadora, entrenamiento de la red, determinación de la razón de aprendizaje y ajuste de pesos.

4.1.3.1. Normalización de la entrada

La red neuronal requiere que la entrada sea normalizada. En este caso el rango de variables de entrada debe ser [-1,1] y cada una de las variables de entrada deben usar este rango libremente. Según Heaton (2005)[8] si una o varias neuronas están en el rango [0,1] se perjudica el desempeño de la red.

La normalización de la entrada se lleva a cabo mediante dos métodos:

- Normalización Multiplicativa
  Para implementar la "normalización multiplicativa", se debe calcular la longitud del vector de los datos de entrada o vector. Esto se hace sumando los cuadrados del vector de entrada y luego tomando la raíz cuadrada de este número, como se muestra en la siguiente ecuación.

$$f = \frac{1}{\sqrt{\sum_{i=0}^{n-1} x_i^2}}$$

- Normalización eje z
  A diferencia del algoritmo multiplicativo para la normalización, el algoritmo de normalización del eje z no depende de los propios datos reales. En su lugar, los datos sin procesar se multiplican por una constante. Para calcular el factor de normalización utilizando la normalización del eje z, utilizamos la ecuación:

$$f = \frac{1}{\sqrt{n}}$$

  Como puede verse en la ecuación anterior, el factor de normalización sólo depende del tamaño de la entrada, designado por la variable n. Sin embargo, no queremos hacer caso omiso de las entradas reales por completo; por lo que se crea una entrada sintética, basada en los valores de entrada. La entrada sintética se calcula usando la ecuación:

$$s = f\sqrt{n - l^2}$$

  Donde n representa el tamaño de la entrada, f el factor de normalización, l es la longitud del vector, s la entrada sintética que se añadirá al vector de entrada que se presentó a la red neuronal.

4.1.3.2. Cálculo del rendimiento de cada neurona

Para calcular la salida, se debe tener en consideración el vector de entrada y los pesos de conexión. Una de las maneras más usuales para relacionar estos es hacer el producto punto entre neuronas de entrada y sus pesos.

$$x_i \cdot w_i$$

Donde x_i es el vector de neuronas de entrada y w_i son los pesos de la conexión.

Posteriormente se debe normalizar la salida multiplicandola por el factor de normalización que fue determinado en el paso anterior.

4.1.3.3. Eligiendo la neurona ganadora

Una vez calculado el valor de la primera neurona de salida, tomamos inicialmente este valor como el de la neurona ganadora sin embargo se va comparando con el resto de la entrada y se toma el mejor, es decir, comparamos el valor actual con el nuevo valor presentado y conservamos el valor mayor. Al final de la iteración con toda la entrada obtenemos el valor de la neurona ganadora.

Para elegir la neurona ganadora, seleccionamos la neurona que produce el mayor valor de salida. De esta manera obtenemos la salida del mapa autoorganizado, siendo los pesos entre las neuronas de entrada y salida los que determinan la salida. En la siguiente sección veremos cómo estos pesos pueden ser ajustados para producir una salida que sea más adecuada para la tarea deseada. El proceso de entrenamiento modifica estos pesos y se describe en la siguiente sección.

4.1.3.4. Cómo aprende un mapa autoorganizado

Para entrenar un mapa autoorganizado debemos involucrar varios pasos en el proceso de formación. En general implica pasar por varias épocas hasta que el error del mapa autoorganizado es inferior a un nivel aceptable. Se debe calcular la tasa de error para el mapa autoorganizado y ajustar los pesos para cada época. También se debe determinar cuándo no son necesarias épocas adicionales para seguir entrenando la red neuronal.

El proceso de formación para el mapa autoorganizado es competitivo. Para cada conjunto de entrenamiento, una neurona "ganará". Esta neurona ganadora tendrá su peso ajustado para que reaccione aún más fuertemente a la entrada la próxima vez que lo vea. A medida que diferentes neuronas ganan por diferentes patrones, su capacidad de reconocer ese patrón particular aumentará.

El mapa autoorganizado es entrenado repitiendo épocas hasta que suceda una de dos cosas: Si el error calculado está por debajo de un nivel aceptable, el proceso de entrenamiento es completo. Si, por el contrario, la tasa de error ha cambiado sólo una cantidad muy pequeña, este ciclo individual será abortado sin que se produzcan otras épocas. Si se determina que el ciclo debe ser abortado, los pesos son inicializados con valores aleatorios y comenzará un nuevo ciclo de entrenamiento.

4.1.3.5. Proceso de entrenamiento

El entrenamiento ocurre en varios pasos y numerosas iteraciones, de la siguiente manera [70]:

1. Designar número de neuronas y establece un valor de error máximo.

2. Inicializar los pesos sinápticos. Existen tres formas de realizar éste proceso:

   - Aleatoria, donde los pesos son inicializados con valores aleatorios pequeños (esta es la forma que se utilizó en la implementación de la aplicación).

   - Por muestras, donde los pesos son inicializados empleando muestras aleatorias del grupo de datos de entrada.

   - Lineal, donde los pesos son inicializados en una manera ordenada a lo largo del subespacio lineal que pasa por los dos vectores propios principales del grupo de datos de entrada. Los vectores propios pueden ser calculados usando el procedimiento Gram-Schmidt [Hoffman et. al, 1973]

3. Un vector se escoge del juego de datos de entrenamiento y es presentado a la matriz.

4. Se presenta un patrón p de entrada $x_p : x_{p1}, ..., x_{pi}, ..., x_{pN}$, el cual se transmite directamente de la capa de entrada a la capa de salida, para posteriormente buscar el vector de pesos w_j que mejor empareja al vector x_p . Esto se hace calculando

$$w_j^T x$$

y seleccionar el mayor. Así tendremos la localización donde la vecindad de neuronas excitadas será centrada, es decir, la neurona ganadora, determinada como la Mejor Unidad de Coincidencia (MUC). El mejor emparejamiento se logra calculando el mínimo de la distancia euclidiana entre x_p y w_j.

5. El radio de la MUC es calculado; este valor inicia con un valor elevado, generalmente el radio de la matriz, pero disminuye en cada iteración del entrenamiento al igual que error calculado. Todas las neuronas que se encuentren dentro de este radio, se consideran que pertenecen a la vecindad de la MUC.

6. Para cada uno de las neuronas encontradas en el vecindario de la MUC, se ajustan los pesos con el fin de hacerlos "más similares" al vector de entrada. Entre más cerca se encuentre una neurona a la Mejor Unidad de Coincidencia, sus pesos serán alterados de mayor manera.

7. Repetir el paso 2 por N iteraciones o cuando el error calculado es aceptable (es decir, es menor al valor del error máximo establecido).

4.1.3.6. Tasa de aprendizaje

La tasa de aprendizaje es una variable que es utilizada por el algoritmo de aprendizaje para ajustar los pesos de las neuronas, este puede tener valor constante o variable durante el proceso, sin embargo este debe tener un valor positivo menor que 1; normalmente se toma un valor entre 0.4 y 0.5 (en la implementación de la aplicación se usó 0.5) y este es representado con la letra griega alpha (α).

Por lo general, establecer la velocidad de aprendizaje a un valor más alto hará que el entrenamiento progrese más rápidamente. Sin embargo, la red puede fallar al converger si la tasa de aprendizaje es establecida en un número que sea demasiado alto. Esto se debe a que las oscilaciones de los vectores de peso serán demasiado grandes para que los patrones de clasificación nunca emerjan.

Otra técnica es empezar con una tasa de aprendizaje relativamente alta y disminuir esta tasa a medida que avanza el entrenamiento. Esto permite un rápido entrenamiento inicial de la red neuronal que luego se "ajusta" a medida que avanza el entrenamiento.

4.1.3.7. Ajuste de pesos

La memoria del mapa de auto-organización es almacenada dentro de las conexiones ponderadas entre la capa de entrada y la capa de salida. Los pesos son ajustados en cada época. Una época se produce cuando los datos de entrenamiento se presentan al mapa autoorganizado y los pesos son ajustados en base a los resultados de estos datos. Los ajustes a los pesos deben producir una red que produzca resultados más favorables la próxima vez que se presenten los mismos datos de entrenamiento.

Eventualmente, el retorno de estos ajustes de peso disminuirá hasta el punto de que ya no es valioso para continuar con este conjunto particular de pesos. Cuando esto ocurre, toda la matriz de pesos se restablece a nuevos valores aleatorios; comenzando así un nuevo ciclo. La matriz de peso final que se utilizará será la mejor matriz de peso de cada uno de los ciclos. Ahora examinaremos cómo se transforman los pesos.

El método original para calcular los cambios en los pesos, que fue propuesto por Kohonen en 1984, comúnmente conocido como el método aditivo. Este método utiliza la siguiente ecuación:

$$w^{t+1} = \frac{w^t + \alpha x}{\|w^t + \alpha x\|}$$

Donde la variable x es el vector de entrenamiento que fue presentado a la red, la variable w^t es el peso de la neurona ganadora, y la variable w^{t+1} es el nuevo peso.

Este método aditivo generalmente funciona bien para la mayoría de los mapas autoorganizados; Sin embargo, en los casos en los que el método aditivo muestra inestabilidad excesiva y falla en la convergencia es utilizado un método alternativo. Este método se denomina método sustractivo. El método sustractivo (propuesto por Heaton en 2005) utiliza las siguientes ecuaciones:

$$e = x - w^t$$
$$w^{t+1} = w^t + \alpha e$$

Estas dos ecuaciones describen la transformación básica que ocurrirá en los pesos de la red.

4.1.3.8. Calculo del error

Antes de que podamos entender cómo calcular el error para la red neuronal, primero debemos definir "error". En el entrenamiento supervisado el error es la diferencia entre la salida anticipada de la red neural y la salida real de la red neuronal. Como ya hemos mencionado en el entrenamiento no supervisado, no hay salida anticipada; por lo que el error que se calcula no es un verdadero error, o al menos no un error en el sentido normal de la palabra.

El propósito del mapa autoorganizado es clasificar la entrada en varios conjuntos. El error del mapa autoorganizado, por lo tanto, proporciona una medida de lo bien que la red está clasificando la entrada en los grupos de salida.

El error se define como la distancia euclidiana entre la neurona ganadora y su peso de conexión. Este valor debe ser minimizado a medida que avanza el aprendizaje. Esto da una aproximación general de lo bien que el mapa autoorganizado ha sido entrenado.

4.2. Árboles de decisión n-arios

Un árbol n-ario es una estructura de datos constituida por un elemento denominado nodo raíz y por una o más estructuras que se le asocian, que son a su vez árboles n-arios. Estas estructuras asociadas se denominan sub-árboles n-arios. Por lo que podemos decir un árbol n-ario es una estructura en la que cada elemento puede tener cualquier número de subárboles n-arios asociados. En este caso el orden de los subárboles no es importante, en el sentido de que no es necesario saber cuál es el primero o el último, sino simplemente saber que es un subárbol [71].

4.2.1. Generalidades y partes de un árbol

En esta sección se presentan algunas definiciones que serán útiles para describir los árboles n-arios y sus propiedades:

Nodo Un árbol n-ario es un conjunto de elementos, cada uno de los cuales se denomina nodo. Un árbol n-ario puede tener cero nodos, en ese caso se dice que el árbol es vacío. Un árbol n-ario puede tener un solo nodo, y en ese caso se dice que solo existe la raíz del árbol o puede tener un número finito de nodos.

Se puede identificar diferentes tipos de nodos:

Nodo hijo Cualquiera de los nodos apuntados por uno de los nodos del árbol. Por ejemplo en el árbol de la figura 4.3 se puede ver que que B, C y D son nodos hijos de A.

Nodo padre nodo que contiene un puntero al nodo actual. Por ejemplo, A es el nodo padre de B.

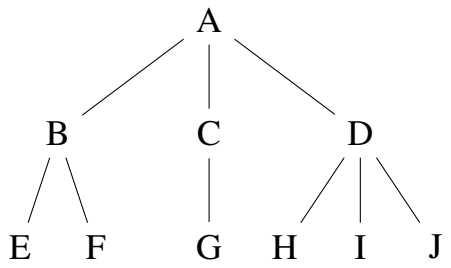


Figura 4.3: Representación de un árbol de decisión n-ario

Una característica importante de los árboles es que cada nodo sólo puede ser apuntado por otro nodo, es decir, cada nodo sólo tendrá un padre. Esto hace que los árboles sean fuertemente jerarquizados y es lo que le da la apariencia de árbol.

En cuanto a la posición dentro del árbol n-ario se tiene:

Nodo raíz, no tiene nodo padre y tiene cero o más nodos hijos.

Nodo interno, cada uno de los cuales tiene exactamente un nodo padre y uno o más nodos hijos.

Nodo hoja, cada uno de los cuales tiene exactamente un nodo padre y no tiene nodos hijos, se podría decir que sus subárboles asociados son vacíos.

Siguiendo el ejemplo de la figura 4.3, el nodo raíz es A, los nodos internos serian B, C, D y los nodos hojas seria E, F, G, H, I, J.

Nivel La distancia medida en nodos que se encuentra entre cualquier nodo del árbol a la raíz, en otras palabras el nivel de la raíz siempre será cero y el de sus hijos uno. Así sucesivamente para los demás nodos. Por ejemplo el nodo D se encuentra en el nivel 1, y el nodo H se encuentra en el nivel 2.

Altura La altura se define como el nivel del nodo de mayor nivel. En el ejemplo el nivel del árbol es dos.

Rama Una rama es un camino que lleva del nodo raíz a un nodo hoja del árbol. Eso quiere decir que en un árbol n-ario existe el mismo número de ramas que de hojas. También se dice que la altura de un árbol es la longitud de la rama más larga del árbol. Por ejemplo A, D y H son un una de las ramas del árbol anterior.

4.2.2. Árboles de decisión

Hasta ahora hemos hablado de los árboles n-arios sin embargo no se ha mencionado nada acerca de la toma de decisión. Por lo que es muy importante mencionar que los árboles de decisión son la técnica para la toma de decisiones más sencilla. Son rápidos, fáciles de implementar y son fáciles de entender. Tienen la ventaja de ser muy modular y fácil de crear [72].

4.2.2.1. Funcionamiento visto con ejemplos

Un árbol de decisiones se compone de puntos de decisión conectados. El árbol tiene una decisión de partida, su raíz. Para cada decisión, a partir de la raíz, se selecciona una de un conjunto de opciones disponibles.

Cada elección se hace en base a un conocimiento previo. Los árboles de decisión se usan a menudo como mecanismos de decisión simple.

El algoritmo continúa a lo largo del árbol, haciendo elecciones en cada nodo de decisión hasta que el proceso de decisión no tenga más decisiones que considerar. En cada hoja del árbol se realiza una acción. Cuando el algoritmo de decisión llega a una acción, esa acción es llevada a cabo inmediatamente.

La mayoría de los nodos de árboles de decisión toman decisiones sencillas, normalmente con sólo dos posibles opciones. Tomemos como ejemplo a un personaje el cual debe hacer una toma de decisiones sobre qué debe hacer respecto a un enemigo, y sus posibles acciones, como se pueden ver en la figura 4.4.

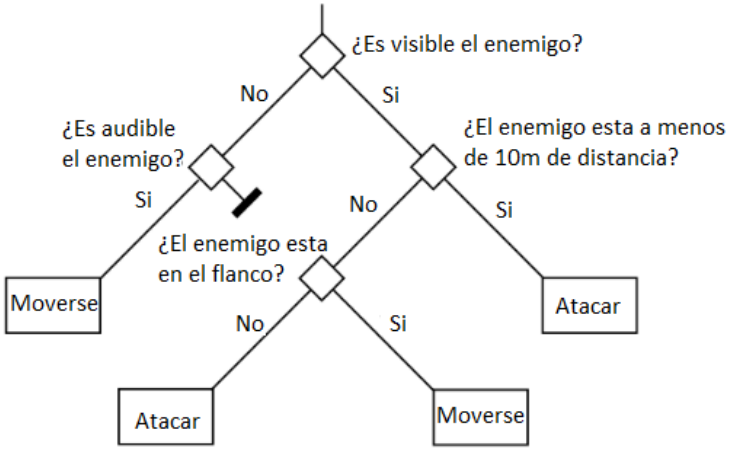


Figura 4.4: Arbol de decisión

Observe que una misma acción puede colocarse al final de varias ramas. En la figura 4.4 el personaje elegirá atacar a menos que no pueda ver al enemigo o que esté flanqueado. La acción de ataque está presente en dos hojas.

La Figura 4.5 muestra el mismo árbol de decisión con una decisión tomada. El camino tomado por el algoritmo se resalta, mostrando la llegada a una sola acción, que luego puede ser ejecutada por el personaje.

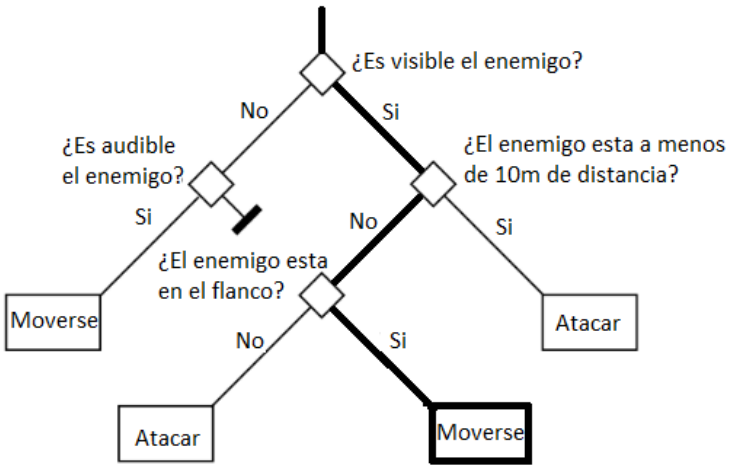


Figura 4.5: Árbol de decisión con decisión realizada

4.2.2.2. Decisiones

Las decisiones en un árbol son simples. Normalmente comprueban un valor único y no contienen ninguna lógica booleana (es decir, no unen pruebas en conjunto con AND y OR). Dependiendo de la implementación y los tipos de datos de los valores almacenados, pueden ser posibles diferentes tipos de pruebas, como se puede ver en la tabla 4.1.

Tipo de Datos	Decisiones
Booleana	Valor es true
Enumeración (es decir, un conjunto de valores, sólo uno de los cuales podría ser permitido)	Corresponde a uno de un conjunto dado de valores
Valor numérico (entero o punto flotante)	El valor está dentro de un rango dado

Tabla 4.1: Tipos de datos

Además de los tipos primitivos, en las aplicaciones orientados a objetos es común permitir que el árbol de decisión acceda a métodos de instancias. Esto permite que el árbol de decisiones delegue un procesamiento más complejo al código optimizado y compilado, al tiempo que sigue aplicando las decisiones simples de la tabla anterior al valor devuelto.

4.2.2.3. Complejidad de la Decisión

Debido a que las decisiones están integradas en un árbol, el número de decisiones que deben ser consideradas es generalmente mucho menor que el número de decisiones en el árbol. La figura 4.6 muestra un árbol de decisiones con 15 decisiones diferentes y 16 acciones posibles. Después de ejecutar el algoritmo, vemos que sólo cuatro decisiones son consideradas.

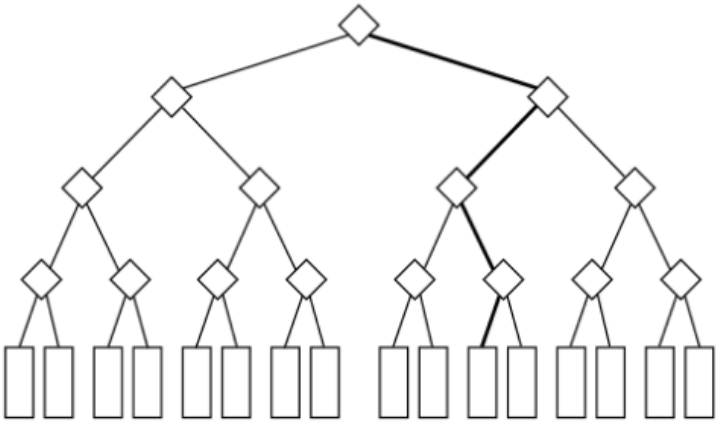


Figura 4.6: Ancho de un árbol de decisión

En el ejemplo anterior, las decisiones elegirán entre dos opciones. Esto se llama un árbol binario de decisión. Sin embargo no hay ninguna razón por la que no pueda construir un árbol de decisiones para que las decisiones puedan tener cualquier número de opciones.

Imagine tener un guardia en una instalación militar. El guardia necesita tomar una decisión basada en el estado de alerta actual de la base. Este estado de alerta puede ser uno de un conjunto de estados: “verde”, “amarillo”, “rojo” o “negro”, por ejemplo. Utilizando el árbol de toma de decisiones binario simple descrito anteriormente, podríamos construir el árbol en la Figura 4.7 para tomar una decisión.

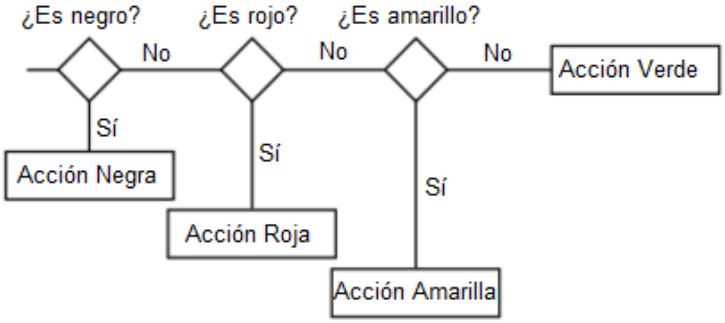


Figura 4.7: Árbol de decisión binario profundo

El mismo valor (el estado de alerta) se debe comprobar tres veces. Esto no sera un problema si hace los chequeos de estado más probables primero. Aun así, el árbol de decisión puede tener que hacer el mismo trabajo varias veces para tomar una decisión.

Podríamos permitir que nuestro árbol de decisión tenga varias ramas en cada punto de decisión. Con cuatro ramas, el mismo árbol de decisiones ahora se parece a la Figura 4.8.

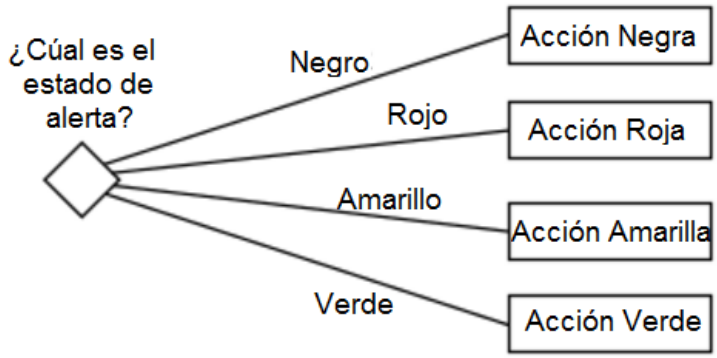


Figura 4.8: Árbol de decisión con cuatro ramas

Esta estructura sólo requiere una decisión, y es más eficiente.

Un árbol de decisiones es más simple con múltiples ramas y la velocidad de implementación no suele ser significativamente diferente con respecto a un árbol binario.

4.2.2.4. Rendimiento de los árboles de decisión

El árbol de decisiones es eficiente porque las decisiones son típicamente muy simples. Cada decisión hace una sola prueba. No ocupa demasiada memoria, y su rendimiento es lineal con el número de nodos visitados.

El rendimiento es lineal ya que si asumimos que cada decisión toma una cantidad de tiempo constante y que el árbol está equilibrado, entonces el rendimiento del algoritmo es $O(log_2 n)$, donde n es el número de nodos de decisión en el árbol.

Es muy común que las decisiones tomen un tiempo constante. Las decisiones de ejemplo que dimos en la tabla al comienzo de la sección son todos procesos de tiempo constantes. sin embargo hay algunas decisiones que toman más tiempo. Una decisión que comprueba si algún enemigo es visible, por ejemplo, puede implicar verificaciones complejas de rayos a través de la geometría del nivel. Si esta decisión se coloca en un árbol de decisión, entonces el tiempo de ejecución del árbol de decisión se verá afectado por el tiempo de ejecución de esta decisión.

4.3. Metodología desarrollada

De acuerdo con lo que fue discutido en el capítulo 1, podemos decir que un kanji está conformado por un conjunto de trazos, y por las reglas del lenguaje, estos se realizan en un cierto orden específico lo que hace que los kanjis sean únicos.

En el capítulo 3 mencionamos los métodos de reconocimiento de kanjis en diccionarios actuales, entre estos se encuentra el método de búsqueda por número de trazos, sin embargo esta búsqueda solo acota el conjunto de posibles kanjis para ese número de trazos.

También mencionamos el método de búsqueda por radical, a partir del cual la búsqueda en diccionarios puede ser acotada a partir del radical.

Ahora bien, sería deseable aprovechar las bondades de ambos métodos y mejorar el proceso de búsqueda.

Partiendo de la idea del uso de radicales, podemos notar que al trazar un kanji veremos algunos trazos en común, que son trazados en el mismo orden, a estos los nombraremos **trazos básicos**.

Usando esta idea de trazos básicos se propone acotar la búsqueda en el diccionario de kanjis usando el siguiente algoritmo:

1. Reconocimiento de trazos básicos usando mapas autoorganizados
2. Almacenamiento de trazos básicos en el orden que fueron trazados
3. Selección de árbol de decisiones a partir del número de trazos
4. Obteniendo un kanji en el árbol de decisión a partir de los trazos almacenados

4.3.1. Reconocimiento de trazos básicos usando mapas autoorganizados

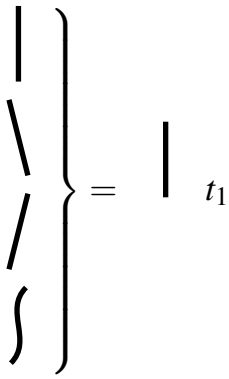
Para identificar los trazos básicos, se hizo una búsqueda exhaustiva en los libros de enseñanza del Centro de Enseñanza de Lengua Extranjera [66]. La tabla 4.2 muestra una lista de los trazos básicos que se consideraron para este trabajo.

	t_1		t_6		t_{11}		t_{16}		t_{21}
	t_2		t_7		t_{12}		t_{17}		t_{22}
	t_3		t_8		t_{13}		t_{18}		t_{23}
	t_4		t_9		t_{14}		t_{19}		t_{24}
	t_5		t_{10}		t_{15}		t_{20}		t_{25}

Tabla 4.2: Trazos básicos

Es importante resaltar que aunque algunos trazos se parecen son trazos diferentes, como el 15 y el 16 que los distingue por el はねる (haneru, que es la terminación en gancho que tiene el trazo 16).

Ahora bien el reconocer los trazos no es una tarea fácil ya que es fácil imaginar que cada vez que un usuario haga un trazo no le quedará exactamente igual, es decir, tomemos de ejemplo el trazo t_1 que es una línea vertical, aparentemente este es el trazo más sencillo de trazar, sin embargo el usuario puede que haga el trazo con una ligera inclinación a la izquierda o a la derecha, o que le tiemble la mano al usuario, estos cuatro trazos serían diferentes.

 $= t_1$ (4.1)

Por lo que surge la necesidad de una metodología que pueda inferir cual es el trazo que el usuario quiso trazar con un cierto margen de error de parte de él ó en otras palabras con la suficiente flexibilidad para soportar estos errores de parte del usuario y funcionar como es de esperar.

4.3.1.1. Binarización del trazo

Ya que vamos a trabajar con mapas autoorganizados, necesitaremos hacer una fase previa, la binarización del trazo.

Todos los kanjis se trazarán en un espacio en blanco de tamaño fijo, el cual llamaremos **lienzo**.

Por ahora concentrémonos en la digitalización de los trazos, el inicio de un trazo empieza desde que el usuario toca el lienzo hasta que el usuario despega el dedo del lienzo, un ejemplo de esto se puede apreciar en la figura 4.9.

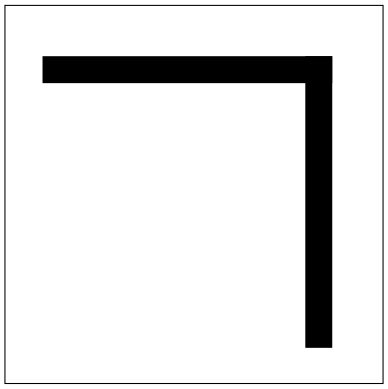


Figura 4.9: Trazo t_3 en el lienzo

Este lienzo es dividido en una malla de 10x10 como en la figura 4.10

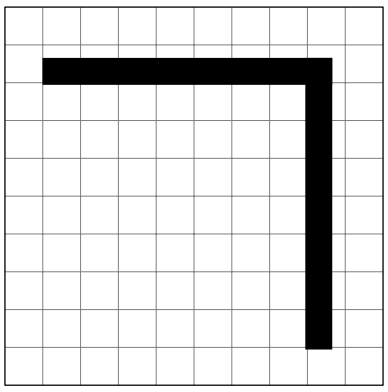


Figura 4.10: División del lienzo en una malla de 10x10

Se termina de rellenar los cuadros de la malla con la siguiente función

$$f(n) = \begin{cases} blanco & \text{si } n < 0.5 \\ negro & \text{si } n > 0.5 \end{cases}$$

Donde n es la cantidad de espacio que abarca dentro de un cuadro

Terminando con el lienzo como en la figura 4.11

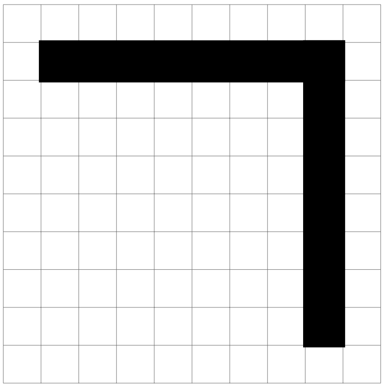


Figura 4.11: Rellenado de la representación

Una vez que tenemos el lienzo de esta manera podemos decir que los cuadros blancos los podemos representar con el número 0 y los cuadros negros con el número 1. Con esa representación de 0' s y 1's podemos crear un matriz como la siguiente:

$$\begin{pmatrix} 0&0&0&0&0&0&0&0&0&0\\ 0&1&1&1&1&1&1&1&1&0\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&0 \end{pmatrix} \tag{4.2}$$

Una vez que ya definimos que los trazos se pueden ver como una matriz de 0's y 1's, podemos hacer una conversión a un vector de entrada para nuestro mapa autoorganizado.

Retomando la matriz anterior, el vector resultante es el siguiente: [0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0, 0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0, 0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0]

Una vez que definimos todos los trazos básicos de la manera anterior empecemos a trabajar con nuestros mapas autoorganizados.

Estos vectores los usaremos como los vectores de entrada e inicializamos los pesos de las neuronas aleatoriamente en un rango de [-1,1], con estos valores podemos inicializar nuestros Mapas autoorganizados como se mencionó anteriormente pasando por los siguientes pasos:

- Normalización de la entrada (normalización del vector de entrada)
- Calcular la neurona ganadora (de salida)
- Selección de la neurona ganadora
- Entrenamiento de la red
- Determinar la razón de aprendizaje
- Determinar el ajuste de pesos

Una vez que los mapas autoorganizados fueron sometidos a un entrenamiento para reconocer los trazos básicos, el sistema es capaz de reconocerlos satisfactoriamente permitiendo continuar con lo siguiente.

4.3.2. Almacenamiento de trazos básicos en el orden que se fueron trazados

Hasta este momento tenemos una técnica capaz de reconocer los trazos básicos. Ahora bien conforme el usuario vaya realizando los trazos, estos deben ser almacenados en una estructura de datos lo suficiente flexible para permitir almacenar cuantos trazos el usuario trace y no desperdiciar espacio de memoria, además de permitir un acceso en memoria inmediato. Por lo que se eligió un arraylist que tiene las bondades de acceso rápido de un arreglo y el fácil agregamiento de elementos de una lista. Veamos un ejemplo, tomemos la representación de un arreglo para esto. En un principio el arraylist es vacío [ ],

El usuario al realizar el primer trazo digamos —, este es procesado por el mapa autoorganizado y clasificado como el trazo t_2, para que a continuación se guarde en el arraylist [t_2] con un segundo trazo digamos |, este análogamente es clasificado y agregada su representación en el arraylist [t_2,t_1], así se continuará agregando los trazos si es que el usuario trazará más.

4.3.3. Selección de árbol de decisiones a partir del número de trazos

En esta sección se toma como punto de partida el hecho de que los kanjis son realizados en una secuencia ordenada de trazos, a partir de esto y del número de trazos que los identifica, se elaboraron tablas que exhiben la posibilidad de plantearlo como árboles de decisión con múltiples ramas, para después mostrar cómo se construye los árboles y las características que tendrán, así como una exploración dentro de ellos.

4.3.3.1. Los kanjis vistos como una secuencia de trazos

En este trabajo usaremos el hecho de que al trazar un kanji, este tiene un orden estricto en la secuencia de trazos.

Por lo que categorizamos los kanjis por número de trazos, creamos tablas de dos, tres y cuatro trazos como la tabla 4.3 que representa a los kanjis de tres trazos.

山	丨	乚	丨	t_1	t_7	t_1
口	丨	冂	一	t_1	t_3	t_2
上	丨	一	一	t_1	t_2	t_2
小	亅	丿	丶	t_{10}	t_9	t_8
川	丿	丨	丨	t_5	t_1	t_1
大	一	丿	乀	t_2	t_5	t_6
土	一	丨	一	t_2	t_1	t_2
三	一	一	一	t_2	t_2	t_2
万	一	乛	丿	t_2	t_{19}	t_5
下	一	丨	丶	t_2	t_1	t_8
子	乛	亅	一	t_{12}	t_{13}	t_2
女	く	丿	一	t_{14}	t_5	t_2
工	一	丨	一	t_2	t_1	t_2
々	丿	乛	丶	t_5	t_{12}	t_8

Tabla 4.3: Tabla de kanjis de tres trazos

La tabla está conformada por una primera columna que contiene al kanji, seguido por los trazos básicos que lo componen y finalmente el código que se les asignó a estos trazos. De acuerdo con la tabla 4.3, una vez que identificado el primer trazo de un kanji podemos descartar a los kanjis cuyo primer trazo no sea igual, reduciendo a una cantidad menor de kanjis, por ejemplo si desea encontrar al kanji 大 dentro de la tabla 4.4, su primer trazo es 一 t_2 por lo que se descarta los kanjis que no estan sombreados.

山				t_1	t_7	t_1
口				t_1	t_3	t_2
上				t_1	t_2	t_2
小				t_{10}	t_9	t_8
川				t_5	t_1	t_1
大				t_2	t_5	t_6
土				t_2	t_1	t_2
三				t_2	t_2	t_2
万				t_2	t_{19}	t_5
下				t_2	t_1	t_8
子				t_{12}	t_{13}	t_2
女				t_{14}	t_5	t_2
工				t_2	t_1	t_2
々				t_5	t_{12}	t_8

Tabla 4.4: Selección de kanjis cuyo primer trazo es t_2

Y si es tomado el segundo trazo del kanji t_5 , son descartados más kanjis (En este caso, se reducen a un solo kanji), como puede apreciar en la tabla 4.5

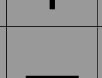
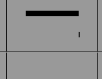
山				t_1	t_7	t_1
口				t_1	t_3	t_2
上				t_1	t_2	t_2
小				t_{10}	t_9	t_8
川				t_5	t_1	t_1
大				t_2	t_5	t_6
土				t_2	t_1	t_2
三				t_2	t_2	t_2
万				t_2	t_{19}	t_5
下				t_2	t_1	t_8
子				t_{12}	t_{13}	t_2
女				t_{14}	t_5	t_2
工				t_2	t_1	t_2
々				t_5	t_{12}	t_8

Tabla 4.5: Selección de kanjis cuyo primer trazo es t_2 y segundo trazo t_5

Cada vez que discriminamos kanjis por el trazo podemos decir que es una toma de decisión, por lo que implementar un árboles de decisiones n-arios es correcto.

4.3.3.2. Construyendo los árboles n-arios

Además de cumplir con las características mencionadas en la sección 4.2 los nodos de estos árboles cumplen con las siguientes particularidades:

Nodo raíz, este nodo contiene el número de trazos que corresponde al árbol, es decir, si es el árbol de tres trazos su nodo raíz contiene el número tres.

Nodo interno, estos nodos contienen la representación de uno de los trazos, por ejemplo t_1.

Nodo hoja, es el nodo que contiene al último trazo de un kanji y este siempre está ligado a al menos un objeto Kanji (un objeto kanji está conformado por los siguientes atributos: el símbolo, significado, lectura onyomi y lectura kunyomi, por ejemplo, símbolo: 月, significado: luna, onyomi: gatsu getsu, kunyomi: tsuki).

Siguiendo la tabla 4.3 puede construir un árbol como se ejemplifica en la figura 4.12.

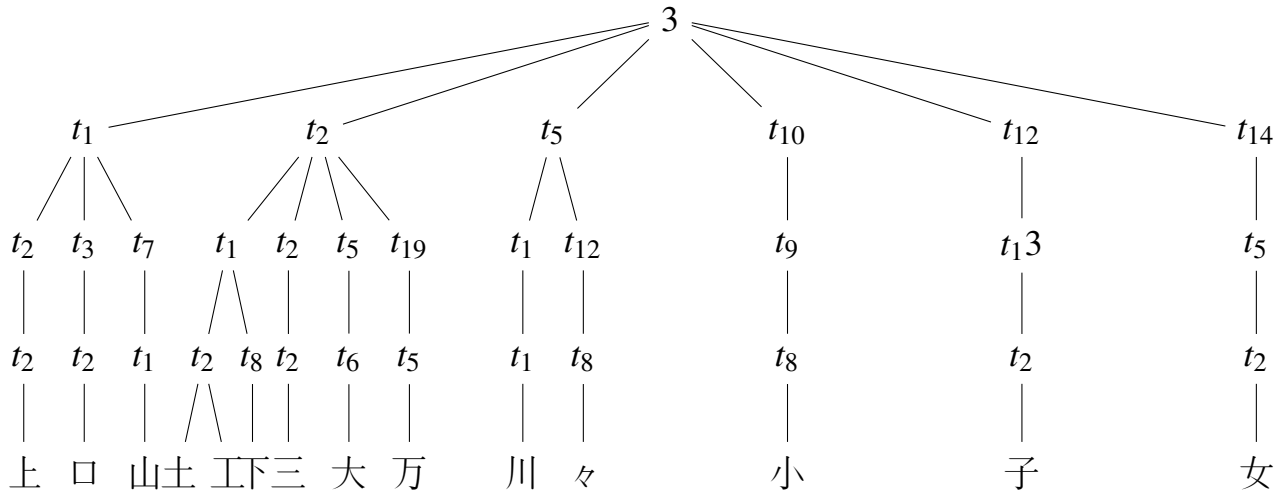


Figura 4.12: Árbol de kanjis

Rama La rama de estos árboles está conformada por la raíz, un número que corresponde al número de trazos, por los nodos internos que serían los trazos que conforman un kanji y el último trazo sería el nodo hoja que apunta al objeto Kanji. Un ejemplo de rama en la figura 4.12 sería la rama del kanji 川, está conformada por 3,t_5,t_1,t_1,川.

4.3.3.2.1. Podando el árbol

Si observamos a detalle el árbol de la figura 4.12 notaremos que algunas ramas se desprenden de la raíz y continúan de manera lineal.

Podemos entonces decir que esas ramas "se cuelgan" ya que a partir de cierto nodo sólo hay un posible camino, esto se debe a que a partir de cierto trazo los kanjis son únicos. Por lo que podemos tomar la decisión de cortar nuestro árbol y para nuestros fines prácticos seguirá funcionando correctamente, y no solo eso, sino que encontrar un kanji será más rápido.

Cabe hacer notar que si nos importara hacer una aplicacion para ver que un estudiante trace adecuadamente un kanji sería adecuado continuar con la estructura tal como está, sin embargo este no es el caso, ya que el objetivo es encontrar kanjis de una manera más rápida.

Esta propuesta de podado se puede contemplar en la figura 4.13.

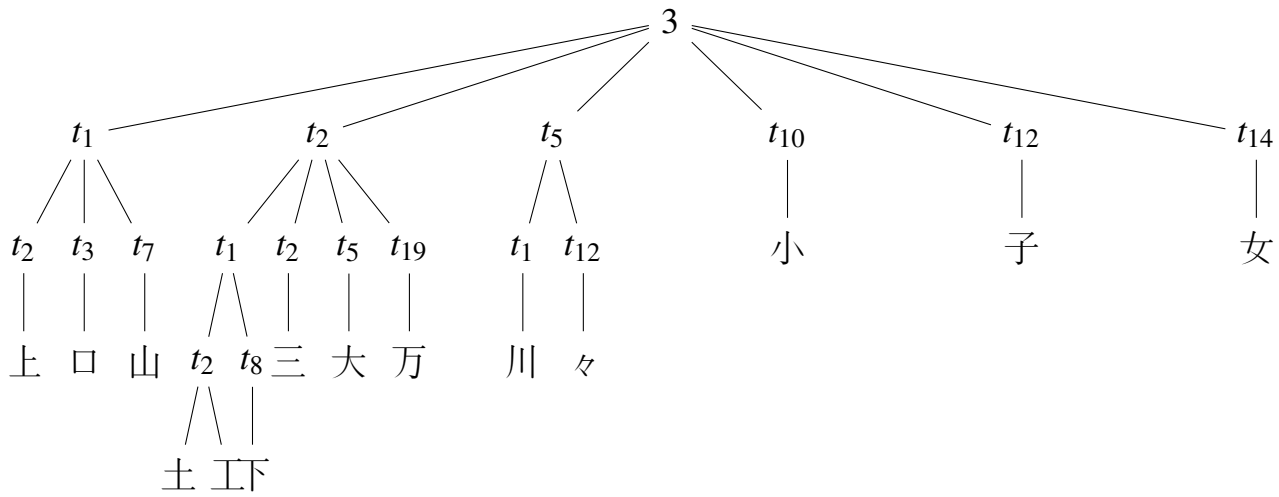


Figura 4.13: Árbol podado

Con esta propuesta cambia ligeramente la definición de nodo hoja a la siguiente:

Nodo hoja, es el nodo que contiene al último trazo significativo de un kanji (con significativo nos referimos a que a partir de este el solo hay un solo posible camino) y este siempre está ligado a al menos un objeto Kanji.

4.3.3.3. Propiedades de los árboles de decisión

Se crearon los árboles de dos y cuatro trazos de manera análoga al árbol de tres trazos de la figura 4.13. Si bien ya fue mencionado como se construyeron estos árboles es importante resaltar sus propiedades como árboles de decisiones.

En este árbol de decisión tiene un nodo raíz que corresponde a la decisión de partida, en este caso el número de trazos de los kanjis representan esa decisión ya que partimos de la agrupación de estos.

Sus nodos de decisión son los nodos internos que contienen los trazos que componen un kanji.

Los nodos hoja (ultimo trazo significativo) están conectados con una acción. En este caso las acciones corresponden a los objetos Kanjis y una vez que llegamos a estas hojas se muestran el contenido de este inmediatamente.

Estos árboles cumplen con que la toma de decisiones es sencilla ya que la elección en cada nodo se lleva a cabo comparando el código que fue asignado a los trazos básicos que componen a los kanjis con los trazos realizados por un usuario.

En la figura 4.14 la primera decisión es llevada a cabo comparando el número de trazos que realiza el usuario con el número trazos del que corresponde al árbol. t_i y t_{ii} son los trazos realizados por el usuario y que se compara con el código asignado a los trazos que componen al kanji. Estas decisiones se detallan en la sección 4.3.3.5.

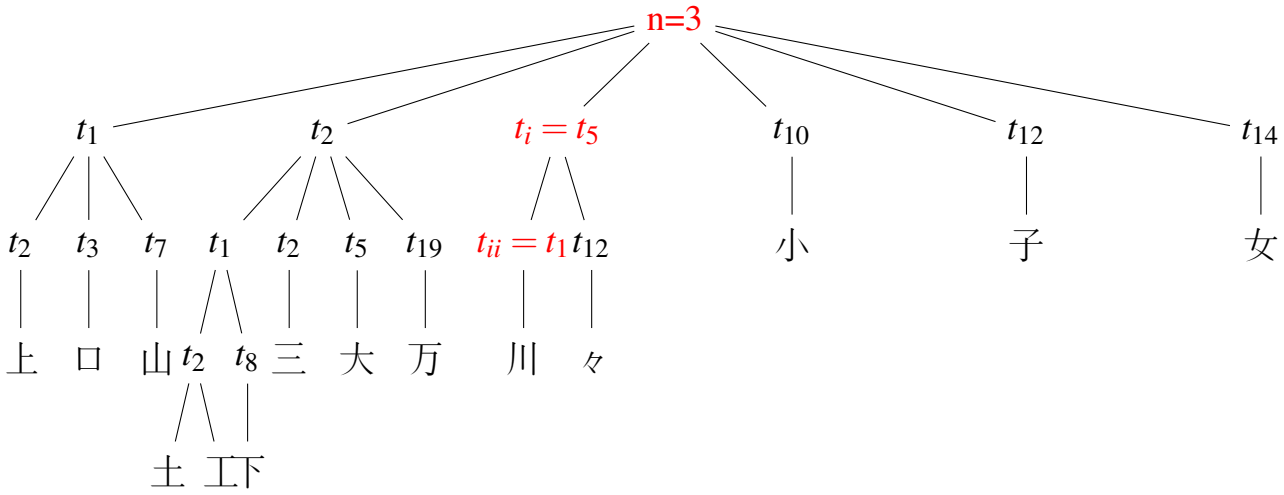


Figura 4.14: Toma de decisiones

Estos árboles cumplen con que el número de decisiones que consideran es menor que el número de decisiones. Siguiendo la figura 4.13 tenemos 6 decisiones diferentes y 14 acciones posibles (14 objetos Kanjis). Después de ejecutar el algoritmo, a lo más solo se consideran 4 decisiones.

De hecho por la forma en que fueron construidos los árboles cuentan con la propiedad de que el máximo número de decisiones que se considerarán será, a lo más, el número de trazos que representa el árbol más uno, ya que en esos casos fue necesario comparar el número de trazos y todos los trazos del kanji.

También podemos apreciar que la ramificación no es binaria, sino que contiene múltiples ramas, ya que considera los trazos que conforman a los kanjis y la manera en que se van descartando los kanjis como fue mostrado en la tabla 4.4 y 4.5.

En cuanto al rendimiento de estos árboles sigue siendo lineal con respecto al número de nodos visitados. ya que los trazos que realiza el usuario fueron procesados antes de hacer las decisiones dentro del árbol, por lo que no afecta al tiempo de ejecución de las decisiones tomadas, conservando así la sencillez que caracteriza a los árboles de decisión.

4.3.3.4. Selección de árbol de decisiones a partir del número de trazos

Ya teniendo construidos los arboles de decision por numero de trazos, y teniendo almacenado los trazos del kanji realizados por el usuario. Procedemos a verificar cuantos trazos fueron almacenados (es decir, los trazos realizados por el usuario) para así discriminar los árboles que no sean de la cantidad de trazos.

Por ejemplo si el usuario ha realizado tres trazos, solo se tomará en cuenta el árbol de decisiones de tres trazos, discriminando el restos de los árboles(el árbol de dos trazos, cuatro trazos, etc) para así continuar con la siguiente etapa del algoritmo.

4.3.3.5. Obteniendo un kanji en el árbol de decisión

Ahora que ha quedado claro cómo construimos estos árboles y sus características, es importante resaltar cómo identificar un kanji, que es propósito de esta estructura.

A partir de la raíz se explora en el primer nivel (los hijos de la raíz), buscando que alguno de los elemento coincida con el primer trazo realizado. Si no coincide se desecha este nodo y todas sus ramas. Si coincide se verifica si este nodo es un nodo hoja, si lo es, obtenemos los objetos Kanjis que están ligados a este, y termina la exploración; si no es un nodo hoja repetimos el proceso con sus nodos hijos comparando con el siguiente trazo almacenado. Como puede ver es una exploración recursiva.

Veamos un ejemplo:

Supongamos que el usuario realizó los siguientes trazos:  ,  ,  , estos son reconocidos como los trazos t_2, t_5, t_6 y son almacenados en ese orden. Como el número de trazos es tres, es seleccionado el árbol de tres trazos, a continuación se hacen las comparaciones con los nodos del primer nivel ($t_1, t_2, t_5, t_{10}, t_{12}, t_{14}$), como

$$t_1 \neq t_2$$

donde t_1 es el nodo del árbol y t_2 el trazo almacenado realizado por el usuario, como son distintos, entonces es desechado ese nodo y sus tres ramas, por lo que

se procede a la siguiente comparación entre t_2 del nodo y t_2 del trazo, como en este caso $t_2 \equiv t_2$, se verifica si este nodo es hoja, como la respuesta es negativa se continúa la exploración tomando como nodo raíz a t_2 con los trazos t_5,t_6. Ahora se hace la comparación entre el siguiente trazo (t_5) y los hijos del nodo actual (que son t_1, t_2, t_5, t_{19}), se desecha los nodos t_1 y t_2 y la comparación coincide con t_5, nuevamente se verifica que si es un nodo hoja y como en este caso sí lo es, obtenemos el kanji cuyo atributo símbolo es 大 terminando así la exploración.

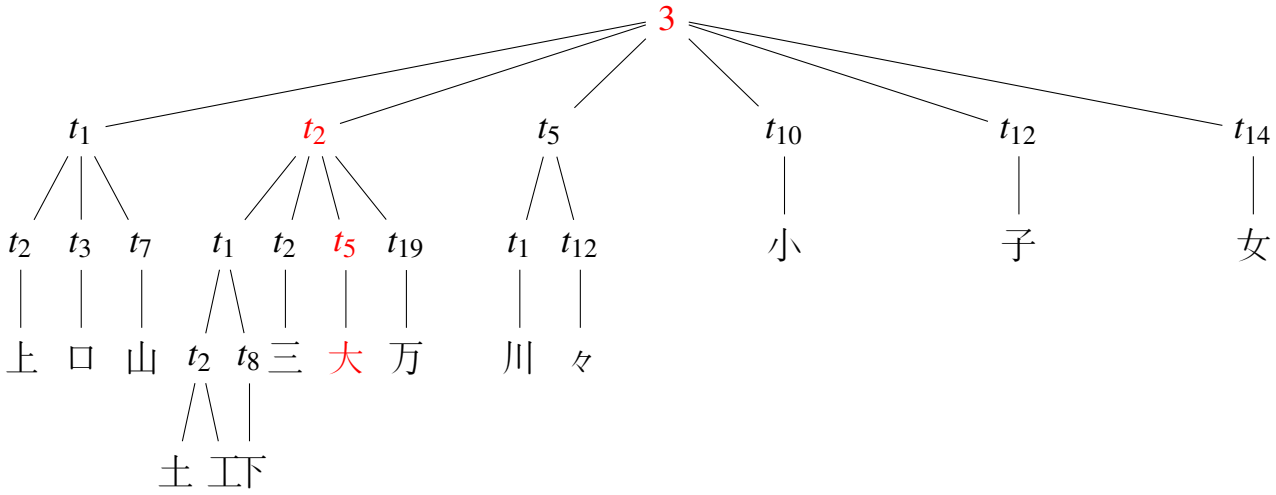


Figura 4.15: Búsqueda en el árbol de tres trazos

4.4. Resumen

En este capítulo ofrecemos una breve introducción a las redes neuronales, las cuales tienen dos tipos de aprendizaje: supervisado y no supervisado; en nuestro caso utilizaremos el no supervisado. Dentro del conjunto de las redes no supervisadas encontramos a los mapas autoorganizados. A esta red se le presenta un patrón de entrada, previamente normalizado a la capa de entrada, la salida de la red está dada por una única neurona ganadora para cada patrón presentado.

Para entrenar al mapa autoorganizado es necesario realizar varias iteraciones donde se presentan los elementos de entrenamiento. Los pesos de esta neurona se modifican para que respondan con mayor fuerza al patrón que la hizo ganar. En el caso en el que una o más neuronas no logran ganar, estas neuronas se podrían considerar como peso muerto en la red neuronal, por lo que debemos identificar esas neuronas y ajustarlas para que reconozcan patrones que ya

son reconocidos por otras neuronas más “sobrecargadas”. Esto permitirá que la carga del reconocimiento caiga más uniformemente sobre las neuronas de salida.

También en este capítulo fueron presentados los árboles de decisión n-arios, estos fueron vistos desde dos enfoques, vistos como árboles n-arios y como árboles de decisión. Se presentan generalidades de los árboles, así como su funcionamiento como árbol de decisión, además de puntualizar la complejidad y rendimientos de estos para así mostrar la metodología desarrollada.

Finalmente, se presenta la metodología desarrollada, basada en las técnicas anteriormente mencionadas.

En el siguiente capítulo se mostrará como se usa la metodología propuesta dentro de una aplicación para dispositivos móviles.

5 Diseño e implementación del reconocimiento de kanjis en dispositivos móviles

En los capítulos anteriores se describieron las bases que sustentan el desarrollo de la aplicación propuesta en este trabajo.

Esta aplicación fue realizada mediante las siguientes fases: toma de referencia una implementación en la que usan Mapas Autoorganizados en java, la adaptación de esta para hacer una implementación del reconocimiento de trazos y la Implementación de la identificación de kanjis, estas fases serán descritas en este capítulo. Así mismo, se hará una descripción del funcionamiento de esta propuesta y finalmente se presentarán las pruebas y resultados obtenidos durante la elaboración de esta aplicación.

5.1. Implementación de referencia

5.1.1. Mapas Autoorganizados

Una vez descrito el funcionamiento de los mapas autoorganizados, vamos a mostrar la implementación que propone Heaton [8] utilizando Java. En esta sección, veremos cómo pueden usar varias clases para crear un mapa autoorganizado.

Las clases que conforman al mapa autoorganizado son las siguientes:

NormalizeInput Clase donde se definen e implementan los tipos de normalización.

SelfOrganizingMap Clase donde se definen los mapas autoorganizados (el valor mínimo del entrenamiento, los pesos de conexión, etc), así como sus operaciones.

TrainSelfOrganizingMap Clase donde se definen todos las operaciones que se realizan durante un entrenamiento de los mapas autoorganizados.

Estas tres clases deben trabajar en conjunto para poder hacer funcionar el mapa autoorganizado, veamos a detalle cómo funcionan estas, empecemos por la clase **NormalizeInput**.

5.1.1.1. Clase de normalización del SOM

La clase **NormalizeInput** recibe un vector el cual representa la entrada y el tipo de normalización que se usará. La firma para el constructor es la siguiente:

```
public NormalizeInput(final double input[], final
    NormalizationType type)
```

El constructor inicializa el tipo de normalización (Esta puede ser una normalización multiplicativa o de eje z). Después, se calculan los factores de normalización (usando el método **calculateFactors**) y de entrada sintética. Y finalmente, se crea la matriz de entrada (**createInputMatrix**).

5.1.1.1.1. Cálculando los factores

El método **calculateFactors** calcula tanto el factor de normalización como el valor de entrada sintético. La firma para el método **calculateFactors** es la siguiente:

```
protected void calculateFactors(final double input[])
```

5.1.1.1.2. Creación de la matriz de entrada

Una vez determinado el factor de normalización y la entrada sintética, se puede crear la matriz de entrada. La matriz de entrada es creada mediante el método **createInputMatrix**. La firma del método **createInputMatrix** es la siguiente:

```
protected Matrix createInputMatrix(final double
    pattern[], final double extra)
```

En este método devuelve una matriz que toma el mismo valor del patrón de entrada más una columna extra que contendrá la entrada sintética.

5.1.1.2. Clase de implementación del SOM

El mapa autoorganizado es implementado en la clase **SelfOrganizingMap**. Un patrón se presenta al SOM utilizando un método **winner**. La firma para este método es la siguiente:

```
public int winner(final double input[])
```

Este método crea una matriz normalizada instanciando a la clase **NormizedInput** y es enviado a un segundo método **winner**.

La firma para este método es la siguiente:

```
public int winner(final NormalizedInput input)
```

Este segundo método winner acepta un objeto **NormizedInput**, que es una matriz normalizada. La neurona ganadora es devuelta por este segundo método winner.

5.1.1.3. Clase de entrenamiento del SOM

El entrenamiento es realizado en la clase llamada **TrainSelfOrganizingMap**. En esta clase se tienen que realizar varias iteraciones hasta que el error sea lo suficientemente pequeño.

5.1.1.3.1. Iteración de entrenamiento

Para realizar una iteración de entrenamiento, se llama al método de **iteration** de la clase **TrainSelfOrganizingMap**. La firma del método **iteration** es la siguiente:

```
public void iteration() throws RuntimeException
```

En este método se llama al método **evaluateErrors** para determinar el nivel actual de error. EEl error actual es evaluado para ver si es mejor que el mejor error encontrado anteriormente. Si es así, los pesos son copiados en la matriz de peso mejor anterior y se determina el número de neuronas que han ganado, desde la última vez que fueron calculados los errores. Si ha habido muy pocos ganadores, uno es forzado. Los pesos son ajustados en el entrenamiento de esta iteración y son copiados a la mejor red, y finalmente estos se normalizan. Esto completa una iteración de entrenamiento. Cabe aclarar que en cada iteración la tasa de aprendizaje se reduce gradualmente hasta 0.01.

5.1.1.3.2. Evaluación de Errores

Evaluar los errores implica determinar cuántas neuronas ganan para un cierto patrón dado. La neurona que tiene la mayor respuesta a un patrón de entrenamiento es ajustada para que respondan con mayor fuerza a este patrón que la hizo ganar. Los errores son calculados utilizando el método **evaluateErrors**. La firma para este método se muestra aquí:

```
public void evaluateErrors()
```

5.1.1.3.3. Forzando una ganadora

En el caso en el que una o más neuronas no logren ganar para ningún patrón de entrenamiento, estas neuronas se podría considerar como peso muerto en la red neuronal, por lo que debemos identificarlas y ajustarlas para que reconozcan patrones que ya son reconocidos por otras neuronas más “sobrecargadas”.

El método **forceWin** ajusta a estas neuronas. La firma del método **forceWin** se muestra aquí.

```
protected void forceWin()
```

5.1.1.3.4. Ajustando los pesos

Una vez que hemos calculado los errores, se llama al método **adjustWeights**.

El método **adjustWeights** como su nombre lo indica ajusta los pesos y permita que la red aprenda. La firma del método **adjustWeights** se muestra aquí.

```
protected void adjustWeights()
```

5.1.2. Reconocimiento de caracteres con mapas autoorganizados

Las clases presentadas en la sección anterior pueden trabajar en conjunto con otras para realizar el reconocimiento de caracteres. Las siguientes clases tienen el siguiente propósito:

Entry Interfaz que permite tener un área de dibujo, en la cual el usuario podrá trazar el carácter o símbolo a reconocer.

Sample Clase que muestra la imagen reducida en resolución (imagen ya binarizada).

SampleData Clase donde almacena la representación interna de Sample (matriz booleana), así como las operaciones de esta.

OCR Clase principal donde inicializa la aplicación OCR.

Ahora que ya tenemos un panorama general de los mapas autoorganizados, examinaremos cada clase que nos ayuda con el reconocimiento de caracteres como lo hicimos en la sección anterior. Comenzaremos examinando cómo un usuario dibuja una imagen.

5.1.2.1. Trazado de imágenes

Aunque no está directamente relacionado con las redes neuronales, el proceso por el cual un usuario es capaz de dibujar caracteres es una parte importante de la aplicación OCR.

En la clase **Entry**, la mayor parte del dibujo realizado por el usuario es manejado por **processMouseMotionEvent**. Una línea es dibujada cuando un usuario toca la pantalla del dispositivo y lo arrastra hasta una cierta posición y suelta

la pantalla. En algunos casos el dedo de un usuario se mueve más rápido que el procesamiento de los eventos por parte de la aplicación; así, que es dibuja una línea estimando los píxeles perdidos. Si no quiere perder precisión al hacer un trazo curvo es necesario ejecutarlo de manera un poco más lenta para así conservar la curvatura.

Este método es llamado repetidamente a medida que el usuario toca el área de dibujo, los trazos realizados por el usuario son almacenados por la clase **SampleData**. Esta clase realiza una reducción de resolución de los trazos, la cual crea una matriz de enteros que permite la manipulación directa de los datos de la imagen.

5.1.2.1.1. Tamaño y posición

Heaton menciona que el tamaño y la posición pueden no ser considerados para el reconocimiento de caracteres alfanuméricos, sin embargo en nuestro caso nos ayudará con el reconocimiento de kanjis más complejos. Según Heaton el área de dibujo es lo suficientemente grande como para permitir al usuario dibuje letras de diferentes tamaños, por lo que recortar la imagenes proporciona imágenes de tamaño consistente, sin embargo en nuestro caso los kanjis respetan el tamaño y posición de los trazos por lo que se decidió que el recorte no se lleve a cabo, pero si la reducción de resolución (binarización del área de dibujo) en las aplicaciones (tanto la aplicación de entrenamiento, como en la aplicación para móviles) presentadas y explicadas con más detalle más adelante.

En la versión de Heaton, cuando es dibujada una imagen, lo primero que hace la aplicación es dibujar una caja alrededor de los límites de su carácter. Esto permite que la aplicación elimine todo el espacio en blanco. El proceso se realiza dentro del método **downsample** de la clase **Entry**. Con el fin de recortar esta imagen, y eventualmente reducir la resolución, debemos tomar su patrón de bits. Esto se hace usando la clase **PixelGrabber**.

5.1.2.1.2. Realización de la reducción de la resolución

Para ser procesada por el mapa autoorganizado, la imagen debe ser sometida a reducción de resolución. Esto implica reducir la imagen a una resolución de 10 X 10. Para entender cómo reducir una imagen de 10 X 10, retomemos la idea de la seccion 4.3.1.1 del capitulo anterior, tomemos la imagen de alta resolución que el usuario dibujó y esta la dividimos en una malla de 10x10. La malla divide la imagen en regiones, diez a lo ancho y diez a lo alto. Si se rellena cualquier píxel de una región, también se rellena el píxel correspondiente en la imagen de

10x10 como podemos ver en la figura 4.11. La mayor parte del trabajo realizado por este proceso es realizado dentro del método **downSampleRegion**, cuya firma es:

```
protected boolean downSampleRegion(final int x, final
    int y)
```

El resultado de aplicar el método **downSampleRegion** es una copia reducida de la imagen, almacenada en la clase **SampleData**. La clase **SampleData** es una clase que contiene una matriz 10x10 de valores booleanos. Es esta estructura la que constituye la entrada tanto para el entrenamiento como para el reconocimiento de caracteres.

5.1.2.1.3. Visualización de la versión binarizada

La clase **Sample** toma la matriz de 10x10 de la clase **SampleData** y muestra en pantalla la representación visual del carácter o trazo. Esto es posible a través del método paint. Cuya firma es la siguiente:

```
public void paint(final Graphics g)
```

5.1.2.1.4. Uso del mapa de autoorganización

El patrón de caracteres binarizado que es dibujado por el usuario, ahora alimenta a las neuronas de entrada del mapa de autoorganizado. Hay una neurona de entrada para cada píxel en la imagen binarizada. Debido a que la imagen binarizada es una matriz de 10x10, hay 100 neuronas de entrada.

La red neuronal comunica que carácter cree que el usuario dibujo a través de las neuronas de salida. El número de neuronas de salida siempre coincide con el número de muestras de caracteres únicos proporcionados. En la implementación de Heaton, dado que proporciona 26 letras en la muestra, hay 26 neuronas de salida. Esta aplicación está diseñada para soportar múltiples muestras por letra individual, y aun así todavía serían 26 neuronas de salida.

Además de las neuronas de entrada y salida, también hay conexiones entre las neuronas individuales. Estas conexiones no son todas iguales. A cada conexión se le asigna un peso. Los pesos son, en última instancia, los únicos factores que determinan lo que la red emitirá para un patrón de entrada dado. Para determinar el número total de conexiones, se debe multiplicar el número de neuronas de entrada por el número de neuronas de salida. Una red neuronal con 26 neuronas de salida y 100 neuronas de entrada tendrá un total de 2600 pesos de conexión.

El proceso de entrenamiento está dedicado a encontrar los valores correctos para estos pesos.

Las neuronas requieren una entrada de punto flotante; por lo que el programa utiliza el valor de 0.5 para representar un píxel negro y -0.5 para representar un píxel blanco. La matriz 10x10 de 100 valores alimenta a las neuronas de entrada. Esto se logra pasando la matriz de entrada al método **winner** de la red neuronal. Este método identificará cuál de las 100 neuronas ganó, y almacenará esta información.

Conocer la neurona ganadora no es muy útil, porque no le muestra qué letra fue realmente reconocida. Para determinar qué neurona está asociada con cada letra, la red debe ser alimentada con cada letra para ver qué neurona gana. Por ejemplo, si tuviera que alimentar la imagen de entrenamiento para "J" en la red neuronal, y la neurona # 4 fue devuelta como el ganador, sabríamos que la neurona # 4 es la neurona que fue entrenada para reconocer el patrón de J. Este proceso se logra llamando al método **mapNeurons**. El método **mapNeurons** devuelve una matriz de caracteres. El índice de cada elemento de la matriz corresponde al número de neuronas que reconoce el carácter particular.

5.1.2.1.5. Entrenamiento de la Red Neural

Definiremos al aprendizaje como el proceso de selección de una matriz de pesos de una neurona que reconocerá correctamente los patrones de entrada que recibe. Un mapa autoorganizado aprende constantemente evaluando y optimizando su matriz de pesos. Para ello, debe establecerse una matriz de pesos inicial. Esto se logra mediante la selección de números aleatorios. Esta matriz de pesos inicial probablemente hará un trabajo pobre de reconocer letras, pero proporciona un punto de partida.

Una vez que inicialmente se crea la matriz de peso al azar, el entrenamiento puede comenzar. En primer lugar, evalúa la matriz de pesos para determinar su nivel de error actual. El error es determinado evaluando qué tan bien las entradas de entrenamiento (las letras que creó) corresponden con las neuronas de salida. Este error se calcula mediante el método **evaluateErrors** de la clase **TrainSelfOrganizingMap**. Cuando el nivel de error es bajo, digamos por debajo del 10 %, el proceso está completo y termina el entrenamiento.

Esta parte del programa es flexible y se puede modificar para entrenar con más de una muestra por cada letra. Por ejemplo, si queremos utilizar 4 muestras por letra, tendremos que asegurarnos de que el recuento de neuronas de sali-

da permanezca 26, aunque se proporcionarán 104 muestras de entrada para el entrenamiento, 4 para cada una de las 26 letras.

El entrenamiento es ejecutado en un subproceso en un método **run** de la clase **OCR**. La firma para el método run es la siguiente:

```
public void run()
```

En este método, primero calcula el número de neuronas de entrada necesarias. Este es el producto de la altura y el ancho de la imagen reducida. Después, se asigna el conjunto de entrenamiento. Se trata una matriz bidimensional con filas igual al número de elementos de entrenamiento, que en este ejemplo son las 26 letras del alfabeto. El número de columnas es igual al número de neuronas de entrada. Recorremos todas las muestras de letras, obtenemos los datos de las letras de muestra. Los datos son transferidos a la red y los booleanos true y false se transforman en 0.5 y -0.5.

Una vez que los datos de entrenamiento fueron creados, se crea el nuevo objeto **SelfOrganizingMap**. Este objeto utilizará la normalización multiplicativa, Posteriormente se crea un objeto **TrainSelfOrganizingMap** para entrenar el mapa de autoorganizado recién creado. Este entrenamiento usa el método de entrenamiento sustractivo. Se realiza un seguimiento del número de intentos de entrenamiento. El número de intentos y la información de error son actualizados en cada iteración de entrenamiento. Cuando el error de entrenamiento ha alcanzado un nivel aceptable, termina el entrenamiento y la red neuronal está ahora lista para su uso.

5.2. Implementación del reconocimiento de trazos

En la sección anterior se presentó el funcionamiento de la aplicación OCR realizada por Heaton, esta idea fue adaptada para que reconociera trazos en lugar de caracteres alfanuméricos, para así presentar una aplicación que utilizando los mapas autoorganizados pueda reconocer los trazos realizados por un ser humano. Esta aplicación no pretende escanear un kanji de un texto; más bien, reconocerá los trazos individuales a medida que sean dibujados por el usuario en la aplicación.

Como se mencionó anteriormente la propuesta de Heaton de SOM está imple-

mentada en java, por lo tanto esta aplicación también funciona en Java y para probar su funcionamiento en un dispositivo móvil se utilizarán dispositivos que soporten Java. Al momento de realizar este trabajo, los dispositivos que soportan java funcionan con Windows 7 o 10.

5.2.1. Aplicación para entrenamiento del mapa autoorganizado

La siguiente aplicación nos sirve para crear el entrenamiento del mapa autoorganizado de la aplicación del apoyo al aprendizaje japonés, esta aplicación permite a un experto realizar los trazos básicos definidos en el capítulo anterior, para así crear el conjunto de entrenamiento.

5.2.1.1. Entrenamiento

Cuando inicia la aplicación se muestra la pantalla que puede apreciar en la figura 5.1

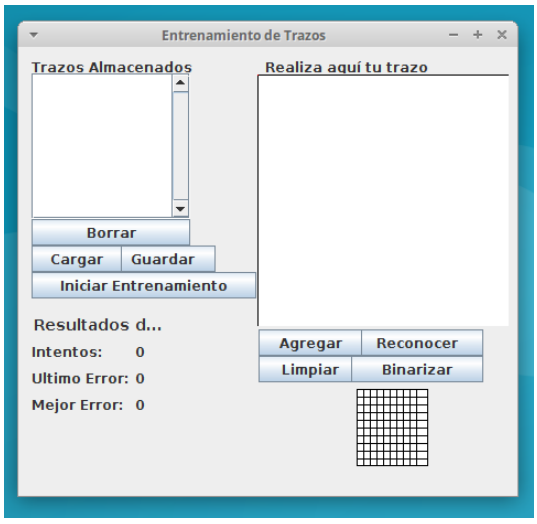


Figura 5.1: Inicio de la aplicación de Entrenamiento

Una vez inicializada la aplicación, el usuario puede realizar un trazo en el lienzo señalado, al presionar el botón “Agregar”, pedirá al usuario que indique el carácter de codificación que le asignará a la muestra realizada como se muestra en la figura 5.2,

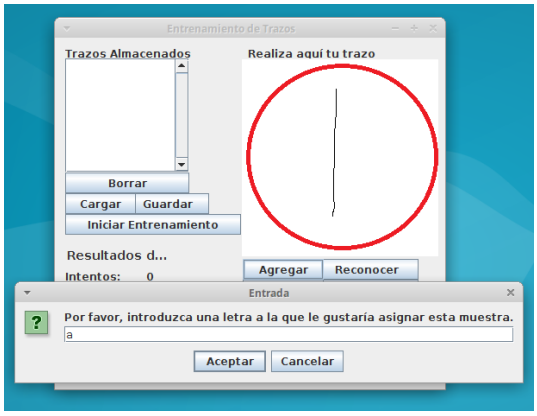


Figura 5.2: Agregar una muestra al conjunto de entrenamiento

Una vez que el usuario indica el carácter, este trazo es binarizado, es decir, la imagen es asignada a una matriz que tiene diez píxeles de ancho y diez píxeles de alto. La imagen se guarda en memoria temporalmente mientras la aplicación esté abierta: Puede acceder a la imagen binarizada seleccionando el carácter asignado que puede encuentrar en la lista debajo de la frase "Trazos almacenados" como se muestra en la figura 5.3.

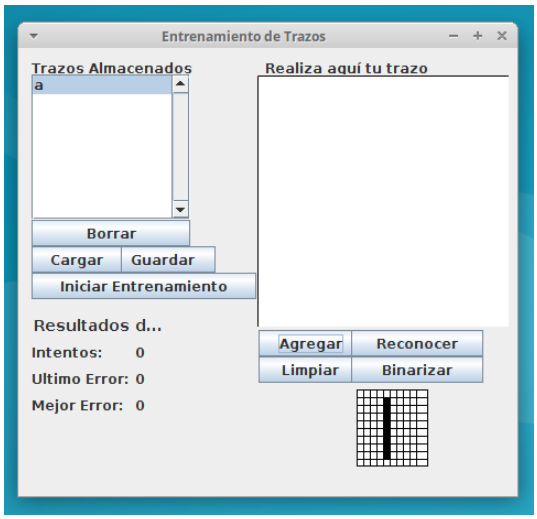


Figura 5.3: Imagen binarizada de un trazo muestra

Este proceso se repite varias veces hasta genera un conjunto de entrenamiento que cubra a todos los trazos básicos definidos anteriormente.

Una vez que ya cuenta con las muestras de los trazos básicos, la red neuronal debe ser entrenada. El proceso de entrenamiento comienza cuando el usuario presiona el botón "Iniciar Entrenamiento". Esto inicia el entrenamiento y cal-

cula el número de neuronas de entrada y salida. En primer lugar, el número de neuronas de entrada es determinado a partir del tamaño de la imagen binarizada. Dado que la altura es de diez y el ancho es de diez para este ejemplo, el número de neuronas de entrada es 100. El número de neuronas de salida coincide con el número de trazos que el programa ha recibido. Este proceso de entrenamiento puede durar de unos segundos a varios minutos, dependiendo de la velocidad del dispositivo utilizado. Una vez terminado el entrenamiento mostrará un mensaje de terminación, como se muestra en la figura 5.4.

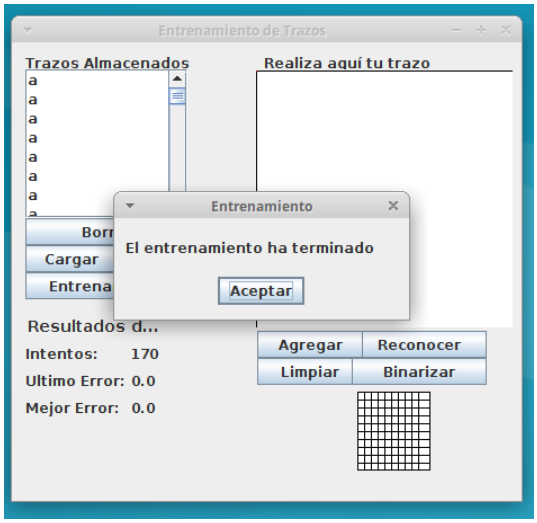


Figura 5.4: Entrenamiento terminado

Ahora que la red neuronal ha sido entrenada, la aplicación está lista para reconocer los trazos básicos. La aplicación permite probar el desempeño del entrenamiento, ya que puedes dibujar un trazo que quiera reconocer en la misma región donde fueron trazadas las muestras y al presionar el botón “Reconocer” comenzará el proceso de reconocimiento. El trazo es binarizado y se pasa al mapa autoorganizado, entonces al mapa autoorganizado indica la neurona ganadora como vemos en la figura 5.5 con lo cual puede observar si el trazo fue reconocido correctamente.

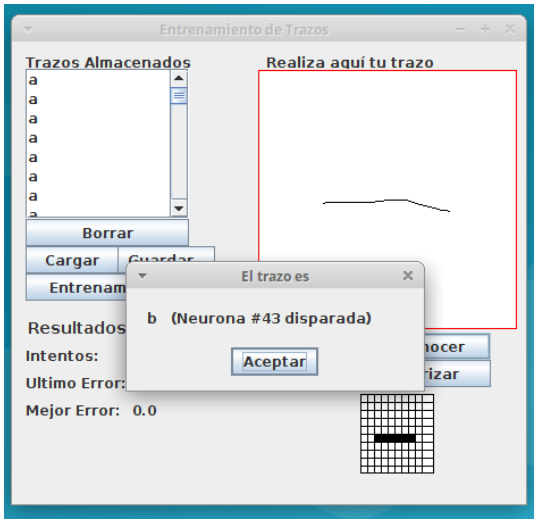


Figura 5.5: Trazo reconocido por el mapa autoorganizado

El mapa autoorganizado utilizado por esta aplicación tiene 25 neuronas de salida que coinciden con los 25 trazos básicos en el conjunto de muestras.

5.2.1.2. Guardar y cargar un conjunto de entrenamiento

Una vez creado un conjunto de entrenamiento es deseable guardarlo, esto es posible presionando el botón “Guardar”, al ejecutar esta acción almacenará un archivo “muestra.dat” como el que se observa en la figura 5.6.

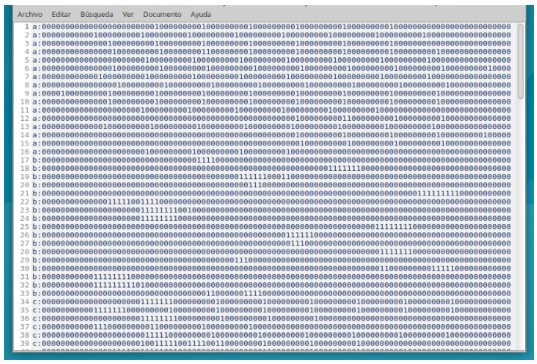


Figura 5.6: Archivo “muestra.dat”

Este archivo se genera en el mismo directorio que la aplicación y contendrá las muestras generadas hasta el momento de guardarlo. Al término de la creación de este archivo se mostrará el mensaje de la figura 5.7.

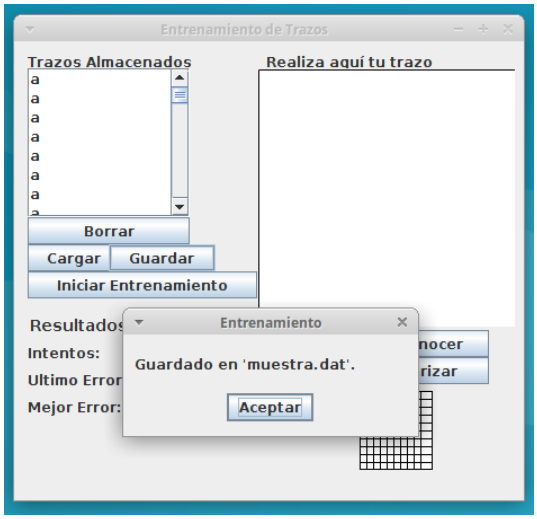


Figura 5.7: Guardar conjunto de entrenamiento

Si bien ya ha guardado un conjunto de entrenamiento, puede realizar el entrenamiento, esto es posible hacerlo presionando el botón “Cargar” de la aplicación. Al ejecutar esta acción, la aplicación intentará cargar el archivo de formación “muestra.dat”, al cargarse el archivo muestra el mensaje que indica que el archivo ha sido cargado correctamente. Una vez que haya cargado el archivo, la aplicación mostrará los códigos de los trazos almacenados.

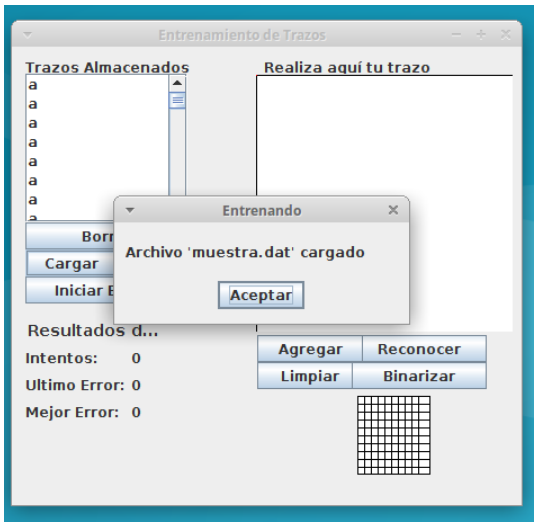


Figura 5.8: Carga del archivo “muestra.dat”

5.2.1.3. Verificación del entrenamiento

Una vez obtenido un conjunto de entrenamiento y realizado el entrenamiento del mapa autoorganizado, podrá verificar la tasa de reconocimiento ingresando nuevos trazos y presionando el botón “Reconocer”.

También pueden analizar los trazos almacenados para identificar las muestras incorrecta, ya que estas muestras pueden causar ruido a la aplicación, lo que provoca un mal reconocimiento así que puede ser retirada del conjunto de entrenamiento.

Para borrar una muestra de un trazo, debe seleccionar la muestra de la lista de “Trazos Almacenados”, al seleccionarla podrá visualizar la versión binarizada de esta muestra como se ve en la figura 5.9.

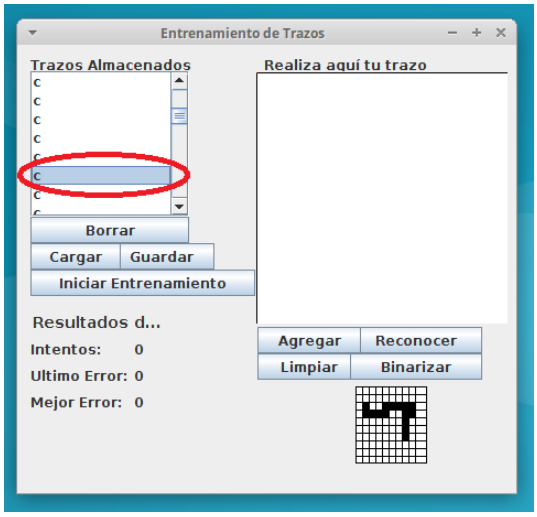


Figura 5.9: Borrar muestra

Una vez que está seleccionado, presiona el botón “Borrar” y la muestra seleccionada será borrada del conjunto de entrenamiento.

Al poder agregar y borrar trazos podemos hacer ajustes a un conjunto de entrenamiento, conforme vaya haciendo esto, es conveniente hacer pruebas y verificar la tasa de reconocimiento que obtenemos.

Para la elaboración de la aplicación de apoyo al aprendizaje del idioma japonés, se realizó un conjunto de entrenamiento conformado por los trazos básicos definidos anteriormente realizados por un experto, en este caso un estudiante de

nivel avanzado del idioma. El conjunto de entrenamiento consta de 169 muestras.

5.2.1.4. Descripcion de otros elementos de la aplicación

Si bien, hasta este momento se ha mostrado gran parte del funcionamiento de la aplicación, en esta sección se muestra elementos aún no mencionados.

Si el usuario realiza un trazo y solo quiere ver la versión binarizada de este trazo, puede presionar el botón “Binarizar”, con esto visualizará la versión binarizada del trazo realizado como el de la figura 5.10

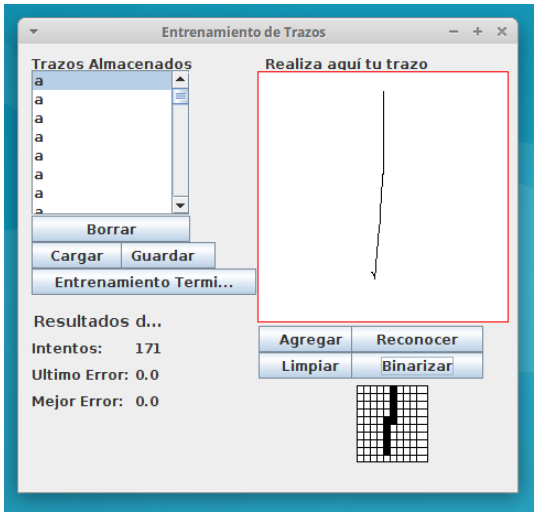


Figura 5.10: Parte inferior muestra imagen binarizada del trazo realizado

Si al usuario no le gusto su trazo este puede borrarlo presionando el botón “Limpiar”, al presionarlo borrarán la región de dibujo y la binarización del trazo como se ve en la figura 5.11.

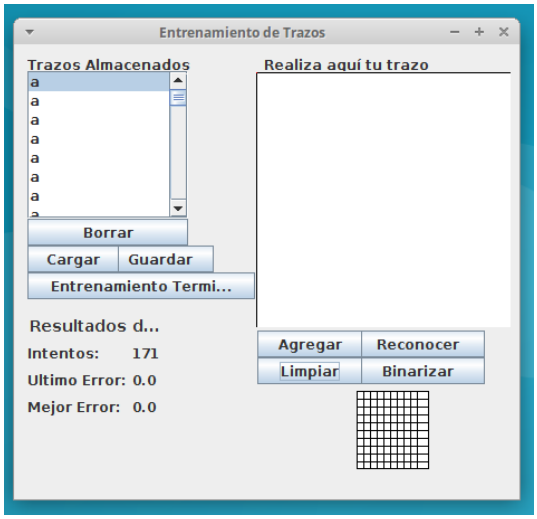


Figura 5.11: Limpieza del lienzo y la imagen binarizada

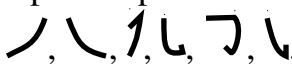
5.2.2. Pruebas con la división del lienzo y selección del actual

Como ya se mencionó el lienzo es dividido en una matriz, la cual es procesada para el entrenamiento y reconocimiento del trazo, la división del lienzo es un factor muy importante para el buen funcionamiento del reconocimiento, por lo que surge la pregunta ¿cuál es el la división idónea para un óptimo funcionamiento de la aplicación?, para responder esta pregunta fueron realizadas las siguientes pruebas con ayuda de un experto, en este caso un estudiante de nivel avanzado del idioma.

El experto proporcionó 47 muestras de los 25 trazos básicos, estas muestras se usaron como entrada para red considerando las diferentes divisiones del lienzo: 10x10, 15x15, 20x20 y 30x30. Los resultados se muestran en la tabla 5.1.

Division del lienzo	Número de trazos reconocidos	Tasa de reconocimiento (%)
10x10	16	64
15x15	10	40
20x20	10	40
30x30	5	20

Tabla 5.1: Resultados del reconocimiento con lienzos de 10x10, 15x15,20x20 y 30x30

En estas primeras pruebas resaltó que los trazos que no se pudieron ser reconocidos por la aplicación en las divisiones de 15x15, 20x20 y 30x30 fueron los trazos  .

Los primero dos son confundidos por los trazos  ,  los cuales si bien la dirección del trazado es la misma, estos carecen de la curvatura que caracteriza a los trazos  ,  .

Estos últimos tres trazos básicos  ,  ,  tiene una terminación de gancho y estos fueron confundidos por los trazos básicos que no contienen ese gancho, como es el caso del trazo  que fue confundido por el trazo  .

Aunque el trazo  no tiene terminación en gancho tiene un problema similar a los anteriores trazos, ya que este es confundido con el trazo  , el cual carece del comienzo en pico del trazo  .

Con el fin de identificar el número de muestras necesarias para un correcto reconocimiento con las diferentes divisiones del lienzo, se agregaron más muestras en la etapa de entrenamiento y se observó lo siguiente:

- En la división de 10x10 con tres muestras por trazo básico era necesarias para su reconocimiento correcto.
- En la división de 15x15, ocho muestras por trazo básico eran necesarias para su reconocimiento correcto.
- En la división de 20x20, quince muestras por trazo básico eran necesarias para su reconocimiento correcto.
- Finalmente, en la división de 30x30,veinte muestras por trazo básico eran necesarias para su reconocimiento correcto.

Con base en las pruebas anteriores se tomó la decisión de tomar la división de 10x10 del lienzo, ya que tuvo mayor tasa de reconocimiento con la menor cantidad de muestras.

Como se puede observar la división del lienzo es un factor fundamental para el buen funcionamiento del reconocimiento del trazo, siendo parte clave para el reconocimiento de kanjis.

5.2.3. Pruebas con los trazos

A partir de la decisión de usar el lienzo con una división de 10x10 se procedió a hacer pruebas con un experto, en este caso un estudiante de nivel avanzado del idioma con la cantidad de muestras por cada trazo, las pruebas fueron realizadas en tres ciclos.

Trazo	Representación del trazo	Núm. de muestras Prueba 01	Núm. de muestras Prueba 02	Núm. de muestras Prueba 03
	a	4	7	16
	b	6	12	17
	c	2	3	10
	d	1	5	7
	e	4	10	18
	f	3	5	7
	g	1	1	3
	h	1	6	9
	i	2	8	11
	j	1	3	7
	k	1	4	11
	l	3	7	7
	m	1	1	2
	n	1	1	3
	o	1	4	4
	p	1	3	5
	q	1	1	4
	r	2	4	4
	s	3	7	7
	t	3	3	3
	u	1	1	3
	v	1	1	3
	w	1	2	2
	x	1	1	1
	y	1	5	5
Total	de Trazos	47	105	169

Tabla 5.2: Número de muestras por trazo realizadas en tres fases de pruebas

	Número de trazos reconocidos	Tasa de reconocimiento (%)
Prueba 01	16	64
Prueba 02	20	80
Prueba 03	25	100

Tabla 5.3: Resultados de las pruebas de reconocimiento de trazos básicos

En base a las pruebas anteriores se procedió a incrementar el número de muestras de 47 a 105. La aplicación reconoció 20 de los 25 trazos básicos con lo que la tasa de reconocimiento aumentó de 64 % a 80 %. Con un número mayor de muestras (169), mejoró el rendimiento del funcionamiento del SOM para el reconocimiento de trazos básicos.

Como se puede observar en la tabla 5.2, los trazos que requirieron mayor número de muestras fueron  y  , ya que el SOM presentó problemas para identificar estos trazos cuando variaba la inclinación y eran reconocidos como los trazos  y  . A sus vez, los trazos anteriores, eran confundidos con los trazos  ,  , ya que tiene tienen la misma dirección pero la curvatura de estos los distingue.

Otro ejemplo de los trazos que presentaron problemas fueron los trazos  ,  ,  , si observamos a los dos primeros trazos podremos ver que los distingue la terminación en gancho (problema que se maximiza con lienzos muy grandes), ahora bien entre  y el problema radica en la curvatura del trazo, sin embargo estos problemas eran solucionados ingresando un mayor número de muestras.

5.2.4. Parámetros finales

Finalmente los parámetros usados fueron: Una división del lienzo de 10x10, 169 muestras de los 25 trazos básicos, el número de muestras por trazo se puede observar en la tabla 5.2.

5.3. Implementación de la identificación del kanji

5.3.1. Codificación de árboles

Para la realización de la codificación de los árboles de decisión fue necesario la realización de las siguientes clases:

Kanji La clase kanji representa un kanji y lo que conlleva, este conformado por su representación (el carácter en sí), su significado en español, lecturas onyomi y kunyomi (las dos tipos de lecturas posibles que puede tener en el idioma japonés), así mismo esta clase tiene tiene los métodos getters y setters para estos atributos.

Nodo La clase nodo representa los nodos que conforman el árbol, estos contiene un carácter (la representación de un trazo), así como apunta a un único nodo padre, también tienen la característica booleana de ser nodo hoja, de ser nodo hoja implicaría que este nodo está ligado a un objeto kanji, en caso contrario este está ligado a nodos hijos.

Arbol La clase arbol esta conformada por un nodo raíz (este contiene el número de trazos), en esta clase se hace la asociación (o ligamiento) de nodo padre con nodos hijos (los trazos que conforman un kanji, lo que mencionamos como nodos internos en el capítulo anterior), también tiene definido una búsqueda recursiva que recorre el árbol para la identificación de un kanji, por último tiene definido la construcción de los árboles de dos, tres y cuatro trazos.

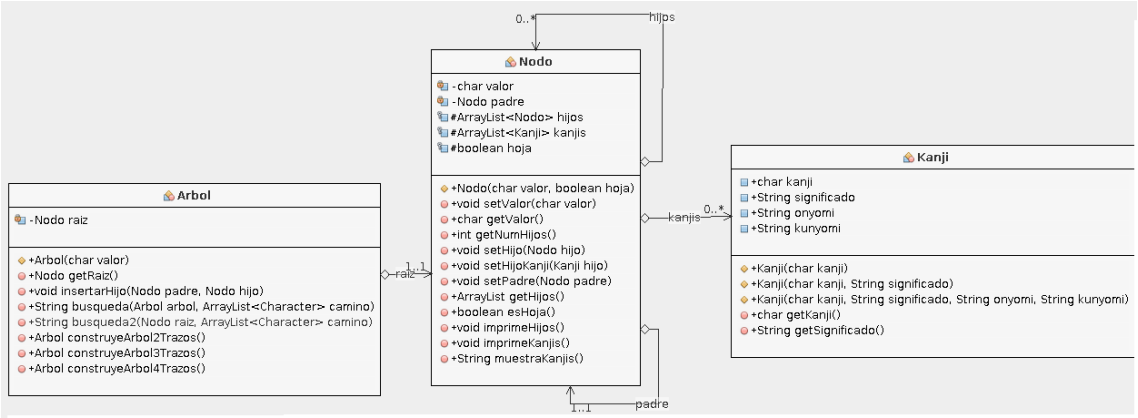


Figura 5.12: UML de las Clases involucradas en la codificación de los árboles

Es importante recordar que los trazos se codificaron como caracteres alfabéticos, esto implica que la representación de los kanjis se realiza a partir de esta codificación, como se muestra en la tabla 5.4.

山	丨	乚	丨	a	g	a
口	丨	冂	乚	a	c	b
上	丨	一	乚	a	b	b
小	亅	丿	丶	j	i	h
川	丿	丨	丨	e	a	a
大	一	丿	乀	b	e	f
土	一	丨	乚	b	a	b
三	一	一	乚	b	b	b
万	一	乛	丿	b	s	e
下	一	丨	丶	b	a	h
子	乛	亅	一	l	m	b
女	𠂉	丿	一	n	e	b
工	一	丨	乚	b	a	b
々	丿	乛	丶	e	l	h

Tabla 5.4: Kanjis visto como secuencia de caracteres (trazos)

Con base en este cambio, los árboles de decisión están representados como se ve en la Figura 5.13

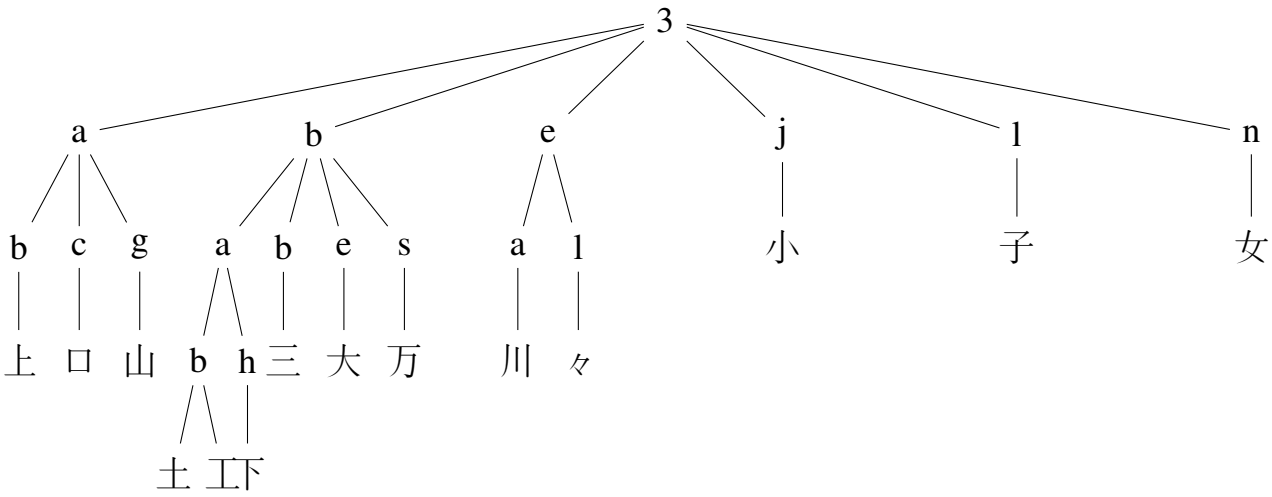


Figura 5.13: Representación de visual del árbol de tres trazos

5.3.2. Identificación del kanji

Para realizar la identificación de un kanji la aplicación tiene que tener plenamente determinado el número de trazos, así como un arraylist cuyo contenido es la representación alfabética obtenida en la codificación de los trazos realizados por el usuario. A partir de estos datos se toma el árbol cuyo número de trazos coincide con el dato almacenado, descartando así a los demás árboles. Por ejemplo si la aplicación identifica que el usuario realizó tres trazos se hará la identificación del kanji en el árbol de tres trazos, como podemos ver en la representación visual del árbol de tres trazos de la figura 5.13.

El arraylist al estar conformado por caracteres alfabéticos simplemente estos se van comparando con los caracteres del árbol y se da una respuesta. Siguiendo con el ejemplo. Se ha identificado que el usuario hizo tres trazos, y el arraylist generado es [**b,e,f**] Partiendo así de la raíz y primero compara **b** con **a**, como no coinciden se pasa al siguiente hijo de la raíz, en este caso **b**, compara **b** con **b** y coinciden, como no es nodo hoja, actualiza la identificación del kanji ahora considerando como nodo raíz **b** y con un arraylist conformado por[**e,f**]

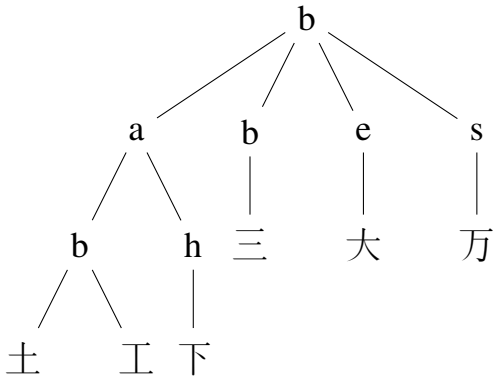


Figura 5.14: Actualización de búsqueda de kanjis

se compara sin éxito con **a, b** y al comparar con **e**, coincide, se pregunta si el nodo **e** es nodo hoja, como la respuesta es afirmativa toma a los kanjis ligados a este, terminando así la identificación del kanji.

5.4. Aplicación para móviles de apoyo al aprendizaje del idioma japonés

Como ya se mencionó en el capítulo 1 buscar el significado de un kanji puede ser complicado. Considerando la estrategia de diseñar diccionarios japonés-español para dispositivos móviles, que permitan al usuario trazar el kanji en el dispositivo, en este trabajo se realizó un prototipo para dispositivos móviles con Windows.

Esta aplicación está orientada a usuarios que estudian o han estudiando japonés de manera formal, ya que asume que tiene conocimiento de las reglas básicas para el trazado de kanjis. La aplicación permite que el usuario realice el trazado de un kanji y muestra el kanji reconocido y su significado.

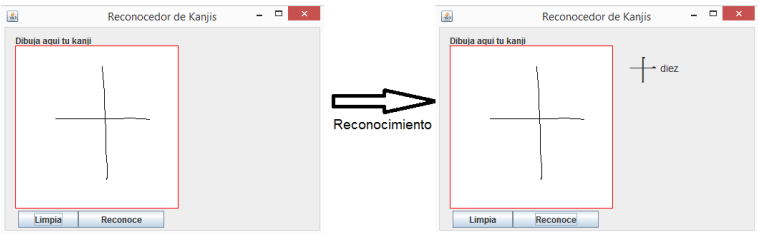


Figura 5.15: Diagrama del funcionamiento de la aplicación.

A continuación presentamos una explicación más detallada de su funcionamiento.

Una vez que inicializa la aplicación se muestra la pantalla inicial y debemos esperar que cargue el archivo de inicialización como podemos ver en la figura 5.16

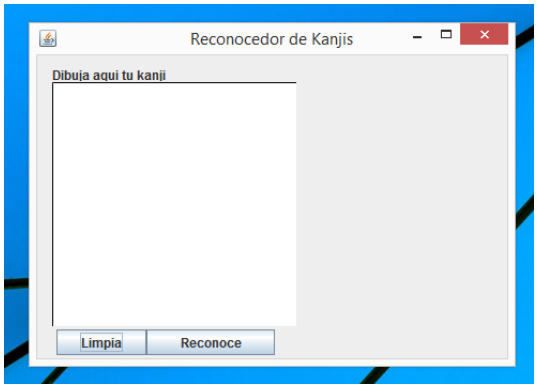


Figura 5.16: Pantalla inicial

En esta etapa se está entrenando el mapa autoorganizado y hace la construcción de los árboles de decisión para realizar la identificación del kanji.

Una vez terminada la inicialización aparecerá el mensaje que lo indica, como se puede ver en la figura 5.17.

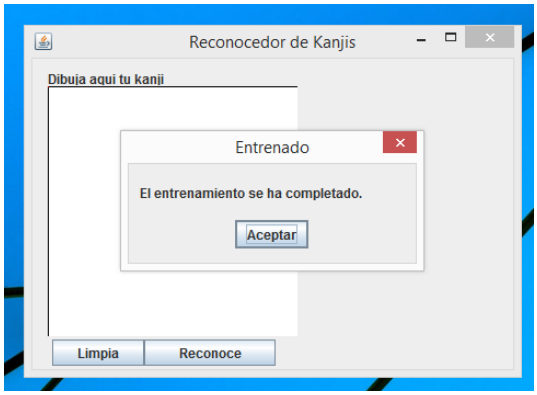


Figura 5.17: Inicialización terminada

Cerramos el mensaje y procedemos a realizar un primer trazo, como la figura 5.18. Cada vez que el usuario realice un trazo internamente identificara el trazo actualizando así la lista de trazos reconocidos hasta el momento y del número

de trazos, hasta que presione el botón “Reconoce” se realizará la identificación del kanji.

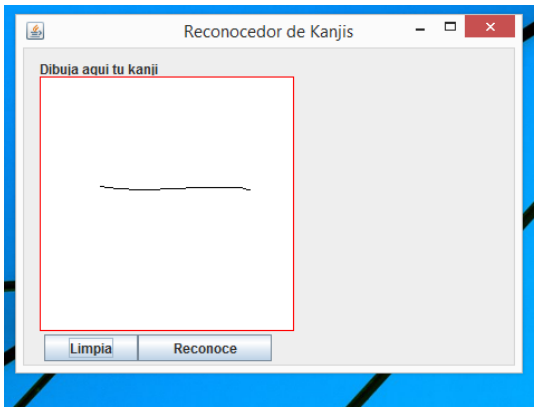


Figura 5.18: Realización de un trazo

Al terminar de hacer un trazo, la aplicación reconoce el trazo y almacena su representación en un arraylist. En el ejemplo almacena al trazo **b**, y tiene como número de trazos “uno”.

Si presionamos el botón “Reconocer” nos mostrará a un lado del lienzo el kanji y su significado, como se puede ver en la figura 5.19. Como solo existe un kanji de un solo trazo no es necesario buscar en algún árbol pero si verifica que sea el trazo **b**.

Figura 5.19: Reconocimiento de kanji de un trazo

Considerando que el usuario hiciera dos trazos como los de la siguiente figura 5.20.

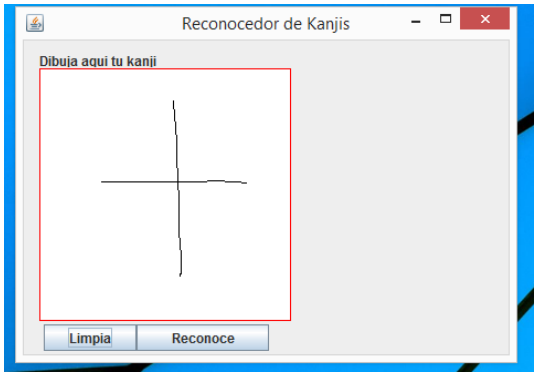


Figura 5.20: Realización de un segundo trazo

La aplicación almacena la representación de los dos trazos [**b,a**] y el número de trazos ahora es dos, por lo que si usamos el botón "Reconocer" buscará en el árbol de dos trazos como se describe a continuación: buscará en los nodos hijos al nodo que represente a **b**; a continuación, busca el nodo que represente a **a**, si el nodo es una hoja, muestra los kanjis que están ligados a este nodo, en este caso solo es un kanji, como lo podemos ver en la figura 5.21.

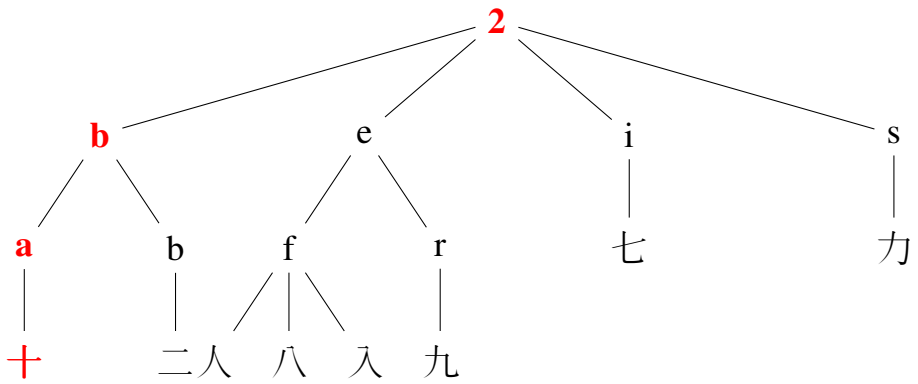


Figura 5.21: Árbol de dos trazos

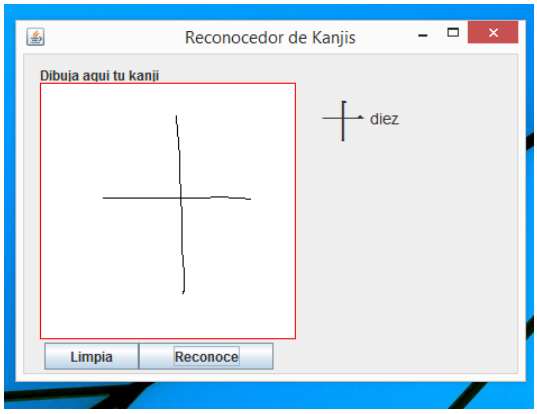


Figura 5.22: Reconocimiento de kanji de dos trazos

Si el usuario así lo quiere, puede continuar realizando trazos, extendiendo este ejemplo, el usuario realiza un trazo más, así que la aplicación actualizará el número de trazos a tres y actualizará la representación del trazo [**b,a,b**]. Hasta que el usuario presiona el botón "Reconocer" empezará la identificación del kanji en el árbol de tres trazos que podemos ver en la figura 5.23,

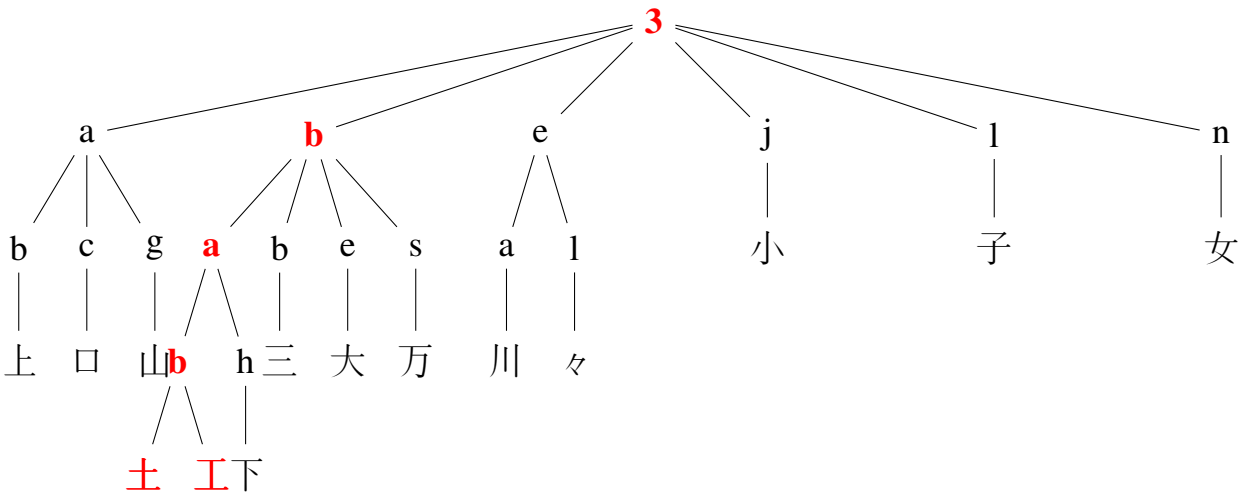


Figura 5.23: Árbol de tres trazos

en este ejemplo podemos ver que la aplicación al considerar los trazos y el orden nos puede dar varios kanjis (los kanjis que concuerden con los trazos y el orden), en este caso nos muestra dos kanjis, los cuales podemos ver en la figura 5.24.

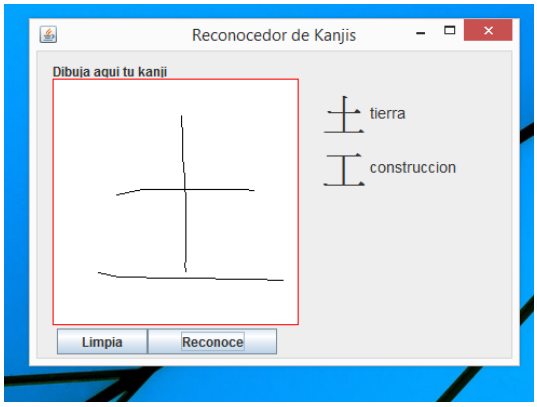


Figura 5.24: Reconocimiento de kanjis conformados por los trazos **b,a,b**

Ahora si el usuario hace un garabato en lugar de un kanji, como el de la figura 5.25

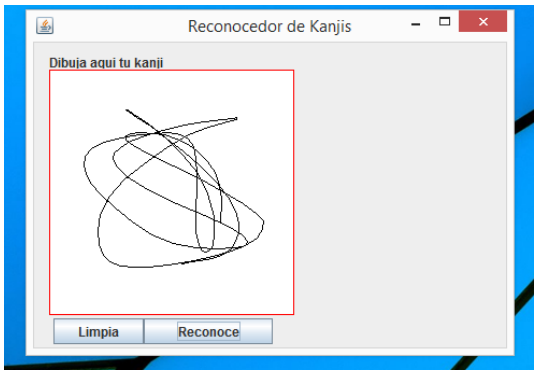


Figura 5.25: Realización de garabato

La aplicación tratará de reconocer al “trazo”, otorgando un símbolo, sin embargo al realizar la identificación del kanji en los árboles fallará y mostrará el mensaje de la figura 5.26

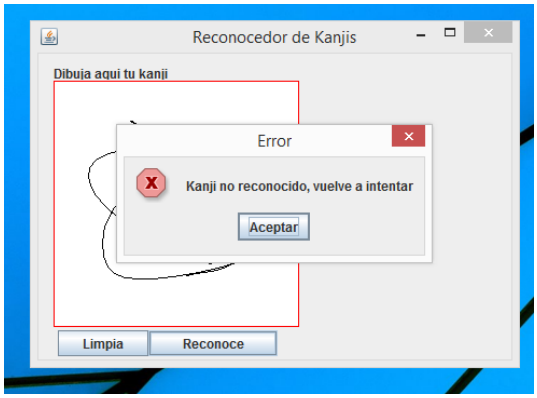


Figura 5.26: Mensaje de fallo en el reconocimiento de kanji

Cerramos el mensaje y presionamos el botón “Limpia” dejara el lienzo en blanco como puede ver en la figura 5.27, dejando los valores de número de trazos en cero y el arraylist vacío en espera de un nuevo trazo

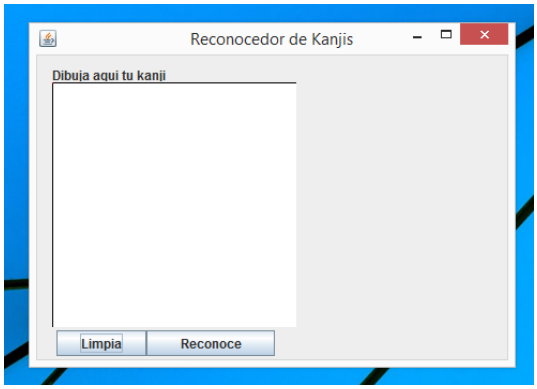


Figura 5.27: Limpieza del lienzo

Cabe aclarar que no es necesario presionar el botón “Reconoce” cada vez que se realiza un trazo, de hecho lo esperado es presionarlo hasta terminar de trazar el kanji que se desea identificar.

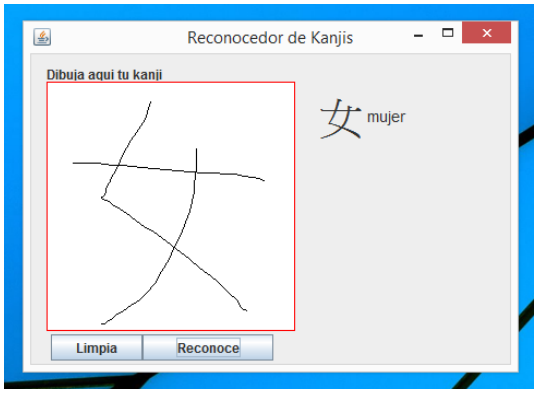


Figura 5.28: Realización de todos los trazos antes de presionar el botón “Reconoce”

Como puede observar en la figura 5.28 la aplicación otorga una cierta flexibilidad al reconocer el kanji, ya que los trazos no son exactamente iguales a los del kanji que muestra.

5.5. Pruebas de reconocimiento de kanjis trazados por usuarios reales

Recordemos que el conjunto de muestra fue construido por un estudiante de nivel avanzado del idioma, en esta sección se describe los resultados con usuarios de diferente nivel de conocimiento de la lengua japonesa.

Se hicieron pruebas con kanjis de un trazo (un kanji: 一), de dos trazos (ocho kanjis: 人,二,七,八,十,入,力,九), de tres trazos (catorce kanjis: 山,口,上,小,川,大,土,三,万,下,子,女,工,々) y de cuatro trazos(treinta y cuatro kanjis: 日,月,木,火,水,五,六,円,中,分,手,文,牛,少,方,午,今,友,父,夫,元,切,不,止,内,公,区,化,欠,予,天,引,心,太), un total de cincuenta y siete kanjis.

La aplicación fue sometida a pruebas con tres tipos de usuarios:

- Usuario con completo desconocimiento del idioma
- Usuarios con conocimientos adquiridos por su propia cuenta (sin instrucción formal)
- Usuario con conocimientos adquiridos en clases con instrucción formal (en el instituto Nihongo Kyoukai)

5.5.1. Usuario con completo desconocimiento del idioma

Para el primer tipo usuario mencionado (con completo desconocimiento del idioma), solo se tomó un solo usuario y durante sus pruebas en un comienzo se usaron los kanjis más simples como son 一, 二,三,川. La aplicación tuvo éxito en el reconocimiento de estos. Para kanjis más complejos y no tan intuitivos la aplicación no logró reconocer el kanji trazado por el usuario, mostraba el mensaje de kanji no reconocido, o bien, la aplicación mostraba otro kanji.

Al trazar el kanji 口 , la aplicación arrojó el kanji 止, esto nos da la impresión de que la aplicación ha fallado rotundamente ya que no se parecen estos kanjis, sin embargo analizando la forma en que el usuario realizó el trazo vemos lo siguiente:

Una persona sin conocimiento del idioma puede pensar que el kanji está conformado por cuatro líneas rectas, dos horizontales y dos verticales que se pueden trazar en cualquier orden. en el caso de nuestro usuario trazó el kanji 口 en el orden (a) de la figura 5.29, mientras que de acuerdo con las reglas del idioma se debe trazar en el orden (b) mostrado en la figura 5.29.

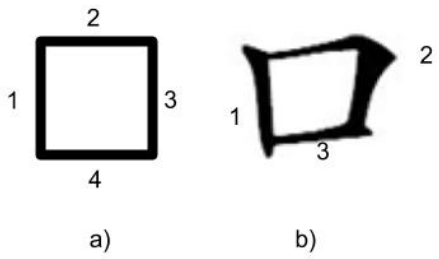


Figura 5.29: Kanji trazado
(a)Kanji realizado por el usuario.
(b)Kanji trazado correctamente.

Recordemos cómo funciona la aplicación, en primer lugar conforme el usuario va trazando los trazos, estos son reconocidos,

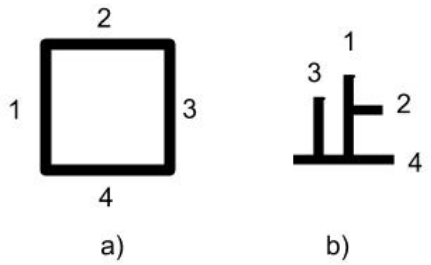


Figura 5.30: Lógica del error al mostrar kanji equivocado.
(a)Trazos realizados por el usuario.
(bTrazos correndientes del kanji arrojado por la aplicación.

si observamos sólo el orden de los trazos la figura 5.30 veremos que los trazos de (a) y (b) coinciden: el trazo 1 tanto de la figura a) y b) son iguales, son líneas verticales, para el segundo trazo ambos casos son una línea horizontal y para los trazos 3 y 4 son casos análogos, eso quiere decir que la aplicación si esta reconociendo los trazos individuales correctamente. Como segundo paso, la aplicación tiene la tarea de identificar el kanji, entonces ¿en esta parte fallo la aplicación? recordemos que la segunda parte del funcionamiento de la aplicación se basa en la identificación del kanji en los árboles de decisión, pero estos fueron construidos a partir del hecho que los kanjis deben seguir una secuencia ordenada de trazos, en este caso el kanji debe ser realizado en tres trazos mientras que el el usuario realizó 4 trazos. La aplicación intenta buscar el kanji en el árbol de cuatro trazos siguiendo el orden en el que el usuario lo realizó y la aplicación identifica el kanji b), por lo que podemos concluir que la aplicación es consistente con el enfoque seguido en su desarrollo, pero falla cuando el usuario no cuenta con las nociones de las reglas de escritura de los kanjis.

5.5.2. Usuarios con conocimientos adquiridos sin instrucción formal

En estas pruebas se contó con dos usuarios con similar conocimiento, según lo reportado por estos usuarios de esta categoría, su fuente de información sobre el idioma son blogs de gente apasionada por la cultura japonesa. Al no contar con un instructor que les indicará que la tipografía que manejan las computadoras no es la 100 % correcta, no son conscientes de algunos aspectos importantes al trazar un kanji como la inclinación correcta de los trazos, en la figura 5.26 podemos ver un ejemplo.

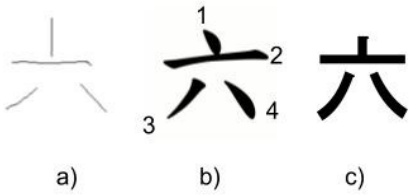


Figura 5.31: Falta de inclinación del primer trazo.
(a)Kanji realizado por el usuario.
(b)Kanji trazado correctamente.
(c)Kanji de tipografía de computadora

En la figura 5.26.a se observa el kanji realizado por el usuario, el cual se parece al que ofrece la tipografía de la computadora (figura 5.31.c) y no presenta los detalles del trazo correcto que se observa en la figura 5.31.b.

Otro error común que tuvo este tipo de usuario fue la falta de cuidado en la realización de los ángulos de los trazos, como podemos ver en la figura 5.31

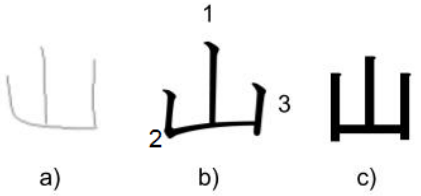


Figura 5.32: Segundo trazo con curvatura no propia del kanji.
(a)Kanji realizado por el usuario.
(b)Kanji trazado correctamente.
(c)Kanji de tipografía de computadora

Al no realizar el segundo trazo con ángulo más agudo, se considera que el kanji no está bien trazado y la aplicación no lo reconoce. El kanji del inciso 5.32.c) muestra cómo la tipografía de la computadora tampoco hace énfasis en estos detalles.

También se observó durante las pruebas que los usuarios agregaron curvaturas no propias de los trazos de los kanji que se convirtieron en ruido para la aplicación, impidiendo el reconocimiento del trazo adecuado, como por se puede ver la figura 5.33

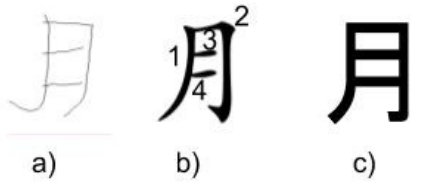


Figura 5.33: Trazado incorrecto del segundo trazo.
(a)Kanji realizado por el usuario.
(b)Kanji trazado correctamente.
(c)Kanji de tipografía de computadora

En el ejemplo de la figura 5.33 puede ver que en lugar del gancho que caracteriza al segundo trazo, el usuario hizo una curvatura impropia del trazo.

Otro ejemplo donde el usuario no da la importancia al gancho del kanji se puede ver en la figura 5.34.

Figura 5.34: Omisión del gancho en segundo trazo.
(a)Kanji realizado por el usuario.
(b)Kanji trazado correctamente.
(c)Kanji de tipografía de computadora

Cuando estos usuarios respetaron las reglas del trazado de los kanjis, la aplicación mostraba los kanjis esperados.

Por último se muestra un ejemplo donde existe una falta de proporción de los trazos, ya que el usuario intenta trazarlo de forma similar a la tipografía de la computadora y no de acuerdo a las reglas del idioma.

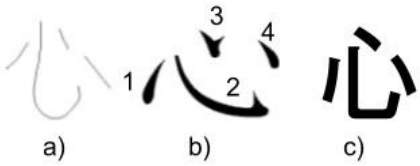


Figura 5.35: Kanji mal trazado.
(a)Kanji realizado por el usuario.
(b)Kanji trazado correctamente.
(c)Kanji de tipografía de computadora

5.5.3. Usuario con conocimientos adquiridos en clases con instrucción formal

Solo se contó con un solo usuario de este tipo, el cual estudió en el instituto Nihongo Kyoukai, es conveniente resaltar que este usuario tomó clases de profesores de una sede distinta del CELE [1], sin embargo, el método de enseñanza es el mismo, eso quiere decir, que las reglas de enseñanza de la lengua son iguales y podemos hacer una afirmación aún más fuerte, dado que las clases son impartidas por personas de nacionalidad japonesa, podemos decir que las reglas del idioma son las mismas aquí y en japón.

Regresando a las pruebas con este usuario, esté realizó el trazo de 16 kanjis, de los cuales reconoció al primer intento 15, de hecho, la única falla al reconocer un kanji surgió al no respetar las reglas, al indicarle que la aplicación era tan estricta con las reglas del idioma, el usuario tuvo el debido cuidado al trazar dando por resultado el reconocimiento de los 16 kanjis que realizó, el usuario no realizó más kanjis debido que no recordó en su momento más kanjis de esa cantidad de trazos, haciendo el comentario final de que sería una excelente aplicación si reconociera una mayor gama de kanjis.

5.6. Resumen

En este capítulo se describe la implementación de Heaton [8] para el reconocimiento óptico de caracteres alfanuméricos, en la que hace uso de los mapas

[1] Centro de Enseñanza de Lengua Extranjera

autoorganizados. Con esta propuesta como referencia, se mostró como fue adaptada esta implementación para que reconozca los trazos básicos definidos en el capítulo anterior. En este capítulo también se describen las pruebas y el entrenamiento realizado para un óptimo reconocimiento de trazos. A continuación, se muestra la implementación del proceso de identificación de un kanji usando los árboles propuestos.

Una vez que fueron presentados los fundamentos del funcionamiento del reconocimiento de kanjis, se describe el funcionamiento de la aplicación para el entrenamiento del mapa autoorganizado, así como la aplicación para el apoyo al aprendizaje del idioma japonés.

Finalmente, se discuten los resultados de las pruebas con tres tipos de usuarios con los que se concluye que esta aplicación no es para todo tipo de usuario, si no que esta está diseñada para estudiantes que tienen noción de las reglas del idioma, que es el tipo de usuario que se esperaría que ocupará o necesitara de esta aplicación.

6 Conclusiones

6.1. Resumen

La idea de hacer esta aplicación surgió cuando la autora de esta se encontraba realizando la lectura de un texto japonés, al no reconocer todos los kanjis de la lectura y tener que recurrir a diccionarios de papel para hacer la búsqueda de estos, con lo cual se reconoce la complejidad y el tiempo para localizar un kanji. Al identificar esta problemática surgió la idea de hacer una aplicación que apoye al rápido reconocimiento de estos.

En el capítulo uno se presentaron a detalle la problemática, así como una muestra de las reglas del idioma japonés.

En el capítulo dos se abordó el problema del reconocimiento óptico de caracteres OCR, que es un acercamiento a la problemática que fue atacada, se muestran los inicios de la investigación sobre el OCR desde el dispositivo de Gustav Tauschek en el 1929, hasta investigaciones más actuales. En este capítulo se observa que el desarrollo de estos sistemas tuvo dos grandes vertientes: comparación de plantillas y análisis de estructuras. Estas vertientes han crecido hasta convertirse en una tecnología concreta de OCR y durante el proceso de investigación, estas se fueron fusionando para convertirse en una amplia corriente que constituye una parte esencial de las tecnologías del reconocimiento de patrones de OCR.

En el capítulo tres se abordan los métodos desarrollados por las culturas asiáticas (China, Corea y Japón) para atender la problemática del reconocimiento de su propia escritura.

China desarrolla el método de *función de extracción y clasificación de kanjis*, para este método es fundamental la posición, dirección y densidad de los trazos que componen a los kanjis para así identificarlos. También desarrolla el método de *coincidencia estructural*, que si bien el anterior método funcionaba de manera aceptable, este último fue desarrollado para escrituras más distorsionadas.

Para este método es importante contar con una base de datos de la información de los tipos de trazos horizontal, vertical, punto, gancho, etc.

Por otro lado en Corea desarrolla el método de *enfoque holístico*, este método está basado en la categorización de clases de caracteres Hangul, y el método *basado en la segmentación*, donde segmentan grafemas, sin embargo estos son muy impreciso por lo que desarrollaron el método de *aproximación en línea*, donde extraen los trazos y la secuencia de cómo fueron escritos.

En Japón se propone por primera vez un preprocesamiento conocido como normalización, los métodos desarrollados por ellos se basan en la obtención de la posición y tamaño de los kanjis, también plantearon el análisis de estos a partir de sus radicales.

Al final del capítulo tres, se muestra las técnicas de reconocimiento de kanjis en diccionarios actuales, ya sean digitales o de papel, los cuales están basados en el número de trazos, el radical principal y lectura.

En el capítulo cuatro ofrecemos una breve introducción a las redes neuronales, las cuales tienen dos tipos de aprendizaje: supervisado y no supervisado; en nuestro caso utilizamos el no supervisado. Dentro del conjunto de las redes no supervisadas encontramos a los mapas autoorganizados. A esta red se le presenta un patrón de entrada, previamente normalizado a la capa de entrada, la salida de la red está dada por una única neurona ganadora para cada patrón presentado.

Para entrenar al mapa autoorganizado es necesario realizar varias iteraciones donde se presentan los elementos de entrenamiento. Los pesos de esta neurona se modifican para que respondan con mayor fuerza al patrón que la hizo ganar.

También en este capítulo fueron presentados los árboles de decisión n-arios, estos fueron vistos desde dos enfoques, vistos como árboles n-arios y como árboles de decisión. Se presentaron generalidades de los árboles, así como su funcionamiento como árbol de decisión, además de puntualizar la complejidad y rendimientos de estos para así mostrar la metodología desarrollada.

Al final del capítulo cuatro, se presenta la metodología desarrollada, que se basa en las técnicas anteriormente mencionadas. A partir de un conjunto de 172 kanjis (los primeros kanjis que enseñan a un estudiante del idioma japonés en

el CELE [1]) se identificaron los trazos que los conforman, para así encontrar los trazos más comunes que los conformaban, a estos se les asignó el nombre de "trazos básicos". Usando los mapas autoorganizados se propuso el reconocieron de los trazos básicos.

Dado que los kanjis se pueden ver como un conjunto ordenado de trazos, se propuso agruparlos por el número de trazos que los conformaban, creando grupos de dos, tres, cuatro, cinco, seis y siete trazos. A partir de estos grupos se crearon las tablas que sirven como base para la creación de los árboles de decisión. Al analizar estos árboles de trazos, se observó que muchas ramas de estos tendían a definir un solo kanji desde antes de llegar a una hoja, esto se debe a que los kanjis son únicos a partir de un cierto trazo, por lo que se tomó la decisión de podar a los árboles, esto permite lograr una identificación rápida y sin la necesidad de verificar que el usuario haga todos los trazos correctamente.

Una vez construidos estos árboles, el proceso de identificación del kanji trazado por el usuario inicia seleccionando el árbol cuyo número de trazos coincida con los trazos realizados hasta ese momento, y a partir del orden de los trazos del usuario se puede identificar el kanji que realizó.

En el capítulo cinco se describe la implementación de Heaton [8] para el reconocimiento óptico de caracteres alfanuméricos, en la que se hace uso de los mapas autoorganizados. Con esta propuesta como referencia, se mostró como fue adaptada esta implementación para que reconozca los trazos básicos definidos en el capítulo cuatro. En este capítulo también se describen las pruebas y el entrenamiento realizado para un óptimo reconocimiento de trazos. A continuación, se muestra la implementación del proceso de identificación de un kanji usando los árboles propuestos.

Una vez que fueron presentados los fundamentos del funcionamiento del reconocimiento de kanjis, se describe el funcionamiento de la aplicación para el entrenamiento del mapa autoorganizado, así como la aplicación para el apoyo al aprendizaje del idioma japonés. Finalmente en el capítulo cinco, se discuten los resultados de las pruebas con tres tipos de usuarios.

[1] Centro de Enseñanza de Lengua Extranjera

6.2. Conclusiones

El primer reto a resolver en este trabajo fue la selección de trazos básicos, para ello se realizaron diferentes propuestas de trazos y se observaba el desempeño del SOM. Esta actividad fue fundamental para el funcionamiento final de la aplicación.

Al inicio de este trabajo en 2014, se pensó que había pocos desarrollos sobre reconocimiento de kanjis en aplicaciones móviles ya que eran escasas las aplicaciones disponibles para Android. “Kanji Recognizer”, una de las más completas fue la que se analizó y se observó que usa como estrategia el número de trazos pero solo brinda una traducción al inglés. Actualmente ya están disponibles más aplicaciones de este tipo pero solo algunas consideran el idioma español.

En este trabajo se considero como usuario final a los estudiantes de japonés y los métodos utilizados para su enseñanza, por lo cual la aplicación propuesta requiere que el kanji sea trazado siguiente las reglas gramaticales.

La aplicación fue probada con tres tipos de usuarios: usuario con completo desconocimiento del idioma, usuarios con conocimientos adquiridos sin instrucción formal y usuario con conocimientos adquiridos en clases con instrucción formal. En estas pruebas fue notable que dependiendo del tipo de usuario la aplicación funcionaba de mejor manera, en otras palabras, entre mayor conocimiento de la lengua y sus reglas gramaticales mejor fue el desempeño de la misma y esto se debe a que esta aplicación fue construida en base a estas reglas.

Esto es consistente con los objetivo de la aplicación, ya que esta aplicación no es para todo tipo de usuario, si no que está diseñada para estudiantes que tienen noción de las reglas del idioma.

Uno de los comentarios de los usuarios es que la aplicación sólo reconoce una pequeña gama de kanjis.

Finalmente, si bien esta aplicación fue implementada por las necesidades de la autora, esta aplicación puede ayudar a estudiantes principiantes del idioma.

6.3. Trabajo a Futuro

Cuando se empezó a realizar la construcción de los árboles a partir de los trazos fueron analizados 172 kanjis, los cuales fueron agrupados por el número de trazos que lo conformaban creando grupos de dos, tres, cuatro, cinco, seis y siete trazos. La aplicación incluye solo los árboles de dos, tres y cuatro trazos, se espera que pueda ser ampliada, a corto plazo, para que contenga a los árboles de cinco, seis y siete trazos. Cabe mencionar que la construcción de estos árboles ya se realizó a papel. Con esta extensión, la aplicación podría reconocer 172 kanjis en lugar de solo 57 kanjis.

Cabe mencionar que con este prototipo se buscó probar la metodología propuesta para el reconocimiento de kanjis, por lo cual no se trabajó a profundidad la interfaz gráfica de la aplicación. Una posible mejora es trabajar en el diseño de esta interfaz y realizar las pruebas de usabilidad correspondientes.

Por otro lado, cabe la posibilidad de realizar una aplicación de evaluación de estudiantes del idioma japonés, teniendo como base los árboles sin podar, esta aplicación verificará que el usuario esté trazando los kanjis de manera correcta, lo cual es muy común en exámenes que suelen hacer en las instituciones de enseñanza del idioma.

Como ya fue anteriormente mencionado, la aplicación solo funciona en dispositivos móviles con Windows, por lo que se podría hacer una versión para dispositivos Android, que son de uso más común.

Bibliografía

[1] G. Tauschek, "Reading machine," Patent U.S. 2 026 329, Diciembre, 1935. [Online]. Available: https://history-computer.com/Library/US2026329.pdf

[2] S. Mori, C. Y. Suen, and K. Yamamoto, "Historical review of ocr research and development," *Proceedings of the IEEE*, vol. 80, pp. 1029–1058, 1992.

[3] R. B. Johnson, "Indicia controlled record perforating machine," Patent U.S. 2741 312, Abril, 1956. [Online]. Available: http://www.google.tl/patents/US2741312

[4] C. Y. Suen, S. Mori, S. H. Kim, and C. H. Leung, "Analysis and recognition of asian scripts - the state of the art," *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, vol. 2, pp. 1–13, Agosto 2003.

[5] S. Kim, "Performance improvement strategies on template matching for large set character recognition," *Proc. 17th International Conference on Computer Processing of Oriental Languages*, pp. 250–253, Abril 1997.

[6] K.-W. Kang and J. H. Kim, "Utilization of hierarchical, stochastic relationship modeling for hangul character recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, pp. 1185–1196, 2004.

[7] H. Kim and P. Kim, "Recognition of off-line handwritten korean characters," *Pattern Recognition*, vol. 29(2), pp. 245–254, 1996.

[8] J. Heaton, *Introduction to Neural Networks for Java, 2Nd Edition*, 2nd ed. Heaton Research, Inc., 2008.

[9] M. of education, culture, sports, science, and tecnology Japan, "学習指導要領「生きる力」." [Online]. Available: http://www.mext.go.jp/a_menu/shotou/new-cs/youryou/syo/koku/001.htm

[10] M. Bernabé, *Japonés en Viñetas. Curso básico de japonés a través del manga*. Editorial Norma, 2010.

[11] G. Company, "Google translator." [Online]. Available: https://translate.google.com/?hl=es#ja/es/%E5%8F%A3%0A%E3%81%8F%E3%81%A1

[12] J. Breen, "El instituto yamasa - オンライン日本語学習 - diccionario de kanji online ," 2002. [Online]. Available: http://www.yamasa.org/ocjs/kanjidic.nsf/SortedByJoyoStrokeTHSpanish?OpenView&Start=1&Count=30&Expand=2.3#2.3

[13] Spacehamster, "Japanese." [Online]. Available: https://play.google.com/store/apps/details?id=com.space.japanese&hl=es

[14] hoang8f, "Japanese english dictionary." [Online]. Available: https://play.google.com/store/apps/details?id=info.hoang8f.japanese.plus&hl=es

[15] N. Elenkov, "Kanji recognizer." [Online]. Available: https://play.google.com/store/apps/details?id=org.nick.kanjirecognizer&hl=es

[16] Craxic, "Akebi japanese dictionary." [Online]. Available: https://play.google.com/store/apps/details?id=com.craxic.akebifree&hl=es

[17] renzoInc, "Japanese for android." [Online]. Available: https://play.google.com/store/apps/details?id=com.renzo.japanese&hl=es

[18] CJKI, "Kodansha kanji learner's dict." [Online]. Available: https://play.google.com/store/apps/details?id=org.kanji.cjki.kkld&hl=es

[19] P. A. Maker, "Filipino japanese dictionary." [Online]. Available: https://play.google.com/store/apps/details?id=com.pasawahanappmaker.filipino.tagalog.japanese.dictionary.translator&hl=es

[20] hoang8f, "Vietnamese japanese dictionary." [Online]. Available: https://play.google.com/store/apps/details?id=info.hoang8f.jdict&hl=es

[21] P. A. Maker, "Nepali japanese dictionary." [Online]. Available: https://play.google.com/store/apps/details?id=com.pasawahanappmaker.nepali.japanese.dictionary.translator&hl=es

[22] M. Eduplay, "Malaysia japanese dictionary." [Online]. Available: https://play.google.com/store/apps/details?id=id.melatieduplay.dictionary.translator.malay.malaysia.japanese&hl=es

[23] ——, "German japanese dictionary." [Online]. Available: https://play.google.com/store/apps/details?id=id.melatieduplay.dictionary.translator.german.japanese&hl=es

[24] S. Entertainment, "diccionario japonés." [Online]. Available: https://play.google.com/store/apps/details?id=org.lakedaemon.japanese.dictionary&hl=es

[25] S. Mori and T.Sakakura, "Fundamentals of image recognition," *Tokyo: Ohm*, vol. I,II, 1986 and 1990.

[26] T. Iijima, *Theory of Pattern Recognition*. Morishita Publishing, 1989.

[27] P. Handel, "Statistical machine," Patent U.S. 1915 993, Junio, 1933. [Online]. Available: https://www.google.com/patents/US1915993

[28] M. H. Glauberman, "Character recognition for business machines," *Electronics*, pp. 132–136, Febrero 1956.

[29] ERA, "An electronic reading automaton," *Electronic Eng*, pp. 189–190, Abril 1957.

[30] T. Iijima, Y. Okumura, and K. Kuwabara, "New process of character recognition using sieving method," *Information and Control Research*, vol. I, pp. 30–35, 1963.

[31] W. S. Rohland, "Character sensing system," Patent U.S. 2 877 951, Marzo, 1959. [Online]. Available: https://www.google.com/patents/US2877951

[32] R. W. Weeks, "Rotating raster character recognition system," *AIEE Trans*, vol. 80, pp. 353–359, Septiembre 1961.

[33] W. H. Highleyman, "Linear decision functions with applications to pattern recognition," *Proc. IRE*, vol. 50, pp. 1501–1514, Junio 1962.

[34] R. . Duda and P. E. Hart, "Pattern recognition and scene analysis," *New York: Wiley*, 1973.

[35] G. Nagy, "Optical character recognition-theory and practice," *Amsterdam: North-Holland*, vol. 2, pp. 621–649, 1982.

[36] T. L. Dimond, "Devices for reading handwritten characters," *Proc. Eastern Joint Computer Conf*, pp. 207–213, Junio 1968.

[37] L. A. Kamentsky, "he simulation of three machines which read rows of handwritten arabic numerals," *RE Trans. Electron. Comput*, vol. EC-10, pp. 489–501, Septiembre 1961.

[38] T. Sakai, M. Nagao, and Y. Shinmi, "A character recognition system," *Proc. Annual Conj IECE Japan*, p. 450, 1963.

[39] N. N. E. Company), "Improvements in or relating to character recognition apparatus," Patent U.K. 1 124 130, Agosto, 1968.

[40] J. H. Munson, "Experiments in the recognition of hand-printed text: Part i, character recognition," *Proc. I968 Fall Joint Comput. Conj, AIlPS Conf*, vol. 33, pp. 1125–1138, 1968.

[41] E. C. Greanias, P. E. Meagher, R. J. Norman, and P. Essinger, "The recognition of handwritten numerals by contour analysis," *IBM J . Res. Develop*, vol. 7, pp. 2–13, 1963.

[42] S. Mori, K. Yamamoto, and M. Yasuda, "Research on machine recognition of handprinted characters," *IEEE Trans. on PAMI*, vol. 6, pp. 386–405, 1984.

[43] N. Kato, M. Suzuki, S. Omachi, H. Aso, and Y. Nemoto, "A handwritten character recognition system using directional element feature and asymmetric mahalanobis distance," *IEEE Trans. on PAMI*, vol. 21, pp. 258–262, 1999.

[44] Y. Tang, L. Tu, J. Liu, S. Lee, W. Lin, and I. Shyu, "Offline recognition of chinese handwriting by multifeature and multilevel classification," *IEEE Trans on PAMI*, vol. 20, pp. 556–561, 1998.

[45] T. Wakahara and Y. Kimura, "Toward robust handwritten kanji character recognition," *Pattern Recognition Letters*, vol. 20, pp. 979–990, 1999.

[46] C. Leung and C. Suen, "Matching of complex patterns by energy minimi-

zation," *IEEE Trans. Systems, Man and Cybernetic*, vol. 28, pp. 712–720, 1998.

[47] F. Cheng, "Multi-stroke relaxation matching method for handwritten chinese character recognition," *Pattern Recognition*, vol. 31, pp. 401–410, 1998.

[48] C. Liu, I. Kim, and J. Kim, "Model-based stroke extraction and matching for handwritten chinese character recognition," *Pattern Recognition*, vol. 34, pp. 2339–2352, 2001.

[49] Y. Mizukami, "A handwritten chinese character recognition system using hierarchical displacement extraction based on directional features," *Pattern Recognition Letters*, vol. 19, pp. 595–604, 1998.

[50] S. Lee and H. Park, "A survey on korean character recognition researches," *Proc. 1st Workshop on Character Recognition*, pp. 3–46, 1993.

[51] S. Kim, S. Jeong, and I. Oh, "A survey on the off-line recognition of handwritten korean characters," *Proc. Conf. Korean Information System Society*, pp. 396–398, 1998.

[52] S. Jeong, K. Lim, and Y. Nam, "A combination method of two classifiers based on the information of confusion," *Proc. 8th Int. Workshop on Frontiers in Handwriting Recognition*, pp. 519–523, 2002.

[53] H. Song and S. Lee, "A self-organizing neural tree for large-set pattern classification," *IEEE Trans. Neural Networks*, pp. 369–380, 1998.

[54] K. Kang, "Hierarchical hangul character recognition with stochastic relationship modeling and candidate pruning," *Ph.D. Dissertation*, 2002.

[55] H. Kim and J. Kim, "Heirarchical random graph representation of handwritten characters and its application to hangul recognition," *Pattern Recognition*, vol. 34, pp. 187–201, 2001.

[56] Y. Hwang and S. Bang, "Recognition of a handwritten korean character by combining segments using constraints satisfying graph," *Proc. 6th Int. Workshop on Frontiers in Handwriting Recognition*, pp. 515–526, 1998.

[57] J. Tsukumo and H. Tanaka, "Classification of handprinted chinese cha-

racters using nonlinear normalization methods," *Proc. 9th Int. Conf. on Pattern Recognition*, pp. 168–171, Noviembre 1988.

[58] H. Yamada, K. Yamamoto, and T. Saito, "A nonlinear normalization method for handprinted kanji character recognition - line density equalization," *Pattern Recognition*, vol. 23, pp. 1023–1029, 1990.

[59] T. Ha and H. Bunke, "Off-line, handwritten numeral recognition by perturbation method," *IEEE Trans. Pattern Anal. Machine Intell*, vol. 19, pp. 535–539, 1997.

[60] P. Simard, Y. L. Cun, and J. Denker, "Efficient pattern recognition using a new transformation distance," *Advances in Neural Information Processing Systems*, vol. 5, pp. 50–58, 1993.

[61] T. Wakahara, "Handwritten japanese character recognition using adaptive shape normalization by global affine transformation," *Int. Journal of Comp. Processing of Oriental Languages*, vol. 15(2), pp. 117–131, 2002.

[62] T. Wakahara and K. Odaka, "Adaptive normalization of handwritten characters using global/local affine transformation," *IEEE Trans. Pattern Anal. Machine Intell*, vol. 20, pp. 1332–1341, 1998.

[63] H. Sakano and N. Miyamoto, "Handprinted character recognition by extended peripheral direction contributivity," *Natl. Conv. Rec*, p. 556, 1995.

[64] T. Saito, H. Yamada, and K. Yamamoto, "On the database etl9b of handprinted characters in kanjis chinese characters and its analysis," *IEICE Trans*, vol. J68-D, pp. 757–767, 1985.

[65] T. N., T. Wakabayashi, F. Kimura, and Y. Miyake, "High accuracy in similar characters using compound function," *EICE Trans*, vol. D-II, J83-D-II(2), pp. 623–633, Febrero 2000.

[66] C. de Enseñanza de Lengua Extranjera, *Kanji I*. México: UNAM, 2010.

[67] S.Haykin, *Neural Networks: A Comprehensive Foundations*. New York: Macmillan College Publishing Co., 1994.

[68] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. D. Jesús, *Neural Network Design*. University of Colorado, 1997.

[69] Kohonen, *Self-organizing Maps*. Springer Series in Information Sciences. Springer, 1995.

[70] B. M., "Kohonen's self organizing feature maps," 2005. [Online]. Available: http://www.ai-junkie.com/ann/som/som1.html

[71] V. S. and J. A., *Introducción a las Estructuras de Datos. Aprendizaje Activo Basado en Casos*. Pearson Educación de Colombia Ltda, 2008.

[72] I. Millington and J. Funge, *Artificial Intelligence for Games, Second Edition*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009.