DESARROLLO DE UNA HERRAMIENTA DE CÓDIGO ABIERTO PARA CONSTRUCCIÓN AUTOMÁTICA DE CARIOGRAMAS

Cristian Vizzarri

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Ciudad, Santa Fe
Año 2018

Desarrollo de una herramienta de código abierto para construcción automática de cariogramas

Cristian Vizzarri

Trabajo de grado presentado como requisito para optar al título de:
Ingeniero en Informática

Director:
Dr. Matías F. Gerard

Co-Director:
Dr. César E. Martínez

Facultad de Ingeniería y Ciencias Hídricas
Universidad Nacional del Litoral
Santa Fe
Argentina

Agradecimientos

Este trabajo es el fruto de un profundo período formativo en el que me sumergí durante este tramo de mi vida. Luego de atravesar la experiencia de todo el proyecto aprendí que no hubiera podido lograrlo solo. Sino que hubo mucha gente que me acompaño a lo largo de este recorrido. En estas líneas quiero agradecerles el tiempo que me brindaron.
En primer lugar quiero agradecerles a mis padres, los cuales me acompañaron en mis inicios en esta campaña y hasta este momento de una u otra manera. A mi madre estando de cuerpo presente y mi padre que desde el cielo todavía guía mi camino. A *Natalia*, mi amor, por ser mi compañera inseparable y por haberme sostenido y apoyado en cada paso con amor, comprensión y alegría. Y a mi compañero *Agustín* por las charlas de café.
En lo profesional, quiero agradecer enormemente a mis directores, *Matías* y *César*, por aceptar dirigirme. Convencido de que esta tesis no hubiese llegado a ser lo que es sin su guía. Me socorrieron pacientemente ante mis dudas y guiaron mis ideas por buen camino. También les quiero agradecer por la confianza depositada y dedicación desde el inicio de este trabajo.

Resumen

Las enfermedades genéticas son difíciles de diagnosticar. Existen distintas metodologías de diagnóstico, una de ellas es a través del estudio de anomalías en los cromosomas. Una forma de poner en evidencia esas anomalías es realizando una representación gráfica de los cromosomas llamada cariograma, e inferir a partir de ésta posibles afecciones en el individuo. Este proyecto se centra en la realización de una herramienta de código abierto para automatizar la construcción del cariograma a partir de imágenes segmentadas de cromosomas en metafase.

Palabras clave: clasificación de cromosomas, citogenética, segmentación de imágenes, cariotipo, aprendizaje automático..

Abstract

Genetic diseases are difficult to diagnose. There are different diagnostic methodologies, one of them is through the study of anomalies in chromosomes. One way to highlight these anomalies is by making a graphic representation of the chromosomes called a cariogram, and inferring possible conditions in the individual. This project focuses on the realization of an open source tool to automate the construction of the karyogram from segmented images of metaphase chromosomes.

Keywords: chromosome classification, cytogenetics, image segmentation, karyotype, automatic learning.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

En este capítulo se detallan los elementos que dan origen a este proyecto. Se hace una reseña de los antecedentes identificados, cómo se justifica el trabajo realizado, los objetivos planteados para ello y las restricciones tenidas en cuenta para el alcance. Por último se detalla la organización de capítulos del informe del proyecto.

1.1. Motivación

El análisis de cromosomas en estado de metafase, para la construcción del cariograma, implica la búsqueda visual e identificación de los mismos usando microscopios ópticos. Este proceso manual para la construcción del cariograma, normalmente se lleva a cabo por clínicos especializados que miran las imágenes de los cromosomas en estado metafásico, los identifican y, finalmente con el uso de conocimiento experto, los colocan en las ubicaciones que le corresponden. Son organizados en tamaños decrecientes por pares, esto lo deben realizar por cada célula de prueba y para cada uno de los pares cromosómicos que tenga la especie en estudio [26]. Por lo cual, este método que requiere la inspección visual, es tedioso, laborioso, consume mucho tiempo, necesita de un experto y es costoso en términos monetarios. Además, se requiere de mucho entrenamiento y experiencia por parte del citogenetista. Incluso en el caso del personal entrenado, se necesita de hasta 3 pasadas de revisión para asegurar su correcto armado[1]. Los cariogramas confeccionados por un operador humano se considera que presentan un porcentaje mínimo de error. Los sistemas informáticos buscan acercarse a éste porcentaje.

Por ello, el armado automático de cariogramas ha sido tema de extensa investigación en todo el mundo en las últimas décadas [38, 45, 33, 52]. Constituye básicamente un problema de reconocimiento y clasificación. Al igual que con cualquier problema de reconocimiento de patrones, el éxito de un algoritmo depende en gran medida de la efectividad del sistema subyacente de representación de entidades que posea [73], de las cuales se van a extraer características para su clasificación. Se han propuesto varios esquemas para derivar características de los patrones de bandeo cromosómico. Estos métodos se pueden clasificar en

[1] http://pendientedemigracion.ucm.es/info/genetica/AVG/practicas/cariotipo/cario.htm

Tabla 1-1: Cuadro comparativo de las opciones comerciales existentes.

	Microscopio con Cámara Incluida	Hardware y Software Incluido[1]	Trabajo multiusuario en red	Procedencia
MetaSystems[2]	SI	SI	SI	Alemania
Microptic[3]	NO	SI	NO	España
Leica Biosystems[4]	SI	SI	SI	Alemania o EEUU
Imstar[5]	SI	SI	NO	Francia

[1] Se refiere a un PC, un monitor y periféricos.
[2] https://metasystems-international.com/us/products/neon/
[3] http://www.micropticsl.com/es/productos/metaclass-cariotipador-fish/
[4] http://www.leicabiosystems.com/es/flujo-de-trabajo/cytogenetics/detalles/product/cytovision/
[5] http://www.imstarsa.com/products-2/pathfinder/morphoscan/karyo/

general en técnicas globales o locales [47, 14]. Los enfoques globales calculan las propiedades matemáticas de los patrones de bandas que generalmente están representados por imágenes 1-D longitudinales de intensidad de la imagen de cromosomas. Los enfoques locales buscan identificar y describir las estructuras particulares en imágenes 1-D o 2-D de los cromosomas.

En lo referente a la etapa de clasificación, existe una gran variedad de algoritmos que han sido empleados exitosamente, y entre los que se encuentran métodos estadísticos [52], las redes neuronales artificiales (ANN) [72], Modelos ocultos de Markov (HMM) [11], Deep Learning [48], Transportation Algorithm [68],y los sistemas de lógica difusa [28].

Aplicando una o algunas de este tipo de técnicas o similares, existen en el mercado herramientas de software propietario que permiten construir el cariograma de forma automática a partir de imágenes de cromosomas en estado metafásico. Estas aglomeran la implementación de una gran parte del conocimiento acumulado sobre técnicas automáticas de cariotipado, que se han conseguido a partir de distintos enfoques a lo largo de los años. La tabla **1-1** resume las características de algunas de las opciones más importantes que se pueden encontrar en el mercado.

Como se puede observar, los sistemas MetaSystems y Leica Biosystems incluyen el equipo de adquisición (Microscopio con cámara adecuada), el Hardware y Software de cariotipado, adicionalmente presentan la posibilidad de que distintos usuarios trabajen en red. Permiten ser instalados es un servidor central y la conexión a este de múltiples terminales que pueden ejecutar distintas funcionalidades. De esta manera garantizan la centralización de la información (mantener los datos en un solo lugar). Además es más fácil administrar los datos propiamente dichos; así como el acceso remoto a ellos. El sistema de Imstar soporta las mismas características que los dos sistemas nombrados anteriormente, y también incluye la cámara, hardware y software para procesamiento de imágenes y la construcción automática del cariograma. Sin embargo, no cuenta con la posibilidad de trabajo en red. Microptic sólo ofrece el hardware y el software, y deja a cargo del comprador la adquisición del equipo de captura de imágenes; recomendando los requisitos mínimos requeridos para el microscopio y la cámara que se deben adquirir para el correcto funcionamiento del sistema. Este último punto no es menor, ya que se debe tener en cuenta este costo extra en caso de no contar con el equipo, además de su calibración y puesta a punto.

1.2. Estado del arte

La cuestión reside, en que si bien existe una diversidad de herramientas disponibles en el mercado que realizan la tarea de cariotipado automático, no existe ninguna que sea de código abierto. Traen aparejado las limitaciones que acompañan a todos los sistemas cerrados tales como un alto costo de adaptación de nuevos módulos, imposibilidad de compartir, problemas al discontinuar una línea de software por desaparición o venta de la compañía, quedar sin soporte y/o código secreto. La causa de ello radica en que el desarrollo de una herramienta de este tipo requiere de conocimiento específico en técnicas de aprendizaje automático, las cuales no están al alcance de todos por diversas razones, ya sea por formación, recursos económicos, recursos de equipamiento, sociales, el posible desinterés en la temática de las comunidades open source, entre otras.

Este proyecto surge con la motivación de cubrir la falta de una herramienta de código abierto que realice esta tarea. Para ello, se propone el desarrollo de una herramienta de código abierto en base a técnicas de aprendizaje automático, investigando sobre ellas y su aplicabilidad.

El beneficio en el aspecto social relacionado al uso de la herramienta es amplio. En principio, permitiría a grupos heterogéneos de personas la posibilidad de experimentar el uso de la herramienta sin costo de licencias y obtener una aproximación para determinar una patología sin recurrir a caros especialistas que ordenen los cromosomas en metafase. Más en particular se puede observar su aplicación en el ámbito educativo donde serviría para que los estudiantes practiquen y aprendan; de otra manera, es muy probable que la primera vez que experimenten con herramientas de este tipo sea en el ámbito profesional. No se pierde de vista su uso profesional, en pequeños laboratorios que no cuenten con los recursos económicos para adquirir una solución privativa.

En un aspecto social desde un punto de vista filosófico, fomenta el impulso de herramientas que contribuyan al desarrollo de una sociedad libre, así como de la libre circulación del conocimiento. Busca aportar un cambio en la percepción social de la creación y producción intelectual y creativa, así como una búsqueda para trascender espacios individualistas de producción y dar paso a espacios colaborativos que conformen comunidades de desarrollo.

Aparte de los beneficios descriptos, no hay que olvidar el principal beneficio práctico de poseer el acceso al código fuente, lo que no es poco, ya que permite examinar el código en detalle y determinar exactamente su capacidad, modificarlo si se desea, aportando a la independencia tecnológica. Ese acceso también contribuye a sistemas más seguros, sin puerta trasera, a tener soporte no limitado a la existencia de la empresa que vendió la solución privativa y compatibilidad a largo plazo[2].

En la formación de quien lleva adelante el proyecto, colaboraría en la profundización de sus conocimientos específicos acerca del área de la inteligencia computacional, su magnitud y diversidad; también mejoraría las destrezas del individuo para aplicar con criterio y visión práctica las metodologías utilizadas en la temática en distintos ámbitos que exceden el

[2]http://geekland.eu/ventajas-software-libre-sobre-privativo/

proyecto mismo.

1.3. Objetivos y alcance

1.3.1. Objetivos

En este proyecto se propone como objetivo general el desarrollo de una herramienta de código abierto para automatizar la construcción del cariograma a partir de imágenes segmentadas de cromosomas en metafase. Para llevar adelante esta tarea, se proponen los siguientes objetivos específicos:

- Seleccionar técnicas de clasificación de cromosomas en metafase más apropiadas para los datos disponibles.
- Desarrollar e implementar un método para cariotipado basado en el estado del arte.
- Diseñar pruebas apropiadas a los datos disponibles para la etapa de segmentación y/o clasificación de cromosomas en metafase.
- Realizar ajustes en el método implementado a fin de reducir errores en clasificación.
- Evaluar el desempeño de las técnicas implementadas.
- Desarrollar un prototipo funcional de la aplicación.

1.3.2. Alcance

Se desea que la herramienta cuente con las siguientes características:

- Licenciada como Código Abierto.
- Análisis de bandeos Q.

Se excluye del siguiente proyecto:

- Construcción de cariogramas de cromosomas no humanos.
- Cariotipado de cromosomas no segmentados.

1.4. Organización

En el Capítulo 2, se explica el marco teórico. Este pretende dar al lector los fundamentos teóricos mínimos para comprender el funcionamiento de la herramienta. Luego, en el Capítulo 3, se describe la metodología aplicada para la clasificación automática de cromosomas humanos en metafase. Posteriormente, en el Capítulo 4, se presentan las configuraciones experimentales para la extracción de características y clasificación. También se exponen los

resultados obtenidos en los experimentos orientados a comprobar el desempeño. Seguidamente, en el Capítulo 5, se muestra la integración de los algoritmos que lograron el mejor desempeño, en un entorno gráfico que permite la construcción automatizada del cariograma. Finalmente, en el Capítulo 6, se exponen las conclusiones surgidas de este trabajo y se proponen algunos trabajos futuros para continuar la evolución de la herramienta.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
Cristian Vizarri, C. E. Martínez & M. Gerard; "Herramienta de código abierto para la construcción automática de cariogramas (Undergraduate project)"
Facultad de Ingeniería y Ciencias Hídricas -Universidad Nacional del Litoral, 2018.

Capítulo 2

Soporte teórico

El presente capítulo pretende dar al lector los fundamentos teóricos mínimos para comprender el funcionamiento de la herramienta. En primer lugar se hará una breve introducción a como es el proceso de construcción del cariograma de forma manual, luego se describe el proceso general para la segmentación, selección y extracción de características. Posteriormente se describen los algoritmos aplicados al proceso de clasificación y sus particularidades. Por último se detallan las medidas de evaluación que se aplicaran.

2.1. Descripción de la estructura de los cromosomas y el cariograma

Los cromosomas de cada especie poseen una serie de características como la forma, el tamaño, la posición del centrómero y las bandas que presentan cuando se aplican distintas técnicas de tinción. Este conjunto de particularidades es el patrón cromosómico expresado a través de un código y se denomina cariotipo. Su representación gráfica, ordenada por pares de cromosomas homólogos, se denomina cariograma [11]. El cariograma, dependiendo de la tinción empleada, representa un patrón de bandas claras y oscuras diferente y específico para cada par cromosómico. Los bandeos más comunes utilizados son: banda Q, banda G, banda R, banda C y banda T [3]. El cariograma es muy importante en el diagnóstico clínico para la detección de alteraciones cromosómicas, tanto numéricas como estructurales. Las primeras están asociadas a cariotipos con un número de cromosomas diferente al normal, y las segundas a cariotipos en los que los cromosomas poseen un número diferente de genes a los que se esperan para los casos normales, causadas por duplicaciones, deleciones, o translocaciones del material genético. Por lo tanto se puede decir que, los cromosomas son las estructuras que contienen la información genética dentro de las células.

La construcción del cariograma es un proceso manual que se realiza en 3 etapas. La primera consiste en la búsqueda visual de los cromosomas dentro de la imagen, y su extracción individual de la misma. Luego se realiza la identificación y ensamblado de los pares de cromosomas homólogos, teniendo en cuenta el tamaño del cromosoma, la posición del centrómero

y el patrón de bandas[1]. Finalmente, se realiza el ordenamiento por tamaño decreciente de todos los pares. Este método, aunque resulta altamente confiable, consume mucho tiempo, es costoso y requiere de conocimiento experto.

Automatizar el proceso de construcción del cariograma implica resolver un problema típico de reconocimiento y clasificación de imágenes. Existen dos escenarios diferentes al momento de abordar el problema, y cada uno requiere una serie de etapas específicas para la construcción del cariograma. Cuando se emplean imágenes ya segmentadas, el desafío se centra en la extracción de características y la optimización de los clasificadores seleccionados. En cambio, cuando se utilizan imágenes completas (cromosomas fotografiados en estado natural) el desafío es mayor, dado que se requiere una etapa previa donde realizar la separación de los cromosomas individuales y su diferenciación respecto del fondo de la imagen. Esta etapa previa habitualmente involucra tareas como la binarización, limpieza de objetos o impurezas, separación de cromosomas unidos o traslapados y la corrección de algunos cromosomas recortados o incompletos.

La Figura **2-1** muestra un esquema del proceso general seguido para la construcción del cariograma. Como puede observarse, el proceso comienza con la adquisición de las imágenes. Luego se aplica un preprocesamiento sobre ellas con la finalidad de mejorar la calidad de la información contenida, lo que facilita la subsecuente fase de segmentación. Posteriormente a la segmentación se realiza la extracción de características por cromosoma y con base en éstas, se identifica la clase de pertenencia. Finalmente, se construye el cariograma para mostrar los cromosomas ordenados.

Por último, antes de abordar la descripción del proceso aplicado a los cromosomas para extraer sus características, clasificarlos, y posteriormente construir el cariograma, es importante describir la estructura general de un cromosoma (Figura **2-2**). La cromátida es el filamento longitudinal que constituye el cromosoma y está unida a su cromátida hermana por el centrómero. Las cromátidas hermanas son similares en morfología e información. El centrómero corresponde a la región más estrecha del cromosoma y divide cada cromátida en dos brazos. El brazo corto (denominado “p”) y largo (denominado “q”) que son los que resultan de la división de la cromátida por el centrómero. El telómero es la porción terminal del cromosoma. Por convención, los brazos “p” y “q” son colocados en la parte superior e inferior del cariograma, respectivamente.

2.2. Extracción de características

En lo referente a la extracción de características se estudiaron dos enfoques para su extracción. Inicialmente se investigó la extracción mediante técnicas clásicas de procesamiento de imágenes que se dio a llamar *forma estándar o clásica* y por otro lado se aplicaron técnicas de aprendizaje profundo (del inglés ”deep learning”).

[1]El patrón de bandas depende de la tinción utilizada.

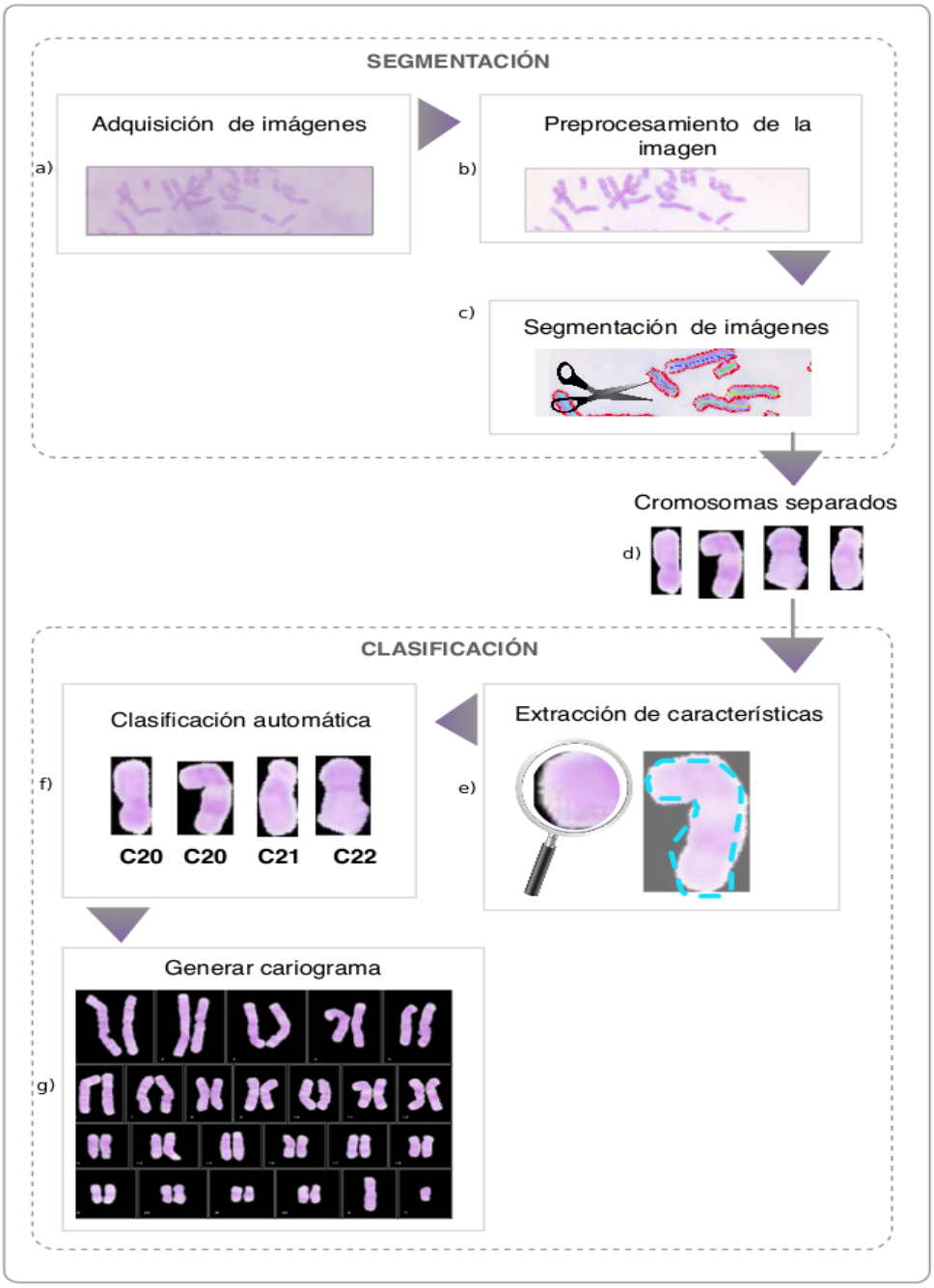


Figura 2-1: Flujo de trabajo: a) Adquisición de imágenes; b) Preprocesamiento; c) Segmentación; d) Cromosomas separados; e) Extracción de características; f) Clasificación; g) Construcción del cariograma.

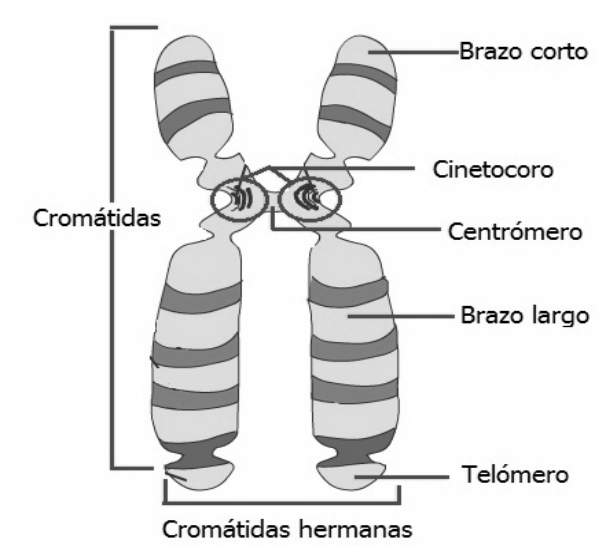


Figura 2-2: Partes de un cromosoma.

2.2.1. Enfoque estándar

Selección de las características a extraer

Un conjunto básico de características empleado por los citogenetistas son el tamaño del cromosoma, la posición del centrómero y los patrones de las bandas. Sin embargo, para el contexto del análisis cromosómico computacional automatizado es conveniente un número mayor de características para lograr la clasificación, y además discriminarlas en características globales y locales. Las primeras son obtenidas a partir de la información del cromosoma completo y un ejemplo típico es la densidad óptica promedio. Por otro lado, las características locales describen propiedades de las subestructuras del cromosoma, tales como la ubicación del centrómero o de una banda de referencia particular y presentan cierta dificultad para obtenerlas. Mientras que las características globales pueden ser extraídas con mayor facilidad dado que no requieren el reconocimiento de subestructuras, pero presentan la desventaja de no describir de manera precisa la variabilidad cromosómica [44]. En lo referente a la selección de características, investigaciones han permitido generar un listado de las 30 características más utilizadas en la literatura, divididas en una estructura jerárquica ordenada de acuerdo a la cantidad de información requerida para obtenerlas [44]. Esta jerarquía presenta los siguientes niveles:

- Nivel 1: son características obtenidas directamente sobre la imagen. Ej.: área del cromosoma, densidad relativa (la relación entre la densidad óptica total y el área) y envoltura convexa del perímetro del cromosoma.
- Nivel 2: se requiere la identificación del eje mayor del cromosoma. Ej.: longitud del cromosoma, perfil de densidad y gradiente.
- Nivel 3: se requiere conocer el eje mayor, el perfil de densidad y la polaridad cromosómica.

Tabla 2-1: Categorías de características

Categoría	Número de características	Descripción
Distribución de píxeles	3	Área, largo del cromosoma y relación de aspecto
Índice centromérico (CI)	2	Área y largo del CI
Perfil de densidad y características asociadas a bandas	124: 7 características asociadas a bandas del perfil de densidad + 116 valores provenientes del perfil de densidad[2]	Características extraídas del perfil de densidad del cromosomas + perfil de densidad

- Nivel 4: requiere conocer el eje mayor, la polaridad y la localización del centrómero. Ej.: índice centromérico.

De este conjunto de características, es importante la extracción del subconjunto de características óptimas que permitan lograr un buen porcentaje de clasificación. La extracción de ese subconjunto no es trivial [44], ya que su elección depende de las características particulares de las imágenes con las que se cuenta. En general, las características utilizadas para la clasificación del cromosoma incluyen las características de Nivel 1. Estas, si bien ayudan a determinar si el cromosoma fue segmentado apropiadamente, son insuficientes para ser utilizadas como únicos descriptores del cromosoma. Por ello vienen a complementarlas las características de los niveles 2, 3 y 4. Estos niveles adicionales permiten identificar características como el perfil de las bandas, los niveles de intensidad a lo largo del esqueleto del cromosoma, entre otras.

Se implementó una selección de características *ad-hoc* basada en diferentes investigaciones [62, 40, 71, 72, 2, 35, 36, 37, 26, 30]. La Tabla **2-1** presenta un resumen de las 128 características utilizadas para describir cada cromosoma. Como se observa en la tabla, se extraen tres familias de características para cada cromosoma. Para su extracción se calculó el perfil de densidad de cada cromosoma y se recorrió su eje medio. Luego para cada punto del eje medio se calculó las intensidades de los valores de gris sobre los segmentos de línea perpendiculares al mismo.

Segmentación del cromosoma: extracción del eje medio

La segmentación de una imagen consiste en la división o partición de la imagen en varias zonas o regiones homogéneas y disjuntas a partir de su contorno, su conectividad, o en términos de un conjunto de características de los píxeles de la imagen que permitan discriminar unas regiones de otras. Este proceso puede ser abordado empleando diferentes estrategias. Se han propuesto aproximaciones basadas en umbralado [23, 34], morfología [13, 22, 46, 64, 53, 39, 72, 33], técnicas geométricas [23, 1, 64], teoría de la decisión estadística [58, 53], entre otras. Para una descripción detallada de las técnicas de segmentación de imágenes puede consultarse [20, 29].

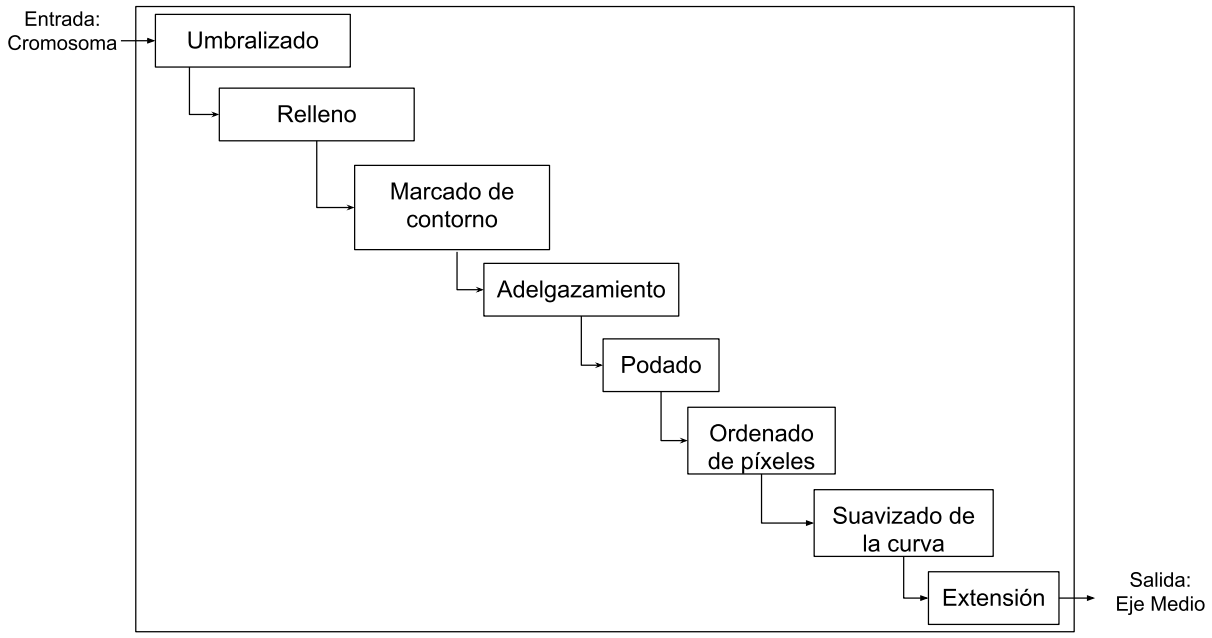
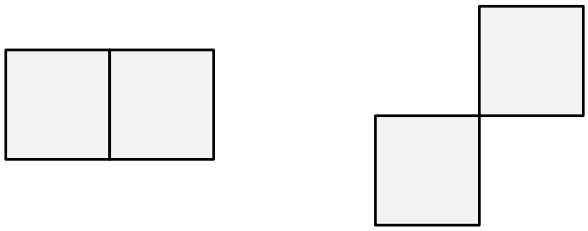


Figura 2-3: Diagrama de bloques de las etapas requeridas para obtener el eje medio.

En este proyecto, el paso inicial para la extracción de características del cromosoma fue la obtención de su eje medio. Para ello primero, fue primordial individualizar el área de la imagen que pertenece exclusivamente al cromosoma lo cual se logra con un umbralizado y eliminando las impurezas que podrían quedar después del umbralizado para luego aplicar los procesos que extraen el eje medio. La Figura **2-3** muestra las diferentes etapas implicadas en este proceso.

El primer paso consiste en el *umbralizado* de la imagen. Para esto, se aplica un umbral a la imagen en metafase (Figura **2-5**a) para separar los cromosomas del fondo (Figura **2-5**b). Esto significa que cada píxel de la imagen que tiene un valor mayor que el umbral, es considerado parte del cromosoma.

El siguiente paso consiste en el *relleno* de puntos espurios que puedan quedar en la imagen umbral de la imagen cromosómica. Las imágenes tienen una profundidad de 8 bits por píxel, por lo que los niveles de gris están codificados entre 0 y 255. Un valor de umbral fijo de 240 produce buenos resultados. Sin embargo, algunas partes dentro de los cromosomas pueden segmentarse erróneamente como fondo. Esto suele ocurrir con bandas muy claras o entre las dos cromátidas hermanas (Figura **2-2**). Este problema se ha discutido en la literatura [27, 46] y distorsiona el cálculo de las características. Para superar este problema, se realiza una detección y eliminación de objetos pequeños y puntos espurios (Figura **2-5**c). Primero se eliminan todos los objetos pequeños de la imagen. Se considera objeto pequeño a todo elemento que esté constituido por menos de 20 píxeles conectados. Para esta operación se toma la imagen cromosómica umbralada y se le aplica un proceso de etiquetado de las componentes conectadas utilizando una región de 8 vecinos. Dada la imagen umbralada se retornara una imagen con etiquetas numéricas, de manera que todos los píxeles que pertenecen a una misma región conectada compartirán la misma etiqueta numérica. Para comprender esta operación se definirá que dos píxeles están conectados si y solo si existe



(a) Adyacentes por frontera (b) Adyacente por esquina

Figura 2-4: Píxeles adyacentes.

un camino del uno al otro a través de píxeles vecinos. El concepto píxel vecino se expondrá en función de la definición de adyacencia. Dos píxeles son adyacentes si, y solo si, tienen en común una de sus fronteras, o al menos una de sus esquinas. Dos píxeles son vecinos si cumplen con la definición de adyacencia. Si los píxeles comparten una de sus fronteras, se dice que los mismos son vecinos directos (Figura **2-4**a) y si sólo se tocan en una de sus esquinas, se llaman vecinos indirectos (Figura **2-4**b).

Una vecindad de un píxel p_0 se define como

$$V_p = \{p : p \in M_{KL}\}; M_{KL} \subset i_{MN}; K = L, \tag{2-1}$$

donde V_p es una submatriz M_{KL} de tamaño $K \times L$, con K y L enteros impares pequeños, contenida en la matriz imagen (i_{MN}), la cual está formada por un número finito de píxeles. En la búsqueda de las componentes conectadas se aplicó una región de vecindad de 8 vecinos, definida como se explica a continuación. Dado un píxel p en las coordenadas (x, y) con $K = L = 3$, presenta 4 vecinos directos cuyas coordenadas están dadas por $(x + 1, y), (x - 1, y), (x, y+1), (x, y-1)$. Este grupo de píxeles se denotara como $N_4(p)$; y 4 vecinos indirectos denotados como $N_D(p)$, cuyas coordenadas están dadas por $(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)$. La región de vecindad de los 8 vecinos queda definida como:

$$N_8(p) = N_4(p) \cup N_D(p). \tag{2-2}$$

Paso seguido, se recorre cada una de las componentes detectadas y se realiza un conteo de las componentes conexas que presenta cada región y se descartan todos los elementos con menos de 20 componentes. Luego de esta operación se considera que la imagen umbral quedó limpia sin elementos espurios fuera de ella. Todavía pueden quedar, por efecto del umbral, huecos en el interior del contorno del cromosoma, por lo que se debe realizar una operación de relleno de huecos.

El tercer paso es aplicar el proceso de *Marcado de contorno*. Este contorno es principalmente necesario para calcular el perímetro, el área y otras propiedades de la forma cromosómica.

Se aplica detección de borde basado en [66] para individualizar el contorno del cromosoma. La Figura **2-5**d muestra un ejemplo del resultado de este proceso.

Luego se aplica el proceso de *Adelgazamiento*. La detección del eje medio cromosómico (Figura **2-5**e) es necesaria para determinar la longitud y el perfil de bandas. La obtención de dicho eje requiere una serie de etapas. Inicialmente se toma la máscara generada en el segundo paso. Luego, se aplica un suavizado para eliminar posibles ruidos en su contorno, ya que la presencia de ruido en el contorno afecta la obtención del eje medio real. Esto se realiza mediante la aplicación de dos operaciones morfológicas con un elemento estructurante fijo. Primero se aplica la operación de cierre y seguidamente una operación de apertura, ambos con un elemento estructurante rectangular de 3×3. La salida obtenida es pasada a través del algoritmo de adelgazamiento Zhang-Suen Thinning Algorithm [77], que retorna el eje medio. Este eje es posible que presente todavía ciertos artefactos y curvatura no deseadas ya que los cromosomas a menudo tienen, al menos, alguna curvatura.

Posteriormente se realiza el *Podado*. El algoritmo de adelgazamiento garantiza un eje conectado, pero deja algunos componentes no deseados a lo largo de la línea principal. Como consecuencia, los esqueletos obtenidos por el algoritmo necesitan ser depurados para seleccionar solamente el eje medio. Se aplica un algoritmo que depura los artefactos no pertenecientes al eje medio, que aplica detección de intersecciones y puntos finales. El algoritmo evalúa las intersecciones que existen y los puntos finales que posee el esqueleto. En cada intersección, se calcula el largo del segmento que lo constituye y se elimina el segmento más corto. Esta solución descarta todas las pequeñas ramificaciones a lo largo del esqueleto principal (Figura **2-5**f).

Seguidamente se procede al *Ordenado de Píxeles*. Después de la poda, el eje medio presenta sólo dos puntos finales, pero no se tiene un mapeo ordenado de los puntos intermedios que permita determinar un "camino" entre ellos. El algoritmo ordenador realiza dicha tarea; comienza desde el punto final superior y visitando uno a uno los puntos vecinos. Etiqueta las componentes conexas hasta unir con el punto final inferior. El algoritmo supone que se conoce la polaridad correcta (orientación) de un cromosoma.

La polaridad define la orientación que presentan los brazos del cromosoma y es requerida para la interpretación fehaciente de la secuencia de bandas. Se define en términos del brazo corto y largo del cromosoma, y utiliza el centrómero como punto de referencia. Para su cálculo se obtienen las longitudes de los brazos cromosómicos en términos de la longitud del eje medio entre cada punto final del eje y la posición del centrómero. Luego se rota el cromosoma de manera que el brazo con la menor longitud quede en la "parte superior"[3].

Finalmente el ante último paso es el *Suavizado de la curva*. El objetivo es mejorar la precisión de la estimación de la curva perteneciente al eje medio y eliminar ruidos/artefactos por la que esté afectada [40, 71]. El procedimiento de suavizado es simple. El eje medio detectado es discretizado con un paso de cinco píxeles. Luego los píxeles seleccionados son conectados mediante rectas y un nuevo eje medio suavizado es generado.

Por último se realiza una *Extensión* del esqueleto. Por definición, el algoritmo de adelgaza-

[3]La parte superior es seleccionada por convención.

miento produce un esqueleto que no llega a las extremidades de los telómeros del cromosoma. La solución es extender el esqueleto desde los puntos finales calculados. Para extenderlo, primero se calcula la pendiente de cada extremo del esqueleto. Luego los extremos del esqueleto se extienden con la misma pendiente hasta que alcanzan el borde del cromosoma. El resultado es el eje medio completo (Figura **2-5**g) que se utiliza para extraer las características cromosómicas.

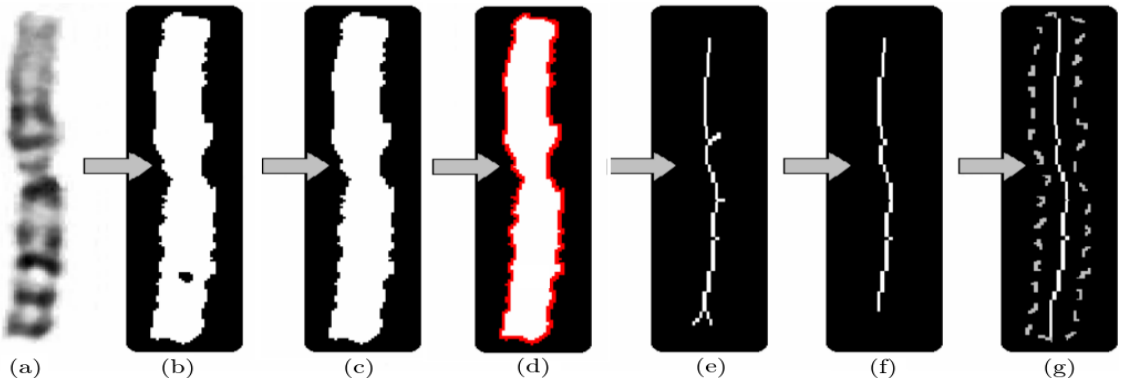


Figura 2-5: Segmentación cromosómica. a) imagen de entrada, b) umbral, c) relleno, d) detección de fronteras, e) adelgazamiento, f) podado, ordenado de píxeles y suavizado de la curva, g) extensión del esqueleto.

Extracción de características

Una vez obtenido el eje medio, el siguiente paso es la extracción de las características del cromosoma, que constituirán su vector de características. Se detallan a continuación las características obtenidas.

Perfil de densidad

Para el cálculo del perfil se aplica filtrado de mediana al cromosoma, para reducir posibles impulsos y ruido que pudieran afectar el cálculo. Por definición, el perfil de densidad registra el valor de gris medio de cada línea perpendicular a lo largo del eje medio, y se calcula como:

$$D(j) = \frac{\sum_{i=1}^{n} g_i(j)}{n}, \tag{2-3}$$

donde $\sum_{i=1}^{n} g_i(j)$ es la suma de los valores de gris g_i a lo largo de la j-ésima línea perpendicular, y n es el número de píxeles pertenecientes a la j-ésima línea perpendicular [71].

Área y largo

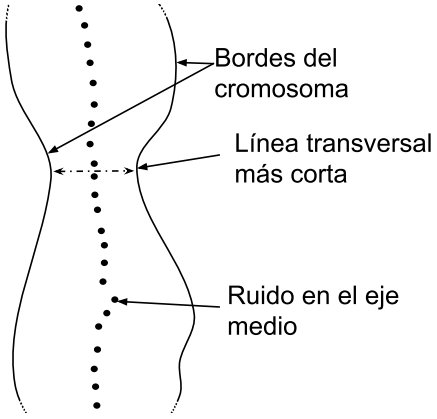


Figura 2-6: Bordes del cromosoma.

El área es calculada luego de obtener el umbral depurado (Figura **2-5**c). Se cuenta el número de píxeles blancos en la imagen. Por otro lado, el largo del eje medio es calculado luego de depurar y extender el esqueleto (Figura **2-5**g) contando el número de píxeles blancos en dicho esqueleto.

Relación de aspecto

Esta relación se calcula como el cociente entre el ancho y la altura del rectángulo delimitador del cromosoma.

Detección del centrómero

El centrómero es la región del cromosoma donde las cromátidas se unen [72]. Este separa el cromosoma en dos brazos (Figura **2-2**). La asignación de polaridad determina la orientación de un cromosoma a través de la identificación de sus brazos. En la bibliografía [40, 72, 2, 9, 41] se han reportado diversos métodos para determinar la ubicación del centrómero. Sin embargo, ninguno es completamente confiable, y no existe una tendencia definida al respecto. En este informe se optó por definir el centrómero como la sección más estrecha a lo largo del cromosoma [9]. Se empleó los valores de anchura del cromosoma calculados durante la determinación del perfil de densidad, y se asignó el centrómero al punto donde el cromosoma es más delgado (Figura **2-6**).

Índice centromérico

El índice centromérico, indica la relación que existe entre la longitud del brazo corto y la longitud total del cromosoma. Permite establecer un valor para la simetría del cromosoma e inferir información sobre su morfología. Por definición el índice centromérico (CI) es la relación entre la longitud del brazo corto y la longitud total del cromosoma; denotado aquí

como *CI(L)*, y se calcula como:

$$CI(L) = \frac{L_p}{L_p + L_q}, \tag{2-4}$$

donde L_p es la longitud del brazo corto y L_q es la longitud del brazo largo. En este informe se incorpora una definición adicional, que lo calcula en función de las magnitudes relativas de las áreas pertenecientes a los brazos del cromosoma [72], denotado aquí como *CI(A)*. Su valor se calcula como:

$$CI(A) = \frac{A_p}{A_p + A_q}, \tag{2-5}$$

donde A_p es el área del brazo corto y A_p es el área del brazo largo.

Características adicionales

Por último, se extrajeron siete características adicionales que obtienen información de los patrones de bandas presentes en el cromosoma a partir del perfil de densidad [71, 72]. Los patrones de bandas no fueron tenidos en cuenta, de forma explícita, hasta el momento en las características anteriores descriptas, por lo que estas características adicionales vienen a complementar este aspecto. Las características obtenidas son: número total de bandas detectadas en un cromosoma, número de bandas en el brazo largo, número de bandas en el brazo corto, número total de bandas oscuras en el cromosoma, número total de bandas claras en un cromosoma, número de bandas oscuras en el brazo largo, número de bandas oscuras en el brazo corto.

Normalización del perfil de densidad

El perfil de densidad calculado, además de aplicarse para la extracción de los descriptores explicados en los párrafos anteriores, se seleccionó para ser utilizado como vector de características en sí mismo [30]. Como se puede inferir, la longitud de un vector de perfil de densidad depende de la longitud del propio cromosoma, que varía, no sólo entre las diferentes clases de cromosomas, sino también entre los cromosomas de la misma clase. Con el fin de estandarizar la longitud del perfil de densidad, se aplica una función de transformación, que remuestrea los valores del perfil de manera que todos presenten la misma longitud. En base a esto, el perfil de densidad puede ser visto como un vector $P = [p_1, p_2, ..., p_L]$, con una longitud L variable de elementos. Para normalizar su longitud se aplica la transformación $P_i^* = P_j, i = 1, ..., L$, donde $j = round(i * \frac{L-1}{N-1} + \frac{N-L}{N-1})$ es el paso con que se tomará cada muestra, N es el número de puntos que debe contener el vector P luego de aplicada la transformación, y P^* es el valor seleccionado del i-ésimo punto.

2.2.2. Enfoque basado en aprendizaje profundo

En los últimos años, el aprendizaje profundo ha conducido a avances en una variedad de problemas. Entre los diferentes tipos de redes neuronales profundas, las redes neuronales convolucionales (CNN) se han aplicado en distintos estudios. Este es un tipo de red neuronal artificial donde las neuronas actúan como sensores receptivos de una manera muy similar a las neuronas en la corteza visual. En particular, diversos estudios reportados en la literatura sugieren que el uso de redes neuronales convolucionales logran buenos resultados en problemas de reconocimiento visual [8, 32, 55, 48, 15].

El soporte teórico de este tipo de arquitectura requiere el desarrollo de una serie de elementos relacionados al aprendizaje maquinal, que por orden de presentación de las tareas a desarrollar no son posibles de explicar todavía en este punto. Los detalles de sus partes constitutivas y funcionalidades aplicadas a la extracción de características en imágenes serán abordadas en la subsección 2.3.5 en el apartado *Redes convolucionales*.

2.3. Algoritmos de Clasificación

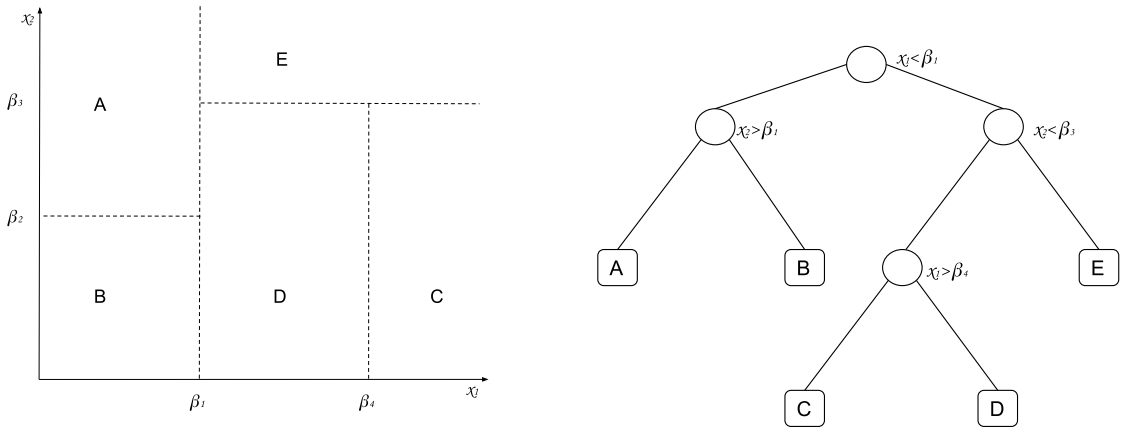
El aprendizaje maquinal es la automatización de un proceso de aprendizaje, donde se construyen reglas que están basadas en las observaciones de un conjunto de datos.

Teniendo en cuenta la forma en la que aprenden estos algoritmos pueden ser clasificados en *supervisados* y *no supervisados*. Los primeros aprenden a partir de ejemplos que fueron resueltos por un experto humano, que previamente ha construido un modelo que resuelve el problema. En el caso de los segundos, el sistema observa los casos y reconoce los patrones por sí mismo y establece las clases.

Por otro lado e independientemente del tipo de aprendizaje, se los puede clasificar en algoritmos de clasificación o regresión. Los algoritmos de clasificación asignan una de las clases disponibles a un objeto dado en base a las propiedades empleadas para describirlo; y los algoritmos de regresión permiten predecir variables continuas basándose en los atributos del conjunto de datos analizado.

En las secciones siguientes se detallaran los sistemas de aprendizaje supervisado para clasificación que fueron aplicados. En los sistemas de aprendizaje maquinal supervisado se ingresa un conjunto de instancias de entrenamiento donde cada instancia se describe por un vector de valores y una clase etiquetada. La tarea del algoritmo es aprender a clasificar correctamente nuevas instancias no etiquetadas.

Se expone a continuación la descripción de los algoritmos de aprendizaje aplicados y las bases de los mismos.



(a) Entrada bidimencional particionada en 5 regiones

(b) Árbol binario correspondiente a la partición (a)

Figura 2-7: Árbol de decisión.

2.3.1. Árbol de decisión

Un Árbol de decisión (DT), es un modelo jerárquico para el aprendizaje supervisado mediante el cual una región es identificada en una secuencia de divisiones recursivas [4]. Comienza con un único nodo (padre) que luego se ramifica. Cada una de esas ramas crea nodos adicionales (hijos), que se ramifican en otras posibilidades. Está compuesto por los nodos de decisión (típicamente representados como círculos) que son los puntos donde se evalúan las características para realizar la toma de decisión. Cada uno de estos nodos implementa una función de prueba $f(\cdot)$, que tiene como salida una cantidad discreta de resultados que etiquetan las ramas (típicamente representadas como líneas continuas) que conectan un nodo padre con sus nodos hijos. Dada una entrada, en cada nodo, la función de prueba es aplicada y una rama de salida es seleccionada dependiendo de la salida de $f(\cdot)$. Este es un proceso iterativo que comienza en la raíz, y que divide el conjunto de fuentes en subconjuntos; el cual es repetido hasta alcanzar un nodo hoja que cumple con un criterio dado de terminación (no se continúa con su ramificación). En ese punto el valor del nodo hoja constituye el valor de salida.

En la Figura **2-7**a se muestra la partición binaria para un espacio de entradas y el camino de esta subdivisión puede ser representada esquemáticamente como el árbol binario expuesto en la Figura **2-7**b. El primer paso es dividir el espacio de entradas de acuerdo a $x_1 < \beta_1$ o $x_1 \geq \beta_1$, donde β_1 es un parámetro del modelo. Esto crea dos subregiones que a su vez son divididas de forma independiente. Para $x_1 < \beta_1$ se divide en $x_2 > \beta_2$ o $x_2 \leq \beta_2$ y da como resultado las regiones A y B. De esa forma se continúa con los demás nodos hasta llegar a las hojas.

Cada $f(\cdot)$ es una función simple que define un discriminante en el espacio de entrada d-dimensional dividiéndolo en regiones más pequeñas. A medida que se toma un camino desde la raíz hacia las hojas esta función divide una función compleja en una serie de decisiones simples. Se han desarrollado diferentes métodos de resolución que asumen distintos modelos y/o variantes de $f(\cdot)$. Entre los más famosos están los sugeridos por Quinlan como ID3 [49],

C4.5 [56] y CART (Classification And Regression Trees) [6]. Uno de los pasos fundamentales de estos algoritmos es la medida de selección aplicada para encontrar el atributo de prueba apropiado en cada nodo de decisión mientras crece el árbol.

En [49] Quinlan ha definido una medida llamada ganancia de información también conocida como la ganancia de criterio basada originalmente en la teoría de la información de Shannon. La idea es calcular la entropía de cada atributo y tomar el atributo con menor *entropía*. Para definir la ganancia de información con precisión, se debe definir primero una medida comúnmente utilizada en la teoría de la información, llamada entropía que caracteriza la impureza de una colección arbitraria de ejemplos.

La entropía $H(S)$ es una medida de la cantidad de incertidumbre en el conjunto (de datos) S y se define como [49]:

$$H(S) = \sum_{x \in X} -p(x) log_2 p(x), \tag{2-6}$$

donde, S es el conjunto donde la entropía es calculada, X las clases de conjuntos existentes en S y *p(x)* la proporción de la cantidad de elementos de la clase x con respecto a la cantidad de elementos en el conjunto S. Cuando $H(S) = 0$ el conjunto S tiene clasificación perfecta. En ID3 [49], la entropía se calcula para cada atributo restante. El atributo con la entropía más pequeña se usa para dividir el conjunto S en cada iteración.

Por otro lado el algoritmo CART basado en [6], define como métrica para la división de los atributos una medida llamada *índice de impureza gini*. Se puede calcular sumando la probabilidad p_i de ser seleccionado un elemento etiquetado como i multiplicado por la probabilidad de haber etiquetado mal ese ítem $\sum_{k \neq i} p_k = 1 - p_i$. Para calcular el *índice de impureza gini* para un conjunto de elementos, supongamos que $i \in \{1, ..., M\}$ donde M es el número de clases y p_i es la fracción de elementos etiquetados con la clase i en el conjunto M para un nodo t, se define como:

$$i(t) = 1 - \sum_{i=1}^{M} p_i^2 \tag{2-7}$$

Definido el índice, el criterio se basa en minimizar:

$$\Delta i(s,t) = i(t) - p_L i(t_L) - p_R i(t_R), \tag{2-8}$$

donde, $\Delta i(s,t)$ es la disminución de la impureza en el nodo t con respecto a la división s, p_L es la probabilidad de enviar el caso al nodo hijo izquierdo t_L, p_R es la probabilidad de enviar el caso al nodo hijo derecho t_R, $i(t_L)$ es medida de impureza de *gini* para el nodo hijo izquierdo y $i(t_R)$ es medida de impureza de *gini* para el nodo hijo derecho.

Tabla 2-2: Ejemplo cálculo índice *gini*.

	Malestar		
Característica	Si	No	Total
A	6	2	8
B	6	10	16
Total	12	12	24

Para interpretar mejor esta medida a continuación se plantea un ejemplo. La muestra tiene 24 individuos los cuales presentan dos características (A y B) para separarlos (Tabla **2-2**). De esta muestra se considera que 12 están afectados con un malestar, por lo que la taza de afectados con malestar es del 50 %. El valor del índice *gini* para este se calcula de la siguiente manera.

$$i(t) = 1 - 0{,}5^2 - 0{,}5^2 = 1 - 0{,}25 - 0{,}25 = 0{,}5$$

Ahora, calculemos el índice *gini* izquierdo ($i(t_L)$) y derecho ($i(t_R)$) de la división usando la variable A

$$i(t_L) = 1 - \left(\frac{6}{8}\right)^2 - \left(\frac{2}{8}\right)^2 = 0{,}375$$

$$i(t_R) = 1 - \left(\frac{6}{16}\right)^2 - \left(\frac{10}{16}\right)^2 = 0{,}469$$

Finalmente calculamos $\Delta i(s,t)$ como:

$$\Delta i(s,t) = i(t) - p_L i(t_L) - p_R i(t_R) = 0{,}5 - \frac{8}{24} * 0{,}375 - \frac{16}{24} * 0{,}469 = 0{,}0624$$

De manera similar, se repite para todos los puntos de división y variables. Luego entre todas las variables y puntos de división se selecciona aquella en la que el índice sea mínimo.

2.3.2. Random Forest

Random Forests (RF), es un modelo que consiste en una colección independiente, idénticamente distribuida y al azar de M árboles de decisión [7]. Cuando está configurado para

clasificación y un *caso* es ingresado, este *caso* es evaluado por todos los árboles en el bosque para tomar una decisión. En este proceso cada árbol le asigna de forma independiente una etiqueta al *caso*. Al terminar este proceso la etiqueta que obtenga la mayor cantidad de incidencias es reportada como la predicción. Es posible denotarlo por $m_n(\mathbf{x}; \Theta_j, H_n)$, donde $H_n = (\mathbf{X}_1, Y_1), ..., (\mathbf{X}_n, Y_n), \mathbf{X} \in [0,1]$ e $Y \in \mathbb{R}$ es el vector de datos de entrenamiento y Θ_j con $j \geq 1$ son los subconjuntos aleatorios, independientes e idénticamente distribuidos de características de prueba seleccionadas del conjunto de características posibles Θ. Cada árbol crece usando los datos de entrenamiento y Θ_j de forma independiente. Luego la predicción individual de cada árbol se combina para obtener la predicción del bosque (Ecuación 2-9).

$$m_{M,n}(\mathbf{x}; \Theta_1, ..., \Theta_M, H_n) = \frac{1}{M} \sum_{j=1}^{M} m_n(\mathbf{x}; \Theta_j, H_n) \tag{2-9}$$

En la Figura **2-8** se expone de modo gráfico lo que describe la Ecuación 2-9. Para el entrenamiento, Random Forest, selecciona al azar un subconjunto de características de prueba del conjunto de características posibles, para luego seleccionar el mejor ajuste entre estas. Posteriormente el proceso se repite en cada uno de los árboles (muchos árboles crecen de la misma manera) para así construir un bosque.

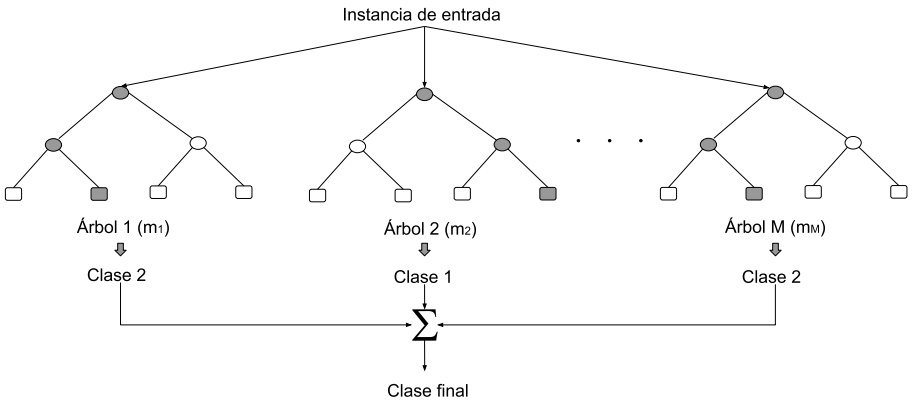


Figura 2-8: Random Forest.

La proporción de árboles que toman una misma respuesta se interpreta como la probabilidad de la misma [7]. Este clasificador presenta dos parámetros cruciales a tener en cuenta para su ajuste. La profundidad máxima que puede tener cada árbol y la cantidad máxima de árboles que se le permite contener.

2.3.3. *K*-Nearest Neighbors

K-vecinos más cercanos (K-Nearest Neighbors - KNN) es un método no paramétrico que puede ser usado para clasificación [4]. De forma general supongamos observaciones distribuidas con alguna densidad de probabilidad $p(\mathbf{x})$ en algún espacio d-dimensional, y deseamos

estimar el valor de $p(\mathbf{x})$. Consideremos una pequeña región R que contiene x. La probabilidad asociada con esta región está dada por

$$P = \int_R p(\mathbf{x}) d(x). \tag{2-10}$$

Además supongamos que se tiene N datos con una distribución de probabilidad $p(\mathbf{x})$, donde cada punto tiene una probabilidad P de caer dentro de R, el número total K de puntos que se encuentran dentro de R se distribuirá de acuerdo a la distribución binomial

$$P(K|N,P) = \frac{N!}{K!(N-K)!} P^K (1-P)^{1-K}, \tag{2-11}$$

donde para N tendiendo a infinito esta distribución se puede simplificar como

$$K \simeq NP. \tag{2-12}$$

Si además se supone que la región R tiende a cero de tal manera que $p(\mathbf{x})$ pueda ser considerada constante en la región se obtiene que

$$P \simeq p(\mathbf{x})V, \tag{2-13}$$

donde V es el volumen de R. Combinando ambas ecuaciones se obtiene

$$p(\mathbf{x}) = \frac{K}{NV}. \tag{2-14}$$

Ahora considerar a K como una constante y a V como el valor a determinar. Para esto suponer una esfera centrada en el punto x que puede crecer hasta contener K puntos, en la cual se estimará la densidad de $p(\mathbf{x})$. La estimación de la densidad $p(\mathbf{x})$ viene dada por la Ecuación 2-14 con V ajustada al volumen de la esfera resultante. Esta técnica se conoce K-vecinos más cercanos, la cual puede ser extendida a problemas de clasificación.

Supongamos que se tiene C_k clases con N_k puntos por clase y N puntos en total, de forma tal que $\sum_k N_k = N$. Si la esfera de volumen V contiene K_k puntos de la clase C_k por Ecuación 2-14 la probabilidad condicional asociada es:

$$p(\mathbf{x}|C_k) = \frac{K_k}{N_k V}, \tag{2-15}$$

la probabilidad incondicional es

$$p(\mathbf{x}) = \frac{K}{NV}, \tag{2-16}$$

y probabilidad a priori

$$p(C_k) = \frac{N_k}{N}, \tag{2-17}$$

combinando estos valores por el Teorema de Bayes se obtiene la probabilidad a posteriori como:

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})} = \frac{K_k}{K}. \tag{2-18}$$

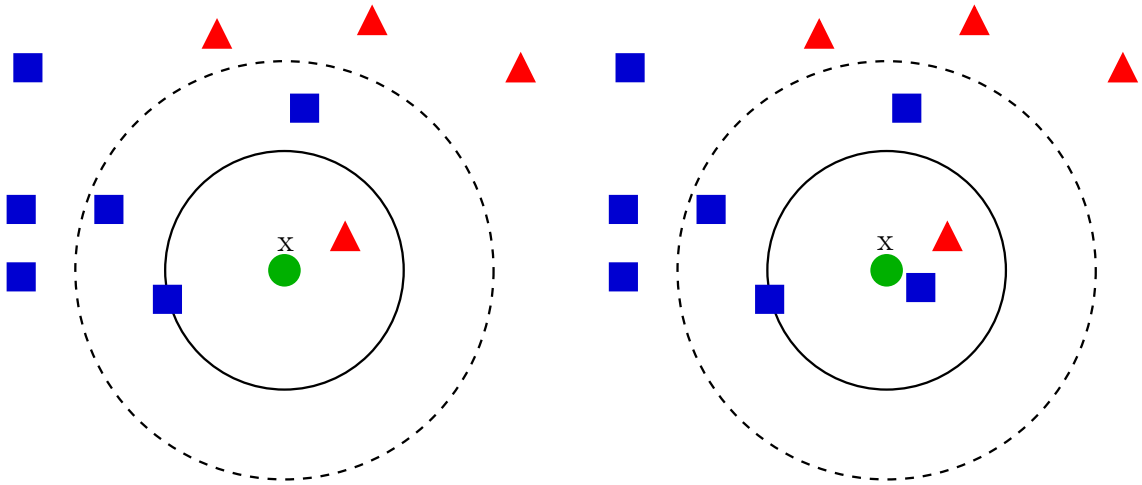
Si deseamos minimizar la probabilidad de clasificación errónea, esto se hace asignando el punto de prueba x a la clase que tiene la mayor probabilidad a posteriori, que corresponde al valor más grande de K_k/K. Por lo tanto, para clasificar un nuevo punto, identificamos los K puntos más cercanos del conjunto de datos de entrenamiento midiendo la distancia mediante alguna métrica y luego asignamos el nuevo punto a la clase que tiene el mayor número de representantes en este conjunto [59].

A manera de ejemplo, se puede observar cómo es la clasificación del vecino más cercano. Se tienen los datos pertenecientes al conjunto de entrenamiento (triángulos y cuadrados), y se quiere conocer la etiqueta de un nuevo dato (círculo). Entonces el procedimiento a seguir consiste en buscar el ejemplo que esté más cerca de este nuevo dato x, y asignarle su etiqueta (triángulo). Esto se esquematiza en la Figura **2-9**a.

Ahora, si se considera el caso donde hay un cuadrado dentro de los datos correspondientes a los triángulos (ruido), y se desea clasificar el nuevo dato (círculo), este se clasificara erróneamente como cuadrado (Figura **2-9**b). Para considerar el problema del ruido se puede cambiar el algoritmo de clasificación y utilizar un mayor número de vecinos. Así generar la etiqueta del nuevo dato y hacer un conteo de las etiquetas por mayoría simple de los vecinos que se encuentran dentro de la distancia considerada, y no un solo vecino. Esta generalización del método se llama k-vecinos más cercanos.

Para la estimación de distancia existen distintas estrategias, pero entre las más aplicadas encontramos a:

$$distancia = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}, \tag{2-19}$$



(a) Vecino más cercano (b) Vecino más cercano con ruido

Figura 2-9: Clasificación del vecino más cercano.

con *p=1* o *p=2* donde $p \geq 1 \in \mathbb{R}$ y representan a las distancias de manhattan y euclídea respectivamente.

K-vecinos más cercanos presenta la ventaja de su simple uso, interpretación y rapidez de ejecución. La clasificación de nuevos objetos sólo implica una medida de la distancia entre objetos y no requiere de un cálculo iterativo.

2.3.4. Máquinas de Vector Soporte

Las Máquinas de Vector Soporte (Support Vector Machines - SVM) son un tipo de clasificador de *margen máximo*. Su potencial radica en la idea de transformar o proyectar un conjunto de datos pertenecientes a una dimensión *N* dada, hacia un espacio de dimensión superior. Están basadas en el principio de minimización del riesgo estructural (Structural Risk Minimization ó SRM), originado de la teoría de aprendizaje estadístico desarrollada por Vapnik [10].

A continuación se hará una revisión de la teoría básica de las SVM en problemas de clasificación [30, 75]. Dado un conjunto de ejemplos $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ de dimensión *d*, donde $\mathrm{x}_i \in \mathbb{R}^d, y_i \in \{+1, -1\}$. Ahora a modo de ejemplo, supongamos que el conjunto *S* contiene L ejemplos y dos clases linealmente separables. Cada punto $\mathbf{x}_i \in \mathbb{R}^d$ pertenece a alguna de ellas y se les puede asignar una etiqueta $y_i \in \{+1, -1\}$ para i=1,...,L. El objetivo es encontrar un hiperplano que los pueda separar, con la condición de que la distancia entre el vector $\mathbf{x}$ (de cada clase) más cercano al hiperplano sea máxima. Si representamos el hiperplano en su forma canónica, ($\mathbf{w}$,b), donde $\mathbf{w}$ es el vector normal al hiperplano y b es un factor de desplazamiento desde el origen hasta el hiperplano, entonces la ecuación del hiperplano óptimo es de la forma:

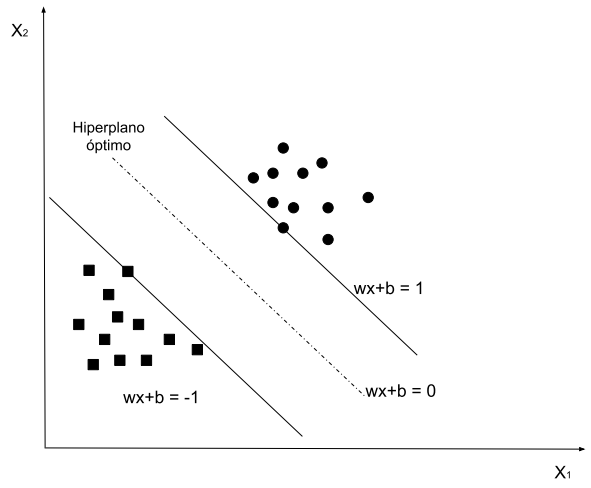


Figura 2-10: Hiperplano óptimo de separación.

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \tag{2-20}$$

y los hiperplanos $\langle \mathbf{w}, \mathbf{x} \rangle + b = +1$ y $\langle \mathbf{w}, \mathbf{x} \rangle + b = -1$ son los generados por los puntos (vectores) que se encuentran más cercanos al hiperplano óptimo (Figura **2-10**).

Entonces, para maximizar el margen geométrico se debe minimizar la norma de $\mathbf{w}$, y el problema se transforma en un problema de optimización:

$$\begin{aligned} &\text{mín} \quad \frac{1}{2}||\mathbf{w}^2|| \\ &\text{s.t.} \quad y_i(\langle \mathbf{w}, \mathbf{x} \rangle + b \geq 1) \quad \forall_i = 1, ...n \end{aligned} \tag{2-21}$$

Ahora, si el conjunto S no es linealmente separable, se debe permitir un *margen blando* que permita conseguir el hiperplano de margen máximo que clasifique de la mejor forma posible los datos. Se introdujo para esto una variable de holgura (ξ_i) para cada dato ($\mathbf{x}_i$) que mida el error, donde la sumatoria $\sum_i \xi_i$ es el acumulado. Reformulando la Ecuación 2-21 ahora se tiene un problema primal:

$$\begin{aligned} &\min_{v_{\mathbf{w},b,\xi_i}} \quad \frac{1}{2}||\mathbf{w}^2|| + C\sum_{i=1}^{n} \xi_i \ , \\ &\text{s.t.} \quad y_i(\langle \mathbf{w}, \mathbf{x} \rangle + b) \geq 1 - \xi_i, \quad i = 1, ...L; \xi_i \geq 0 \quad i = 1, ...L \end{aligned} \tag{2-22}$$

donde $C > 0$ es una constante de regularización, que ajusta la superficie de decisión. Un C pequeño hace que la superficie de la decisión sea suave, mientras que un C grande busca que la superficie de decisión se ajuste lo mejor posible a los ejemplos de entrenamiento. Este es un problema de programación cuadrática que puede ser resuelto construyendo un Lagrangiano y transformándolo en el dual:

$$\begin{aligned} \text{máx} \quad & \sum_{i=1}^{L} \alpha_i - \frac{1}{2} \sum_{i=1}^{L} \sum_{j=1}^{L} \alpha_i \alpha_j y_i y_j x_i \cdot x_j, \\ \text{s.t.} \quad & \sum_{i=1}^{L} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C, \quad i = 1, ..., L \end{aligned} \tag{2-23}$$

donde $\alpha = (\alpha_1, ..., \alpha_L)$ es un vector de multiplicadores de Lagrange positivos asociados a las constantes en $y_i(\langle \mathbf{w}, \mathbf{x} \rangle + b) \geq 1 - \xi_i, \quad i = 1, ...L$. Luego por el teorema de Khun-Tucker [31], la solución del problema de la Ecuación 2-23 satisface:

$$\alpha_i(y_i(\langle \mathbf{w}, \mathbf{x} \rangle + b) - 1 + \xi_i) = 0 \quad i = 1, ..., L \tag{2-24}$$

$$(\text{C-}\alpha_i)\xi_i = 0 \quad i = 1, ..., L \tag{2-25}$$

Finalmente para construir el hiperplano óptimo $\langle \mathbf{w}, \mathbf{x} \rangle + b$, se utiliza $w = \sum_{i=1}^{L} \alpha_i y_i x_i$. Y el escalar b puede se determinado de la Ecuación 2-25. La función de decisión generalizada queda entonces definida como:

$$f(x) = sign(\langle \mathbf{w}, \mathbf{x} \rangle + b) = sign(\sum_{i=1}^{L} \alpha_i y_i x_i \cdot x + b) \tag{2-26}$$

Por ultimo falta definir la función de kernel. Una función de kernel es una función K: $\mathbb{X} \times \mathbb{X} \to \mathbb{R}$, que asigna a cada par de elementos del espacio de entrada, $\mathbb{X}$, un valor real correspondiente al producto escalar de las imágenes de dichos elementos en un nuevo espacio $\mathfrak{F}$. Por lo tanto la función de kernel puede sustituir el producto escalar en la Ecuación 2-26 quedando:

$$f(x) = sign(\langle \mathbf{w}, \mathbf{x} \rangle + b) = sign(\sum_{i=1}^{L} \alpha_i y_i K(x_i, x_j) + b) \tag{2-27}$$

Con esta función se logra trasladar los datos del problema a un hiperplano donde la solución es lineal.

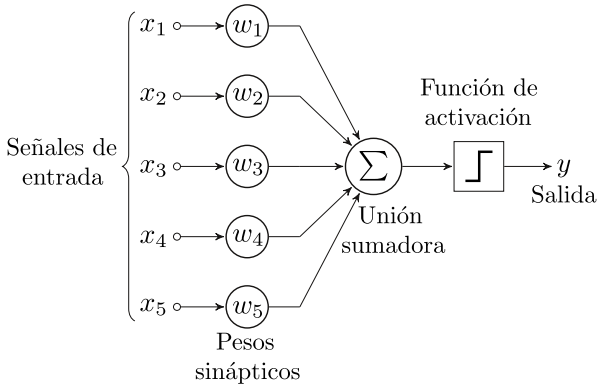


Figura 2-11: Diagrama de un Perceptrón con cinco señales de entrada.

2.3.5. Redes Neuronales

Las redes neuronales son modelos matemáticos de procesamiento de información. Están inspiradas en el procesamiento de la información que tiene lugar en el cerebro. En su génesis constituyeron un intento de reproducir o imitar las capacidades de procesamiento de información de un sistema biológico, como puede ser el sistema nervioso central humano, o de cualquier organismo biológico en general.

A continuación se revisa el funcionamiento básico de una única neurona artificial. El modelo de una neurona artificial es una imitación del proceso de una neurona biológica.

El Perceptrón constituye la red neuronal más simple, constituida solamente por una neurona. Recibe estímulos de su entorno por medio de un conjunto de sensores simbolizados por las entradas, realiza sobre ellas un procesamiento simple y elemental, y luego obtiene un valor de salida que representa la actividad de la neurona. La arquitectura se ilustra en la Figura **2-11**.

Las entradas al Perceptrón se definen mediante un vector dado por $\mathbf{x} = [x_1, x_2, ..., x_N]^T$. Las conexiones sinápticas desde las fuentes de estímulos externos se representan mediante enlaces denominados pesos sinápticos, los cuales ponderan de manera diferente cada entrada de acuerdo a su importancia en el procesamiento de los estímulos. El conjunto de pesos está dado por $\mathbf{w} = [w_1, w_2, ..., w_N]^T$. Además de estos se debe tener en cuenta el valor del sesgo o *bias* el cual representa un valor mínimo a superar por la sumatoria de las entradas. Cuando este valor es superado se dispara la señal neuronal de excitar o inhibir la actividad neuronal. Este valor puede ser agregado al vector de las entradas como el valor x_0 y al vector de pesos como w_0. El potencial total de las entradas se calcula como la suma ponderada de las entradas: $v = \sum_{i=0}^{N} w_i x_i$. Y la salida final es calculada al pasar por medio de una función de activación

$$y = f(v). \tag{2-28}$$

Existen diferentes definiciones para esta función de activación que modifican el rango de la

salida. Independientemente de la función aplicada, cuando la salida se encuentra en un nivel cercano al máximo del rango que permite la función se considera la salida como una salida activa o excitada, mientras que si la salida es cercana al mínimo se considera una salida no excitada.

Una función usualmente aplicada es la función logística que se define como:

$$f(v) = \frac{1}{1 + e^{-v}} \tag{2-29}$$

Además de esta, existen otras funciones de activación que suelen ser aplicadas. Dependiendo la función de activación que se use y la regla de corrección de error se puede describir la regla de aprendizaje del Perceptrón simple. A continuación se hará la descripción para la función logística y la regla de corrección de error será minimizar el error cuadrático instantáneo entre la salida calculada y la salida esperada. El error viene dado por

$$e(l) = d(l) - y(l) \tag{2-30}$$

donde *d(l)* $\in (0, 1)$ es el valor de salida esperado e *y(l)* $\in 0, 1$ es la salida calculada para el vector de entrada $\mathbf{x}(l)$. Se busca minimizar:

$$J((w)) = \frac{1}{2}(d(l) - y(l))^2 \tag{2-31}$$

En donde su gradiente es:

$$\Delta J((w)) = -(d(l) - y(l))y^{'}(l)\mathbf{x}(l) \tag{2-32}$$

Finalmente la regla de aprendizaje queda expresada por:

$$w_i(l + 1) = w_i(l) + \nu(d(l) - y(l))y^{'}(l)x_i(l), 0 \leq i \leq N \tag{2-33}$$

Donde los pesos $w_i(0)$ son inicializados de manera aleatoria, la derivada de *y(l)* es $y^{'}(l) = y(l)(1 - y(l))$, y ν es una constante positiva denominada *coeficiente de aprendizaje*.

El Perceptrón simple supone la base para las redes neuronales. Pero solamente es aplicable a problemas linealmente separables que impliquen dos clases. Para superar esta limitación se han planteado distintas arquitecturas. A continuación se describen algunas de esas arquitecturas las cuales fueron usadas en el desarrollo de este proyecto.

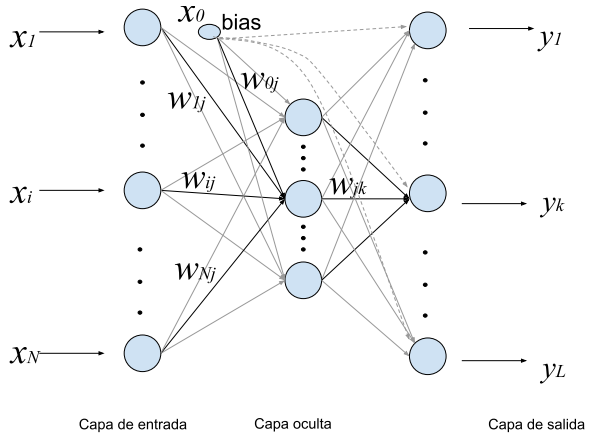


Figura 2-12: Diagrama de un Perceptrón multicapa.

Perceptrón Multicapa

El Perceptrón multicapa (Multilayer Perceptrón - MLP), es una generalización del Perceptrón simple y surgió como consecuencia de las limitaciones de dicha arquitectura en lo referente al problema de la separabilidad no lineal. Es una red neuronal feedforward multicapa, por lo tanto los datos fluyen desde la capa de entrada y atraviesan todas las capas intermedias hasta la capa de salida. La red consta de nodos o neuronas y cada uno contiene un multiplicador, un sumador y una función de activación *f*. En la Figura **2-12** vemos un esquema del MLP genérico. Las entradas $x_i, i = 1, \ldots, N$ están multiplicadas por los pesos y sumadas todas añadiendo además una constante llamada *bias*. El resultado de esa operación da la entrada a la función de activación *f*. Al igual que para el Perceptrón simple existen distintas funciones de activación.

Conectando varios nodos en paralelo se crea una típica red MLP. Cada sensor de entrada está conectado con las neuronas de la segunda capa, y cada neurona de proceso de la segunda capa está conectada con las neuronas de la primera capa y con las neuronas de la tercera capa, y así sucesivamente. Las salidas están conectadas solamente con las neuronas de la última capa oculta, como se muestra en la Figura **2-12**.

El aprendizaje ocurre en el MLP cuando ajusta los pesos de la conexión basado en la cantidad de error en la salida comparada al resultado esperado a través del algoritmo de retropropagación, una generalización del algoritmo de mínimos cuadrados. En primer lugar se realiza la propagación hacia adelante donde se obtiene la combinación lineal de las entradas y los pesos de cada Perceptrón simple pasado por la función de activación que se aplique. Las salidas para la capa oculta y para la capa de salida se obtienen como se exponen a continuación:

$$o_j = f(\sum_{i=0}^{N} w_{ij} x_i), \tag{2-34}$$

donde i y j pueden tomar valores entre $1 \leq i \leq N, 1 \leq j \leq M$,

$$y_k = f(\sum_{j=0}^{M} w_{jk} o_j), \tag{2-35}$$

donde j y k pueden tomar valores entre $1 \leq j \leq M, 1 \leq k \leq L$.

El aprendizaje se da al minimizar el error, tal como ocurre para el Perceptrón simple en la Ecuación 2-32, pero con algunos cambios. Ahora se debe tener en cuenta la capa a la que pertenece el error. Queda expresada de la siguiente forma:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^{M} (d_k - y_k)^2 \tag{2-36}$$

donde $\mathbf{w}$ es la matriz de pesos de toda la red y d_k es la k-ésima salida asociada al g-ésimo patrón.

Con esto definido se establece por la regla delta [74] que la actualización de pesos de la última capa de salida se hará según:

$$\Delta w_{jk} = w_{jk}(l+1) - w_{jk}(l) = -\nu \frac{\partial E}{\partial w_{jk}} = \nu (d_k - y_k) f^{'}(\sum_{j=0}^{M} w_{jk} o_j) o_j, \tag{2-37}$$

donde $1 \leq j \leq M, 1 \leq k \leq L$ y ν es el coeficiente de aprendizaje.

Para actualizar los pesos de las capas se aplica este razonamiento a la matriz de pesos de la capa oculta, así:

$$\Delta w_{ij} = -\nu \frac{\partial E}{\partial w_{ij}}, \tag{2-38}$$

donde $1 \leq j \leq N, 1 \leq k \leq M$.

Entonces, después de calcular la derivada parcial del error por la regla de la cadena y reemplazando términos se llega a la regla de actualización de pesos para las capas ocultas:

$$\Delta w_{ij} = -\nu \left[\sum_{k=1}^{M} (d_k - y_k) f^{'}(\sum_{j=0}^{M} w_{jk} o_j) w_{jk} \right] f^{'}(\sum_{j=0}^{M} w_{ij} x_i) x_i, \tag{2-39}$$

donde $1 \leq j \leq N, 1 \leq k \leq M$.

Con esta arquitectura los MLPs consiguieron solventar las limitaciones de las redes de una sola capa y consiguen aproximar cualquier función con el grado de precisión deseado.

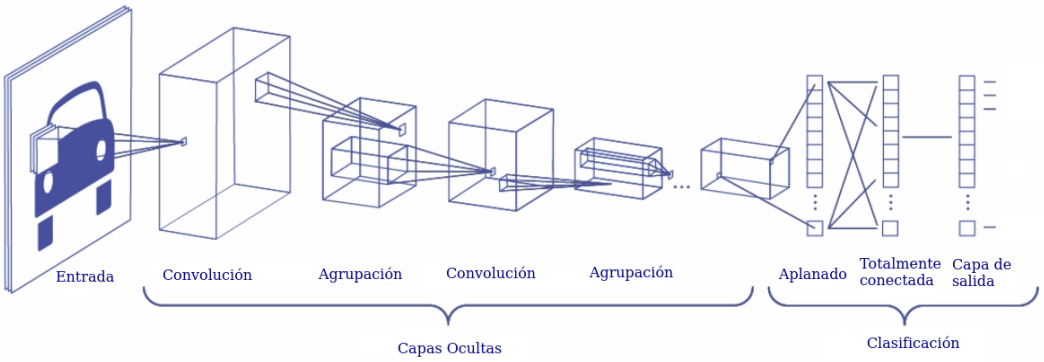


Figura 2-13: Arquitectura general de las CNN, figura modificada de https://www.mathworks.com.

Redes convolucionales

Las CNN son una arquitectura de aprendizaje profundo inspirada en el mecanismo de percepción visual natural de los seres vivos. Una CNN se compone de diferentes capas. Las básicas son la capa convolucional, la capa de agrupación y la capa totalmente conectada [15, 63], y por ultimo más recientemente se agrega el Dropout [65]. La Figura **2-13** muestra un esquema de la arquitectura presente en una CNN. La primer capa convolucional está diseñada para detectar características básicas, mientras que las capas subsiguientes aprenden a codificar características más abstractas. Al apilar varias capas convolucionales y de agrupación, gradualmente se pueden extraer representaciones de características con niveles más abstractos.

La capa 0 o de entrada consiste en una imagen que ingresa a la red.

La capa de convolución consiste en un conjunto de filtros entrenables que se utilizan para calcular diferentes mapas de características. Al deslizar (convolucionar) el filtro sobre el espacio de entrada, se produce un mapa de activación bidimensional que da las respuestas de ese filtro en cada posición espacial. Formalmente la convolución se puede definir de la siguiente manera. Sea I una imagen bidimensional de entrada y K un filtro de tamaño hxw, donde h es la altura en píxeles y w el ancho en píxeles. La imagen convolucionada es igual a:

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{i=j}^{w} K_{ij} \cdot I_{x+i-1,y+j-1} \tag{2-40}$$

La cantidad de convoluciones finales que se llevaran a cabo van a están definidas en base a la cantidad de filtros que se definan y por los parámetros de *Stride* y *Padding*. El *Stride* indica que tan grande es el paso del filtro al convolucionarse con la imagen de entrada, mientras que el *Padding* indica cuantos píxeles de borde se agregaran en la imagen de entrada. El *Padding* viene a que en algunas implementaciones se desea hacer que el resultado de la convolución tenga un ancho y una altura determinado, de manera de hacer que la salida tenga el mismo tamaño que la entrada.

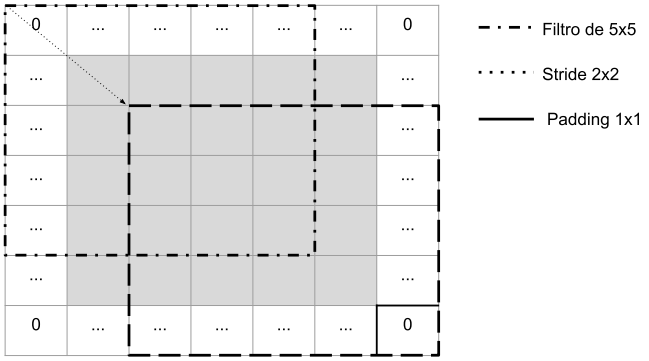


Figura 2-14: Stride y Padding.

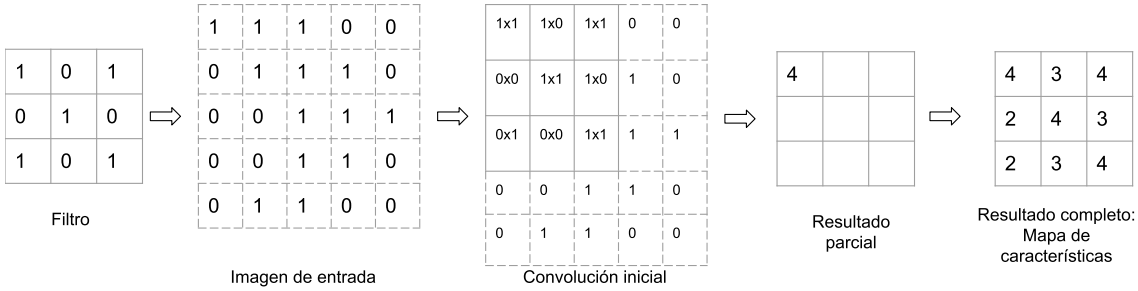
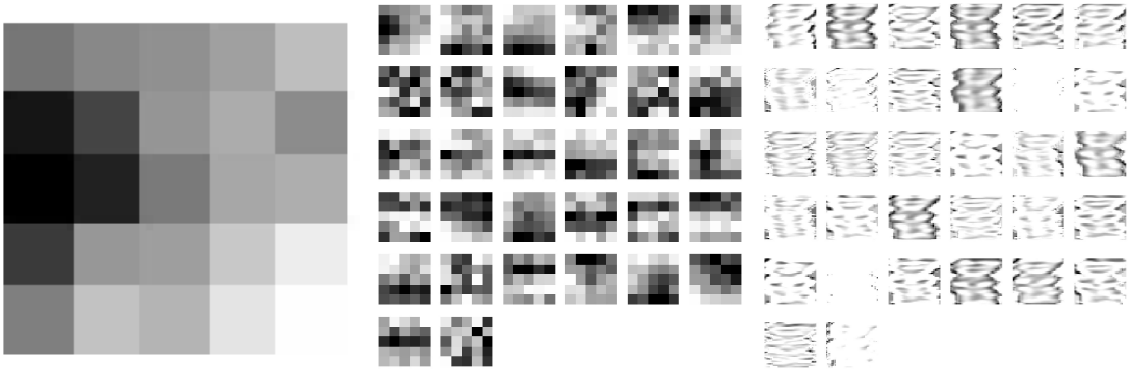


Figura 2-15: Convolución con un filtro de entrada de 3x3 sobre una imagen de 5x5 con Stride de 1x1 y Padding cero.

A continuación se expone un ejemplo para una convolución bidimensional con *Stride* de 1x1 y *Padding* 0 (Figura **2-15**). En cada ubicación se realiza la multiplicación de la matriz del filtro contra un área de la imagen de entrada y se suma. El resultado es el mapa de características para ese filtro.

Finalmente, una vez aplicadas todas las convoluciones, se obtiene como resultado diferentes mapas de características. Así la red aprenderá los filtros que se activan cuando detectan algún tipo de característica visual como un borde. Cada uno de los filtros aprenderá un mapa de activación bidimensional distinto, esas activaciones representan las características que la red va extrayendo. Los mapas de características individuales se "apilan" uno a uno a lo largo de la dimensión de profundidad dando lugar a los mapas de activación o también llamados filtro completos.

En la Figura **2-16** se muestra un ejemplo gráfico de la forma de los filtros entrenables. En este punto es importante entender que si, por ejemplo, se define que la i-ésima capa convolucional tendrá 32 filtros bidimensionales en su entrada, de un tamaño de 5x5 píxeles cada uno (Figura **2-16**a), el filtro completo para esa capa no será de 5x5 píxeles, sino un filtro de 5x5x32 ya que nuestra entrada tiene 32 dimensiones (Figura **2-16**b). Este filtro todavía no fue convolucionado con ninguna información de entrada. Si ahora consideramos una entrada de 50x50 valores (la imagen de entrada en la Capa 0 o la dimensión de salida de la capa anterior si se está parado en una capa interna, por lo general una capa de agrupación), con un *Stride* de 1x1 y un *Padding* cero, la salida final para la capa convolucional actual será 46x46x32 (Figura **2-16**c). Esto es debido a la naturaleza de la convolución. Para el ejemplo,



(a) Filtro 5x5 (b) Banco de filtros de 5x5 (c) Mapa de características de la capa convolucional

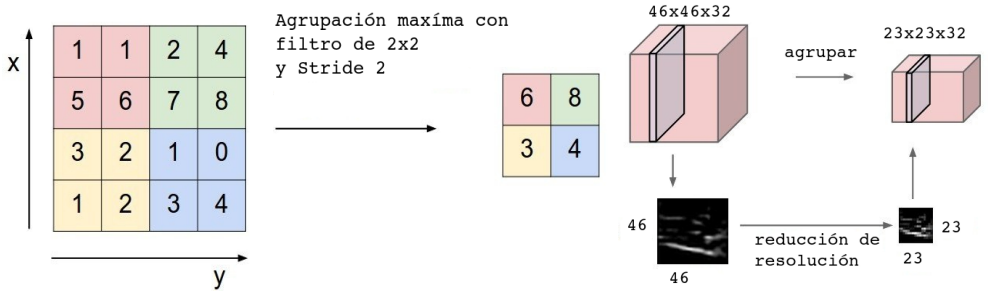
Figura 2-16: Ejemplo de filtros generados en la primera capa de convolución para una entrada de 50x50 píxeles con un tamaño de filtro de 5x5 píxeles.

usamos un vecindario de 5x5 para calcular un punto, pero las filas y columnas externas no tienen un vecindario de 5x5, por lo que no podemos calcular ninguna salida y se perderá esa información. Esto se repite para cada capa convolucional.

La capa de agrupación tiene como objetivo lograr invariancia de cambio (Figura **2-17**), con el fin de mejorar la robustez y capacidad de generalización[4] de la red [57]. Se colocan después de una capa convolucional, de manera que cada mapa de características de una capa de agrupación esté conectado a su correspondiente mapa de características de la capa convolucional precedente. La operación típica es la agrupación máxima [75, 5, 50]. Se basa en aplicar un filtro de máximo a regiones no superpuestas del mapa de características obtenido de la capa convolucional. A modo de ejemplo, supongamos que se tiene como entrada inicial una matriz de 4x4 y que aplicamos sobre ella un filtro de 2x2 con paso 2. Para cada una de las regiones representadas por el filtro, se toma el máximo de esa región y se crea una nueva matriz de salida donde cada elemento es el máximo de la región en la entrada original (Figura **2-17**a). Si aplicamos este razonamiento al mapa de características de la Figura **2-16**c que tiene una dimensión de 46x46x32, el mapa de características de salida de la capa de agrupación tendrá una dimensión de 23x23x32 como se puede observar en la Figura **2-17**b.

Además existe un elemento aplicable luego de cada capa de agrupación o de una capa totalmente conectada denominado Dropout [65]. Es adecuado para reducir la probabilidad de que ocurra un sobreajuste de la red, ya que es deseable que las redes puedan generalizar las aptitudes adquiridas durante el entrenamiento para poder resolver correctamente patrones nunca vistos. La elección de las unidades (neuronas) es aleatoria. Dado un ejemplo de entrada en el entrenamiento, cada unidad se conserva o se descarta con una probabilidad fija "p"definida como parámetro e independiente de otras unidades. Si la unidad es conservada se permite su activación, y si no lo es se le asigna el valor de salida 0. Esto hace que en cada iteración solo una parte de las neuronas se activen. En la Figura **2-18** se puede apreciar gráficamente cómo afecta el Dropout a las conexiones de una red neuronal. Por lo tanto,

[4]Capacidad de una red neuronal de discernir entre ejemplos de entrada independientemente de las características particulares de cada ejemplo.



(a) Agrupación

(b) Agrupamiento en un ejemplo real

Figura 2-17: Figuras adaptadas de Stanford's CS231n github, Capa de agrupación para reducción de invariancias al cambio.

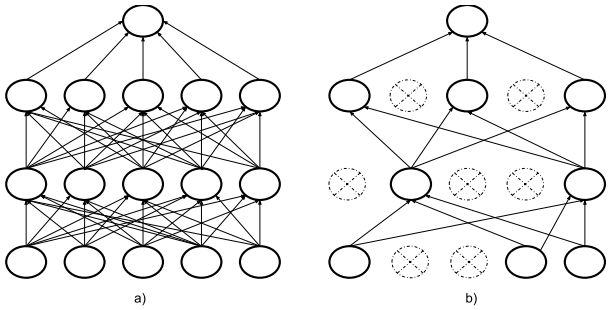


Figura 2-18: a) Red neuronal estándar con 2 capas ocultas. b) Red luego de aplicar Dropout a sus unidades. [65]

el resultado es que en cada iteración se está muestreando aleatoriamente la red neuronal completa, actualizando sólo los pesos de las unidades que fueron seleccionadas.

Después de varios bloques convolucionales y de agrupación, puede haber una o más capas totalmente conectadas que apuntan a realizar un razonamiento de alto nivel [76, 60, 16]. Se toman todas las neuronas de la capa anterior y se conectan a cada neurona de la capa actual para generar información semántica global. Pero para formar la capa totalmente conectada es requerido previamente un proceso de aplanado, que puede ser visto como una nueva convolución de todas las matrices bidimensionales resultantes en un vector. La profundidad o largo de dicho vector va venir dada por la cantidad de características que se desea poseer para trabajar en las capas totalmente conectadas. El aplanado es necesario para que se puedan introducir en las capas completamente conectadas las características extraídas por las capas convolucionales y de agrupación. Las capas totalmente conectadas no tienen una limitación local, como las capas convolucionales (que solo observan una parte local de una imagen mediante el uso de filtros convolucionales). Esto significa que pueden combinar todas las características locales encontradas de las capas convolucionales previas.

La última capa, es la capa de salida. Esta capa puede ser interpretada cómo la capa de clasificación, ya que básicamente se constituye de un clasificador que es entrenado en base a las características extraídas por las capas anteriores. Esta capa puede considerarse independiente de las demás capas anteriores, y permite ser intercambiada. En los experimentos

de este informe, se aplica la red completa a los datos de entrada solo en la etapa de ajuste de la CNN. Luego, una vez ajustada, son empleadas las capas anteriores a la capa de salida, como extractores de características, las cuales una vez extraídas se guardan. Posteriormente se tomaron esas características extraídas y se clasificaron mediante un clasificador distinto al definido en la red.

Máquinas de Aprendizaje Extremo

El algoritmo Máquinas de Aprendizaje Extremo (Extreme Learning Machine - ELM) fue propuesto por Huang [19]. Es una estructura multicapa con una sola capa oculta (Single Layer Feedforward Network, SLFN). En su paso inicial, el algoritmo inicializa aleatoriamente los pesos que unen la capa de entrada y la capa oculta. De esta manera, sólo será necesario optimizar los pesos que están entre la capa oculta y la capa de salida. Para optimizar estos últimos pesos se utiliza la matriz pseudoinversa de *Moore-Penrose* [51]. Uno de los parámetros principales a definir en este tipo de redes neuronales es la cantidad de neuronas en la capa oculta, ya que el rendimiento computacional y la tasa de clasificación es altamente dependiente de estos dos parámetros. En este informe se implementó un desarrollo de ELM que utiliza [43] basado en [18, 17]. A continuación se detalla el método de la matriz pseudoinversa de *Moore-Penrose*.

Dado un conjunto de N patrones, $D = (\mathbf{x}_i, \mathbf{o}_i), i = 1, 2, ..., N$, donde las entradas $\mathbf{x}_i \in \Re^{d1}$ y la salida deseada $\mathbf{o}_i \in \Re^{d2}$, donde *d1* y *d2* representan las dimensiones del espacio de las entradas y las salidas respectivamente. El objetivo es encontrar la relación entre $\mathbf{x}_i$ y $\mathbf{o}_i$. Siendo M el número de neuronas en la capa oculta e y_j la salida j-ésima del SLFN, se tiene:

$$y_j = \sum_{k=1}^{M} h_k \cdot f(\mathbf{w}_k, \mathbf{x}_j), \tag{2-41}$$

donde $1 \leq j \leq N$, $\mathbf{w}_k$ representa los parámetros del k-ésimo elemento de la capa de entrada, h_k es el k-ésimo peso que conecta la capa de entrada con la capa de salida y f es una función de activación aplicada al producto escalar entre el vector de entrada y los pesos de la capa de salida.

Otra forma de expresar la Ecuación 2-41 es como $\mathbf{y}=\mathbf{G} \cdot \mathbf{h}$, donde $\boldsymbol{h}$ es el vector de pesos de la capa de salida, $\boldsymbol{y}$ es el vector de salida y $\boldsymbol{G}$ se calcula mediante:

$$\mathbf{G} = \begin{bmatrix} f(\mathbf{w}_1, \mathbf{x}_1) & \cdot & \cdot & \cdot & f(\mathbf{w}_M, \mathbf{x}_1) \\ \cdot & \cdot & & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot \\ f(\mathbf{w}_N, \mathbf{x}_1) & . & . & . & f(\mathbf{w}_M, \mathbf{x}_N) \end{bmatrix}$$

$$\mathbf{h} = (h_1 \dots h_M)^T \tag{2-42}$$

$$\mathbf{y} = (y_1 \dots y_N)^T. \tag{2-43}$$

Se inicializan aleatoriamente los pesos de la capa de entrada. Los pesos de la capa de salida se obtienen mediante la matriz pseudoinversa de Moore-Penrose ($\mathbf{G}^+$) con las siguientes expresiones [51]:

$$\mathbf{h} = \mathbf{G}^+ \cdot \mathbf{o}, \tag{2-44}$$

donde la matriz pseudoinversa $\mathbf{G}^+$ se calcula mediante la expresión:

$$\mathbf{G}^+ = (\mathbf{G}^T \cdot \mathbf{G})^{-1} \cdot \mathbf{G}^T. \tag{2-45}$$

Siendo $\mathbf{G}^T$ la matriz transpuesta y $\mathbf{o} = (o_1 \dots o_N)^T$ es el vector de la salida deseado.

2.4. Medidas de evaluación

Para comparar los clasificadores se emplean medidas que permiten determinar qué tan bien funcionan. Para obtener dichas medidas se utilizan una serie de métricas basadas en el cálculo del número de ejemplos y cómo fueron clasificados. Estas métricas se calculan en función de la matriz de confusión asociada al modelo. Una matriz de confusión es la matriz que para dos o más clases tendrá de manera ordenada los casos obtenidos por el clasificador (valor predicho) y lo que realmente debería haber sido según los datos reales (valor real).

Suponga que se tienen 2 clases o estados en los que los objetos pueden ser clasificados: Positivo y Negativo. Llamaremos Clase "1"a los objetos que realmente son positivos, Clase "1' "a los que el clasificador dice que son positivos, Clase "2"a los que realmente son negativos y por último Clase "2' "a los que el clasificador dice que son Negativos. Luego esto se ordenara como se muestra en la Tabla **2-3**:

Tabla 2-3: Matriz de confusión para un clasificador binario.

		Valor real	
		Positivo (real): Clase "1"	Negativo (real): Clase "2"
Valor predicho	Positivo (predicho): Clase " 1' "	VP	FP
	Negativo (predicho): Clase " 2' "	FN	VN

La Tabla **2-3** presenta la estructura de esta matriz para el caso de un clasificador binario, donde las entradas de la matriz de confusión tienen el siguiente significado:

- Verdaderos positivos (VP): Casos que pertenecen a la Clase “1” y el clasificador los definió como Clase “ 1’ ”: se corresponde al número total de ejemplos que el clasificador definió como positivos y cuyas etiquetas correctas son las positivas.
- Falsos positivos (FP): Casos que pertenecen a la Clase “2” y el clasificador los definió como Clase “ 1’ ”: se corresponde al número de ejemplos que el clasificador definió como clase positiva, pero cuyas etiquetas reales son negativas.
- Falsos negativos (FN): Casos que pertenecen a la Clase “1” y el clasificador los definió como Clase “ 2’ ”: indican el número total de ejemplos que el clasificador definió como negativos, pero que realmente pertenecen a la clase positiva.
- Verdaderos negativos (VN): Casos que pertenecen a la Clase “2” y el clasificador los definió como Clase “ 2’ ”: se corresponde con el número total de ejemplos que el clasificador definió como clase negativa, y cuyas etiquetas reales son negativas.

En la diagonal principal se indica el número de ejemplos correctamente asignados a cada clase, y fuera de ella los ejemplos de cada clase que son asignados incorrectamente a otra. Siguiendo las mismas ideas, esta matriz puede ser fácilmente extendida para su uso con un clasificador multiclase [21].

Con estas medidas la forma más sencilla de medir el desempeño de un clasificador es calcular su exactitud mediante la cantidad de aciertos obtenidos para la clase dada sobre el total de las predicciones hechas (medida conocida como accuracy o ACC). Esta se define como:

$$Exactitud(ACC) = \frac{VP + VN}{VP + VN + FP + FN} \tag{2-46}$$

En un ámbito de recuperación de información la *exactitud* (Ecuación 2-46) indica la capacidad del método para asociar la clase correcta a cada patrón. Esta medida puede tomar valores en el intervalo [0;1], siendo ”1” el valor asociado a una clasificación correcta de todos los patrones. Sólo puede emplearse para problemas de 2 clases. En problemas multiclase, suele considerarse como clase negativa a todas aquellas que no forman parte de la positiva. No obstante, esta medida no es buena especialmente cuando se tratan conjuntos de datos no balanceados por clases. Si se quiere modelar un clasificador, y los datos utilizados tienen más ejemplos de una clase particular, y si se obtiene un 95 % de predicciones correctas sobre el total, no hay seguridad de que el modelo sea bueno si el 5 % restante abarca muchos o todos los casos de las clases con menos elementos.

Para evitar esto, se adiciono un conjunto de medidas ampliamente empleadas en la literatura [40, 72, 26, 30, 61] que tienen en cuenta el tipo de error cometido. Son las medidas de *precisión*, *sensibilidad* y *Unweighted Average Recall*.

En tareas de clasificación, el valor de *precisión* determina para una clase la cantidad de Verdaderos Positivos dividido por el número total de elementos clasificados como pertenecientes a dicha clase (es decir la suma de Verdaderos Positivos y Falsos Positivos). Se define como:

$$Precisión = \frac{VP}{VP + FP} \tag{2-47}$$

La *precisión* (Ecuación 2-47) tiene en cuenta el "tipo" de error cometido, en base a la cantidad de Falsos Positivos detectados por el clasificador. Ya que para una clase dada, esta medida indica la proporción de Verdaderos Positivos (VP) presentes en la totalidad de positivos devuelta por el clasificador. Esta medida puede tomar valores en el intervalo [0;1], siendo "1" el valor asociado a una clasificación correcta de todos los patrones identificados como VP que realmente los son.

Por su parte, la *sensibilidad* (recall) por definición representa la cantidad de Verdaderos Positivos predichos sobre el total de elementos que realmente pertenecen a la clase en cuestión. Se define como:

$$Sensibilidad = \frac{VP}{VP + FN} \tag{2-48}$$

La *sensibilidad* (Ecuación 2-48) tiene en cuenta el "tipo" de error cometido, "pesado" en base a la cantidad de Falsos Negativos detectados por el clasificador. Ya que para una clase dada, esta medida evalúa la proporción de los VP que son efectivamente recuperados. Puede tomar valores en el intervalo [0,1], siendo "1" cuando el clasificador clasifica correctamente todas las instancias de la clase positiva.

Por último, el *Unweighted Average Recall* (UAR) (Ecuación 2-49) indica la precisión promedio por clase.

$$UAR = \frac{1}{|C|} \sum_{k=1}^{|C|} \frac{VP_k}{VP_k + FN_k} \tag{2-49}$$

Es una medida útil en problemas multiclase por que tiene en cuenta el desbalance en su resultado. Puede tomar valores en el rango [0;1], donde 1 indica el clasificador ideal, que no asigna ningún Falso Negativo en ninguna clase. Donde la variable C indica el número total de clases del problema, y k es la clase particular que está siendo considerada.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
Cristian Vizarri, C. E. Martínez & M. Gerard; "Herramienta de código abierto para la construcción automática de cariogramas (Undergraduate project)"
Facultad de Ingeniería y Ciencias Hídricas -Universidad Nacional del Litoral, 2018.

Capítulo 3

Metodología para la construcción automática de cariogramas

En este capítulo se introduce una metodología para la clasificación automática de cromosomas humanos en metafase. Inicialmente se describe el proceso general implicado en la clasificación de cromosomas. Luego se explica el preprocesamiento requerido por las imágenes individuales de cromosomas. Por último se explican dos formas de extracción de características aplicables a las imágenes de los cromosomas, una que se denominó *forma estándar* basada en técnicas de procesamiento de imágenes clásicas y otra basada en técnicas de deep learning.

3.1. Flujo de trabajo

El proceso general de clasificación de cromosomas puede ser dividido en dos grandes bloques (Figura **3-1**). El primero consiste en un bloque de extracción de características, el cual es el encargado de extraer el vector de características de cada cromosoma; el segundo bloque, llamado módulo de clasificación, se encarga de identificar la clase del cromosoma en base a sus características. El bloque de clasificación es independiente y puede ser intercambiado, de manera que las características extraídas pueden combinarse con distintos clasificadores.

En este estudio, se aborda cada paso del proceso de extracción de características mediante dos enfoques: de *forma estándar*, basado en técnicas de procesamiento de imágenes clásicas, y otro basado en técnicas de deep learning, mediante una red convolucional. Para ambos enfoques se realiza primero el preproceso del conjunto de imágenes cromosómicas individuales (Figura **3-2**), donde es analizada su curvatura y son enderezadas. Posteriormente se extrae para cada cromosoma una serie de descriptores (características) con los cuales se define un vector que lo representa. Para la *forma estándar* estos descriptores están perfectamente identificados y de forma explícita en la Tabla **2-1**, mientras que los de la técnica basada en deep learning con la red neuronal convolucional no son posibles de determinar de forma explícita. Finalmente, con el objetivo de determinar el modelo de clasificación que mejor se ajusta a cada método de extracción de características, se evaluaron diferentes tipos de

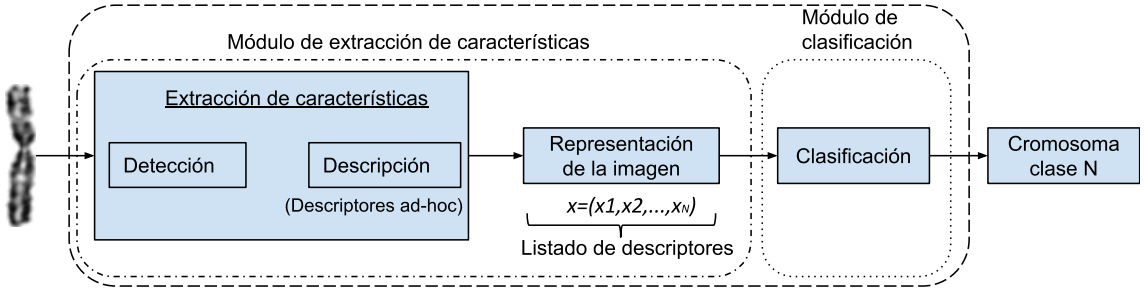


Figura 3-1: Proceso general de extracción de características y clasificación

clasificadores. A partir de estos resultados, se seleccionó el conjunto de características que mejor representa a los cromosomas de la base de datos seleccionada.

3.2. Preprocesamiento de imagen individual: Enderezamiento de cromosomas con alta curvatura

Dado que en las fotografías suelen aparecer cromosomas que presentan curvatura, se debe realizar previamente a la extracción de características un preprocesamiento que permita enderezarlos. El esquema del proceso general se expone en la Figura **3-2**, donde se explicita el pipeline del preprocesamiento de enderezado.

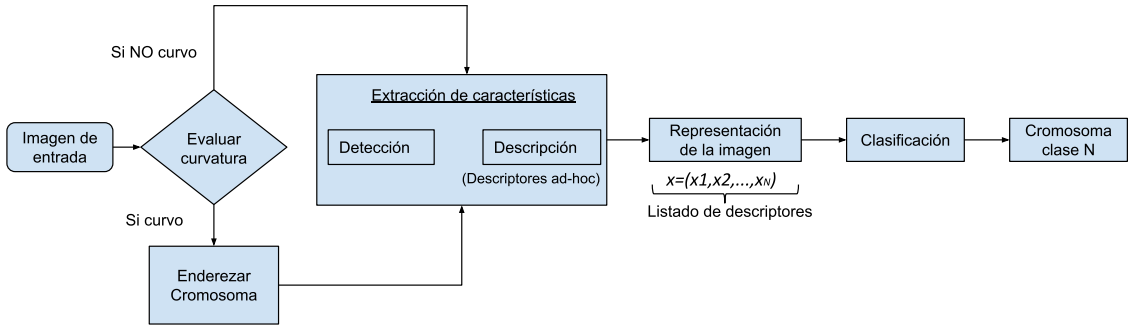


Figura 3-2: Esquema general del procesamiento de las imágenes cromosómicas

Los cromosomas se curvan a partir de su centrómero [54]. La técnica de enderezado consiste en localizar el centrómero del cromosoma, a partir de utilizar una representación binaria de la imagen original en escala de grises. La imagen de un cromosoma puede ser considerada como una imagen bimodal, en donde un objeto está situado sobre un fondo de un único color (generalmente negro o blanco), y conforma una imagen con diferentes escalas de grises. Los vectores de proyección ortogonal (horizontal y vertical) de una imagen binaria contienen toda la información morfológica del objeto [54]. Para calcular el vector de proyección horizontal, se suman los valores de los píxeles de cada fila de la imagen binaria. Teniendo en cuenta que dicha representación binaria, sólo posee 0s (píxeles negros) y 1s (píxeles blancos), los elementos en el vector de proyección representan la cantidad de píxeles blancos (1s) en la fila correspondiente. En el caso de los cromosomas sin curvatura, el míni-

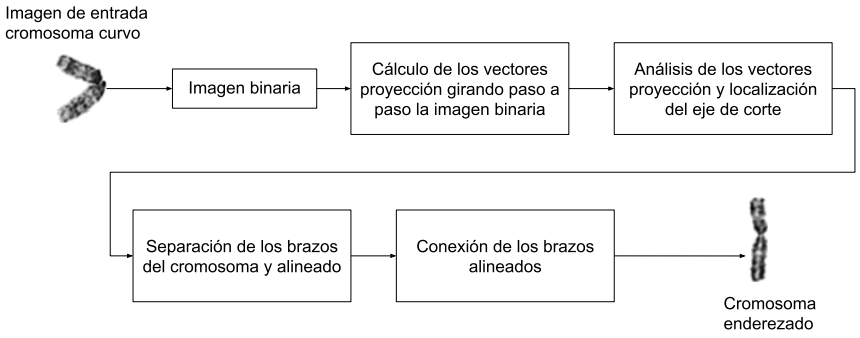


Figura 3-3: Diagrama de bloques del algoritmo de enderezamiento automático de cromosomas.

mo global de la proyección horizontal se encuentra entre los dos puntos de máximo global[1] (Figura **3-4**a y **3-4**b), en la parte central del vector de proyección, y se corresponde con el centrómero del cromosoma. Localizando los dos máximos y el mínimo global en el vector de proyección horizontal, se puede encontrar de forma automática la posición del centrómero del cromosoma.

Una aproximación similar puede aplicarse en los cromosomas curvos, rotando la imagen y obteniendo, para cada rotación un nuevo vector de proyección. Luego analizar dichas proyecciones para estimar la posición del centrómero; y finalmente separar y alinear los brazos del cromosoma. En la Figura **3-3** se muestra un ejemplo esquemático del flujo de trabajo del método de enderezado automático de cromosomas. Toma como entrada un cromosoma, obtiene su imagen binaria, luego calcula las proyecciones sobre la imagen binaria con un paso de rotación de 5° en el rango 0° a 180°. Entonces ese conjunto de proyecciones es analizado[2] y se busca sus máximos y mínimos. Cada vector proyección correspondiente al cromosoma girado, contiene dos puntos de máximos globales con amplitudes similares y un punto de mínimo global entre ellos (Figura **3-4**b).

Sean P_1 la magnitud del primer máximo global para una proyección, P_2 la magnitud del segundo máximo global, P_3 la magnitud del mínimo global, se define la puntuación S de rotación para esa proyección como:

$$S = w_1 \times R_1 + w_2 \times R_2, \tag{3-1}$$

donde $R1 = |P_1 - P_2|/(P_1 + P_2)$ es la diferencia relativa entre las magnitudes de los máximos, y $R_2 = P_3/(P_1 + P_2)$ indica la amplitud relativa del mínimo global respecto de los máximos globales. Los coeficientes w_1 y w_2 son los parámetros de ajuste para controlar el peso de cada término en el índice de rotación S donde $w_1 < 1$, $w_2 < 1$ y $w_1 + w_2 = 1$. La imagen girada objetivo es aquella para la cual la puntuación de rotación S es mínima. El mínimo global

[1]Se considera cromosoma curvo al que no cumple con esta premisa

[2]Se descartan los valores iniciales y terminales de la proyección, ya que representan el inicio y final del cromosoma.

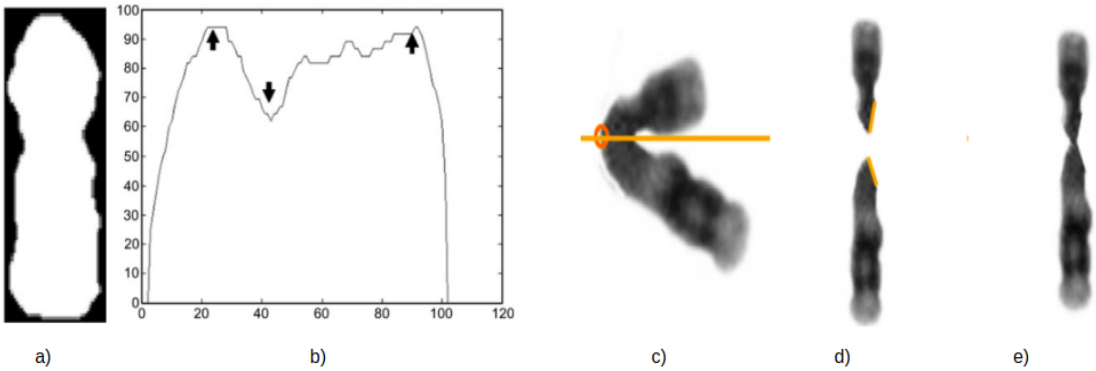


Figura 3-4: a) Imagen Binaria de un cromosoma, b) Representación del vector proyección horizontal, c) Eje de corte, d) Brazos después del proceso de enderezado, e) Resultados después de conectar los brazos.

en el vector de proyección horizontal de dicha imagen corresponde al centro de flexión del cromosoma y es por donde se corta para separar los brazos del cromosoma. Luego se realiza el cálculo de las proyecciones verticales sobre la imagen binaria de cada brazo del cromosoma con paso 5° en el rango -50° a 65° y se seleccionan las proyecciones que minimicen su ancho. Una vez determinadas las proyecciones de mínimo ancho (y cómo se tiene un mapeo directo de cuantos grados fue girada la imagen binaria para obtenerla), se toma el ángulo de giro de las imágenes binarias que dieron lugar a las proyecciones y se aplican esos ángulos de giro a cada brazo del cromosoma respectivamente. Por último se realiza una traslación para el pegado de los brazos.

En la Figura **3-4**a se observa el cromosoma binarizado mediante umbralizado. En la Figura **3-4**b su vector de proyección horizontal para un determinado ángulo, donde las flechas exteriores indican las posiciones de los máximos globales y la interna la posición del mínimo global. En la Figura **3-4**c se puede observar el eje de corte, el cual permite separar al cromosoma en sus brazos (Figura **3-4**d). Por último esos brazos son pegados para formar nuevamente el cromosoma (Figura **3-4**e).

3.3. Extracción de características

La selección y extracción de características son dos etapas que determinan fuertemente la eficiencia y precisión del proceso de clasificación [24, 69]. La selección de características es el proceso por el cual se selecciona un subconjunto de ellas para utilizarlas luego en la construcción de un modelo. Su importancia es marcada en casos donde se dispone de un elevado número de características que describen, por ejemplo, una imagen y se cuenta con una base de datos reducida. Por otro lado, la extracción de características es un proceso que consiste en reducir la cantidad de información necesaria para describir ese subconjunto de datos seleccionado. Partiendo de los datos originales, el proceso consiste en generar información derivada que permita describir de forma más compacta los mismos datos.

Para la *forma estándar*, la definición de las características seleccionadas y la explicación de

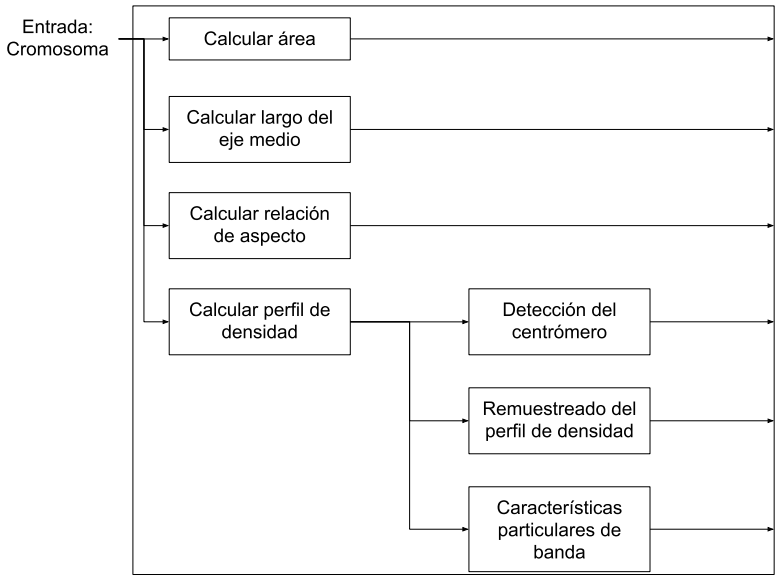


Figura 3-5: Orden de procedimientos seguidos para obtener características mediante *forma estándar.*

las tarea aplicadas sobre la imagen cromosómica para extraerlas se realizó en la Sección 2. En la Figura **3-5** se muestra un esquema general con el orden de procesos seguidos para obtener dicho conjunto de características. Se puede observar, que en una primera etapa, se obtienen las características asociadas a la "Distribución de píxeles". Su cálculo no presenta mayores dificultades ya que se obtienen directamente de la información presente en los píxeles de la imagen. Luego se calcula el perfil de densidad y finalmente se obtienen las demás características que están directamente relacionadas al entorno local del cromosoma.

Por otro lado el orden llevado adelante para la técnica de deep learning por la CNN se puede describir en base a la configuración usada en la red y de la funcionalidad de cada una de dichas partes constituyentes. Cada una de sus partes y funcionalidades fueron descriptas en la subsección 2.3.5. La arquitectura detallada de la CNN aplicada en este trabajo para la extracción de las características cromosómicas será desarrollada en el capítulo 4.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
Cristian Vizarri, C. E. Martínez & M. Gerard; "Herramienta de código abierto para la construcción automática de cariogramas (Undergraduate project)"
Facultad de Ingeniería y Ciencias Hídricas -Universidad Nacional del Litoral, 2018.

Capítulo 4

Configuración experimental y resultados

En este capítulo se expondrán las configuraciones experimentales seleccionadas para la extracción de características y para su clasificación. Se describirá qué parámetros se optimizaron, que experimentos se realizaron y los resultados obtenidos. Finalmente se expondrá un análisis y conclusiones de dichos resultados.

4.1. Descripción de la base de datos

Un requisito esencial para entrenar los sistemas de clasificación es contar con un conjunto de imágenes adecuadamente etiquetadas, que permitan identificar correctamente los cromosomas en las mismas. El cuadro siguiente presenta un resumen de las características de 4 bases de datos, de libre acceso, que son de las más citadas en la literatura.

En la Tabla **4-1** puede apreciarse que la base de datos pki-3 es la que cuenta con el mayor número de células y con las imágenes de mayor resolución. Lisbon-K por su parte, cuenta con características similares a pki-3; pero el número de células es considerablemente menor, lo cual dependiendo del método de extracción de características y clasificación que se seleccione

Tabla 4-1: Información disponible de base de datos recolectadas.

	Cant. Células	Cant. Imágenes[5]	Tinción	Segmentada	Etiqueta	Formato	Resolución
pki-3[1]	612	28144	G	SI	SI	TIFF	768x582,8 bits/píxel
Lisbon-K[2]	41	41	G	SÍ	SI	TIFF	768x512,8 bits/píxel
BioImLab[3]	117	162	Q	SI	SI	BMP	768x576,8 bits/píxel
ADIR[4]	76	1398(200 M-FISH Images)	MFISH	NO	NO	TIFF	517x645,8 bits/píxel

[1] http://www.fim.uni-passau.de/en/faculty/former-professors/mathematical-stochastics/chromosome-image-data
[2] http://mediawiki.isr.ist.utl.pt/wiki/Lisbon-K_Chromosome_Datase
[3] http://bioimlab.dei.unipd.it/Chromosome%20Data%20Set%204Seg.htm
[4] http://live.ece.utexas.edu/research/mfish.html, github.com/jeanpat/MFISH, github.com/mpsampat/M-FISH-chromosome
[5] En algunos casos, el número de imágenes resulta mayor al de células debido a que los cromosomas se encuentran distribuidos en un área más grande que el campo de observación de la cámara, debiendo registrarse varias fotografías para construir la imagen completa.

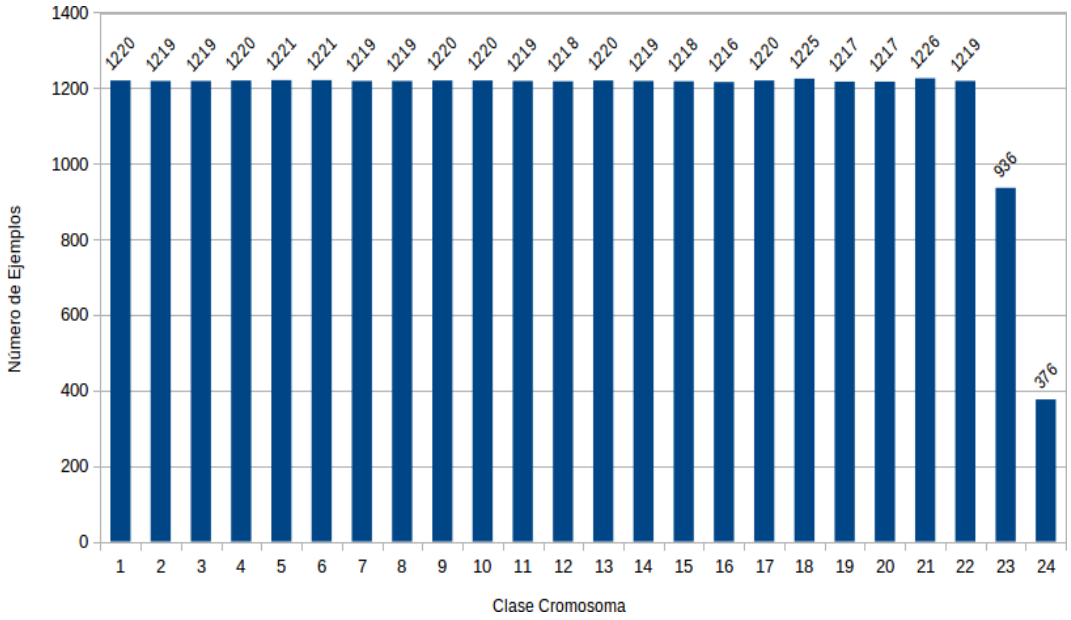


Figura 4-1: Número de cromosomas por clase perteneciente a la base de datos pki-3.

puede ser una limitante en dicha etapa. Por su parte, la base de datos BioImLab presenta un mayor grupo de células que Lisbon-K y ADIR M-FISH[1], y las imágenes son obtenidas por bandeo Q, por lo cual, los cromosomas se tiñen con un colorante fluorescente que genera un patrón complementario al de las bandas G. Finalmente, ADIR M-FISH cuenta con el mayor número de imágenes, aunque presenta un número relativamente pequeño de células. Esto es consecuencia de que el formato de almacenamiento no soporta múltiples planos, la base de datos aplica el formato TIFF, por lo que cada plano de la imagen se almacena en un archivo separado. Todas las base de datos explicitadas son públicas, y se encuentran disponibles directamente o por pedido en sus respectivos sitios web. Sólo se solicita como requisito que sean citados los autores originales de las mismas en la investigación en la que sean utilizadas.

La base de datos de cromosomas que se empleo fue pki-3 [53]. Contiene 612 células y 28144 cromosomas con bandeo G. Las imágenes se encuentran en escala de grises con 8 bits de profundidad y en formato TIFF. Las longitudes cromosómicas presentan una variabilidad de entre 10 píxeles y 180 píxeles aproximadamente, dependiendo del tipo de cromosoma y del factor de zoom. Por cuestiones experimentales se seleccionó ésta base de datos dado que posee la mayor cantidad de casos de estudio para extraer características, lo cual resulta necesario debido a que el problema será abordado mediante algoritmos de aprendizaje maquinal, los cuales requieren grandes volúmenes de información para su entrenamiento. En la Figura **4-1** se muestra la estructura de los datos por clases de cromosomas.

[1]Técnica que utiliza 5 colorantes que se unen a una secuencia específica de ADN de tal manera que cada clase de cromosoma absorbe una combinación única de colorante.

4.2. Configuración experimental

Para ambos grupos de características, las extraídas mediante el enfoque *forma estándar* y las extraídas mediante el enfoque basado en técnicas de deep learning, se fijó el número de características extraídas en 128. En particular, esto permitirá evaluar si el mismo número de características extraídas mediante deep learning aportan más información a los clasificadores que las características extraídas de *forma estándar*.

Para evaluar el desempeño de los algoritmos de clasificación el conjunto de datos se dividió en dos particiones: una de entrenamiento (aproximadamente un 85 % del total de datos) y otra de validación (aproximadamente un 15 % del total de datos). Para el entrenamiento de los clasificadores se utilizó la partición de entrenamiento y se aplicó particionamiento k-fold con k=10. Posteriormente, los clasificadores fueron evaluados con el conjunto de datos reservados para validación.

Todos los experimentos se llevaron a cabo sobre un PC Intel® Core™ i5-3230M CPU @ 2.60GHz × 4, placa de vídeo GeForce GT 740M/PCIe/SSE2, 6 Gb memoria RAM DDR3 y sistema operativo Ubuntu 16.04 LTS.

Configuración de la extracción de características de forma estándar

En particular para la etapa de extracción de características de forma estándar, el rango de sintonización ideal para los valores w_1 y w_2 empleados en la Ecuación (3-1) se calculó de acuerdo a [54], y quedaron éstos comprendidos en los intervalos [0,42;0,46] y [0,54;0,58], respectivamente. Para los experimentos del informe se empleó w_1=0,42 y w_2=0,54, dado que produjeron los mejores resultados durante experimentos preliminares.

Configuración CNN para extracción de características de forma automática

Para la extracción de características basada en deep learning se aplicó una red CNN con una topología de seis capas (capa de entrada no considerada en el conteo) y dos Dropout teniendo en cuenta las investigaciones aplicadas en [48, 60, 67] e implementada con las librerías [12, 42].

La capa de entrada a la CNN (Capa 0), es la encargada de leer las imágenes crudas y determina el tamaño de la imagen que puede leer la red, se definió con un tamaño de 50x50 píxeles. Lo cual requirió remuestrear todas las imágenes a un tamaño de 50x50 píxeles. Este tamaño fue seleccionado en base a las limitaciones del hardware que se disponía durante la implementación de la herramienta, principalmente al recurso de memoria. Si se dispone de un hardware de mayor capacidad este tamaño puede cambiarse para realizar nuevos experimentos. Cada una de las capas convolucionales fue construida con 32 filtros de entrada, cada uno con un tamaño de 5x5 píxeles. En las capas convolucionales se dispuso un *Stride* de 1x1 y un *Padding* cero. Para las capas de agrupación, se definió una agrupación máxima de 2x2 con *Stride* 2. Posteriormente, a la capa de aplanado se la definió con 128 valores. El mapa de características de la capa de aplanado fue especialmente diseñado con esta

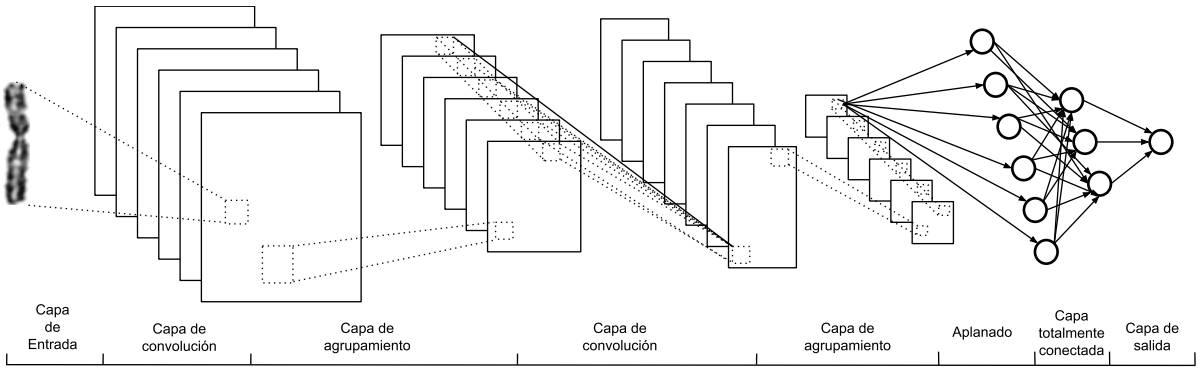


Figura 4-2: Arquitectura definida para la extracción de características.

forma de salida ya que define la cantidad de características extraídas por la red CNN. Así esta cantidad coincide con la cantidad de características extraídas de la *forma estándar*. Por último, la capa de salida se diseñó con 24 valores, correspondientes a la cantidad de clases. Es importante remarcar que la capa de salida sólo se utilizó con la CNN durante su entrenamiento y ajuste de parámetros. Luego una vez ajustada, para la extracción de características no se utilizó, y se tomó directamente la salida de la capa de aplanado como entrada a los clasificadores seleccionados. Con todos los parámetros definidos la red quedó constituida como se expone en la Figura **4-2**.

Ajuste de la CNN para extracción de características

La red fue entrenada con una tasa de aprendizaje de 10^{-2} y un valor de momento ajustado a 0,9 basado en los ajustes aplicados en investigaciones [48, 60, 67]. Estos valores fueron entrenados con {1; 5; 10; 20; 30; 40; 50} épocas y se buscó maximizar el UAR (Tabla **4-2**). Para el entrenamiento se emplearon, en promedio, 14,38 segundos por época, con un desvío estándar de 1 segundo.

Tabla 4-2: Valores UAR tomados por la CNN durante el ajuste.

Número de época	1	5	10	20	30	40	50
UAR entrenamiento	0,6891	0,9236	0,9507	0,9637	0,9741	0,9822	0,9828
UAR evaluación	0,6694	0,8985	0,9312	0,9313	0,9381	0,9456	0,9346

En la en la Tabla **4-2** se muestra la variación del UAR en función del número de épocas para los datos de entrenamiento y evaluación. En general, el UAR de evaluación mejora más de un 25 % al pasar de la época 1 a la época 10 y luego se estabiliza, se observó una mejora lenta hasta la época 40 donde posteriormente decrece. La red logra proporcionar el mejor UAR con 40 épocas, utilizando para el ajuste un clasificador tipo MLP (sexta capa de la CNN). De una forma más gráfica esto se muestra en la Figura **4-3**.

La Figura **4-4** presenta un ejemplo de las salidas intermedias obtenidas para la red entrenada con 40 épocas. Es relevante mostrar estas salidas intermedias ya que permiten mostrar de una manera gráfica cómo la red aprende las características que considera más relevantes

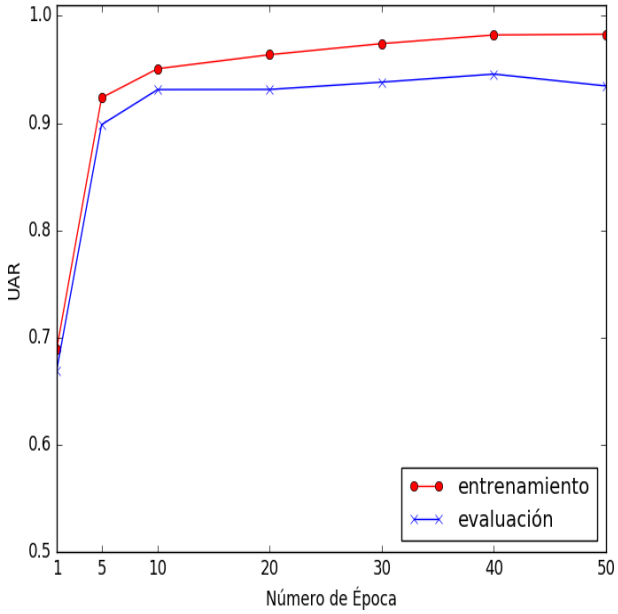


Figura 4-3: Ajuste de CNN: UAR vs Número de épocas.

de un cromosoma. La Figura **4-4**a corresponde a la imagen de entrada. En la Figura **4-4**b se visualizan los pesos que tomaron los 32 filtros de la primera capa convolucional. En la Figura **4-4**c se observa la salida de la primera capa de convolución, la cual es la entrada a la primera capa de agrupación máxima. En la Figura **4-4**d se visualiza la salida de la primera capa de agrupación máxima, la que a su vez es la entrada a la segunda capa de convolución. El proceso se repite y, en la Figura **4-4**e se observa la salida de la segunda capa de convolución la cual es la entrada a la segunda capa de agrupación máxima de la red. La salida de esta se puede observar en la Figura **4-4**f.

A partir de la Figura **4-4**c se puede observar como la red comienza a retener las características más importantes, mientras elimina detalles poco relevantes del mapa de características. A medida que avanzamos en las capas más internas de la red, las características de salida ya no son similares a la imagen de entrada porque se eliminan los detalles sutiles. Son características altamente abstraídas e independientes.

4.3. Optimización de los parámetros de los clasificadores

Se probaron los algoritmos de aprendizaje supervisado de SVM, RF, ELM, KNN, DT. Los parámetros a testear se seleccionaron de acuerdo a recomendaciones en la literatura y experimentos preliminares sobre el uso de cada uno de los clasificadores. Para lograr el ajuste se realizó validación cruzada con K-Fold, con K=10. En los experimentos se emplearon los datos de la partición de entrenamiento y se buscó un ajuste que maximice el UAR.

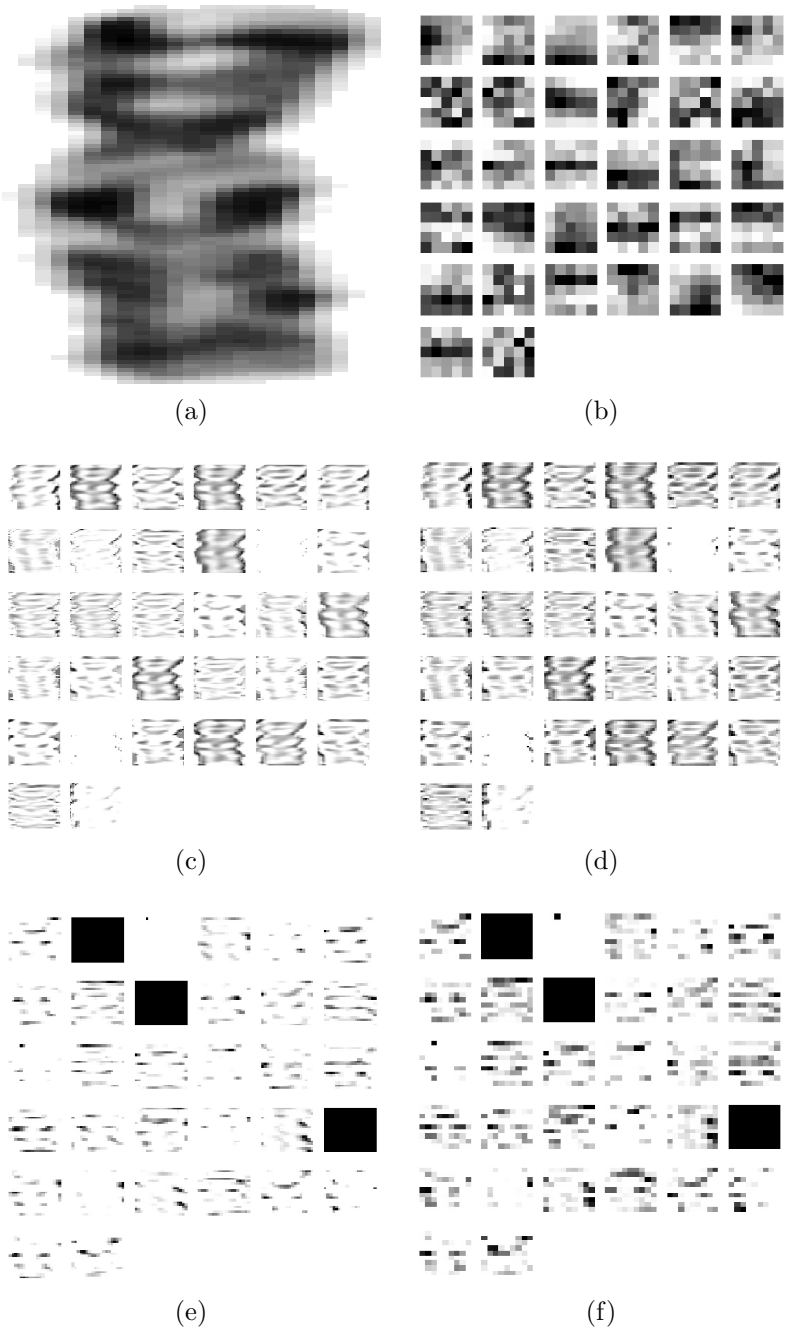


Figura 4-4: Filtros y mapas de características: a) imagen de entrada, b) filtros bidimensionales de la primera capa convolucional, c) mapa de características resultante de la primera capa de convolución, d) mapa de características resultante de la primera capa de agrupación máxima, e) mapa de características resultante de la segunda capa de convolución, f) mapa de características resultante de la segunda capa de agrupación máxima.

4.4. Descripción de experimentos

Para ambos grupos de características los experimentos aplicados sobre los parámetros de los clasificadores fueron los mismos de manera que los resultados fueran comparables. El proceso de optimización de KNN consistió en la identificación del número de vecinos más adecuado para realizar la clasificación. Para ello, se realizó la validación cruzada y se varió el número de vecinos en el intervalo [1;10], usando la distancia de *Manhattan* como métrica de distancia a los vecinos en el algoritmo.

Para el caso de DT, el proceso de optimización consistió en la identificación de la profundidad máxima más adecuada para realizar la clasificación. Para lograrlo, se realizó la variación de la profundidad en el intervalo [1;20] con entropía como función para medir la calidad de la división del árbol.

Con ELM el objetivo fue minimizar el número de neuronas en la capa oculta que logre el mejor resultado de clasificación. Se evaluó la ELM con {100; 500; 1000; 1500; 2000; 2500; 3000; 3500; 4000} neuronas en la capa oculta, valor alpha en {0; 0,1; 0,2; 0,3; 0,4; 0,5; 0,6; 0,7; 0,8; 0,9; 1} y las funciones de activación "multiquadric", "tanh", "sigmoid", "gaussian". Se realizó el ajuste de estos parámetros por separado en tres experimentos. En el experimento inicial se fijó los valores del número de neuronas en 100 y alpha en 0,8 y se buscó la función de activación que logre el mejor resultado. Posteriormente, en un segundo experimento, se buscó optimizar el valor de alpha, para lo cual se fijó el número de neuronas en 100 y como función de activación se tomó la que obtuvo mejor resultados en el experimento anterior. Por último se buscó el número de neuronas que mejorara el resultado de UAR, para ello se fijó la función de activación y el valor de alpha teniendo en cuenta los resultados obtenidos en los experimentes anteriores.

La optimización realizada para Random Forest consistió en determinar el tamaño del bosque y la profundidad de los árboles que maximizaran el UAR. Se realizaron los ajustes de estos parámetros por separado. En un primer paso se implementó el experimento evaluándose bosques con {1; 3; 5; 10; 15; 20; 25; 30; 35; 40} árboles y se dejó fija la profundidad. Luego en un segundo paso y por separado se implementó otro experimento en el que se hizo variar la profundidad en {1; 3; 5; 10; 15; 20; 25; 30; 35; 40}, y se dejó fijó el número de árboles.

Finalmente, para SVM se ajustó el tipo de kernel y el parámetro C del clasificador. Este proceso se dividió en dos etapas. Primero se ajustó el tipo de kernel manteniendo el parámetro C constante en 1. Las funciones de kernel evaluadas fueron las siguientes:

- lineal: $\langle x, x' \rangle$.
- polinomial: $(\gamma \langle x, x' \rangle + r)^d$, donde d especifica el grado del polinomio, $\gamma > 0$ y r el coeficiente.
- gaussiano: $\exp(-\gamma \mid x - x' \mid^2)$, donde $\gamma > 0$.
- sigmoidea: $(\tanh(\gamma \langle x, x' \rangle + r))$, donde r especifica el coeficiente.

El valor de γ se estableció para todos los experimentos de acuerdo con la dimensión del vector de características siguiendo los lineamientos definidos en [36]. De esta manera quedó

definido como $\gamma = \frac{1}{nc}$, donde nc es el número de características (definido en 128 para nuestros experimentos). Por otro lado, en base a experimentos preliminares se definió el grado del polinomio como $d = 5$, y el coeficiente como $r = 0$. Una vez encontrado el mejor kernel se evaluó el parámetro C para los valores {0,1; 0,5; 1; 5; 10}.

4.5. Experimentos de ajuste

En esta sección se mostraran los experimentos de ajustes llevados adelantes para ambos juegos de características y los resultado obtenidos en dichos ajustes. Así, en la Figura **4-5** se exponen la métricas UAR obtenidas por los distintos clasificadores en los distintos experimentos de ajuste, en los que se empleó la partición de entrenamiento con particionamiento k-fold con k=10. En rojo se indican los valores UAR obtenidos por los clasificadores con las características extraídas de forma estándar, mientras que en azul se indican los valores de UAR obtenidos empleando las características extraídas mediante deep learning. Estos resultados serán analizados en detalle en las subsecciones siguientes.

4.5.1. Características extraídas de forma estándar

Se inició el análisis con el clasificador KNN. Este mostró un incremento en el valor del UAR a medida que se consideraba un mayor número de vecinos (Figura **4-5**a). Así, aunque inicialmente se obtuvo un UAR de 0,8456 para k=1, éste valor se incrementó hasta 0,8781 para k=8 y luego permaneció constante al incrementar el número de vecinos. En base a estos resultados se seleccionaron 8 vecinos para realizar futuros experimentos con este clasificador.

Para el caso de DT (Figura **4-5**b), se observó un incremento importante del UAR al permitir la construcción de árboles con profundidades cada vez mayores. Así, mientras que árboles con profundidad 1 lograban un valor de UAR de 0,0829, para el caso de una profundidad 10 se obtuvo un UAR de 0,7745. Sin embargo, al incrementar aún más la profundidad (hasta 20 niveles) no se observó una mejora en los valores de clasificación. En consecuencia, se decidió emplear una profundidad 10 para futuros experimentos que se realicen con este clasificador.

En lo referente a los experimentos con ELM se dividieron en tres etapas. En primer lugar, se determinó que función de activación se adaptaba mejor a los datos con lo que se cuenta, por los que se fijó el número de neuronas en 100, alpha en 0,8 y se probaron las distintas funciones de activación postuladas. En base a este experimento la función seleccionada para los experimentos futuros con ELM y esta base de datos de características fue "multiquadric", ya que se observó que lograba una mejor métrica UAR (Tabla **4-3**).

Tabla 4-3: Características estándar: ajuste función de activación de ELM. Se muestran los valores promedio sobre 10 réplicas del experimento.

Función de activación	tanh	sigmoid	gaussian	multiquadric
UAR	0,7277	0,7827	0,6991	0,7942

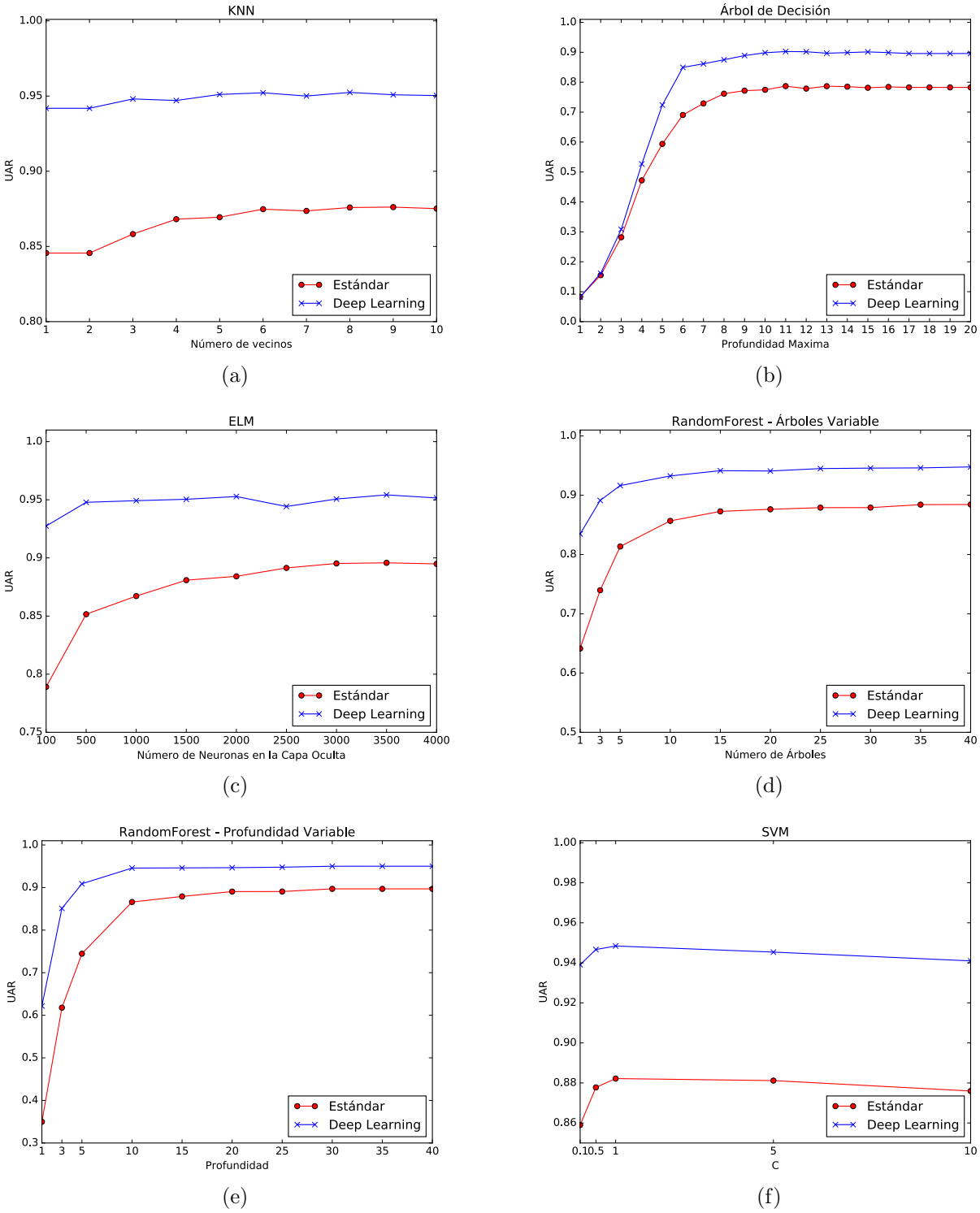


Figura 4-5: Comparación de resultados obtenidos por los clasificadores para ambos juegos de características en su etapa de ajuste. a) KNN, b) Árbol de decisión, c) ELM, d) Random Forest con árboles variables y profundidad fija, e) Random Forest con profundidad variable y árboles fijos, f) SVM

En la segunda etapa del experimento se optimizo el valor de alpha, y se realizó la variación en el rango [0;1] en los valores postulados. El número de neuronas se fijó en 100 y la función de activación en "multiquadric". Los resultados obtenidos se observan en la Tabla **4-4** y en base a estos se seleccionó 0,9 como valor de alpha para futuros experimentos con esta base de datos y ELM.

Tabla 4-4: Características estándar: ajuste alpha de ELM. Se muestran los valores promedio sobre 10 réplicas del experimento.

Alpha	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
UAR	0, 7868	0, 7907	0, 7881	0, 7873	0, 7865	0, 7885	0, 7913	0, 7932	0, 7960	**0,7972**	0, 7950

Finalmente se optimizo el número de neuronas, donde se percibió una notable mejora a medida que se aumentó el número de neuronas en la capa oculta. Inicialmente se obtuvo un UAR de 0,7890 para 100 neuronas en la capa oculta, éste valor se incrementó hasta 0,8952 para 3000 neuronas y luego permaneció constante al incrementar las neuronas (Figura **4-5**c). Dado este comportamiento se tomó 3000 neuronas en la capa oculta para futuros experimentos con la base de datos estándar y este clasificador.

En el caso de RF, para el primer experimento se tomó la profundidad del árbol fija, que fue seleccionada en 15 en base a experimentos preliminares. Mostró un incremento notable del valor de UAR a medida que se consideraban bosques de mayor tamaño (Figura **4-5**d). De esta forma, se entrenó los bosques con un solo árbol, y se logró un valor de UAR de 0,6413, el cual mejoró hasta 0,8762 con bosques de 20 árboles. No obstante, al incrementar más aún la cantidad de árboles en el bosque (hasta 40) no se percibió una mejora significativa en los valores del UAR que acompañase el aumento de tamaño en el bosque de árboles. El segundo experimento realizado, en el cual se fijó en 20 el número de árboles del bosque en base al experimento anterior, se observó una mejora importante en el valor del UAR hasta alcanzar una profundidad 15, y luego se mantuvo constante (Figura **4-5**e). Dado estos dos experimentos se seleccionó un bosque de 20 árboles con una profundidad 15 para experimentos futuros sobre la base de datos estándar con este clasificador.

Por último, los experimentos para ajuste SVM también se dividieron en dos etapas. En primer lugar, se evaluó los distintos tipos de kernels, donde se mantuvo constante el parámetro C (Tabla **4-5**). Se observó que el kernel lineal obtuvo el mejor resultado, con un valor de UAR de 0,8890. A continuación, se buscó el valor de C en el rango definido que maximizaba el UAR, empleando el kernel lineal previamente seleccionado. El mejor UAR fue logrado con C=1 (Figura **4-5**f). Por lo tanto se fijó para experimentos posteriores con SVM y la base de datos estándar el kernel lineal y parámetro C=1.

4.5.2. Para características automáticas

El ajuste logrado por los clasificadores para este set de características fue similar al comportamiento logrado en el set extraído de *forma estándar*. Se siguió una correlación con los experimentos anteriores manteniendo el mismo orden de exposición aquí. Se inició con KNN, en el cual se observó un incremento en el valor del UAR a medida que se consideraba

Tabla 4-5: Resultados del experimento de selección de kernel para SVM sobre la base de datos estándar. Se muestran los valores promedio sobre 10 réplicas del experimento.

Número de Experimento	UAR
1. kernel: función sigmoidea; C=1	$0,8287$
2. kernel: gaussiano; C=1	$0,8485$
3. kernel: polinomial; C=1	$0,4672$
4. kernel: lineal; C=1	$0,8890$

un mayor número de vecinos (Figura **4-5**a). Así, aunque inicialmente se obtuvo un UAR de 0,9418 para k=1, éste valor se incrementó hasta 0,9524 para k=8 y luego permaneció constante al incrementar el número de vecinos. En base a estos resultados se seleccionó 8 vecinos para realizar futuros experimentos sobre la base de datos de características obtenidas mediante técnicas de deep learning con este clasificador.

Para el caso de DT (Figura **4-5**b), se observó un incremento importante del UAR al permitir la construcción de árboles con profundidades cada vez mayores. Puede notarse que el ajuste se realizó con más rapidez que para las características estándar y superó el UAR obtenidas por éstas a partir de profundidad 6 donde obtiene un UAR de 0,8495 y para el caso de una profundidad 10 se obtuvo un UAR de 0,8985. Sin embargo, al incrementar aún más la profundidad (hasta 20 niveles) no se observó una mejora en los valores de clasificación. En consecuencia, se decidió emplear una profundidad 10 para futuros experimentos sobre la base de datos de características obtenidas mediante técnicas de deep learning con este clasificador.

En lo alusivo a los resultados obtenidos con ELM para el primer experimento la función de activación que logra una mejor métrica fue "multiquadric"(Tabla **4-6**), por lo que fue seleccionada para los experimentos futuros con ELM y esta base de datos.

Tabla 4-6: Características deep learning: ajuste función de activación de ELM. Se muestran los valores promedio sobre 10 réplicas del experimento.

Función de activación	tanh	sigmoid	gaussian	multiquadric
UAR	0,8709	0,9084	0,8468	0,9156

En un segundo experimento se optimizó el valor de alpha, para lo cual se realizó la variación en el rango [0;1] en los valores postulados. Los resultados obtenidos se ven en la Tabla **4-7** y en base a estos se seleccionó 0,9 como valor de alpha para futuros experimentos con esta base de datos y ELM.

Tabla 4-7: Características deep learning: ajuste alpha de ELM. Se muestran los valores promedio sobre 10 réplicas del experimento.

Alpha	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
UAR	$0,9197$	$0,9202$	$0,9214$	$0,9192$	$0,9195$	$0,9194$	$0,9173$	$0,9186$	$0,9191$	**0,9268**	$0,9135$

Tabla 4-8: Resultados del experimento de selección de kernel para SVM sobre la base datos extraída mediante deep learning. Se muestran los valores promedio sobre 10 réplicas del experimento.

Número de Experimento	UAR
1. kernel: función sigmoidea; C=1	0,9452
2. kernel: gaussiano; C=1	0,9400
3. kernel: polinomial; C=1	0,0386
4. kernel: lineal; C=1	0,9527

Finalmente en el experimento final para ELM se hizo variar la cantidad de neuronas de la capa oculta, con valor de alpha en 0,9 y función de activación multiquadric. En la Figura **4-5**c), se pueden observar que los resultados superaron el valor de UAR de 0,90 a partir de las 100 neuronas. Logró el mejor resultado con 2000 neuronas. Dado este comportamiento se tomó 2000 neuronas en la capa oculta para futuros experimentos sobre la base de datos de las características obtenidas mediante técnicas de deep learning con este clasificador.

En el caso de RF (Figura **4-5**d), el primer experimento fue realizado variando la cantidad de árboles del bosque, y profundidad fija en 15 en base a experimentos preliminares. Mostró un incremento importante del valor de UAR a medida que se consideraban bosques de mayor tamaño. De esta forma, se entrenó los bosques con un solo árbol y se logró un valor de UAR de 0,8346. Este valor se incrementó hasta 0,9414 para el caso de los bosques con 15 árboles, y luego permaneció constante al incrementar aún más la cantidad de árboles en el bosque (hasta 40). El segundo experimento aplicado (Figura **4-5**e), en el cual se fijó en 15 el número de árboles del bosque en base al experimento anterior, se observó una mejora sostenida en el valor UAR hasta profundidad 10 y luego se mantuvo constante, de forma similar a como ocurre para las características estándar. Dado estos dos experimentos se seleccionó un bosque de 15 árboles con una profundidad 10 para experimentos futuros sobre la base de datos obtenidas para las características obtenidas mediante técnicas de deep learning con este clasificador.

Los experimentos para el ajuste de SVM también se dividieron en dos etapas al igual que para RF. En primer lugar, se evaluaron distintos tipos de kernels (Tabla **4-8**), donde se evidencio que el kernel lineal obtuvo el mejor resultado, con un UAR del 0,9527. Para el segundo experimento se seleccionó el kernel lineal, en base al experimento anterior y se varió el parámetro C en los valores definidos. El mejor UAR fue logrado con C=1 y luego decrece para valores superiores. Se fijó para experimentos posteriores con SVM con los datos obtenidos mediante técnicas de deep learning, un kernel lineal y parámetro C=1.

Además de los experimentos expuestos en la Tabla **4-5** y Tabla **4-8** para SVM, se realizaron experimentos adicionales con un esquema de descomposición SVM OVO[2] sobre las clases que componen el problema, tanto para las características estándar cómo para las extraídas mediante técnicas de deep learning. Los resultados obtenidos con este enfoque en compa-

[2]OVO: One-vs-One: divide un problema de m clases en un problema de m(m-1)/2 clases binarias. Cada clase binaria es enfrentada por un clasificador binario que es responsable de distinguir entre cada par diferente.

Tabla 4-9: Métricas para características extraídas de forma estándar. Los mejores resultados se indican en negrita. Se muestran los valores promedio sobre 10 réplicas del experimento.

	KNN	DT	ELM	RF	SVM
Accuracy	0,8900	0,7907	**0,9044**	0,8981	0,8915
UAR	0,8858	0,7859	**0,8966**	0,8913	0,8863
Precisión	0,8932	0,7939	**0,9057**	0,8991	0,8924
Sensibilidad	0,8900	0,7907	**0,9044**	0,8981	0,8915
Tiempo Total	3,038	1,376	3,832	**0,468**	4,29

Tabla 4-10: Métricas para características extraídas mediante técnicas de deep learning. Los mejores resultados se indican en negrita. Se muestran los valores promedio sobre 10 réplicas del experimento.

	KNN	DT	ELM	RF	SVM	CNN(MLP)
Accuracy	0,9522	0,8861	**0,9552**	0,9478	0,9547	0.9520
UAR	0,9497	0,8814	**0,9560**	0,9491	0,9525	0.9499
Precisión	0,9529	0,8878	**0,9557**	0,9483	0,9550	0.9554
Sensibilidad	0,9522	0,8861	**0,9550**	0,9478	0,9547	0.9531
Tiempo Total	2,727	1,82	1,802	**0,784**	1,821	2.65

ración a los logrados por SVM obtuvieron diferencias del orden de 10^{-3}, por lo que fueron descartados para el análisis.

Concluida la etapa de ajuste, se realizó dos experimentos que validaran el desempeño preliminar mostrado por los clasificadores.

En el primero de ellos, se extrajeron las características de los datos de entrenamiento mediante la *forma estándar* y mediante la CNN. Luego, se tomaron los clasificadores postulados y se los configuro con los parámetros que lograron los mejores resultados en los experimentos descriptos en los párrafos previos, realizados con las características extraídas mediante la *forma estándar* y mediante la CNN respectivamente. Por último se entrenaron los clasificadores, quedando conformados dos conjuntos de clasificadores de acuerdo al conjunto de característica aplicado para el ajuste. Los ajustados en base a características extraídas de *forma estándar* y los ajustados basados en características extraídas mediantes la CNN. Posteriormente se tomaron los datos reservados para validación, se extrajeron sus características mediante la *forma estándar* y mediante la CNN. Luego estos dos conjuntos de características se las clasifico con los dos grupos de clasificadores (clasificadores previamente ajustados con características extraídas de los datos de entrenamiento). Las Tablas **4-9** y **4-10** presentan los resultados obtenidos con los datos de validación para cada clasificador ajustado con las características extraídas de *forma estándar* y mediante técnicas de deep learning, respectivamente. La última columna de la tabla **4-10** (CNN (MLP)) presenta las métricas obtenidas utilizando la CNN completa ajustada con los datos de entrenamiento, aplicando la capa de clasificación que trae incluida en su arquitectura.

En el segundo experimento se combinaron los grupos de características uniéndolos, las ex-

Tabla 4-11: Métricas para características combinadas: 128 características extraídas mediante *forma estándar* y 128 características extraídas mediante técnicas de deep learning. Los mejores resultados se indican en negrita.

	*K*NN	DT	ELM	RF	SVM
Accuracy	0.9111	0.8633	**0.9397**	0.9250	0.9366
UAR	0.9059	0.8293	0.9315	0.9205	**0.9368**
Precisión	0.9138	0.8667	0.9380	0.9263	**0.9400**
Sensibilidad	0.9109	0.8633	0.9360	0.9254	**0.9394**
Tiempo Total	6.03	3.30	4.14	**2.00**	3.96

traídas de *forma estándar* y las extraídas mediante la CNN del conjunto de datos de entrenamientos respectivamente. Para cada modalidad se extraen 128 características por cada cromosoma, por lo que en este experimento se tendrá una configuración de 256 características por cada cromosoma. En este punto, por un lado se tomaron los clasificadores postulados y se los configuro con los parámetros que lograron los mejores resultados con las características extraídas de *forma estándar*, conformando un grupo de clasificadores; por otro lado y de forma independiente se configuro a los clasificadores con los parámetros que lograron los mejores resultados en las características extraídas mediante la CNN, conformado otro grupo de clasificadores. Posteriormente cada grupo se ajustó mostrándoles el conjunto de características combinado. De esta manera, quedaron conformados dos grupos de clasificadores, denotados por los ajustados con los parámetros que lograron los mejores resultados para características extraídas de *forma estándar* y otro grupo ajustado en base a los parámetros que lograron los mejores resultados para las extraídas mediante la CNN pero ajustados con el grupo de características combinado. Finalmente, se tomaron los datos reservados para validación, se extrajo sus características de *forma estándar* y mediante la CNN, se las combino y se las clasifico con ambos grupos de clasificadores (los ajustados previamente en base a los parámetros que lograron los mejores resultados para características extraídas de *forma estándar* y los ajustados en base a los parámetros que lograron los mejores resultados para características extraídas mediante la CNN en ambos casos con datos de entrenamiento). La Tabla **4-11** muestra las métricas obtenidas para las características combinadas y clasificadas con los clasificadores ajustados en base a los parámetros que lograron los mejores resultados para características extraídas mediante la CNN. Las métricas corresponden a un promedio sobre 10 repeticiones al experimento sobre los datos de validación. Por otro lado, las métricas obtenidas para las características combinadas y clasificadas con los clasificadores ajustados en base a los parámetros que lograron los mejores resultados para características extraídas mediante la *forma estándar* obtuvieron diferencias del orden de 10^{-4}, por lo que fueron descartados para el análisis.

4.5.3. Análisis y conclusiones

El clasificador que logró mejores resultados en las métricas y para los dos conjuntos de características fue ELM, excepto para el tiempo total de ejecución[3]. En contraposición, DT tuvo el peor desempeño para todas las métricas (menor a 0,9) y ambos conjuntos de datos.

En particular, para el caso de ELM se puede visualizar que tanto en la Tabla **4-9** como en la Tabla **4-10** la precisión es mayor a la sensibilidad, lo que indica que la mayoría de las etiquetas predichas son correctas (bajos FP). Por otro lado, se observa que el valor de UAR y el accuracy en la Tabla **4-9** y en la Tabla **4-10** son similares, debiéndose las diferencias a desbalances en algunas clases.

Es importante también hacer notar, que las métricas que logró el clasificador ELM aplicado al conjunto de características extraídas mediante técnicas de deep learning, supera en casi el 5 % a todas las métricas logradas a cuando se lo aplicó a características extraídas de forma estándar. Se puede ver también que Random Forest fue el más eficiente, en términos de tiempo, para ambos casos.

Por último, en el experimento en el que se combinó ambos conjuntos de características (Tabla **4-11**), no se evidencia una mejora en las métricas. Esto sugiere que el agregar los descriptores extraídos de *forma estándar* agrega *ruido* no deseado en la clasificación. Probablemente un subgrupo de estas características logre mejorar las métricas, para lo cual sería requerido aplicar un proceso de selección de características previo.

Dado que ELM proporcionó los mejores resultados de clasificación para características extraídas con la CNN, es interesante realizar un análisis detallado de su comportamiento para cada clase. En la Tabla **4-12** se compara el desempeño de clasificación por clase. En particular, se analizó la precisión y la sensibilidad (recall) como medidas de desempeño dado que resultaba de interés determinar la tasa de Falsos Positivos y Falsos Negativos obtenidos con el clasificador. En general, ambas métricas alcanzaron valores elevados (valores mayores o iguales a 0,90) para todas las clases. Pero se observó ciertas clases donde esto no se cumplió, problema que viene dado por la similaridad existente entre clases de cromosomas. Dependiendo la clase a la que pertenezcan los cromosomas pueden presentar características que sean muy similares entre ellos dando lugar a una reducción de los descriptores efectivos para el clasificador.

Por ejemplo este problema de reducción de descriptores efectivos se presenta en los cromosomas de las clases 16 a 22 ya que todos presentan un tamaño y forma similar entre ellos. Además los cromosomas de las clases 16, 17 y 18 son cromosomas en los cuales el centrómero es prácticamente equidistante entre el centro y uno de los extremos, de forma que uno de los brazos es un poco más corto que el otro. Los brazos de las cromátides no tienen igual longitud, sin embargo la diferencia es mínima. Esta característica está presente en la mayoría de los cromosomas por lo que agrega una dificultad extra a los descriptores

[3]Tiempo total: Tiempo total de ejecución en minutos que tardó el algoritmo en procesar todas las particiones de la validación cruzada K-Fold. No se tiene en cuenta el tiempo de transferencia a memoria de los datos en ningún caso.

Tabla 4-12: Características deep learning: Métricas logradas por el clasificador ELM.

Clase Cromosoma	Características extraídas con técnicas de deep learning	
	Precisión	Sensibilidad
Clase 1	0,97	0,98
Clase 2	0,90	0,98
Clase 3	0,98	0,95
Clase 4	0,98	0,95
Clase 5	0,93	0,98
Clase 6	0,97	0,94
Clase 7	0,93	0,98
Clase 8	0,98	0,97
Clase 9	0,94	0,98
Clase 10	0,98	0,95
Clase 11	0,94	0,96
Clase 12	0,98	0,97
Clase 13	0,94	0,96
Clase 14	0,97	0,96
Clase 15	0,95	0,95
Clase 16	0,92	0,89
Clase 17	0,91	0,82
Clase 18	0,92	0,90
Clase 19	0,93	0,91
Clase 20	0,97	0,95
Clase 21	0,95	0,94
Clase 22	0,97	0,94
Clase 23	0,94	0,92
Clase 24	0,96	0,97

para su clasificación. Especialmente en las clases 16 y 17 donde sufre una mayor confusión. En la matriz de confusión (Figura **4-6**) es posible observar este hecho.

Los valores de sensibilidad más altos se dieron en las clases 1, 2, 5, 7 y 9 lo que indica que es ínfima la posibilidad de que para dichas clases el clasificador etiquete a un cromosoma de una de esas clases cómo que no pertenece a la misma. Además se observa que los valores de precisión son altos, por lo que para estas clases el clasificador devuelve resultados precisos. Excepto para la clase 2, donde la precisión es de 0,90. Siendo el valor más bajo de precisión para todas las clases por lo que presenta cierta tendencia a errores al predecir cromosomas de esta clase.

Por último es importante notar, que la clase 24 a pesar de ser la clase que tiene menor cantidad de casos y se encuentra desbalanceada con respecto a las demás se obtienen buenos resultados en ambas métricas.

En la Figura **4-6** se presenta la matriz de confusión correspondiente al clasificador ELM para las características extraídas mediante técnicas de deep learning. Puede observarse aquí que, en general, el clasificador obtiene buenos resultados ya que presenta una diferencia marcada en los valores concentrados en su diagonal por sobre los que se encuentran fuera de ella. Se observan algunos inconvenientes más altos para la clasificación en las clases 16, 17, 18 donde el número de patrones correctamente clasificados es menor, y el clasificador puede confundirse más repetidamente.

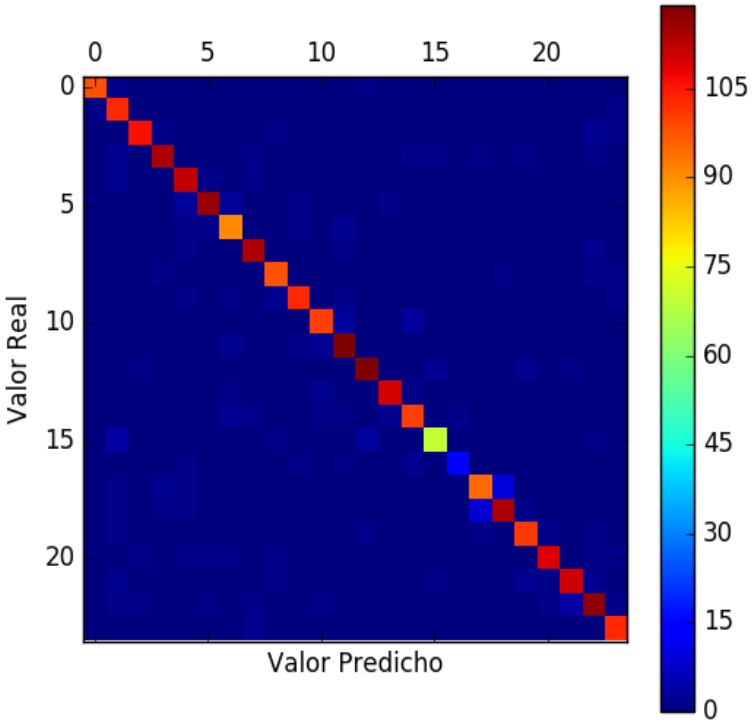


Figura 4-6: Matriz de confusión de ELM para características extraídas mediante deep learning.

En base a los experimentos realizados sobre ambas metodologías de extracción de características y los resultados obtenidos por los clasificadores postulados al aplicarlos sobre ellas se seleccionó la técnica de deep learning (CNN) para la extracción de características y ELM como el clasificador a implementar en la herramienta de cariotipado. En el próximo capítulo se describirá la herramienta, en su arquitectura y funcionalidades con las que se la equipo.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
Cristian Vizarri, C. E. Martínez & M. Gerard; "Herramienta de código abierto para la construcción automática de cariogramas (Undergraduate project)"
Facultad de Ingeniería y Ciencias Hídricas -Universidad Nacional del Litoral, 2018.

Capítulo 5

CarioPyNet

5.1. Diseño de la herramienta

La herramienta está compuesta por el sistema que procesa las imágenes y las clasifica (back-end), desarrollado enteramente en Python3[1] y librerías relacionadas, y por la interfaz gráfica o de usuario (front-end) que muestra por pantalla la información, desarrollada empleando el Framework Electron.js[2]. El código fuente se encuentra disponible en:

https://gitlab.com/cristiaan3003/CaryoPyNet

A continuación se explicará cada una de las partes con más detalle.

5.1.1. Front-End

En el diseño se buscó que el usuario final pueda usarla sin dificultades. Fue desarrollada en Electron.js, una plataforma para desarrollar aplicaciones de escritorio usando tecnologías web (HTML, CSS y JavaScript) creada y mantenida por Github[3]. Electron.js funciona creando dos tipos de procesos: main y renderer. El primero (main) es un proceso de Node.js[4] y es el proceso más importante, ya que tiene acceso a varias APIs de Electron.js que ayudan a comunicarse directamente con el SO y realizar distintas acciones. El segundo (renderer) es un proceso de Chromium[5], el navegador de web de código abierto del que Google Chrome obtiene su código de fuente. Electron.js utiliza un Chromium embebido para mostrar páginas web. Cada *"página web"* en Electron.js se ejecuta en su propio proceso, que se denomina *"proceso de renderizado"*.

[1] https://www.python.org/

[2] https://electronjs.org/

[3] https://github.com

[4] entorno de ejecución para JavaScript construido con el motor de JavaScript del navegador web Google Chrome

[5] https://www.chromium.org/

5.1.2. Back-End

En el desarrollo de la parte lógica del back-end se utilizó Python junto con librerías externas que extienden las funcionalidades. Estas librerías se utilizaron como apoyo para construir el modelo de la red convolucional y para el preprocesado de las imágenes.

- Modelo de la Red Neuronal Convolucional

Las dos librerías principales utilizadas para construir la red neuronal convolucional son Nolearn[6] y Lasagne[7]. Nolearn es un wrapper para Lasagne, el cual hace que el código sea más legible, fácil de escribir y administrar. Lasagne es una librería de alto nivel enfocada en Deep Learning, y se apoya en el uso de Theano[8]. Ésta abstrae el código y los complejos cálculos de Theano proporcionando una interfaz más simple de programación. Por su parte, Theano es una librería desarrollada en Python, que brinda al usuario funcionalidades para definir, optimizar, y evaluar expresiones matemáticas, en especial aquellas que involucran arreglos multidimensionales. Al utilizar Theano es posible aumentar la velocidad de ejecución del software desarrollado solamente empleando Python.

- Librería de procesamiento de imágenes

Para el procesamiento de imágenes se decidió utilizar la biblioteca OpenCV[9] en su versión 3 dado que es completamente Open Source, multiplataforma, tiene una gran comunidad con muchos años de desarrollo y provee funcionalidades como por ejemplo: procesamiento y análisis de imágenes, estadísticas, calibración de la cámara, seguimiento de objetos y algoritmos de flujo óptico, detección de texto, segmentación, operaciones morfológicas, suficientes para el presente proyecto.

Para lograr que la red neuronal convolucional de la herramienta sea capaz de identificar cromosomas, se requiere previamente configurar distintas partes del sistema llamadas **"módulos"**. Estos *"módulos"* constituyen el corazón o back-end de la herramienta y cada uno de ellos tiene una funcionalidad diferente. El resultado que producen es tomado por el front-end y mostrado al usuario. La Figura **5-1** presenta una descripción de los módulos que componen la herramienta. El módulo principal es el de *"Construcción del cariograma"*. Este es el que usualmente utilizará el usuario, ya que es el que le permitirá construir el cariograma a partir de un caso de estudio. Toma relevancia porque es un módulo integrador, ya que se vale de elementos precalculados por los otros módulos que fueron previamente ejecutados y/o configurados.

El primer paso para configurar la herramienta consiste en la ejecución del módulo denominado *"Enderezado de cromosomas"*. El objetivo del mismo es enderezar las imágenes

[6]https://github.com/dnouri/nolearn
[7]https://lasagne.readthedocs.io/en/latest/
[8]http://deeplearning.net/software/theano/
[9]https://opencv.org/

del conjunto de entrenamiento capturadas por el microscopio que pueden tener distintas curvaturas y orientaciones. A continuación se debe ejecutar el módulo de *"Almacenado de imágenes de cromosomas"*. El objetivo de este módulo es procesar el conjunto de imágenes 2D de los cromosomas enderezados y transformarlas en un vector de datos manejables por los algoritmos desarrollados. El siguiente módulo, denominado *"Entrenamiento de la Red Convolucional"*, se encarga del ajuste de la red convolucional. Esta red será la encargada de aprender, a partir de las imágenes cromosómicas, aquellas características más relevantes que permitan identificar a cada cromosoma. El próximo módulo, denominado *"Extracción de características"*, aplica la red convolucional entrenada sobre las imágenes de los cromosomas para obtener un listado de descriptores de cada imagen, que son empleados por el módulo final para clasificar los cromosomas. Finalmente, el módulo de *"Entrenamiento del clasificador"*, toma como entrada las características extraídas y ajusta los parámetros del modelo para clasificar correctamente las imágenes.

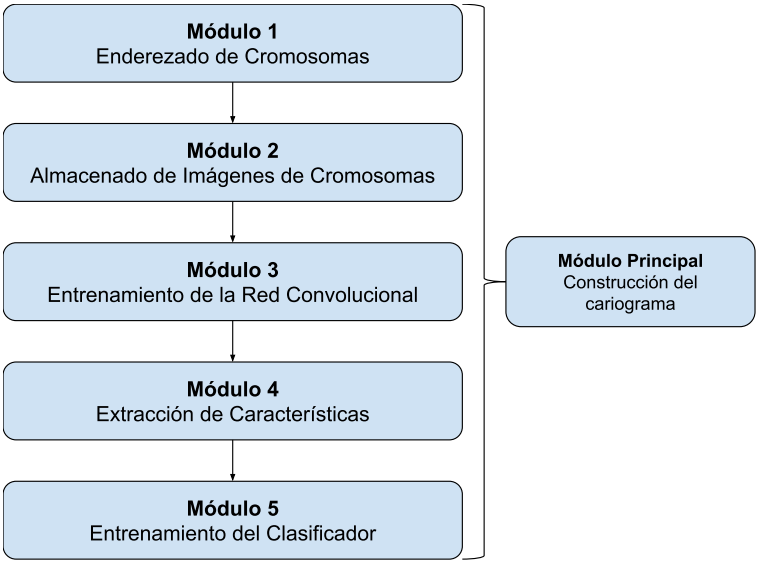


Figura 5-1: Módulos del back-end.

5.1.3. Estructura de las carpetas de la herramienta

La estructura de desarrollo de la herramienta en Electron.js, es una estructura que se compone de 6 carpetas en el directorio raíz (Figura **5-2**). En la figura se observa que la herramienta se encuentra dentro de la carpeta raíz *"CarioPyNet"*, en la cual se encuentran los distintos directorios. La carpeta *"data"* contiene la información referente a la red neuronal, y almacena los cromosomas y las características extraídas. La carpeta *"menu"* contiene el diseño y diagramación de la barra de menú de la herramienta. La carpeta *"node_modules"*, almacena la librería Electron.js. La carpeta *"plugins"* almacena las hojas de estilo css de la herramienta, funciones JavaScript externas a la librería Electron.js, e imágenes empleadas en el front-end. En *"pycalc"* se almacena el código Python que aporta la lógica de la herramienta. Finalmente, en *"tmp"* se almacenan archivos temporales generados por la herramienta.

Se expone a continuación la estructura de carpetas y la organización que deben poseer

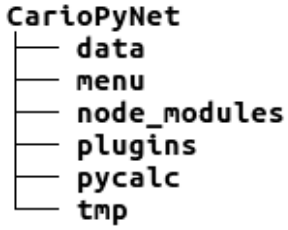


Figura 5-2: Estructura de las carpetas de la herramienta.

las imágenes de entrenamiento para ser procesadas por la herramienta. Se debe disponer de una carpeta independiente que contenga e identifique a cada caso de estudio de forma unívoca. En la Figura **5-3** se puede observar un ejemplo de esta estructura. El directorio raíz de la carpeta ***"cromosomas_entrenamiento"*** que contiene los casos de estudio que se disponen. Ejemplo: 97002023.12.tiff, 97002023.13.tiff, 97002023.16.tiff, etc.

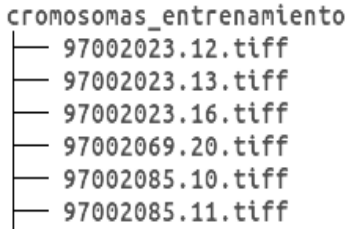


Figura 5-3: Estructura de carpetas de imágenes de entrenamiento.

Por último se detalla la estructura interna que debe posee cada una de las carpetas que identifica los casos de estudio. Se explica con un ejemplo para el caso de estudio *"97002023.12.tiff"*. Dentro de la carpeta que identifica al caso se deberá tener una carpeta identificada por clase de cromosomas, iniciando en la *"clase 1"*. Cada una contendrá los cromosomas de dicha clase que le pertenezcan al caso de estudio. Se puede observar en la Figura **5-4** un ejemplo gráfico de la estructura.

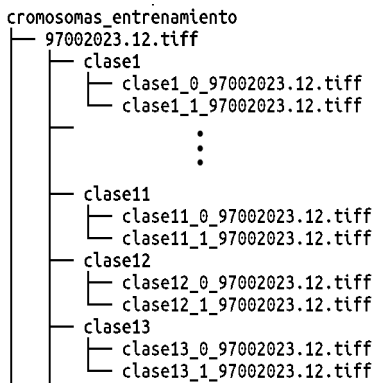


Figura 5-4: Organización internas de las carpetas de los casos de estudio.

5.2. Instalación y prerrequisitos

5.2.1. Entornos Linux en general

Para utilizar la herramienta deben cumplirse algunos requisitos previos de librerías y paquetes que deben instalarse con anterioridad a su ejecución. Estos paquetes y librerías no son muy exigentes con las versiones a instalar. La única excepción a esta regla es Lasagne. Debido a su estrecho acoplamiento con Theano, se requiere que la versión de Theano a instalar sea la requerida por Lasagne. Las siguientes instrucciones suponen que se está ejecutando un sistema Linux o Mac y son genéricas.

En cuanto a los requerimientos de Hardware, la herramienta requiere mínimo 4Gb de RAM y 15 Gb de espacio libre en el disco duro. Aunque es recomendable 6 Gb de RAM o superior, 30 Gb o más de espacio libre en el disco duro y una configuración de pantalla con resolución 1366 x 768 o superior.

Requisitos previos de Software

Descripción General

Las implementaciones se realizaron empleando el lenguaje de programación Python, utilizando las librerías SciKit-Learn [43] y SciPy [25, 70] en una distribución derivada del sistema operativo Debian. Se expone a continuación el esquema de las bibliotecas y dependencias de prerrequisitos[10] a instalar antes de ejecutar la herramienta (Figura **5-5**). El esquema muestras dos grupos. Las dependencias *"de Sistema"* son aquellas a instalar directamente en el sistema anfitrión y que deben estar disponibles en su línea de comandos, y las *"de Python"* que son aquellas librerías y bibliotecas que dependen directamente de Python.

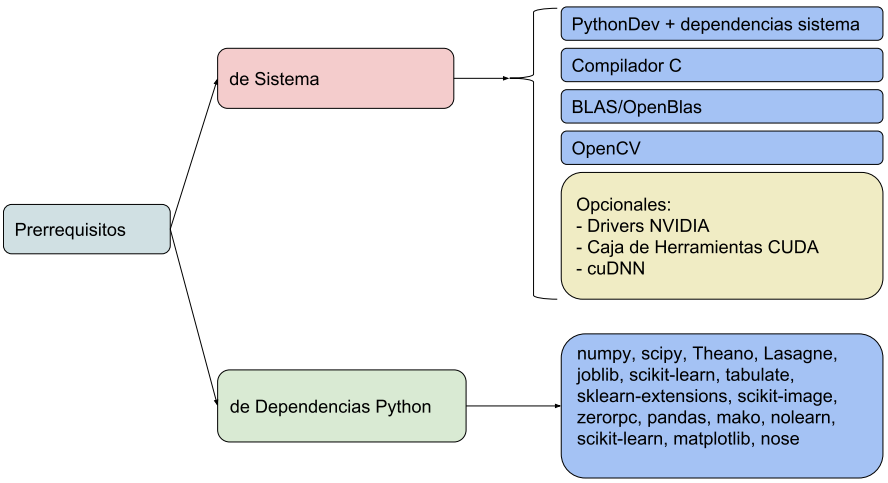


Figura 5-5: Dependencias y bibliotecas requeridas.

[10]Los prerrequisitos están orientados hacia distribuciones Debian o Debian derivadas. Puede que existan pequeñas variaciones en otras distribuciones.

A continuación, se realiza una explicación detallada de los requerimientos para la instalación de las dependencias de cada uno de los puntos expuestos en la Figura **5-5**.

- **PythonDev y Dependencias del sistema**

Se requiere que tenga instalado Python2 y Python3. Debe verificar que los binarios de Python2 y Python3 se encuentren en ***/usr/bin/python2*** y ***/usr/bin/python3*** respectivamente. Adicionalmente chequear que la versión de Python se corresponda con el path donde está alojado el binario con ***python2 -V***, ***python3 -V***. Python2 es requerido obligatoriamente para compilar el complemento de *zerorpc* que es almacenado por Electron.js.

De las dependencias *"de Python"* existen un listado que es necesario que estén instaladas en el sistema que son utilizadas internamente por la herramienta (Tabla **5-1**).

Tabla 5-1: Dependencias que deben estar instaladas en el sistema. El detalle de las versiones está disponible en el apéndice A.

python3-numpy	libtbb-dev	gfortran	libavformat-dev
python3-scipy	libpng-dev	libzmq3	libswscale-dev
python3-dev	curl	libgconf	qt5-default
python3-pip	cython	build-essential	qtbase5-dev
python3-nose	g++	cmake	libopenblas-dev
python3-tk	libtbb2	libgtk	liblapack-dev
libjpeg-dev	libblas-dev	pkg-config	
libtiff5-dev	libatlas	libavcodec-dev	

Es necesario instalar en el sistema un Compilador C, el cual es usado por Theano y OpenCV. En Linux, el compilador predeterminado es generalmente gcc. Se recomienda instalarlo a través del administrador de paquetes de su sistema operativo.

Para el manejo de las imágenes instalar la librería de procesamiento de imágenes OpenCV. Esta librería debe ser bajada y compilada. Luego de la descarga, los pasos de instalación de la misma varían dependiendo de la distribución y versión de sistema operativo que se use. En párrafos posteriores se dará una explicación detallada para su instalación en el sistema operativo Ubuntu 16.04.

Para optimizar los cálculos algebraicos de las implementaciones de Numpy y Scipy se debe instalar una biblioteca BLAS (Subprogramas básicos de Álgebra lineal), la cual proporciona rutinas rápidas de álgebra lineal para el CPU. Esta genera mayores beneficios en las implementaciones en CPU. En este desarrollo particular se decidió utilizar OpenBLAS, una implementación libre de BLAS. Es una librería libre para operaciones multi-hilo en CPU, y aumenta la velocidad de los algoritmos brindando capacidades multi-hilo en un CPU.

Al instalar OpenBLAS se debe tener en cuenta que, si se instalaron Numpy y Scipy a través del administrador de paquetes del sistema operativo, este va enlazar a la biblioteca OpenBLAS instalada en su sistema automáticamente. Si se instaló Numpy y Scipy a través

de pip, deben tenerse instalados los encabezados de desarrollo de la biblioteca BLAS (por ejemplo, en Debian/Ubuntu es el paquete libopenblas-dev) antes de ejecutar el comando de instalación. Para una guía detallada de instalación en los distintos sistemas operativos soportados dirigirse al sitio oficial http://www.openblas.net/.

Para el soporte de la red neuronal convolucional es requerido instalar Theano. En este desarrollo Theano no es usado de forma directa. Sino cómo dependencia de Lasagne y Nolearn, los cuales lo utilizan para sus cálculos internos. Debe recordarse que, la versión para instalar depende de la versión de Lasagne que se elija, por lo que se tratará a continuación. Lasagne es una librería de alto nivel orientada a deep learning. Para este desarrollo utilizó el paquete Lasagne versión 0.2.dev1, el cual requiere Theano 0.8. Una versión aún más reciente de Theano puede que funcione correctamente, pero podría existir incompatibilidad. Se recomienda instalarlo en su directorio de inicio, para ello agregue ***–user*** al comando del gestor de paquetes python cuando ejecute la instalación. Para actualizar desde una instalación anterior, agregue ***–update***. Por otro lado Nolearn es otra librería orientada a deep learning pero que se integra a Lasagne aportando funciones de alto nivel que facilitan algunas tareas.

Si bien no estaba dentro de los objetivos de este proyecto ofrecer soporte de GPU[11] a los algoritmos para el entrenamiento y recuperación, es posible dar a la herramienta dicho soporte si el usuario desea activarlo. Lasagne admite de forma transparente el entrenamiento de las redes en una GPU, que puede ser de 10 a 50 veces más rápido el entrenamiento que una CPU. Esto requiere una GPU NVIDIA con soporte ***CUDA***®. Los respectivos driver para las distintas GPU y sistemas operativos soportados están disponibles en http://www.nvidia.com/Download/index.aspx?lang=es. Por otro lado, se debe instalar la caja de herramientas ***CUDA***®. ***CUDA*** son las siglas de ***C***ompute ***U***nified ***D***evice ***A***rchitecture (Arquitectura Unificada de Dispositivos de Cómputo). Es una arquitectura de cálculo paralelo de NVIDIA[12] que aprovecha la gran potencia de la GPU para proporcionar un incremento extraordinario del rendimiento del sistema. Dichos controladores se encuentran disponibles en el sitio web de NVIDIA. Una vez instalados, se debe verificar que ***/usr/local/cuda/bin*** esté en su PATH, y probar por consola que el comando ***nvcc –version*** funciona correctamente. También se debe verificar que /usr/local/cuda/lib64 esté en su LD_LIBRARY_PATH, para que las bibliotecas del kit de herramientas puedan ser accedidas desde la línea de comandos. Este desarrollo se implementó con CUDA versión 8.0. Para habilitar a Theano usar la GPU de forma automática, se debe crear un archivo con nombre .theanorc en el directorio personal del usuario de su sistema. Por ejemplo, si el usuario de sistema es ***"nautilus"*** el archivo debe crearse en ***/home/naultilus***, con el siguiente contenido:

```
[global]
floatX = float32
device = gpu
```

[11] Unidades de procesamiento gráfico.
[12] https://developer.nvidia.com/cuda-downloads

Si el archivo ya existe con anterioridad para su uso en CPU, sólo debe modificarse el parámetro *"device=cpu"* por *"device=gpu"*.

Adicionalmente a los driver NVIDIA CUDA, es recomendable instalar la biblioteca NVIDIA CUDA® Deep Neural Network (cuDNN), la cual es una biblioteca con soporte para GPU de operaciones altamente optimizadas sobre redes neuronales disponible en

https://developer.nvidia.com/cudnn, que mejoran aún más el rendimiento de ejecución sobre GPU. Se requiere una GPU con Compute Capability 3.0 o superior. Para instalarse deben copiarse los archivos *.h a ***/usr/local/cuda/include*** y los archivos lib* a ***/usr/local/cuda/lib64***.

Por otro lado, existen una serie de dependencias que no son requeridas obligatoriamente en el sistema y pueden ser instaladas mediantes el gestor de paquetes de Python (Tabla **5-2**).

Tabla 5-2: Dependencias que deben ser instaladas mediante **pip**. El detalle de las versiones está disponible en el apéndice A.

pandas	numpy	mako	Lasagne
sklearn-extensions	zerorpc	Theano	scipy
scikit-learn	tabulate	joblib	nose

Instalación y uso de la herramienta

En este punto se considera que se tienen instaladas correctamente todas las dependencias *"de Sistema"* y *"de Python"*. Para ejecutar la herramienta se debe posicionar dentro de la carpeta raíz de la misma. A modo de ejemplo, supondremos que esta carpeta se llama ***"CarioPyNet"***. Una vez dentro de CarioPyNet, descargar las dependencias JavaScript de Electron.js. Estas dependencias son totalmente independientes de las anteriores instaladas, y crearán una carpeta autónoma dentro de la misma carpeta raíz (CarioPyNet) que contiene el código fuente de la herramienta. Esta carpeta tendrá el fin de lanzar el framework Electron.js que utilizara el código desarrollado.

A tal fin se debe configurar con el comando ***export*** algunas variables temporales que deben estar disponibles para la instalación. En estas variables se definirá que se usará Electron (*npm_config_runtime*) y que versión del mismo (*npm_config_target*), desde donde se lo descargara (*npm_config_disturl*) y si se compilara desde el código fuente (*npm_config_build_from_source*).

: $ export npm_config_target=1.7.6 # electron version
: $ export npm_config_runtime=electron
: $ export npm_config_disturl=https://atom.io/download/electron
: $ export npm_config_build_from_source=true

Ahora se debe obtener los valores de configuración de las fuentes seteadas en las variables, por esto se debe ejecutar un *npm config*.

$ npm config ls

Por último se ejecutará un npm install y npm install zerorpc. El primer comando instala el paquete Electron.js y sus dependencias estándar en la carpeta local ***node_modules***, el segundo comando agrega la dependencia zerorpc requerida en este desarrollo por Ejectron.js a la carpeta ***node_modules***.

: $ npm install
: $ npm install zerorpc

Empezar a utilizar la herramienta

Para iniciar la aplicación se debe abrir una consola y posicionarse dentro la carpeta de la herramienta. Luego de esto ejecutar el comando:

"./node_modules/.bin/electron ."

Esto hará que se ejecute la herramienta y se abra la pantalla principal (Figura **5-6**). Si es la primera vez que se utiliza la herramienta, la misma debe configurarse antes de poder generar los cariogramas. Para dicha tarea se dispone de funcionalidades de configuración que ayudarán al usuario durante los distintos pasos. Tener en cuenta que antes de iniciar la configuración de los módulos debe contar con una base de datos de imágenes individuales de cromosomas en estado metafásico y organizadas en la estructura definidas en la Figuras **5-3** y **5-4**.

Figura 5-6: Pantalla principal.

Pasos de configuración:

Se recomienda configurar los módulos de la herramienta en el orden que se indicará a continuación. Siguiendo estos pasos preestablecidos la herramienta se configurará con sus parámetros estándares en base a las pruebas preliminares realizadas.

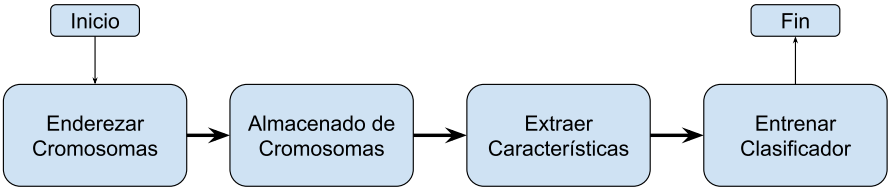


Figura 5-7: Pasos de configuración.

Paso 1: Enderezado de cromosomas

En la barra superior del menú dirigirse a *"Configuración"* y dentro de la misma a *"Enderezar Cromosomas"*. Se mostrará una pantalla cómo la Figura **5-8**. En *"Seleccionar Cromosomas"* se debe seleccionar la carpeta que contiene las imágenes de los cromosomas. Esta carpeta debe contener las subcarpetas con cada uno de los casos de estudio (individuos), y las imágenes de cada uno de los cromosomas separadas por tipo. En *"Salida"*, se debe seleccionar una carpeta vacía en donde se almacenarán las imágenes de los cromosomas enderezados. Una vez completado estos dos requerimientos, presionar el botón *"Enderezar"* y dejar que la herramienta realice el proceso de enderezado de cromosomas. Este proceso puede requerir varias horas, dependiendo de la potencia de cálculo del dispositivo, memoria disponible y cantidad de cromosomas a enderezar.

Figura 5-8: Enderezar cromosomas.

Paso 2: Almacenado de Cromosomas

Debe dirigirse a la barra de menú, opción "Configuración" y dentro de la misma a *"Almacenar Cromosomas"* (Figura **5-9**). En *"Cromosomas"* seleccionar la carpeta que contiene las imágenes de los cromosomas enderezados. Esta carpeta debe contener las subcarpetas con cada uno de los casos de estudio, y las imágenes de los cromosomas separadas por tipo. Es importante que estas imágenes estén enderezadas mediante el algoritmo del *"Paso 1"* y

no por otro algoritmo. Aunque algorítmicamente para la herramienta sea posible almacenar las imágenes de forma indistinta y un método de enderezado diferente puede otorgar un resultado visualmentente similar en el enderezado de cromosomas, no es posible asegurar que las características extraídas sobre las imágenes procesadas con un método de enderezado diferente tengan las mismas propiedades que las que extrae la herramienta con el método implementado en ella. Luego, los resultados de las métricas que se obtengan en la herramienta sobre el conjunto de características extraído desde cromosomas enderezados con un método distinto no estarían validados, tal como los resultados expuestos en este trabajo.

Luego verificar el checkbox *"Agregar al mismo archivo"*, esta opción permite agregar al fichero existente los datos de los nuevos cromosomas. Cuando esta opción se encuentra habilitada, las imágenes de los cromosomas son agregadas al final del fichero ya existente. Esta opción resulta útil para actualizar la base de datos. Se debe deshabilitar esta opción si desea borrar el fichero que almacena los cromosomas y crear un fichero nuevo con nuevas imágenes. Es importante tener en claro que se perderá la información del fichero existente. Finalmente, el botón *"Almacenar"* inicia el proceso de almacenado de las imágenes.

Figura 5-9: Almacenado de cromosomas.

Paso 3: Extraer características

Para acceder a esta pantalla dirigirse a *"Configuración"* opción *"Características"*. Aparecerá una pantalla como la que se observa en la Figura **5-10**. En ella se encontrara el botón *"Extraer"* y un checkbox que permite modificar los parámetros del extractor de características. Si se encuentra configurando la herramienta por primera vez o desea explícitamente modificar los parámetros del extractor de características debe tildar el checkbox *"Modificar parámetros del extractor de características"* que lo llevara a la siguiente pantalla (Figura **5-11**). Aquí se podrá entrenar y guardar el extractor de características.

Las opciones que presenta esta pantalla (Figura **5-11**) permitirán entrenar el extractor de características con los valores estándar predefinidos o bien realizar modificaciones sobre dichos valores y guardar el extractor entrenado. La opción *"Modificar parámetros"* habilita modificar los valores estándares predefinidos para el extractor. Para la *"Tasa de aprendizaje"* acepta valores decimales en el intervalo [0,001;5,0] y se sugiere 0,01. *"Momento"* acepta valores decimales en el intervalo [0,0;1,0] y se sugiere 0,01. *"Número de Épocas"* acepta valores enteros en el intervalo [1;100] y se sugiere 20. *"Guardar/Sobreescribir red entrenada"*

Figura 5-10: Extraer características.

activa o desactiva el grabado en disco de la red entrenada, *"Regresar"* permite regresar a la pantalla de *"Extraer características"* (Figura **5-10**), el botón *"Entrenar"* inicia el entrenamiento del extractor de características, y *"Exactitud"* indica la capacidad del método para asociar la clase correcta a cada patrón.

Figura 5-11: Ajustar extracción de características.

Paso 4: Entrenar clasificador

Para acceder a esta pantalla dirigirse a *"Configuración"* opción *"Entrenar ELM"*. Lo recibirá una pantalla como la que se observa en la Figura **5-12**. En ella se encontrara el botón *"Entrenar"* y un checkbox que permite modificar los parámetros del clasificador. Si se encuentra configurando la herramienta por primera vez o desea explícitamente modificar los parámetros del clasificador debe tildar este checkbox para acceder a la pantalla (Figura **5-13**) que le permitirá entrenar y guardar el clasificador, y adicionalmente también permitirá modificar los parámetros del mismo.

Las opciones que presenta esta pantalla (Figura **5-13**), permitirán entrenar el clasificador con los valores estándar predefinidos o bien realizar modificaciones sobre dichos valores, y guardar el clasificador entrenado. La opción *"Modificar parámetros"* habilita modificar los valores estándares predefinidos para el extractor. *"Número de neuronas"* debe ser un valor entero en el intervalo [5;5000] y se sugiere 2000. *"Alpha"* es un valor decimal en el intervalo [0,0; 1,0] y se sugiere 0,90. *"Función de Activación"* es un menú de selección mediante el cual permite seleccionar de un listado de funciones de activación disponibles, *"Guardar/Sobreescribir red entrenada"* activa o desactiva el grabado en disco de la red

Figura 5-12: Pantalla de inicio del proceso de entrenamiento del clasificador.

entrenada, el botón *"Entrenar"* inicia el entrenamiento del extractor de características, y *"Exactitud"* indica la capacidad del método para asociar la clase correcta a cada patrón.

Figura 5-13: Modificar parámetros del clasificador.

Los cuatro pasos anteriores descriptos permiten configurar los módulos internos de la herramienta de forma independiente. Si no se desea configurar módulo por módulo se incluye una funcionalidad llamada *"Configuración Simple"*, que centraliza la configuración de todos los pasos en una sola pantalla (Figura **5-14**). Esta aporta una mayor simplicidad a la configuración inicial de la herramienta o si se desea reinicia la configuración existente por completo. Se debe tener en cuenta que el proceso de *"Configuración Simple"* incluye todos los módulos del sistema, lo cuales pueden tardar un tiempo considerable en terminar todas la tareas; y si el proceso se ve interrumpido por alguna razón deberá iniciarlo nuevamente desde el principio.

Figura 5-14: Configuración simple.

Luego de realizar los pasos de configuración la herramienta se encuentra lista para su uso. Para construir el cariograma dirigirse a la pantalla principal (Figura **5-6**) y en *"Cromosomas a clasificar"* debe seleccionar la carpeta que contiene las imágenes de los cromosomas del caso de estudio del cual se desea obtener el cariograma. Posteriormente presionar el botón *"Generar cariograma"* y esperar que el proceso de la herramienta termine (Figura **5-15**).

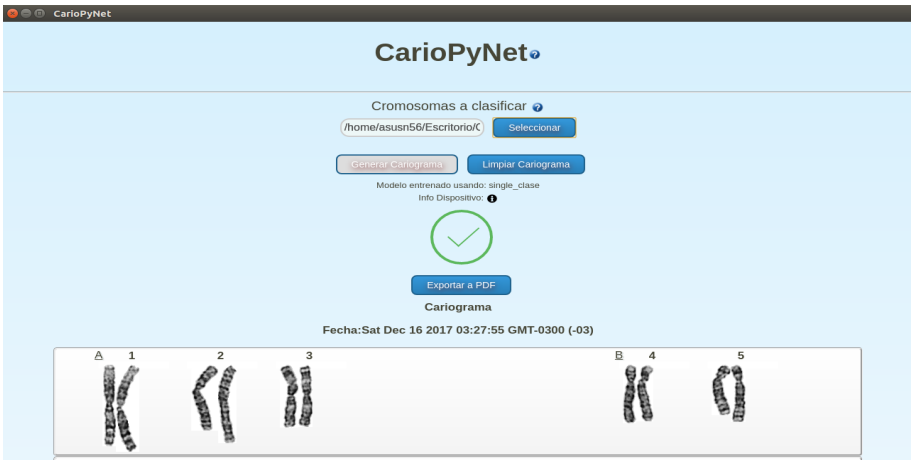


Figura 5-15: Pantalla parcial de la herramienta luego de obtener el cariograma.

Al obtener el resultado se visualizará el botón extra *"Exportar a PDF"* que permitirá descarga a un archivo PDF el cariograma obtenido (Figura **5-16**). Para realizar el proceso nuevamente, debe limpiar el cariograma actual con el botón *"Limpiar Cariograma"*.

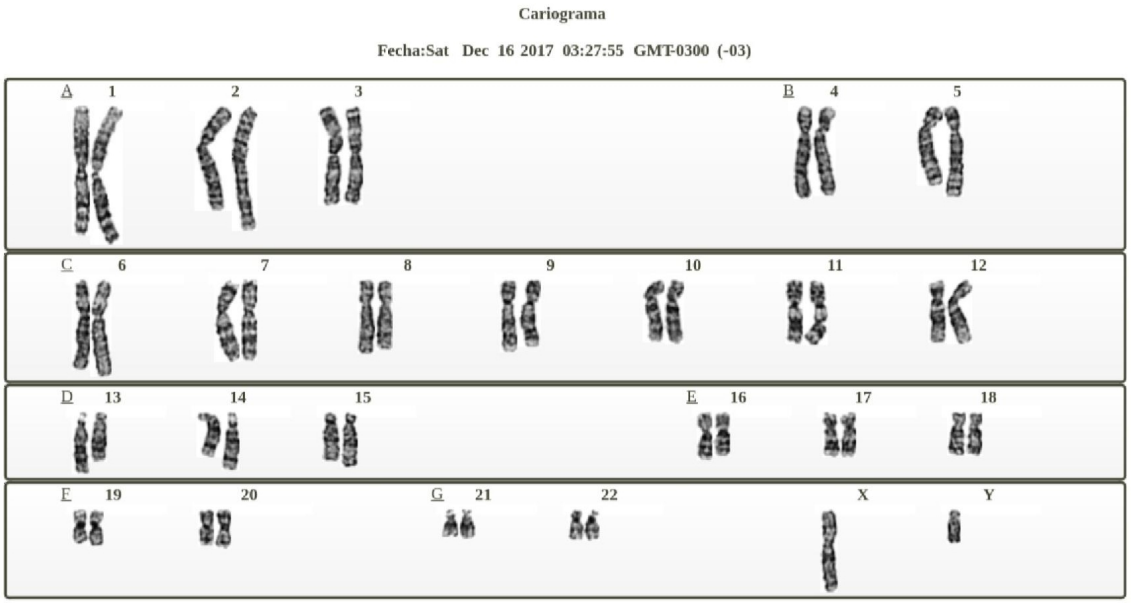


Figura 5-16: Cariograma exportado a PDF.

Capítulo 6

Conclusiones y trabajos futuros

Esta última parte del trabajo está destinada a expresar las conclusiones finales respecto a los resultados obtenidos en la etapa de experimentos, desempeño de la herramienta, las funcionalidades realizadas, y las posibles mejoras a realizarse en un futuro.

6.1. Conclusiones

En este trabajo se abordó el problema construcción de una herramienta para la generación automática de cariogramas a partir de imágenes de cromosomas individuales. Esto implicó primero generar los algoritmos para generar el cariograma. Posteriormente, resuelto este problema, se afrontó el desarrollo de la herramienta de software que integre los algoritmos desarrollados. Se busca en ella una manera de posibilitar un software basado en su totalidad librerías algorítmicas de código abierto, que presente una alternativa a los desarrollos privativos que tienen un gran costo de adquisición.

La primera etapa consistió en extraer las características relevantes de las imágenes que permita identificarlas. En este caso la información relevante a extraer fueron las características de los cromosomas. Para realizar esto se analizaron dos metodologías para la extracción de características, un *enfoque estándar* y un enfoque basado en aprendizaje profundo. En una segunda etapa, se realizó la implementación de la herramienta y se aplicó extracción de características con CNN y el clasificador ELM en el back-end; y para al front-end una interfaz basada en tecnología web que permite ayudar en la configuración que requieren los algoritmos y obtener el cariograma final visualizándolo en la misma interfaz.

Se logró obtener una herramienta OpenSource que brinda apoyo a la construcción de cariogramas a expertos, que aplicó la combinación de extraer las características mediante una CNN y clasificó las mismas con un ELM obteniendo un UAR que supero el 0,90 con un buen margen. La herramienta posibilita la edición del cariograma construido y la generación de informes en ficheros de tipo PDF. Todo esto obtenido con bibliotecas OpenSource, documentadas y disponibles en la web, lo que permite a cualquier individuo si así lo quiere modificar la herramienta de acuerdo a sus necesidades particulares.

Tener en cuenta que debido al volumen de información requerido para el entrenamiento de la redes CNN se trabajó con la base de datos en banda G. Si quiere usar la herramienta ya entrenada, tiene que tener la precaución de pasarle las imágenes con la tinción correcta o entrenarla desde cero con las imágenes correspondientes.

6.2. Trabajos futuros

Se incluyen a continuación posibles aspectos a mejorar y/o agregar en la herramienta, alusivas a sus funcionalidades y funcionamiento interno.

- Explorar el comportamiento de la herramienta modificándola para que esté totalmente basada en una CNN, sin intervención del preprocesamiento de enderezado de cromosomas. Para esto, profundizar el estudio de la configuración de la CNN como extractor de características.
- Agregar la posibilidad de detección temprana de anomalías numéricas y estructurales en la dotación cromosómica de un individuo (cromosomopatías).
- Agregar el recorte automático de cromosomas a partir de las imágenes en estado metafásico obtenidas desde el microscopio.
- Evaluar el desarrollo de una versión completamente web y una móvil.

Bibliografía

[1] Agam, G ; Dinstein, I: Geometric separation of partially overlapping nonrigid objects applied to automatic chromosome classification. En: *IEEE transactions on pattern analysis and machine intelligence* 19 (1997), November, Nr. 11, p. 1212–1222. – ISSN 0162–8828

[2] Badawi, A M. ; Hasan, K G. ; Aly, E E A. ; Messiha, R A.: Chromosomes classification based on neural networks, fuzzy rule based, and template matching classifiers. En: *2003 46th Midwest Symposium on Circuits and Systems* Vol. 1, 2003. – ISSN 1548–3746, p. 383–387 Vol. 1

[3] Bickmore, Wendy A.: Karyotype Analysis and Chromosome Banding. En: *eLS*. John Wiley & Sons, Ltd, 2001. – ISBN 9780470015902

[4] Bishop, Christopher: *Pattern Recognition and Machine Learning*. 1. Springer-Verlag New York. – ISBN 9780387310732

[5] Boureau, Y-Lan ; Ponce, Jean ; LeCun, Yann: A theoretical analysis of feature pooling in visual recognition. En: *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, p. 111–118

[6] Breiman ; Friedman ; Stone ; Olshen ; Hall/CRC, Chapman A. (Ed.): *Classification and Regression Trees*. [Wiley, International Biometric Society], 1984. – ISBN 9780412048418

[7] Breiman, Leo: *Random Forests*. Statistics Department, University of California, Berkeley, 2001. – 5–32 p. ISSN 0885–6125

[8] Chen, J C. ; Patel, V M. ; Chellappa, R: Unconstrained face verification using deep CNN features. En: *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016, p. 1–9

[9] Cho, J M.: Chromosome classification using backpropagation neural networks. En: *IEEE engineering in medicine and biology magazine: the quarterly magazine of the Engineering in Medicine & Biology Society* 19 (2000), Januar, Nr. 1, p. 28–33. – ISSN 0739–5175

[10] Cortes, Corinna ; Vapnik, Vladimir: Support-Vector Networks. En: *Machine Learning* 20 (1995), Sep, Nr. 3, p. 273–297. – ISSN 1573–0565

[11] Curtis, H ; Sue Barnes, N ; Schnek, A ; Massarini, A B.: Séptima edición en espanol. En: *Santiago de Chile, Chile. Editorial Médica Panamericana* (2008)

[12] Dieleman, Sander ; Schlüter, Jan ; Raffel, Colin ; Olson, Eben ; Sønderby, Søren K. ; Nouri, Daniel ; Maturana, Daniel ; Thoma, Martin ; Battenberg, Eric ; Kelly, Jack ; Fauw, Jeffrey D. ; Heilman, Michael ; diogo149 ; McFee, Brian ; Weideman, Hendrik ; takacsg84 ; peterderivaz ; Jon ; instagibbs ; Rasul, Dr. K. ; CongLiu ; Britefury ; Degrave, Jonas. *Lasagne: First release.* Url:https://doi.org/10.5281/zenodo.27878. August 2015

[13] García, Cristina U. ; Rubio, Antonio B. ; Pérez, Fabián A. ; Hernández, Francisco S.: A curvature-based multiresolution automatic karyotyping system. En: *Machine vision and applications* 14 (2003), 1 Juli, Nr. 3, p. 145–156. – ISSN 0932–8092, 1432–1769

[14] Graham, J ; Piper, J: Automatic karyotype analysis. En: *Methods in molecular biology* 29 (1994), p. 141–185. – ISSN 1064–3745

[15] Gu, Jiuxiang ; Wang, Zhenhua ; Kuen, Jason ; Ma, Lianyang ; Shahroudy, Amir ; Shuai, Bing ; Liu, Ting ; Wang, Xingxing ; Wang, Li ; Wang, Gang ; Cai, Jianfei ; Chen, Tsuhan: Recent Advances in Convolutional Neural Networks. (2015), 22 Dezember

[16] Hinton, Geoffrey E. ; Srivastava, Nitish ; Krizhevsky, Alex ; Sutskever, Ilya ; Salakhutdinov, Ruslan R.: Improving neural networks by preventing co-adaptation of feature detectors. (2012), 3 Juli

[17] Huang, Guang-Bin ; Chen, Lei: Convex incremental extreme learning machine. En: *Neurocomputing* 70 (2007), 1 Oktober, Nr. 16, p. 3056–3062. – ISSN 0925–2312

[18] Huang, Guang-Bin ; Chen, Lei ; Siew, Chee-Kheong: Universal approximation using incremental constructive feedforward networks with random hidden nodes. En: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 17 (2006), Juli, Nr. 4, p. 879–892. – ISSN 1045–9227

[19] Huang, Guang-Bin ; Zhu, Qin-Yu ; Siew, Chee-Kheong: Extreme learning machine: Theory and applications. En: *Neurocomputing* 70 (2006), 1 Dezember, Nr. 1, p. 489–501. – ISSN 0925–2312

[20] Irshad, Humayun ; Veillard, Antoine ; Roux, Ludovic ; Racoceanu, Daniel: Methods for nuclei detection, segmentation, and classification in digital histopathology: a review-current status and future potential. En: *IEEE reviews in biomedical engineering* 7 (2014), p. 97–114. – ISSN 1937–3333, 1941–1189

[21] Japkowicz, Nathalie ; Shah, Mohak: *Evaluating Learning Algorithms: A Classification Perspective.* Cambridge University Press, 17 Januar 2011. – ISBN 9781139494144

[22] Ji, L: Fully automatic chromosome segmentation. En: *Cytometry* 17 (1994), 1 November, Nr. 3, p. 196–208. – ISSN 0196–4763

[23] Ji, Liang: Intelligent splitting in the chromosome domain. En: *Pattern recognition* 22 (1989), 1 Januar, Nr. 5, p. 519–532. – ISSN 0031–3203

[24] Johnston, D A. ; Tang, K S. ; Zimmerman, S: Band features as classification measures for G-banded chromosome analysis. En: *Computers in biology and medicine* 23 (1993), März, Nr. 2, p. 115–129. – ISSN 0010–4825

[25] Jones, Eric ; Oliphant, Travis ; Peterson, Pearu: {*SciPy*}*: Open source scientific tools for* {*Python*}. 2001. – url: http://www.scipy.org

[26] Joshi, Prachi ; Munot, Mousami ; Kulkarni, Parag ; Joshi, Madhuri: Efficient karyotyping of metaphase chromosomes using incremental learning. En: *IET Science, Measurement & Technology* 7 (2013), 1 September, Nr. 5, p. 287–295. – ISSN 1751–8830, 1751–8830

[27] Katz, S W. ; Brink, A D.: Segmentation of chromosome images. En: *1993 IEEE South African Symposium on Communications and Signal Processing*, 1993, p. 85–90

[28] Keller, J M. ; Gader, P ; Sjahputera, O ; Caldwell, C W. ; Huang, H-M T.: A fuzzy logic rule-based system for chromosome recognition. En: *Proceedings Eighth IEEE Symposium on Computer-Based Medical Systems*, IEEE Comput. Soc. Press. – ISBN 9780818671173, p. 125–132

[29] Khan, Waseem: Image Segmentation Techniques: A Survey. En: *Journal of Image and Graphics* 1 (2013), p. 166–170. – ISSN 2301–3699

[30] Kou, Zhenzhen ; Ji, Liang ; Zhang, Xuegong: Karyotyping of comparative genomic hybridization human metaphases by using support vector machines. En: *Cytometry* 47 (2002), 1 Januar, Nr. 1, p. 17–23. – ISSN 0196–4763

[31] Kuhn, H W. ; Tucker, A W.: Nonlinear Programming. En: *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, The Regents of the University of California, 1951. – ISSN 0097–0433

[32] Lee, Eun J. ; Ko, Byoung C. ; Nam, Jae-Yeal: Recognizing pedestrian's unsafe behaviors in far-infrared imagery at night. En: *Infrared Physics & Technology* 76 (2016), 1 Mai, p. 261–270. – ISSN 1350–4495

[33] Lerner, B: Toward a completely automatic neural-network-based human chromosome analysis. En: *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics: a publication of the IEEE Systems, Man, and Cybernetics Society* 28 (1998), Nr. 4, p. 544–552. – ISSN 1083–4419

[34] Lerner, B ; Guterman, H ; Dinstein, I: A classification-driven partially occluded object segmentation (CPOOS) method with application to chromosome analysis. En: *IEEE transactions on signal processing: a publication of the IEEE Signal Processing Society* 46 (1998), Oktober, Nr. 10, p. 2841–2847. – ISSN 1053–587X

[35] Loganathan, E ; Anuja, M R. ; Madian, N: Analysis of human chromosome images for the identification of centromere position and length. En: *2013 IEEE Point-of-Care Healthcare Technologies (PHT)*, 2013. – ISSN 2377–5262, p. 314–317

[36] Markou, Christoforos ; Maramis, Christos ; Delopoulos, Anastasios ; Daiou, Chrysa ; Lambropoulos, Alexandros: Automatic Chromosome Classification using Support Vector Machines. En: *Pattern Recognition: Methods and Applications*. 2012. – ISBN 978–1–477554–821

[37] Martínez, César ; García, Héctor ; Juan, Alfons ; Casacuberta, Francisco: Chromosome Classification Using Continuous Hidden Markov Models. En: *Pattern Recognition and Image Analysis*, Springer, Berlin, Heidelberg, 4 Juni 2003 (Lecture Notes in Computer Science). – ISBN 9783540402176, 9783540448716, p. 494–501

[38] Martínez, César ; Juan, Alfons ; Casacuberta, Francisco: Iterative Contextual Recurrent Classification of Chromosomes. En: *Neural Processing Letters* 26 (2007), 1 Dezember, Nr. 3, p. 159–175. – ISSN 1370–4621, 1573–773X

[39] Minaee, S ; Fotouhi, M ; Khalaj, B H.: A geometric approach to fully automatic chromosome segmentation. En: *2014 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, 2014, p. 1–6

[40] Mohammadi, Mohammaed R.: Accurate localization of chromosome centromere based on concave points. En: *Journal of medical signals and sensors* 2 (2012), April, Nr. 2, p. 88–94. – ISSN 2228–7477

[41] Moradi, M ; Setarehdan, S K. ; Ghaffari, S R.: Automatic locating the centromere on human chromosome pictures. En: *16th IEEE Symposium Computer-Based Medical Systems, 2003. Proceedings.*, 2003, p. 56–61

[42] Nouri, Daniel: *nolearn: scikit-learn compatible neural network library*. 2014. – url: https://github.com/dnouri/nolearn

[43] Pedregosa, Fabian ; Varoquaux, Gaël ; Gramfort, Alexandre ; Michel, Vincent ; Thirion, Bertrand ; Grisel, Olivier ; Blondel, Mathieu ; Prettenhofer, Peter ; Weiss, Ron ; Dubourg, Vincent ; Vanderplas, Jake ; Passos, Alexandre ; Cournapeau, David ; Brucher, Matthieu ; Perrot, Matthieu ; Duchesnay, Édouard: Scikit-learn: Machine Learning in Python. En: *Journal of machine learning research: JMLR* 12 (2011), Nr. Oct, p. 2825–2830. – ISSN 1532–4435, 1533–7928

[44] Piper, J ; Granum, E: On fully automatic feature measurement for banded chromosome classification. En: *Cytometry* 10 (1989), Mai, Nr. 3, p. 242–255. – ISSN 0196–4763

[45] Piper, J ; Granum, E ; Rutovitz, D ; Ruttledge, H: Automation of chromosome analysis. En: *Signal processing* 2 (1980), 1 Juli, Nr. 3, p. 203–221. – ISSN 0165–1684

[46] Popescu, M ; Gader, P ; Keller, J ; Klein, C ; Stanley, J ; Caldwell, C: Automatic karyotyping of metaphase cells with overlapping chromosomes. En: *Computers in biology and medicine* 29 (1999), Januar, Nr. 1, p. 61–82. – ISSN 0010–4825

[47] Qiang Wu ; Zhongmin Liu ; Chen, T ; Zixiang Xiong ; Castleman, K R.: Subspace-based prototyping and classification of chromosome images. En: *IEEE*

transactions on image processing: a publication of the IEEE Signal Processing Society 14 (2005), September, Nr. 9, p. 1277–1287. – ISSN 1057–7149

[48] Qiu, Yuchen ; Lu, Xianglan ; Yan, Shiju ; Tan, Maxine ; Cheng, Samuel ; Li, Shibo ; Liu, Hong ; Zheng, Bin: Applying deep learning technology to automatically identify metaphase chromosomes using scanning microscopic images: an initial investigation. En: *Biophotonics and Immune Responses XI* Vol. 9709, International Society for Optics and Photonics, 9 März 2016, p. 97090K

[49] Quinlan, J R.: Induction of Decision Trees. En: *Machine learning* 1 (1986), 1 März, Nr. 1, p. 81–106. – ISSN 0885–6125, 1573–0565

[50] Ranzato, M ; Huang, F J. ; Boureau, Y L. ; LeCun, Y: Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. En: *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007. – ISSN 1063–6919, p. 1–8

[51] Rao, C R. ; Mitra, Sujit K. ; Others: Generalized inverse of a matrix and its applications. En: *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Theory of Statistics*, 1972

[52] Ritter, Gunter ; Gaggermeier, Karl: Automatic classification of chromosomes by means of quadratically asymmetric statistical distributions. En: *Pattern recognition* 32 (1999), 1 Juni, Nr. 6, p. 997–1008. – ISSN 0031–3203

[53] Ritter, Gunter ; Gao, Le: Automatic segmentation of metaphase cells based on global context and variant analysis. En: *Pattern recognition* 41 (2008), 1 Januar, Nr. 1, p. 38–55. – ISSN 0031–3203

[54] Roshtkhari, Mehrsan J. ; Setarehdan, Seyed K.: A novel algorithm for straightening highly curved images of human chromosome. En: *Pattern recognition letters* 29 (2008), 1 Juli, Nr. 9, p. 1208–1217. – ISSN 0167–8655

[55] Sagawa, R ; Shiba, Y ; Hirukawa, T ; Ono, S ; Kawasaki, H ; Furukawa, R: Automatic feature extraction using CNN for robust active one-shot scanning. En: *2016 23rd International Conference on Pattern Recognition (ICPR)*, 2016, p. 234–239

[56] Salzberg, Steven L.: C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993. En: *Machine learning* 16 (1994), 1 September, Nr. 3, p. 235–240. – ISSN 0885–6125, 1573–0565

[57] Scherer, Dominik ; Müller, Andreas ; Behnke, Sven: Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. En: *Artificial Neural Networks – ICANN 2010*, Springer, Berlin, Heidelberg, 15 September 2010 (Lecture Notes in Computer Science). – ISBN 9783642158247, 9783642158254, p. 92–101

[58] Schwartzkopf, Wade C. ; Bovik, Alan C. ; Evans, Brian L.: Maximum-likelihood techniques for joint segmentation-classification of multispectral chromosome images. En: *IEEE transactions on medical imaging* 24 (2005), Dezember, Nr. 12, p. 1593–1610. – ISSN 0278–0062

[59] Silverman, B W. ; Jones, M C.: E. Fix and J.L. Hodges (1951): An Important Contribution to Nonparametric Discriminant Analysis and Density Estimation: Commentary on Fix and Hodges (1951). En: *International statistical review = Revue internationale de statistique* 57 (1989), Nr. 3, p. 233–238. – ISSN 0306–7734, 1751–5823

[60] Simonyan, Karen ; Zisserman, Andrew: Very Deep Convolutional Networks for Large-Scale Image Recognition. (2014), 4 September

[61] Sokolova, Marina ; Lapalme, Guy: A systematic analysis of performance measures for classification tasks. En: *Information processing & management* 45 (2009), 1 Juli, Nr. 4, p. 427–437. – ISSN 0306–4573

[62] Somasundaram, D ; Palaniswami, S ; Vijayabhasker, R ; Venkatesakumar, V: G-band chromosome segmentation, overlapped chromosome separation and visible band calculation. En: *International journal of human genetics* 14 (2014), Nr. 2, p. 73–81. – ISSN 0972–3757

[63] Srinivas, Suraj ; Sarvadevabhatla, Ravi K. ; Mopuri, Konda R. ; Prabhu, Nikita ; Kruthiventi, Srinivas S S. ; Venkatesh Babu, R: A Taxonomy of Deep Convolutional Neural Nets for Computer Vision. (2016), 25 Januar

[64] Srisang, Wacharapong ; Jaroensutasinee, Krisanadej ; Jaroensutasinee, Mullica: Segmentation of Overlapping Chromosome Images Using Computational Geometry. En: *Walailak Journal of Science and Technology (WJST)* 3 (2011), 16 November, Nr. 2, p. 181–194. – ISSN 2228–835X, 2228–835X

[65] Srivastava, N ; Hinton, G ; Krizhevsky, A ; others: Dropout: A simple way to prevent neural networks from overfitting. En: *The Journal of Machine* (2014)

[66] Suzuki, Satoshi ; Be, Keiichia: Topological structural analysis of digitized binary images by border following. En: *Computer Vision, Graphics, and Image Processing* 30 (1985), 1 April, Nr. 1, p. 32–46. – ISSN 0734–189X

[67] Swati ; Gupta, G. ; Yadav, M. ; Sharma, M. ; Vig, L.: Siamese Networks for Chromosome Classification. En: *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017, p. 72–81

[68] Tso, M K S. ; Graham, J: The transportation algorithm as an aid to chromosome classification. En: *Pattern recognition letters* 1 (1983), 1 Juli, Nr. 5, p. 489–496. – ISSN 0167–8655

[69] Tso, Michael ; Kleinschmidt, Peter ; Mitterreiter, Ilse ; Graham, Jim: An efficient transportation algorithm for automatic chromosome karyotyping. En: *Pattern recognition letters* 12 (1991), 1 Februar, Nr. 2, p. 117–126. – ISSN 0167–8655

[70] Walt, Stéfan van der ; Colbert, S C. ; Varoquaux, Gaël: The NumPy Array: A Structure for Efficient Numerical Computation. En: *Computing in science & engineering* 13 (2011), 1 März, Nr. 2, p. 22–30. – ISSN 1521–9615

[71] Wang, Xingwei ; Zheng, Bin ; Li, Shibo ; Mulvihill, John J. ; Liu, Hong: A rule-based computer scheme for centromere identification and polarity assignment of

metaphase chromosomes. En: *Computer methods and programs in biomedicine* 89 (2008), Januar, Nr. 1, p. 33–42. – ISSN 0169–2607

[72] Wang, Xingwei ; Zheng, Bin ; Li, Shibo ; Mulvihill, John J. ; Wood, Marc C. ; Liu, Hong: Automated classification of metaphase chromosomes: optimization of an adaptive computerized scheme. En: *Journal of biomedical informatics* 42 (2009), Februar, Nr. 1, p. 22–31. – ISSN 1532–0464, 1532–0480

[73] Wang, Xingwei ; Zheng, Bin ; Wood, Marc ; Li, Shibo ; Chen, Wei ; Liu, Hong: Development and evaluation of automated systems for detection and classification of banded chromosomes: current status and future perspectives. En: *Journal of physics D: Applied physics* 38 (2005), 22 Juli, Nr. 15, p. 2536. – ISSN 0022–3727

[74] Widrow, B. ; Hoff, M. E.: Adaptive Switching Circuits. En: *1960 IRE WESCON Convention Record* (1960), p. 96–104. – Reprinted in *Neurocomputing* MIT Press, 1988 .

[75] Yang, Jianchao ; Yu, Kai ; Gong, Yihong ; Huang, T: Linear spatial pyramid matching using sparse coding for image classification. En: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009. – ISSN 1063–6919, p. 1794–1801

[76] Zeiler, Matthew D. ; Fergus, Rob: Visualizing and Understanding Convolutional Networks. En: *Computer Vision and Pattern Recognition,* Vol: abs/1311.2901, Url: https://arxiv.org/abs/1311.2901 (2013)

[77] Zhang, T Y. ; Suen, C Y.: A Fast Parallel Algorithm for Thinning Digital Patterns. En: *Communications of the ACM* 27 (1984), März, Nr. 3, p. 236–239. – ISSN 0001–0782