Universidad de las Ciencias Informáticas

Facultad 3

Identificación de imágenes de estructura ocular de pacientes operados de cataratas

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor:

Claudia Fernández Peña

Tutor:

Lic. Reyder Cruz de la Osa

La Habana, Junio del 2019.

¨Año 61 de la Revolución¨

DECLARACIÓN DE AUTORÍA

Declaro ser el autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los ___ días del mes de _____ del año _____.

_____

Claudia Fernández Peña

Autor

_____

Lic. Reyder Cruz de la Osa

Tutor

DEDICATORIA

A mis padres, mi hermana, mi familia y mis amigos. Gracias por compartir este largo viaje conmigo, y hacer de él, una de mis mejores experiencias.

AGRADECIMIENTOS

A mis padres por ser los pilares que sostienen mi vida. Por brindarme tanto amor, cariño, comprensión, paciencia. Por apoyarme cuando más lo necesite y por hacer de mí, una mejor persona. A mi madre por enseñarme que lo más bello de la vida está en la simpleza y en el amor que le pongamos a lo que realmente nos interesa. A mi padre, por crear en mi esa coraza de valor, dedicación, porque gracias a ti, soy una mujer fuerte. A mi hermana porque la amo, porque eres la niña de mis ojos, porque sabes que moriría por ti. A ustedes, gracias, por tanto. Que no me esfuerce en gritarle al mundo que los amo, no significa que así sea. Ustedes son todo para mí.

A mi familia por la preocupación, por el apoyo, la atención. Gracias por estar siempre en todo momento. A mis abuelas Adriana y Ada, que sería de mi sin ustedes. A mi abuelo Luis, por educarme tanto en la vida. A mis tías Adita, Cristina, Odalys. A mis tíos Luis, Angelito. A mis primos Eileen, Mildred, Luis Angel, Adrianita; no me alcanza la vida para saldar esa deuda tan grande que tengo con ustedes. Gracias de corazón por los buenos momentos, las experiencias, gracias por formar parte de mi vida.

A Diosbel, por preocuparte cada día por cómo iba mi tesis. Por hacerme entender que la vida consiste en valorar y seguir adelante, sin importar costo alguno. A Baby y su familia por las experiencias vividas. A ustedes muchas gracias.

A mis amigos, por hacerme sentir la persona más afortunada del mundo. Más que amigos, somos familia. Gracias por estar presente en todo momento. Por compartir estos años conmigo, y por hacerme entender lo grande que es la vida si te rodeas de gente que aprecia y vive los momentos con la misma intensidad que lo haces tú. A Gretchen, Nailee, Ingrid, Mayrielin, Asiledy, Claudia, Arisleidis, Guido, Loandry, Ernesto, Yordanka. Les estoy eternamente agradecida.

A ustedes porque siento que, si un día los pierdo, voy a morir. Porque uno de los mejores regalos que me ha dado la vida ha sido conocerlos. Gracias por la atención, los consejos. Porque compartir con ustedes lo que realmente me gusta ha sido un honor. Porque cada vez que baile, serán el motivo. A José Luis, Yoslenys, Julio, Ively, Jeiser, Yeisel, Camila, Mariam, a todo Espacio Abierto y Caguairán. Ha sido, es y será un placer haber compartido escenarios juntos. Hacia ustedes, mis mayores respetos.

A mis tutores de tesis, de la vida. Porque son ejemplo de grandeza y humildad. A Reyder por el voto de confianza que puso en mí, por los consejos, por las horas dedicación, le estoy eternamente agradecida. A Michel por ser guía incondicional. A Yoslenys por ser ese ejemplo a seguir. A Reinier por las críticas constructivas. A Dariela por tanto apoyo y comprensión, gracias a la vida por disfrutar de su sutileza y ayuda. Son ese punto de referencia que les doy a todos. Al tribunal por los consejos y recomendaciones siempre recibidas, a la universidad por tan buenos momentos y a los profesores que contribuyeron a mi formación como Ingeniera, muchas gracias.

A mis compañeros Angel, Mayrielin, Yandri, Ramón, Luis Rubén, Pedro, Alejandro, Juan José. Gracias por compartir cada café conmigo. Nunca importó la calidad de este, la compañía siempre fue buena. A mis compañeras de apartamento, gracias por acogerme como una más de ustedes.

A todos, les estaré siempre agradecida por todo y tanto.

RESUMEN

En la actualidad, la detección de la Opacidad de la Cápsula Posterior es uno de los elementos fundamentales en la oftalmología cubana. Con el objetivo de contribuir a esta tarea, el grupo de investigación *Artificial Intelligence: Research and Innovation* de la Universidad de las Ciencias Informáticas y el Instituto cubano de oftalmología Ramón Pando Ferrer, deciden la informatización del proceso de detección de dicho padecimiento. El mismo contribuye a agilizar la tarea de identificación de características relevantes en las imágenes utilizadas provenientes de los equipos oftalmológicos empleados. En base a lo anteriormente expuesto, la presente investigación tiene como objetivo desarrollar una herramienta capaz de detectar estructuras oculares en imágenes de pacientes operados de cataratas. Para lograr este objetivo, se utilizó como metodología de desarrollo Xtreme Programming, así como un conjunto de lenguajes y herramientas, permitiendo de esta forma el desarrollo de la propuesta de solución. Los resultados de la misma fueron validados a través de pruebas de software, donde se pudo verificar la calidad de los artefactos generados, así como las funcionalidades propuestas.

Palabras Claves: *Deep learning*, Imágenes, Opacidad de la cápsula posterior.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

A lo largo de la historia el padecimiento de cataratas ha sido una de las complicaciones postoperatorias más frecuentes, sufriéndose como consecuencia la pérdida total o parcial de la visión. Esta enfermedad no es más que opacidad en el cristalino del ojo, lo cual conlleva a que la luz se disperse dentro del mismo, creando de esta forma una serie de imágenes difusas. La catarata es uno de los ejemplos más comunes de ceguera tratables en cirugía, y aunque tiene varias causas de origen, se les atribuye mayormente el padecimiento de esta enfermedad a personas mayores de cincuenta años. Igualmente puede ocurrir con quienes sufren de diabetes o hipertensión. Después de una intervención quirúrgica el paciente puede recuperar su visibilidad total o parcialmente, pero no en todos los casos la cirugía es un éxito, pues en muchos de ellos el paciente puede presentar, a mediano o largo plazo, complicaciones postoperatorias como la Opacidad de la Cápsula Posterior (OCP) (Tetz, y otros, 2013).

La OCP es uno de los aspectos más importantes en la cirugía de catarata en la actualidad. Según un análisis de 90 estudios publicados en la actualidad su incidencia se encuentra entre 0,7% y 47,6 % en los primeros cinco años de la cirugía, siendo en Cuba una cifra considerable que asciende hasta el 50% de los casos (Grewall, 2008).

Actualmente existen diversos equipos oftalmológicos de alta tecnología, entre ellos se encuentra el PENTACAM, capaz de reconstruir imágenes tridimensionales de alta resolución del polo anterior del ojo, lo cual se realiza a partir de varias fotografías tomadas mediante una cámara rotacional del sistema Scheimpflug perteneciente a dicho equipo. La cuantificación objetiva de la OCP puede ser el resultado del análisis de estos tomogramas (Tetz, y otros, 2013).

La lámpara de hendidura es otro de los mecanismos de diagnóstico más comúnmente usados por un oftalmólogo. Proporciona iluminación y magnificación para examinar las partes del ojo. Una serie de accesorios se pueden añadir a este para convertirlo en un instrumento que puede medir la presión intraocular, la curvatura de la córnea, el espesor de la córnea, la distancia entre la córnea y el lente, el volumen de la cámara anterior, la opacidad y otros elementos. Algunas lámparas de hendidura tienen accesorios para

conectar una cámara fotográfica que se utiliza para proporcionar energía láser en cualquier lugar del ojo para el tratamiento.

Una lámpara de hendidura moderna consta de tres componentes principales:

- Un sistema de iluminación - fuente de luz, espejos y prismas.
- Un sistema de magnificación - el biomicroscopio.
- Un sistema mecánico que une el sistema de magnificación con el sistema de iluminación y proporciona movimientos verticales y laterales para enfocar la luz en la parte deseada del ojo (Hernández López, y otros, 2010).

Por las razones antes expuestas, el grupo de investigación Artificial Intelligence: Research and Innovation (AIRI) de la Universidad de las Ciencias Informáticas (UCI), en conjunto con el Instituto Cubano de Oftalmología Ramón Pando Ferrer deciden crear el software PANDOC o Programa Analizador de Opacidad Capsular. Este software provee al oftalmólogo de una herramienta por medio de la cual es capaz de cuantificar numéricamente y detectar diferencias de opacidad (a veces imperceptibles para el ojo humano), para lograr una evaluación objetiva del grado de la misma. En algunas ocasiones la detección de ciertos detalles resulta difícil debido a que el especialista debe identificar manualmente los puntos que considera opacos, valoración que depende mucho de su nivel de experticia, y se desea evitar este grado de subjetividad. Por lo que en algunos momentos el paciente puede presentar determinadas características, lo que dificulte el proceso de la toma de fotos. Entre estos a personas de la tercera edad que sufren ciertas molestias, a niños o jóvenes que muestren un comportamiento inadecuado, o inclusive a pacientes con alguna dificultad de aprendizaje o características especiales, teniendo como resultado una imagen que no presente las condiciones necesarias para su posterior procesamiento. Es por ello que surge como necesidad la identificación automática de imágenes de estructura ocular provenientes de la lámpara de hendidura, para que con ello el oftalmólogo pueda emplear su tiempo realizando su labor, sin preocuparse por la toma o la calidad de las fotos.

Dada la situación problemática expuesta anteriormente se plantea como **problema:** ¿Cómo identificar imágenes de estructura ocular provenientes de la lámpara de hendidura? A raíz de esto se toma como **objeto de estudio:** visión por computadoras, enmarcándose en el

campo de acción: la identificación de imágenes de OCP provenientes de la lámpara de hendidura.

En aras de dar solución al problema planteado se tiene como **objetivo general:** Desarrollar un algoritmo para la identificación de imágenes de estructura ocular provenientes de la lámpara de hendidura. Para dar cumplimiento al objetivo de la investigación se han tomado los siguientes objetivos específicos:

Objetivos específicos:

1. Elaborar el marco teórico relacionado con la identificación de imágenes médicas para detectar estructuras oculares en ellas.

2. Desarrollar la identificación de los requisitos, análisis, diseño e implementación de un algoritmo para la identificación de imágenes de estructura ocular provenientes de la lámpara de hendidura.

3. Validar la propuesta de solución a través de pruebas unitarias y funcionales para medir el funcionamiento del algoritmo.

Teniendo en cuenta el problema a resolver se formuló la siguiente **Idea a defender:** si se desarrolla un algoritmo para la identificación de imágenes de OCP, se podrán tomar dichas imágenes automáticamente desligando al médico y la cooperación del paciente de esta tarea.

Para dar cumplimiento al objetivo de la investigación se trazaron las siguientes tareas a cumplir:

Tareas a cumplir:

1. Identificación de las diferentes técnicas de procesamiento de imágenes médicas.

2. Estudio del uso de algoritmos de reconocimiento de imágenes.

3. Descripción de los pasos a seguir para el uso del algoritmo propuesto.

4. Diseñar la solución propuesta en función de los requisitos especificados.

5. Implementación del algoritmo de identificación propuesto.

6. Validación de la implementación del sistema a partir de la aplicación de pruebas.

Para dar solución a las tareas antes propuestas se definen los métodos científicos, clasificados en teóricos y empíricos. De dichos métodos se emplearon:

Métodos teóricos:

- **Analítico-sintético:** se utilizó con el objetivo de analizar las teorías, documentos e información, permitiendo la extracción de los elementos más importantes que se relacionan con la segmentación de imágenes médicas.
- **Inductivo-deductivo:** se empleó para inducir una serie de conocimientos referentes a la cuantificación objetiva de la OCP, poder arribar a razonamientos que conlleven a la deducción de conocimientos que puedan ser aplicables al problema a resolver en cuestión.
- **Histórico-lógico:** se utilizó con el objetivo de analizar y estudiar la trayectoria y evolución de software de análisis de OCP en pacientes operados de catarata, para así poder contar con una noción de cuán desarrollado está el tema a nivel mundial y tomar la decisión de cuál desarrollar.
- **Análisis documental:** permitió realizar el estudio bibliográfico como aspecto esencial en la conformación del marco teórico referencial de la investigación, como sustento de las valoraciones realizadas, así como el estudio de los contenidos relacionados con la identificación de la OCP en pacientes operados de catarata.

Método empírico:

- **Entrevista:** se utilizó en el intercambio con el cliente para adquirir información acerca de los referentes teóricos a abordar acerca de la OCP. Permitió también, fomentar las bases para la descripción de los requisitos necesarios para la elaboración de la propuesta.

El documento de tesis está estructurado por 3 capítulos, a continuación se dará una breve descripción de los mismos.

En el **capítulo 1** se realiza un análisis de la literatura consultada y se hace una representación de los conceptos asociados con la investigación. Se enfatiza en los principales métodos para la identificación de estructuras oculares y se expone lo que es el tratamiento de imágenes digitales. Se explican las técnicas existentes en el campo del tratamiento de imágenes, y se realiza un estudio de sistemas homólogos que detecten la OCP. También son tratadas las principales técnicas y herramientas definidas para la implementación de dicho algoritmo.

En el **capítulo 2** se presenta un algoritmo para la identificación de estructuras oculares en imágenes provenientes de la lámpara de hendidura. Además, se realiza una descripción general de la solución propuesta, se especifican los requisitos funcionales y no funcionales que se tendrán en cuenta para el desarrollo del algoritmo y se detallan aspectos relacionados con su diseño y arquitectura. Se especifican además, los patrones del diseño aplicados y los artefactos derivados de la metodología de desarrollo de software seleccionada.

En el **capítulo 3** se presenta la validación del algoritmo propuesto. Se reflejan los resultados reales luego de ser aplicadas las pruebas necesarias para verificar los requisitos definidos durante la propuesta de solución.

CAPÍTULO 1. Fundamentación teórica.

En este capítulo se realiza una descripción y análisis desde el punto de vista teórico del problema general en que se enmarca la investigación. Se abordan los principales conceptos asociados al dominio del problema. Se realiza un estudio para observar el comportamiento de las técnicas más utilizadas en la clasificación de imágenes, así como la metodología que guiará el proceso de desarrollo del software y las herramientas necesarias para el desarrollo de la solución.

1.1: Conceptos asociados.

Opacidad de la cápsula posterior

La cápsula posterior es una estructura que anatómicamente queda por detrás de la lente intraocular implantada en el paciente operado de catarata, su transparencia es imprescindible para una buena recuperación visual del paciente. Desafortunadamente no son pocos los casos en los que se desarrolla esta anomalía, la opacificación de la cápsula posterior repercute negativamente en el resultado visual del paciente. En el campo de la oftalmología esto es conocido como OCP y constituye en la actualidad la complicación tardía más importante en la cirugía de catarata (Alvarez Cancio, y otros, 2013).

Imagen digital

Una imagen digital es un arreglo bidimensional de píxeles, donde el valor de cada píxel se representa mediante una función *f(x,y)* que simboliza el nivel de brillantez, color o intensidad de la imagen en tales coordenadas. Por lo tanto, una imagen en blanco y negro puede ser representada por una matriz de dimensión $M \times N$, donde $f(x,y) \in \{0,1\}$. Además de la representación en blanco y negro, otras opciones son las imágenes en tonos de gris, para las cuales *f(x,y)* constituye un nivel de intensidad típicamente asociado a un entero entre 0 y 255 y otras más sirven para describir imágenes en color (Elizondo, y otros, 2012).

Procesamiento digital de imágenes

El Procesamiento Digital de Imágenes (PDI) es el conjunto de técnicas y procesos para descubrir, resaltar y describir la información contenida en una imagen, usando como herramienta principal un sistema de cómputo. El PDI tiene como objetivo el mejoramiento

de la información gráfica para la interpretación humana y el procesamiento de los datos de la escena para la percepción automática por computadora (Alvarez Cancio, y otros, 2013).

Es importante destacar que no existe un método común que logre realizar correctamente el procesamiento de una imagen digital, por lo que se tiene que analizar según las características propias de la imagen en cuestión, cuál de los algoritmos existentes en cada una de las etapas planteadas tiene un mejor desempeño.

1.2: Sistemas homólogos para tratar la OCP.

Los sistemas más usados en la actualidad por el oftalmólogo que le permite visualizar la OCP son los sistemas basados en la lámpara de hendidura. Es por esta razón que algunos autores han desarrollado soluciones informáticas para lograr una evaluación objetiva y reproducible de esta complicación, los cuales se muestran a continuación.

1.2.1: POCO.

POCO (*Posterior Capsule Opacity*): en este sistema las imágenes obtenidas son por retroiluminación, las mismas son evaluadas usando un análisis de los píxeles basado en la diferencia de texturas. Esta evaluación es semiobjetiva, ya que el examinador debe elegir el área afectada y clasificarla según su severidad: el programa divide la zona dentro de la capsulorrexis en 56 pequeños segmentos de igual área, el examinador debe marcar los segmentos que estén opacificados en más del 50% de su área y luego asignar a éstos un nivel de severidad (leve, moderado o severo). El resultado viene dado en forma de porcentaje de opacificación de 0 a 100%. Se calcula la severidad de la opacificación mediante la fórmula (Madruga, 2016):

[(área de grado 1 x 1) + (área de grado 2 x 2) + (área de grado 3 x 3)]/ área total.

1.2.2: EPCO.

EPCO (Evaluación de la opacidad de la cápsula posterior): se obtienen imágenes de la lente por retroiluminación, se transfieren al programa, se marca el área a estudiar (algunos autores analizan el área tras la óptica, otros el área dentro de la capsulorrexis anterior y también es posible examinar la zona central comprendida por 3-4 mm), luego se remarcan las zonas opacificadas con el ratón y se clasifican subjetivamente en 4 grupos: mínimo, leve, moderado y severo. El índice de OCP viene dado al multiplicar el grado de opacificación por el área seleccionada. El resultado obtenido es un índice de 0 a 4. Al igual

que el software POCO calcula la severidad de la opacificación mediante la fórmula (Madruga, 2016):

[(área de grado 1 x 1) + (área de grado 2 x 2) + (área de grado 3 x 3)]/ área total.

1.2.3: AQUA.

AQUA (*Automated Quantification of After-Cataract*): se obtienen imágenes con retroiluminación y se importan al programa. Se selecciona y analiza el área dentro de la capsulorrexis. Este programa está basado en texturas. Para calcular la no homogeneidad de la imagen se calcula la entropía (grado de desorden) de un mapa de bits. Es totalmente automático y no tiene pasos subjetivos. El resultado que se obtiene es un índice de 0 a 10 (Alvarez Cancio, y otros, 2013).

1.2.4: AA.

AA *(Aslam Analize):* este sistema fue diseñado y programado utilizando la plataforma de programación MatLab. El primer problema encontrado fue uno de iluminación desigual en las imágenes. Incluso con las grandes áreas de reflejos de luz aberrantes retirados, utilizando la fusión con imágenes similares, pero vírgenes, una iluminación de fondo generalmente variables de toda la imagen puede causar errores en el análisis de imágenes. Aunque un sistema para la eliminación de registro basado de imágenes de luz está siendo probado y desarrollado por TM Aslam, todavía se necesita una curva de aprendizaje larga y sigue influyendo el efecto negativo de la iluminación de fondo (Elizondo, y otros, 2012).

Luego de haber realizado un breve estudio se demuestra que en estos sistemas no se da tratamiento al efecto producido por la reflexión de la luz en la córnea en las imágenes provenientes de la lámpara de hendidura. Dichas imágenes con este fenómeno de reflexión que sesga la detección de opacidad capsular son conocidas como imágenes de Purkinje. Para el procesamiento de estas imágenes se crea el software PANDOC ya antes mencionado.

1.3: Imágenes en retroiluminación.

Para detectar OCP se toman imágenes en retroiluminación provenientes de la lámpara de hendidura, que es una imagen digital del fondo del ojo capturada cuando el haz de luz es reflejado por el cristalino del ojo y cuentan con las siguientes propiedades (Tetz, y otros, 2013):

1. Se encuentran en un espacio de color RGB lo que da niveles de intensidades desde 0 a 255 colores.
2. Tienen como formato JPG, lo que significa que la imagen obtenida no es la misma que la deseada. Esto es provocado por el algoritmo de compresión que utiliza dicho formato que es un algoritmo de reducción con pérdida para reducir el tamaño del archivo.
3. Tiene una profundidad de 24 bits.
4. Una resolución de 483 x 333 píxeles.
5. Presenta una luz provocada por el mismo equipo que puede dificultar la obtención de la opacidad.

1.4: Imágenes oblicuas.

En las lámparas de hendidura se puede ajustar también la iluminación y cambiar los filtros utilizados para observar el globo ocular en diferentes condiciones. Esto permite una examinación muy completa del ojo, por lo que el profesional puede detectar enfermedades como las cataratas, el glaucoma o la degeneración macular, aunque también se pueden observar indicios de otras patologías sin relación directa con la vista. Otra de las funciones que tiene es que se puede modificar los aumentos, e incluso se puede desplazar todo el microscopio a los lados del ojo. Las imágenes resultantes de este proceso se conocen como imágenes oblicuas. Estas imágenes presentan propiedades similares a las imágenes en retroiluminación, aunque estas tienen una resolución de 1024 x 768 píxeles (Tetz, y otros, 2013).

1.5: Visión por computadora.

Según (Shapiro, y otros, 2000), el objetivo de la visión por computadora *(computer vision)* es tomar decisiones correctas sobre objetos y escenas del mundo físico a partir de imágenes captadas. Para tomar decisiones a partir de objetos físicos reales, es necesario construir algún modelo de dichos objetos a partir de las imágenes, es por ello que surgen las siguientes inquietudes asociadas a esta disciplina:

1. Captura de la imagen: ¿Cómo las imágenes captan propiedades del mundo exterior, tales como material, forma, iluminación y relaciones espaciales?

2. Información codificada: ¿Cómo las imágenes codifican información del mundo tridimensional, tales como geometría, textura, movimiento o la identificación de objetos?
3. Representaciones: ¿Qué representaciones deben ser utilizadas para almacenar descripciones de objetos, y sus partes, propiedades y relaciones?
4. Algoritmos: ¿Qué métodos existen para el procesamiento de la información de la imagen y construir descripciones del mundo y sus objetos? (Shapiro, y otros, 2000)

Las aplicaciones de las computadoras en el análisis y procesamiento de imágenes son ilimitadas, en contextos tales como examinar el interior del cerebro humano utilizando tomogramas, el análisis de imágenes digitales en la toma de decisiones, procesamiento de páginas de texto escaneadas, reconocimiento de emociones a partir de imágenes de rostros, identificación de objetos en imágenes de video tomadas por cámaras de vigilancia, entre otros.

1.6: Descriptores de color.

El Descriptor de la Distribución del Color (DDC) está diseñado para capturar la distribución espacial del color en una imagen. El DDC captura la disposición espacial de los colores más representativos de una rejilla superpuesta en una región o una imagen. La representación se basa en los coeficientes aplicando la transformada discreta del coseno (DCT) en una matriz 2D con la representación local de los colores en Y o Cb, o Cr del espacio de color. Este es un descriptor muy compacto, altamente eficiente en la navegación y rápido en las aplicaciones de búsqueda. Se puede aplicar tanto en las imágenes fijas cómo en los fragmentos de vídeo (Verdarguer, 2016).

Este es una representación del color de resolución invariante y muy compacta que permite la recuperación de imágenes con alta velocidad y que ha sido diseñado para representar de manera eficaz la distribución espacial de los colores. Es especialmente útil para aplicaciones espaciales de recuperación basadas en estructuras. Las funcionalidades del DDC son básicamente las siguientes:

- Comparación de imagen a imagen.
- Comparación de videoclip a videoclip.

El DDC es uno de los descriptores de color más rápido y preciso En la figura 1 se muestra el procedimiento que siguen ests descriptores .

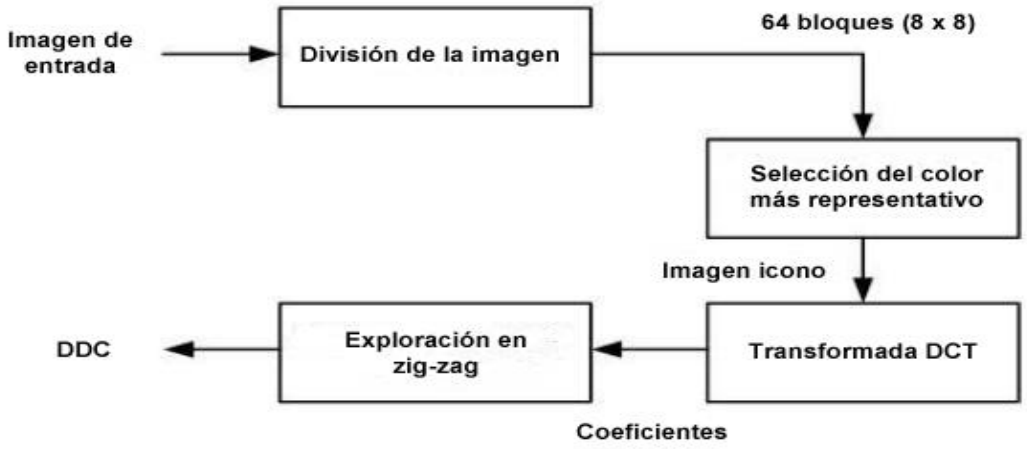


Figura 1. Descripción de descriptores.

Fuente: (Verdarguer, 2016) .

1.7: Aprendizaje automático.

El aprendizaje automático o *machine learning* nació de la búsqueda de Inteligencia Artificial. Ya en los primeros días, algunos investigadores se interesaron en hacer que las máquinas aprendiesen. Trataron de resolver el problema con diversos métodos simbólicos, así como lo que ellos llamaron “redes neurales” que eran en general perceptrones y otros modelos básicamente basados en modelos lineales generalizados como se conocen en las estadísticas. Este tiene como objetivo desarrollar técnicas que permitan que las computadoras aprendan. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos (Gollapudi, 2016).

El aprendizaje automático tiene como resultado un modelo para resolver alguna tarea dada. Entre estos se distinguen (Gollapudi, 2016):

- Modelos Geométricos: Construidos en el espacio de instancias y que pueden tener una, dos o múltiples dimensiones. Si hay un borde de decisión lineal entre las clases, se dice que los datos son linealmente separables.

- Modelos Probabilísticos: Intentan determinar la distribución de probabilidades descriptora de la función que enlaza a los valores de las características con valores determinados.
- Modelos Lógicos: Transforman y expresan las probabilidades en reglas organizadas en forma de árboles de decisión.

Los modelos pueden también clasificarse como modelos de agrupamiento y modelos de gradiente. Los primeros tratan de dividir el espacio de instancias en grupos. Los segundos, como su nombre lo indican, representan un gradiente en el que se puede diferenciar entre cada instancia.

Otro de los algoritmos de aprendizaje automático son los de refuerzo. Estos, no son más que un método de aprendizaje que interactúa con su entorno al producir acciones y descubrir errores o recompensas. La búsqueda de prueba y error y la recompensa diferida son las características más relevantes del aprendizaje por refuerzo. Este método permite que las máquinas y los agentes de software determinen automáticamente el comportamiento ideal dentro de un contexto específico para maximizar su rendimiento. Se requiere una retroalimentación de recompensa simple para que la gente sepa que acción es la mejor; esto se conoce como la señal de refuerzo.

Este aprendizaje permite además el análisis de cantidades masivas de datos. Si bien generalmente proporciona resultados más rápidos y más precisos para identificar oportunidades rentables o riesgos peligrosos, también puede requerir tiempo y recursos adicionales para capacitarlo adecuadamente. La combinación del aprendizaje automático con la inteligencia artificial y las tecnologías cognitivas puede hacer que sean aún más efectivos en el procesamiento de grandes volúmenes de información (Varone, y otros, 2016).

1.7.1: Clasificadores.

La clasificación es el proceso de asignar a un conjunto de observaciones en un conjunto de datos una de las categorías disjuntas previamente especificadas para el dominio de observaciones. En otras palabras, consiste en identificar características representativas de diferentes clases de instancias a partir de un conjunto de instancias de entrenamiento y posteriormente, utilizar estas características para obtener la clase de nuevas observaciones.

Entre los enfoques de clasificación más utilizados en el estado del arte se encuentran los probabilísticos (Duda, y otros, 1973) (Friedman, y otros, 1997), los basados en inducción de reglas (Friedman, y otros, 1997), los árboles de decisión (Quinlan, 1993), las máquinas de soporte vectorial (Dumais, y otros, 2015) y las redes neuronales (Lippmann, 1989).

Los clasificadores estándares requieren que la estructura a ser segmentada posea características cuantificables distintas. Debido a que los datos de entrenamiento pueden ser etiquetados, los clasificadores pueden transferir estas etiquetas a los nuevos datos siempre que el espacio característico distinga cada etiqueta lo suficiente. No son iterativos, por lo que son relativamente eficientes computacionalmente y pueden ser aplicados a imágenes multicanal. Como desventajas se tiene que no obedecen a ningún modelo espacial y la necesidad de la interacción manual para obtener los datos de entrenamiento (Ortega, y otros, 2008).

1.7.1.1: Máquinas de soporte vectorial.

Las máquinas de soporte vectorial, máquinas de vectores soporte o máquinas de vector soporte (*Support Vector Machines*, SVM) son un conjunto de algoritmos de aprendizaje supervisado desarrollados por Vladimir Vapnik y su equipo en los laboratorios AT&T (Castro, 2015). Estos métodos están propiamente relacionados con problemas de clasificación y regresión. Dado un conjunto de ejemplos de entrenamiento (de muestras) podemos etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra. Intuitivamente, una SVM es un modelo que representa a los puntos de muestra en el espacio, separando las clases a 2 espacios lo más amplios posibles mediante un hiperplano de separación definido como el vector entre los 2 puntos, de las 2 clases, más cercanos al que se llama vector soporte (Giacamonte, y otros, 2015). Cuando las nuevas muestras se ponen en correspondencia con dicho modelo, en función de los espacios a los que pertenezcan, pueden ser clasificadas a una o la otra clase. Más formalmente, una SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta (o incluso infinita) que puede ser utilizado en problemas de clasificación o regresión. Una buena separación entre las clases permite una clasificación correcta (Alonso, 2015).

1.7.1.2: Redes neuronales artificiales

Las Redes Neurales Artificiales (RNA) son redes masivamente paralelas de procesamiento de elementos o nodos que simulan el aprendizaje biológico. Cada nodo en una RNA es capaz de llevar a cabo cálculos elementales. El aprendizaje se alcanza a través de la adaptación de pesos asignados a las conexiones entre nodos (Alonso, y otros, 2016).

Una primera clasificación de las redes de neuronas artificiales que se suele hacer es en función del patrón de conexiones que presenta. Así se definen tres tipos básicos de redes (Alonso, y otros, 2016):

- Dos tipos de redes de propagación hacia delante o acíclicas en las que todas las señales van desde la capa de entrada hacia la salida sin existir ciclos, ni conexiones entre neuronas de la misma capa de red neuronal.
  1. Monocapa. Ejemplos: Perceptrón, Adaline.
  2. Multicapa. Ejemplos: Perceptrón multicapa.
- Las redes recurrentes que presentan al menos un ciclo cerrado de activación neuronal. Ejemplos: Elman, Hopfield, máquina de Boltzmann.

Una segunda clasificación que se suele hacer es en función del tipo de aprendizaje de que es capaz (si necesita o no un conjunto de entrenamiento supervisado). Para cada tipo de aprendizaje existen varios modelos propuestos (Alonso, y otros, 2016) (Caparrini, 2015):

- Aprendizaje supervisado: necesitan un conjunto de datos de entrada previamente clasificado o cuya respuesta objetivo se conoce. Ejemplos de este tipo de redes son: el perceptrón simple, la red Adaline, el perceptrón multicapa, *red backpropagation*[1], y la memoria asociativa bidireccional.
- Aprendizaje no supervisado o autoorganizado: no necesitan de tal conjunto previo. Ejemplos de este tipo de redes son: las memorias asociativas, las redes de Hopfield,

[1] Es un método de cálculo del gradiente utilizado en algoritmos de aprendizaje para entrenar redes neuronales artificiales. El método emplea un ciclo propagación – adaptación de dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas siguientes de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas (Michie, y otros, 2016).

la máquina de Boltzmann y la máquina de Cauchy, las redes de aprendizaje competitivo, las redes de Kohonen o mapas autoorganizados y las redes de resonancia adaptativa (ART).

- Redes híbridas: son un enfoque mixto en el que se utiliza una función de mejora para facilitar la convergencia. Un ejemplo de este último tipo son las redes de base radial.
- Aprendizaje reforzado: se sitúa a medio camino entre el supervisado y el auto organizado.

Entre los tipos de redes más utilizados actualmente se encuentran:

Perceptrón

Puede entenderse como la neurona artificial o unidad básica de inferencia en forma de discriminador lineal, a partir de lo cual se desarrolla un algoritmo capaz de generar un criterio para seleccionar un subgrupo a partir de un grupo de componentes más grande. La limitación de este algoritmo es que, si dibujamos en un plot estos elementos, se deben poder separar con un hiperplano únicamente los elementos "deseados" discriminándolos (separándolos) de los "no deseados". El perceptrón puede utilizarse con otros perceptrones u otro tipo de neurona artificial, para formar redes neuronales más complicadas **(RAI, 2016)** En la figura 2 se evidencia su arquitectura.

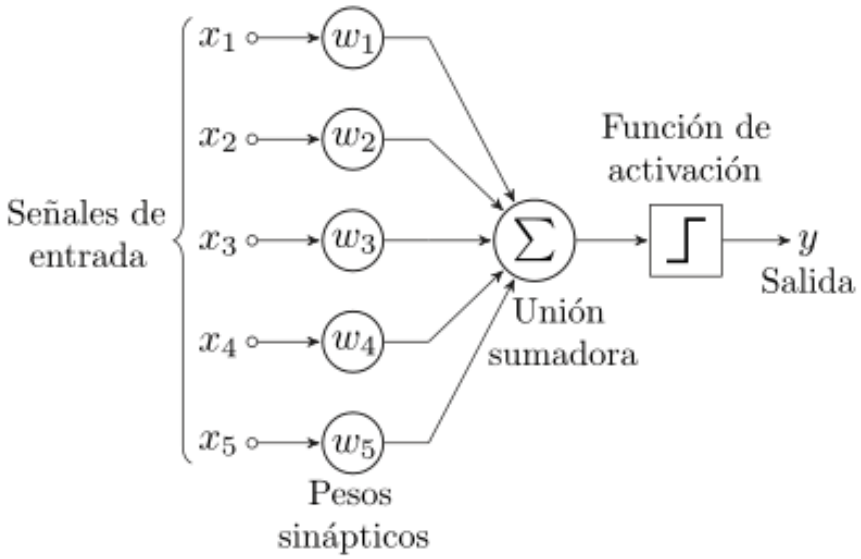


Figura 2. Topología de un perceptrón.

Fuente: (RAI, 2016).

Perceptrón multicapa

Es una red neuronal artificial formada por múltiples capas, esto le permite resolver problemas que no son linealmente separables, lo cual es la principal limitación del perceptrón (también llamado perceptrón simple). El perceptrón multicapa puede ser totalmente o localmente conectado. En el primer caso cada salida de una neurona de la capa *i* es entrada de todas las neuronasde la capa *i+1*, mientras que en el segundo caso cada salida de neurona de la capa *i* es entrada de una serie de neuronas (región) de la capa *i+1* **(RAI, 2016)**. En la figura 3 se muestra la arquitectura que la compone.

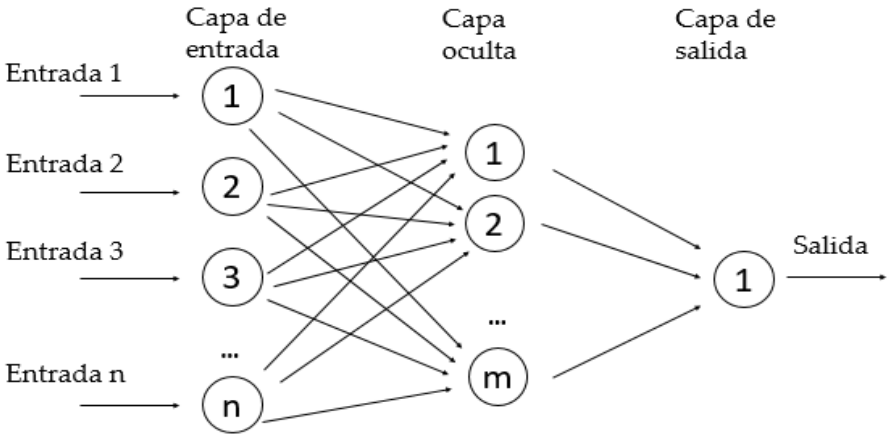


Figura 3. Topología de un perceptrón multicapa.

Fuente: (RAI, 2016).

Adaline

De (*ADAptative LINear Element*) es un tipo de red neuronal artificial desarrollada por el profesor Bernie Widrow y su alumno Ted Hoff en la Universidad de Stanford en 1960. El modelo está basado en la Neurona de McCulloch-Pitts. Con respecto al perceptrón el Adaline posee la ventaja de que su gráfica de error es un hiperparaboloide que posee o bien un único mínimo global, o bien una recta de infinitos mínimos, todos ellos globales. Esto evita la gran cantidad de problemas que da el perceptrón a la hora del entrenamiento debido a que su función de error (también llamada de coste) posee numerosos mínimos locales **(RAI, 2016)**.

Hopfield

Es una red recurrente, es decir, existe realimentación entre las neuronas. De esta forma, al introducir un patrón de entrada, la información se propaga hacia adelante y hacia atrás, produciéndose una dinámica. En algún momento, la evolución se detendrá en algún estado estable. En otros casos, es posible que la red no se detenga nunca (Gómez, 2017). En la figura 4 se observa su arquitectura.

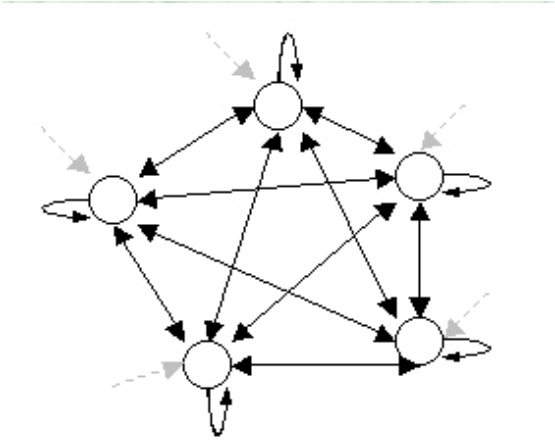


Figura 4. Arquitectura de una red de Hopfield.

Fuente: (Gómez, 2017).

Máquina de Boltzmann

Una máquina de Boltzmann puede considerarse como la contrapartida estocástica y generativa de las redes de Hopfield. Fueron de los primeros tipos de redes neuronales capaces de aprender mediante representaciones internas, son capaces de representar y (con tiempo suficiente) resolver complicados problemas combinatorios. Sin embargo, debido a una serie de cuestiones que se abordan más adelante, las máquinas de Boltzmann sin restricciones de conectividad no han demostrado ser útiles para resolver los problemas que se dan en la práctica en el aprendizaje o inferencia de las máquinas. Aún así, resultan interesantes en la teoría debido a la localización y a la naturaleza hebbiana de su algoritmo de entrenamiento, así como por su paralelismo y por la semejanza de su dinámica a fenómenos físicos sencillos. Si se limita la conectividad, el aprendizaje puede ser lo bastante eficaz como para ser útil en la resolución de problemas (Ackley D H, 2014).

Redes neuronales convolucionales

Una red neuronal convolucional o *Convolutional Neural Network* (CNN) es un tipo de red neuronal artificial donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria de un cerebro biológico. Este tipo de red es una variación de un perceptrón multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones (Fukushima, 2015).

Dichas redes neuronales consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Después de cada capa, por lo general se añade una función para realizar un mapeo causal no-lineal.

Como redes de clasificación, al principio se encuentra la fase de extracción de características, compuesta de neuronas convolucionales y de reducción de muestreo (Ciseran, y otros, 2015). Al final de la red se encuentran neuronas de perceptrón sencillas para realizar la clasificación final sobre las características extraídas. La fase de extracción de características se asemeja al proceso estimulante en las células de la corteza visual. Esta fase se compone de capas alternas de neuronas convolucionales y neuronas de reducción de muestreo. Según progresan los datos a lo largo de esta fase, se disminuye su dimensionalidad, siendo las neuronas en capas lejanas mucho menos sensibles a perturbaciones en los datos de entrada, pero al mismo tiempo son activadas por características cada vez más complejas. En la figura 5 se refleja el proceso que siguen estas redes.

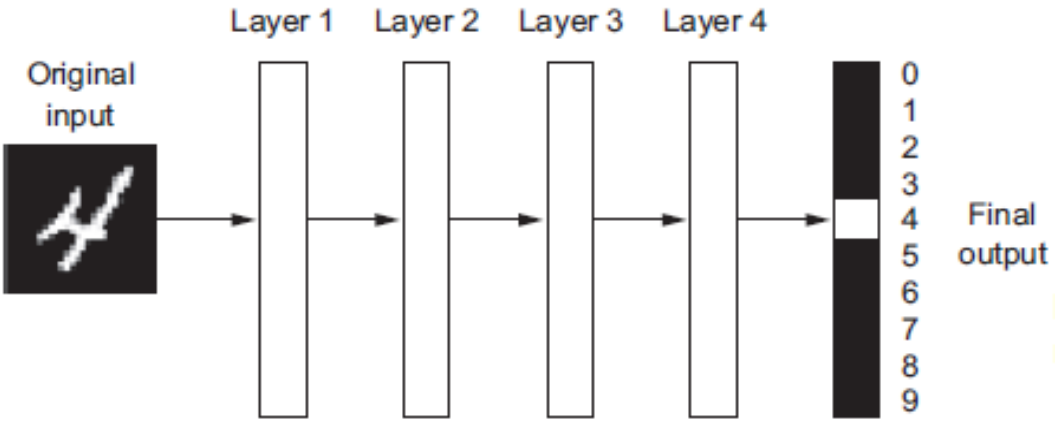


Figura 5. CNN para el reconocimiento de números escritos a mano.

Fuente: (Chollet, 2018).

1.7.1.3: Enfoque bayesiano

El clasificador bayesiano se basa en la suposición de que todos los atributos son independientes dada la clase. Una forma muy sencilla y rápida de clasificar los píxeles de imágenes médicas es en base a sus atributos de color (Sucar, 2016). Este clasificador puede ser suficiente en algunas aplicaciones y en otras proveer un procesamiento inicial de la imagen, para luego utilizar otros métodos más sofisticados en las regiones candidatas.

Una alternativa es considerar un clasificador bayesiano simple o semi-simple utilizando el algoritmo de mejora estructural, incluyendo inicialmente varios modelos de color; considerando para ello los modelos RGB, HSV e YIQ, de forma que inicialmente se tienen 9 atributos en total (Sucar, 2016). Luego de esto se evalúa experimentalmente tanto al clasificador original como al mejorado con imágenes en que hubiera pixeles (diferentes al entrenamiento), teniendo como resultado una mejora de hasta un 94% según los atributos de precisión.

1.7.2: Aprendizaje profundo.

Aprendizaje profundo o *deep learning* es un conjunto de algoritmos de clase *machine learning* que intenta modelar abstracciones de alto nivel en datos usando arquitecturas compuestas de transformaciones no lineales múltiples.

El *deep learning* de las redes neuronales anteriores es que se dispone de hardware más rápido y especializado con más datos de entrenamiento disponibles. Se pueden entrenar redes neuronales con muchas más capas ocultas permitiendo un aprendizaje jerárquico, donde características más simples son aprendidas en las capas bajas y patrones más abstractos en las capas superiores.

La palabra *deep* (profundo) en *deep learning* se refiere a la idea de aprender la jerarquía de los conceptos directamente a partir de los datos. Un término más apropiado técnicamente sería *representation learning* (aprendizaje de representación), pues se utilizan capas sucesivas de representaciones para la clasificación final (Ketkar, 2017).

El ejemplo más representativo de aplicar *deep learning* en el aprendizaje de descriptores de imagen son las CNN (Lecun, 1998). Estas son aplicadas en el reconocimiento de caracteres escritos a mano que automáticamente aprende patrones discriminativos (llamados “filtros”) a partir de las imágenes, generando secuencialmente capas ocultas en la red. Los filtros en las capas más bajas del modelo representan bordes y esquinas, mientras que capas superiores utilizan dichos bordes y esquinas para aprender patrones más abstractos necesarios para discriminar entre las clases deseadas.

No hay un consenso respecto a la cantidad de capas ocultas necesarias para hablar de *deep learning*. Varios autores refieren que hay *deep learning* cuando hay más de una capa oculta en la arquitectura de la red (Sewak, y otros, 2018) (Rosebrock, 2017) . Por definición, una CNN es un tipo de algoritmo de *deep learning*. En muchas aplicaciones, las CNN son consideradas los clasificadores de imágenes más potentes y los responsables de impulsar el estado del arte del aprendizaje automático en la visión por computadoras.

Las dos ideas clave para *deep learning* en visión por computadoras, o sea las CNN y *backpropagation*, ya eran de dominio público en 1989. El algoritmo LSTM (*Long-Short Term Memory*), fundamental para *deep learning* en series temporales, fue desarrollado en 1997 y apenas ha cambiado desde entonces. Sin embargo, el despegue de *deep learning* no ocurre hasta 2012, gracias al desarrollo de hardware y la acumulación de grandes cantidades de datos, como es la aparición del proyecto ImageNet con más de 1.4 millones de imágenes anotadas en 1000 categorías (una categoría por imagen).

Varias arquitecturas de aprendizaje profundo, como redes neuronales profundas, redes neuronales profundas convolucionales, y redes de creencia profundas, han sido aplicadas a campos como visión por computador, reconocimiento automático del habla, y

reconocimiento de señales de audio y música, y han mostrado producir resultados de vanguardia en varias tareas (Schmidhuber, 2014).

A partir de este punto común, diferentes publicaciones se centran en distintas características, como:

- Usar una cascada de capas con unidades de procesamiento no lineal para extraer y transformar variables. Cada capa usa la salida de la capa anterior como entrada. Los algoritmos pueden utilizar aprendizaje supervisado o no supervisado, y las aplicaciones incluyen modelización de datos y reconocimiento de patrones.
- Estar basados en el aprendizaje de múltiples niveles de características o representaciones de datos. Las características de más alto nivel se derivan de las características de nivel inferior para formar una representación jerárquica.
- Aprender múltiples niveles de representación que corresponden con diferentes niveles de abstracción. Estos niveles forman una jerarquía de conceptos (Bengio, y otros, 2013).

Todas estas maneras de definir el aprendizaje profundo tienen en común múltiples capas de procesamiento no lineal y el aprendizaje supervisado o no supervisado de representaciones de características en cada capa. Las capas forman una jerarquía de características desde un nivel de abstracción más bajo a uno más alto (Schmidhuber, 2014).

1.7.3: Aprendizaje jerárquico

En el contexto de aprendizaje automático aplicado en la clasificación de imágenes, el objetivo es tomar un conjunto de imágenes e identificar patrones que pueden ser útiles para discriminar objetos o clases de imagen (Rosebrock, 2017).

Para lograr dicho objetivo, se han utilizado descriptores de imagen, raramente se utilizaban directamente los valores de intensidad de los píxeles como la entrada del modelo clasificador, como ya es habitual con *deep learning*. Para cada imagen en el *dataset* se realizaba la extracción de descriptores, proceso en el cual se retornaba un vector que cuantificaba los contenidos de la imagen. Dichos descriptores permitían representar texturas, forma y color (histogramas, correlogramas) (Rosebrock, 2017).

En el caso de *deep learning*, y específicamente las CNN, en lugar de definir un conjunto de reglas y algoritmos para la extracción de descriptores de la imagen, estos descriptores son aprendidos automáticamente en el proceso de entrenamiento (Rosebrock, 2017).

Dada una imagen, se suministran las intensidades de sus píxeles como entradas a la CNN. Una serie de capas ocultas extraen características de la imagen, escalonadamente, en forma jerárquica. Primeramente, solo se identifican por las capas ocultas más bajas regiones de bordes y esquinas. La combinación de esquinas y contornos permite la identificación de partes de objeto más abstractas en las capas superiores. La capa de salida se utiliza para clasificar la imagen y obtener la etiqueta de clase deseada.

1.7.4: Comparativa entre Máquinas de soporte vectorial, *Deep learning* y Enfoque bayesiano.

Para determinar qué tipo de algoritmo de clasificación se utilizará, se realizó una comparación entre los más usados actualmente, el cual se refleja en la siguiente tabla.

Tabla 1. Tabla comparativa entre algoritmos.

Fuente: (Elaboración propia).

Algoritmos	Dimensionalidad	Espacio en memoria	Tiempo de ejecución	Adaptación de descriptores
Deep learning	Capas ocultas transforman a espacios de cualquier dimensión.	Son representados por parámetros en una forma analítica.	Sigue el mismo patrón que el enfoque bayesiano, aunque necesitan un número de multiplicaciones y adiciones en tiempo de evaluación, lo que las hace lentas.	Adapta automaticamente los descriptores de color.

SVM	*Los kernels*[2] transforman a espacios de dimensión muy superior.	Solo necesita almacenar en memoria los vectores de soporte.	Elevada complejidad temporal, se ha logrado disminuir a O(m^2) [m: cantidad de elementos en el conjunto de entrenamiento]	Necesita adaptar los descriptores de color. Por lo que le resta tiempo en ejecución y memoria.
Enfoque bayesiano	*Los kernels* resuelven problemas de gran dimensionalidad.	Son representados por tablas de probabilidad condicional.	Sigue el mismo patrón que las redes neuronales pero pueden ser fácilmente implementadas como una sencilla tabla.	Necesita adaptar los descriptores de color. Por lo que le resta tiempo en ejecución y memoria.

Luego de haber profundizado las principales características de estos y haberlos comparado mediante los indicadores planteados, se decidió escoger *deep learning* como método de identificación a utilizar, principalmente por la ventaja de poder aprender automáticamente los descriptores de la imagen.

A pesar de que el *deep learning* parece una técnica nueva e innovadora, realmente es un tema en el que se lleva investigando más de 30 años. En el año 2012 un grupo de investigación de la Universidad de Toronto obtuvo unos resultados sorprendentes en la ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*), la principal competición de reconocimiento de imágenes. Desde entonces este tipo de técnicas ha tomado un importante papel en muchas áreas de la inteligencia. Por ello, empresas como Google, IBM y Facebook están invirtiendo en estas técnicas y contratando a los mejores investigadores.

[2] Se utilizan para transformar los vectores de entrada de n-dimensiones en vectores de dimensión más alta donde las clases puedan ser linealmente separables, lo que aumenta la capacidad computacional en las máquinas de aprendizaje lineal (Bernal de Lázaro, y otros, 2011) .

En muchas otras competiciones de aprendizaje automático, las técnicas de *deep learning* superan a otras técnicas (Simonyan, y otros, 2015).

1.8: Metodología de software.

Entre las tareas más difíciles de la ingeniería se encuentra el desarrollo de software, debido a que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. En una primera parte contamos con aquellas metodologías más tradicionales o robustas que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir y las herramientas y notaciones que se usarán. En la última parte tenemos las metodologías ágiles o ligeras las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas (Hernández, y otros, 2013).

Posteriormente al estudio de las metodologías de desarrollo y análisis de sus características, etapas de desarrollo, ventajas que ofrecen y siguiendo la línea de desarrollo del software PANDOC, se determina utilizar una metodología ágil, dado que la prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software; el cliente es parte del equipo de desarrollo.

Metodología de software XP

La metodología de software programación extrema (XP, según sus siglas en inglés) se utiliza para establecer el control y utilizar un marco de trabajo definido y de probada eficiencia. Una de las prácticas más significativas que posee, es que con XP es posible simplificar el diseño para agilizar el desarrollo, facilitar el mantenimiento y descartar las ideas que no se necesiten. Al realizar pruebas unitarias frecuentemente permite descubrir fallos debido a cambios recientes en el código. Además, esta metodología ha sido la utilizada durante el desarrollo del software PANDOC.

Entre las características más significativas de XP se encuentran (Hernández, y otros, 2013):

- Programación en pares: consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

- Pruebas unitarias: se basa en las pruebas realizadas a los principales procesos con el objetivo de detectar futuros errores.

- La tendencia de entregar software en espacios de tiempo cada vez más pequeños con exigencias de costos reducidos y altos estándares de calidad.

- Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

XP plantea cuatro fases para el desarrollo:

- Planificación: Durante esta etapa se lleva a cabo el proceso de identificación y confección de las Historias de Usuario (HU).

- Diseño: Durante esta etapa se crea un diseño evolutivo que va mejorando incrementalmente y que permite hacer entregas pequeñas y frecuentes de valor para el cliente, basado principalmente en el desarrollo de las tarjetas Clase-Responsabilidad- Colaboración (CRC).

- Desarrollo: En esta fase se realiza la implementación de las HU que fueron seleccionadas por cada iteración. Al inicio se lleva a cabo un chequeo del plan de iteraciones por si es necesario realizar modificaciones. Como parte de este plan se crean tareas de ingeniería para ayudar a organizar la implementación exitosa de las HU.

- Pruebas: Esta fase permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones. XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñadas por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñada por el cliente final.

1.9: Herramientas de desarrollo.

En este epígrafe se describen las tecnologías y herramientas a utilizar durante el proceso de desarrollo del software, así como la herramienta para el modelado, el entorno integrado de desarrollo y se argumenta acerca de las características y beneficios que ofrece el lenguaje de programación seleccionado.

1.9.1: Visual Paradigm.

Como herramienta para el modelado de la solución se ha seleccionado Visual Paradigm for UML (VP-UML v8.0), una de las líderes del mercado de las herramientas de Ingeniería de Software Asistida por Computadora (CASE, según sus siglas en inglés).

VP-UML v8.0 soporta los principales estándares de la industria tales como el Lenguaje de Modelado Unificado (UML, según sus siglas en inglés), SysML, BPMN, XMI, entre otros. Entre las bondades de Visual Paradigm destacan: ofrecer un conjunto completo de herramientas que ofrece a los equipos de desarrollo de software todo lo necesario para la captura de requisitos, planificación de software, planificación de controles, modelado de clases y modelado de datos, así como brindar interoperabilidad entre diagramas ya que es capaz de exportar los diagramas de un modelo a otro con mucha facilidad, ahorrando de esta manera tiempo, lo cual es crucial para el desarrollo. Hace posible la generación de código Java desde los diagramas (Started, 2013).

1.9.2: Lenguajes de Programación.

En la fase de desarrollo de la herramienta se eligió como lenguaje a Java. Este lenguaje en sí mismo heredó sintaxis de C y C++, pero elimina herramientas de bajo nivel y tiene un modelo de objetos más simple. La selección de dicho lenguaje se debe a que es puro orientado a objetos, lo que propina una gran reusabilidad, independencia de la plataforma, esto significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware (Started, 2013).

Resaltar que se cuenta con vasta documentación y una amplia comunidad sobre este, además las interfaces del software PANDOC están implementadas en este lenguaje.

Java presenta características como:

- Interpretado: se ejecuta en una máquina virtual.
- Robusto: administra la memoria de la computadora para que el programador no se tenga que preocupar por ello, además de realizar verificaciones en busca de errores lo mismo en tiempo de compilación que en tiempo de ejecución.

- Portable: un programa compilado de Java puede ser utilizado por cualquier computadora que tenga implementado el intérprete de Java, ya que su código compilado es interpretado.

- Simple: se enfoca en el contexto de los lenguajes orientados a objetos y elimina la complejidad de otros lenguajes como C.

- Multiproceso: puede ejecutar diferentes líneas de código al mismo tiempo.

- Dinámico: no es necesario que compile todas las clases de un programa para que este funcione. Al efectuar al menos un cambio en alguna de las clases, Java se encarga de realizar un enlace dinámico o una carga dinámica para encontrar las clases.

1.9.3: Entorno integrado de desarrollo.

IntelliJ IDEA es un Entorno de Desarrollo Integrado (IDE) para el desarrollo de programas informáticos. Es desarrollado por JetBrains (anteriormente conocido como IntelliJ), y está disponible en dos ediciones: edición para la comunidad y edición comercial. IntelliJ IDEA no está basada en Eclipse como MyEclipse u Oracle Enterprise Pack para Eclipse.

La primera versión de IntelliJ IDEA fue liberada en enero 2001, y en aquel momento fue uno de los primeros IDE Java disponibles con navegación avanzada de código y capacidades de refactorización de código integrado (Davydov, y otros, 2005).

En un informe de Infoworld en 2010, IntelliJ recibió la puntuación más alta entre las cuatro mejores herramientas de programación de Java: Eclipse, IntelliJ IDEA, NetBeans y Oracle JDeveloper (Binstock, 2010).

Esta plataforma crea IDEs sensibles al idioma con un conjunto completo de componentes, que incluye (Binstock, 2010):

- Sistema de archivos virtual.
- Marco de la interfaz de usuario (sistema de acción, ventanas de herramientas)
- Editor de texto.
- Análisis sintáctico, árboles de sintaxis abstracta y otra infraestructura específica del idioma.

- Marcos para la implementación de navegación, terminación de código, inspecciones, intenciones, refactorizaciones.
- Integración de control de versión.
- Marco de depuración.
- Corredor de pruebas de gráfico por unidad.

La plataforma IntelliJ es de código abierto y el código está cubierto por la licencia Apache 2.0. Esto significa que puede construir productos tanto de código abierto como comerciales sobre la plataforma sin pagar regalías a JetBrains.

IntelliJ no es un marco de aplicación de escritorio general, esta solo está diseñada para ser utilizada para aplicaciones similares a IDE. Básicamente, si una aplicación requiere un editor de código, existe una gran posibilidad de que la plataforma IntelliJ sea adecuada como marco subyacente (Davydov, y otros, 2005).

1.9.4: MySQL.

El sistema de base de datos a utilizar es MySQL. Está escrito en C y C++, y se destaca por su adaptación a diferentes entornos de desarrollo, permitiendo su interacción con los lenguajes de programación como PHP, Perl y Java, y su integración en distintos sistemas operativos. MySQL sigue políticas de código abierto, lo cual permite que su utilización sea gratuita e incluso se pueda modificar con total libertad, pudiendo descargar su código fuente, lo que ha favorecido su desarrollo y continuas actualizaciones (Márquez, y otros, 2015).

MySQL es un sistema de administración relacional de bases de datos. Una base de datos relacional archiva datos en tablas separadas en vez de colocar todos los datos en un gran archivo. Esto permite velocidad y flexibilidad. Las tablas están conectadas por relaciones definidas que hacen posible combinar datos de diferentes tablas sobre algún pedido.

MySQL tiene un código fuente abierto que significa que es posible para cualquier persona usarlo y modificarlo. Este usa como licencia a GPL/GNU para definir qué puede hacer y qué no puede hacer con el software en diferentes situaciones (Urlocker, 2010).

Conclusiones del capítulo

El estudio realizado sobre la identificación de imágenes de estructura ocular provenientes de la lámpara de hendidura, facilitó una mejor comprensión del campo de acción de la investigación. El análisis de las diferentes técnicas de identificación de características en imágenes, permitió seleccionar a *deep learning* como método de identificación debido a su potencial para reconocer patrones de información en imágenes de cualquier tipo. La metodología de software seleccionada para guiar el proceso de desarrollo, sigue la línea de confección de los sistemas elaborados por el grupo de investigación AIRI. También el empleo de la herramienta CASE Visual Paradigm ayuda a automatizar el proceso de desarrollo del software y el marco de trabajo IntelliJ IDEA facilita la construcción de la solución propuesta.

CAPÍTULO 2. Análisis y diseño de la propuesta de solución.

En este capítulo se describen las actividades a seguir para el proceso de desarrollo de la propuesta de solución. En este se hace un enfoque sobre el análisis y diseño, describiendo la arquitectura seleccionada, así como los requisitos funcionales y no funcionales, en conjunto con los artefactos que plantea la metodología seleccionada. Además, se detalla la fase inicial de la metodología XP, siendo esta la planificación donde se obtienen artefactos importantes, como son las historias de usuario, plan de iteraciones, plan de duración de iteraciones, plan de entregas y tarjetas Clase-Responsabilidad-Colaboración (CRC).

2.1: Arquitectura VGG-16

Durante el entrenamiento de una red neuronal se calculan los pesos óptimos para la clasificación, asociados a las conexiones entre neuronas. Después de entrenarla podemos guardar los pesos, así como la arquitectura de la red. Por otra parte, cabe tener en cuenta que el entrenamiento de una red es largo y tedioso, y requiere una gran cantidad de recursos a nivel computacional. Por este motivo, una forma de reducir el consumo de recursos es partir de una red pre-entrenada (Lerch, 2018).

Dentro de las redes pre-entrenadas para trabajar con *deep learning* se encuentran las redes VGG-16, modelo de *deep learning* desarrollado por el Grupo de Geometría Visual (VGG por sus siglas en inglés) de la Universidad de Oxford. Dicho modelo sigue el esquema arquetípico de las redes convolucionales clásicas: una serie de capas de convoluciones, capas de *max pooling*[3], capas de activación y finalmente algunas capas de clasificación completamente conectadas. VGG-16 utiliza en sus capas ocultas una función de activación de Rectificación no Lineal (ReLu). Es el modelo actualmente más utilizado por la comunidad de trabajo en imágenes para la extracción de características y clasificación (Kasthurirangan, y otros, 2017).

La red entrenada que se utiliza en la propuesta de solución, sigue el enfoque de la arquitectura antes mencionada, por lo que presenta 16 capas ocultas convolucionales y

[3] La operación de max-pooling encuentra el valor máximo entre una ventana de muestra y pasa este valor como resumen de características sobre esa área. Como resultado, el tamaño de los datos se reduce por un factor igual al tamaño de la ventana de muestra sobre la cual se opera (Ciseran, y otros, 2015) .

aplica una convolución 3x3 y un *pooling* espacial de tamaño 2x2. Teniendo como entrada las imágenes a las cuales se le realizará la clasificación, y como salida, las imágenes ya clasificadas en oblicua, retroiluminación o desconocida.

2.2: Descripción de la solución.

Para darle solución al problema planteado, que no es más que realizar la identificación de estructuras oculares en imágenes provenientes de la lámpara de hendidura, se estableció una relación con respecto a las actividades relacionadas con dicha propuesta, evidenciándose en la figura que se muestra a continuación.

A continuación, se muestran los pasos a realizar en la propuesta de solución.

1. Obtener imágenes.
2. Identificar si es una imagen oblicua o en retroiluminación con OCP.
3. Identificar si es ocblicua o en retroiluminación.
4. Almacenar imágenes.
5. Desechar imágenes de poca relevancia.

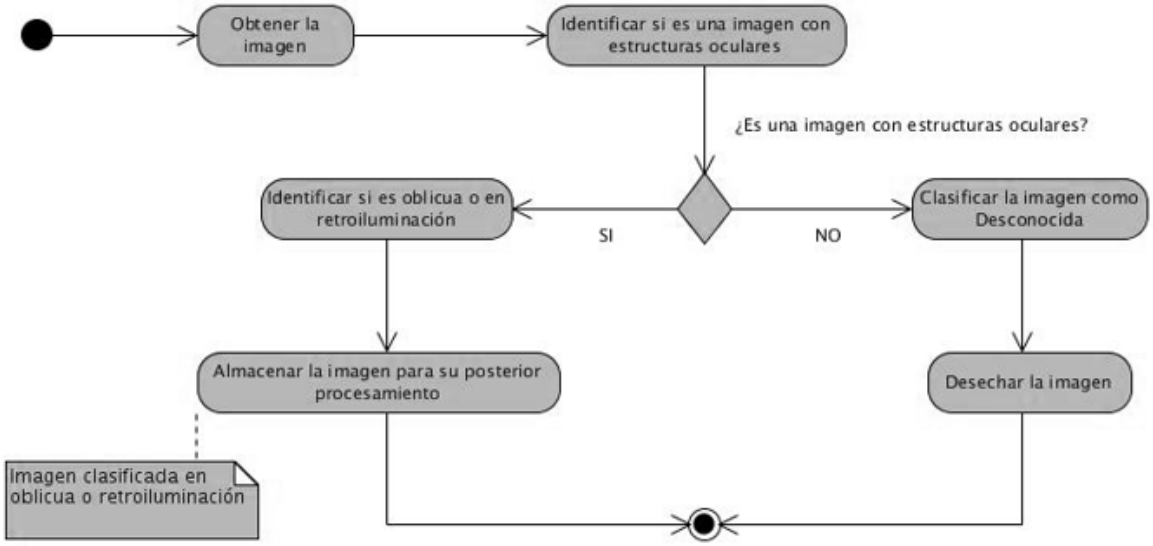


Figura 6. Descripción de la solución.

Fuente: (Elaboración propia).

2.3: Fase de planificación.

Durante el transcurso de esta etapa se realiza el proceso de identificación y elaboración de las historias de usuario, además el equipo de trabajo se familiariza con las tecnologías y herramientas seleccionadas para el desarrollo. El cliente define el nivel de prioridad con que se deben implementar las HU, así como la estimación del esfuerzo que costará implementar las mismas. El resultado de la presente fase es un plan de entregas donde se realiza una estimación de las versiones que tendrá el producto en su elaboración, de forma tal que sea una guía durante el desarrollo (Beck, y otros, 2015).

2.3.1: Especificación de requisitos.

"Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste" (Somerville, 2005). La calidad con que se realiza la selección de estos contribuye en la calidad de todo el proceso de desarrollo, reflejándose en las fases restantes. Además, estos contribuyen a tomar mejores decisiones en los aspectos de diseño y arquitectura.

Requisitos funcionales

Un requisito funcional (RF) define una función del sistema de software o sus componentes. Una función es descrita como un conjunto de entradas, comportamientos y salidas. Los requerimientos funcionales pueden ser: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que se supone que un sistema debe cumplir. Estos son complementados por los requisitos no funcionales, que se enfocan en cambio, en el diseño o la implementación (Somerville, 2005).

A continuación, se evidencian los RF identificados:

- **RF 1**: Cargar la imagen.
- **RF 2**: Identificar imágenes de estructura ocular
- **RF 3**: Identificar si es una imagen oblicua.
- **RF 4**: Identificar si es una imagen en retroiluminación.
- **RF 5**: Almacenar imágenes obtenidas.

Requisitos no funcionales

Los requisitos no funcionales (RNF) son propiedades o cualidades que el sistema debe tener. Estas propiedades o cualidades se refieren a las características que hacen al sistema estable, usable, rápido, confiable y escalable (Somerville, 2005).

A continuación, se muestran los RNF identificados:

Requisito de software

- **RNF1:** Tener instalada la máquina virtual de Java 1.6.

Requisitos de hardware: Las propiedades mínimas de la PC deben ser:

- **RNF2:** Microprocesador: Intel Core i3 o superior.
- **RNF3:** Memoria RAM: 4 GB DDR4.
- **RNF4:** Tarjeta de Red: Fast-Ethernet 100 MB/s.

Requisitos de diseño

- **RNF5:** El sistema debe ofrecer una interfaz es de color gris, negro y azul. El tipo de letra en las interfaces Open Sans, SansSerif siguiendo con el diseño del PANDOC

Requisitos de interfaz

- **RNF6:** El sistema ofrece una interfaz amigable y fácil de operar.

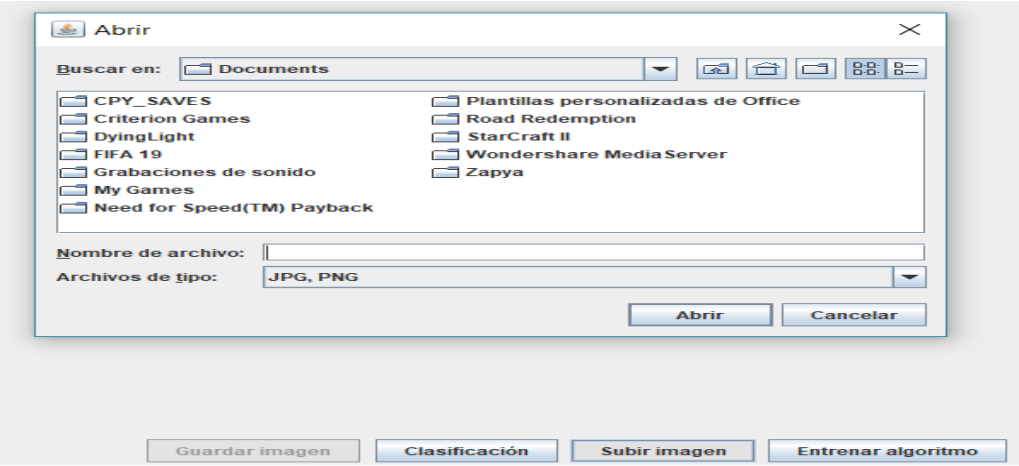
2.3.2: Historias de usuario.

Las HU es la técnica utilizada en XP para especificar los requisitos del software; en ellas el cliente describe brevemente las características que el sistema debe poseer. Se realiza una por cada característica principal del sistema. En cualquier momento pueden reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada HU es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (Wesley, y otros, 2002). La tabla 2, muestra una de las HU elaboradas en la investigacion.

Tabla 2. Historia de usuario: Cargar imagen.

Fuente: (Elaboración propia).

Historia de Usuario	
Número: 1	**Nombre del requisito:** Cargar imagen

Modificación a la Historia de Usuario: 0	
Usuario: Reyder Cruz de la Osa	**Iteración asignada:** 1
Prioridad : Alta	**Tiempo estimado:** 5 días
Riesgo en desarrollo: medio	**Tiempo real:** 1 semana
Programador responsable: Claudia Fernández Peña	
Descripción: Cuando el usuario accede al programa para hacer uso de la identificación de estructuras oculares necesita una imagen, por lo que se hace necesario poder cargar imágenes provenientes de la lámpara de hendidura.	
Observaciones: La imagen debe estar en el formato JPG, PNG.	
Prototipo de Interfaz: 	

2.3.3: Estimación de esfuerzos por historias de usuario.

En el presente epígrafe se realiza la estimación del esfuerzo por HU, donde se hace necesario tener en cuenta que estas deben ser programadas en un tiempo de una a tres semanas. Si la estimación es superior a tres semanas, se divide en dos o más HU. Si es menor de una semana, se combina con otra HU. Estas estimaciones permiten tener una medida de la velocidad del proyecto y ofrecen una guía a la cual ajustarse. Los resultados estimados se muestran en la siguiente tabla.

Tabla 3. Estimación de esfuerzos (semanas).

Fuente: (Elaboración propia).

Historia de usuario	Puntos de estimación (semanas)

HU 1: Cargar imagen.	1
HU 2: Identificar imágenes de estructura ocular.	2
HU 3: Identificar si es una imagen oblicua.	3
HU 4: Identificar si es una imagen en retroiluminación.	2
HU 5: Almacenar imágenes obtenidas.	3

2.3.4: Plan de iteraciones.

Una vez finalizadas las HU se debe crear un plan de iteraciones, indicando cuáles se desarrollarán en cada iteración, mostrándose la información resultante en la tabla 4. A continuación, se muestra cómo queda definido el plan de iteraciones para la solución propuesta.

Iteración 1

En esta iteración se realizarán los procesos de cargar la imagen y entrenar el modelo para las operaciones restantes.

Iteración 2

En esta iteración se identificará la clasificación perteneciente de las imágenes anteriores.

Iteración 3

En esta iteración se almacenarán las imágenes clasificadas para su posterior procesamiento.

Tabla 4. Plan de iteraciones.

Fuente: (Elaboración propia).

Iteraciones	Orden de las historias de usuario a implementar	Duración de las iteraciones (semanas)
Iteración 1	Cargar imagen.	3 semanas
	Identificar imágenes de estructura ocular.	
Iteración 2	Identificar si es una imagen oblicua.	5 semanas

	Identificar si es una imagen en retroiluminación.	
Iteración 3	Almacenar imágenes obtenidas.	3 semanas
Total		11 semanas

2.3.5: Plan de entrega.

El plan de entrega es el cronograma que define cuáles HU serán agrupadas para conformar una entrega, y el orden en que se realizarán estas (Joskowicz, 2008). Este plan se encarga de definir el número de liberaciones que se realizarán en el transcurso del proyecto y las iteraciones que se requieren para desarrollar cada una de ellas. En la tabla 5 se muestra el plan de entrega conformado para la propuesta.

Tabla 5. Plan de entrega.

Fuente: (Elaboración propia).

	Iteración 1	**Iteración 2**	**Iteración 3**
Cantidad de HU	1	3	1
Fecha de entrega	11/3/2019	15/4/2019	6/5/2019

2.4: Fase de diseño.

La metodología de desarrollo XP plantea prácticas especializadas que accionan directamente en la realización del diseño para lograr un sistema robusto y reutilizable. Se trata en todo momento de conservar su simplicidad, o sea, crear un diseño evolutivo que vaya mejorando y que permita hacer entregas pequeñas y frecuentes de valor para el cliente, basado principalmente en el desarrollo de las tarjetas CRC.

2.4.1: Tarjetas CRC.

Las tarjetas CRC son utilizadas para representar las responsabilidades de las clases y sus interacciones. Estas tarjetas permiten trabajar con una metodología basada en objetos, permitiendo que el equipo de desarrollo completo contribuya en la tarea del diseño. En cada tarjeta CRC el nombre de la clase se coloca a modo de título, las responsabilidades se colocan a la izquierda y las clases que se implican en cada responsabilidad a la derecha,

en la misma línea que su requerimiento correspondiente. Dichos resultados se observan en la tabla 6.

Tabla 6. Tarjeta CRC.

Fuente: (Elaboración propia).

Clase: Index	
Responsabilidades	**Colaboradores**
Se encarga de verificar las condiciones para que la imagen sea almacenada.	ModelController
Se encarga de todo el proceso de entrenamiento del modelo a utilizar.	DeepLearningController
Se encarga de validar si la imagen ha sido almacenada correctamente.	ModelRepository
Se encarga de definir los atributos necesarios para el almacenamiento de la imagen.	EntityModel

2.4.2: Patrones de diseño.

Los patrones de diseño constituyen una descripción de la interacción entre las clases, con el objetivo de resolver un problema de diseño general en un determinado contexto. El uso de patrones permite la estandarización de la forma en que se realiza el diseño de manera que el código pueda ser reutilizable, y se le pueda dar mantenimiento cuando se requiera. A continuación, se muestran los patrones de diseño utilizados.

Patrones GRASP

Los patrones GRASP (*General Responsibility Assignment Software Patterns*) describen los principios fundamentales del diseño de objeto y la asignación de responsabilidades, expresados como patrones. Es un sistema orientado a objetos, se compone de objetos que envíen mensajes a otros objetos para que lleven a cabo las operaciones requeridas (Larman, 2003).

Experto

El patrón Experto establece el principio básico de asignación de responsabilidades. Indica que la responsabilidad de la creación de un objeto, o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo (Larman, 2003). Este patrón se ve reflejado en la figura 7, donde se hace referencia al método evaluateModel, debido a que las operaciones de este, recaen en las clases restantes que colaboran con la clase DeepLearningController.

```
private void evaluateModel(Model model, DataSetIterator trainIter, DataSetIterator testIter, JTextArea textAreaSummary){
    Evaluation eval;
    int iter = 0;
    int other = 0;
    boolean hasEval = false;
    while (other < 5) {
        while (trainIter.hasNext()) {
            DataSet trained = trainIter.next();

            if(model instanceof MultiLayerNetwork)
                ((MultiLayerNetwork)model).fit(trained);
            else
                ((ComputationGraph)model).fit(trained);

            if (iter >= 19 && !hasEval) {
                logger(textAreaSummary, text: "Evaluar modelo en la iteración " + iter + " ....");

                if(model instanceof MultiLayerNetwork)
                    eval = ((MultiLayerNetwork)model).evaluate(testIter);
                else
                    eval = ((ComputationGraph)model).evaluate(testIter);
```

Figura 7. Clase DeepLearningController.

Fuente: (Elaboración propia).

Creador

Es el encargado de guiar la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento (Larman, 2003). En dicha aplicación se evidencia en la clase ModelController, siendo esta la responsable de varias de las asignaciones de algunas clases restantes. La figura 8, muestra un ejempo de este patrón..

```
table.setDefaultRenderer(Object.class, new Render());
table.getTableHeader().setDefaultRenderer(new SimpleHeaderRenderer());

DefaultTableModel dt = isCellEditable(row, column) → { return false; };

dt.addColumn( columnName: "Imagen");
dt.addColumn( columnName: "Clasificación");
dt.addColumn( columnName: "Creado");
dt.addColumn( columnName: "Número");

ArrayList<EntityModel> list = data.listElements();

for (EntityModel element : list) {
    Object[] file = new Object[4];
    file[0] = this.getImage( dirUploads: dirUploads + element.getPhoto());
    file[1] = element.getClassification();
    file[2] = element.getCreated().toString();
    file[3] = element.getId();

    dt.addRow(file);
```

Figura 8. Clase ModelController.

Fuente: (Elaboración propia).

Controlador

El patrón Controlador sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, manejando los eventos de entrada de dicha interfaz (Larman, 2003). La figura 9 muestra la clase Index, donde se manifiesta el uso de este patrón en el método clasificationButton, siendo este el encargado de la creación de varios controladores que se encargan de manejar eventos.

```
classificationButton.addActionListener((e) → {

        updateJFrame(LOADING_GIF,  loading: true);

        try {
            int cl;
            if (computationGraphRadioButton.isSelected())
                cl = deepLearningController.classificationByGraph(new File(absolutePath), textAreaSummary, acuracy);
            else
                cl = deepLearningController.classificationByMultiLayer(new File(absolutePath), textAreaSummary, acuracy);

            showResponse(cl, deepLearningController.getOutput1(), deepLearningController.getOutput2());
            updateJFrame(absolutePath,  loading: false);

            if (cl == 0 || cl == 1) {
                saveButton.setEnabled(true);
            }

        } catch (Exception ex) {
            System.out.println(ex.getMessage());

            updateJFrame(ACTION_ERROR,  loading: true);
```

Figura 9. Clase Index.

Fuente: (Elaboración propia).

2.4.3: Modelo de datos.

Un modelo de datos es un lenguaje orientado a hablar de una base de datos. Típicamente un modelo de datos permite describir (González, 2016):

- Las estructuras de datos: El tipo de los datos que hay en la base de datos y la forma en que se relacionan.
- Las restricciones de integridad: Un conjunto de condiciones que deben cumplir los datos para reflejar la realidad deseada.
- Operaciones de manipulación de los datos: típicamente, operaciones de agregado, borrado, modificación y recuperación de los datos de la base.

La figura 10 hace referencia al modelo de datos de la presente solución. En esta se refleja los atributos pertenecientes al mismo, especificándose el nombre y tipo de cada uno de ellos.

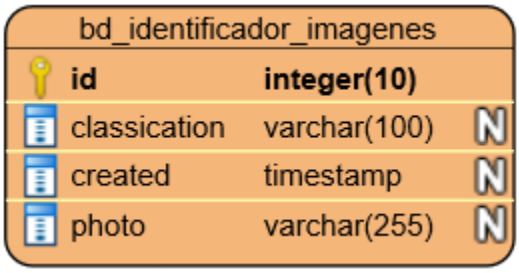


Figura 10. Modelo de datos.

Fuente: (Elaboración propia).

Conclusiones del capítulo

El empleo de la metodología XP, permitió organizar el desarrollo de la solución y generar los productos de trabajo necesarios correspondientes a las etapas de planeación, diseño y desarrollo. La descripción de la propuesta de solución, permitió obtener una visión de los flujos de información y las actividades que componen el algoritmo a implementar. Además, con la obtención de los requisitos, se seleccionó la información necesaria para el desarrollo de la propuesta. De igual forma, la elección de la arquitectura a utilizar, facilitó un mayor entendimiento del problema en que se enmarca la investigación. Por último, con el uso de los patrones de diseño se facilitó el proceso de desarrollo, permitiendo así tener una estructura bien organizada de las clases para su posterior implementación.

CAPÍTULO 3. Validación de la propuesta de solución.

En este capítulo se realiza una evaluación de la calidad y fiabilidad de los resultados obtenidos en el desarrollo del algoritmo propuesto luego de la implementación de la propuesta de solución. Quedarán evidentes los resultados de las pruebas internas con el fin de verificar la solidez de la implementación. Por último, se realizará la aceptación con el cliente para verificar que se cumplan sus necesidades.

3.1: Pruebas de software.

Las pruebas de software son de vital importancia para comprobar que la aplicación desarrollada no presente problemas de ejecución y se encuentre libre de errores, ya que durante el proceso de desarrollo de software es frecuente que los desarrolladores, al programar las diferentes funcionalidades, obvien algunas posibilidades que pueden variar el resultado durante la ejecución del código; por tanto, estas pruebas constituyen la vía a través de la cual se asegura que el producto desarrollado está listo para ser entregado a los usuarios finales (Pressman, 2010).

3.1.1: Pruebas unitarias.

Una prueba unitaria es la verificación de una unidad de código (módulo) determinado dentro de un sistema. Estas pruebas aseguran que un módulo determinado cumpla con un comportamiento esperado en forma aislada antes de ser integrado al sistema, es decir, debe aprobar satisfactoriamente todos los casos de prueba definidos (Malfará, y otros, 2006).

Para la aplicación de estas pruebas se utilizará el método de caja blanca.

Método de caja blanca

Las pruebas de caja blanca se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. Se escogen distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa cerciorándose que se devuelvan los valores de salida adecuados (Pressman, 2010).

Las pruebas de caja blanca intentan garantizar que:

- Se ejecutan al menos una vez todos los caminos independientes de cada módulo.

- Se utilizan las decisiones en su parte verdadera y en su parte falsa.
- Se ejecuten todos los bucles en sus límites.
- Se utilizan todas las estructuras de datos internas.

Para la aplicación de dicho método se define la técnica de ruta básica, la cual se describe a continuación.

Ruta básica

Esta técnica permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un procedimiento y usar esta medida como guía para definir un conjunto básico de rutas de ejecución. Los casos de prueba derivados para revisar el conjunto básico tienen garantía para ejecutar todas las sentencias del programa, al menos una vez durante la prueba (Pressman, 2010).

Para ello se construye un grafo asociado a un método determinado cuyas aristas representan el flujo de control y cuyos nodos son las instrucciones y luego se calcula la complejidad ciclomática. Dicha métrica se calcula sobre un grafo, y se puede realizar mediante las siguientes fórmulas:

1. V(G)= R
2. V(G)= E – N + 2
3. V(G)= P + 1

Conociendo que:

- **G:** Grafo de flujo (grafo).
- **R:** El número de regiones contribuye a estimar el valor de la complejidad ciclomática.
- **E:** Número de aristas.
- **V(G):** Complejidad ciclomática.
- **N:** Número de nodos del grafo.
- **P:** Número de nodos predicados incluidos en el grafo.

En la siguiente figura, se muestra el método evaluateModel() al cual se le aplicará la técnica seleccionada, teniendo como resultado el grafo de flujo que muestra el total de caminos posibles por los cuales el flujo puede circular. En la figura 11 se muestra el código utilizado

para realizar esta prueba, mientras que en la figura 12, se evidencia el grafo de flujo obtenido.

```
private void evaluateModel(Model model, DataSetIterator trainIter, DataSetIterator testIter, JTextArea textAreaSummary){
    Evaluation eval;
    int iter = 0;
    int other = 0;
    boolean hasEval = false;                                    1
    while (other < 5) {                                         2
        while (trainIter.hasNext()) {                           3
            DataSet trained = trainIter.next();                 4

            if(model instanceof MultiLayerNetwork)              5
                ((MultiLayerNetwork)model).fit(trained);        6
            else
                ((ComputationGraph)model).fit(trained);         7

            if (iter >= 19 && !hasEval) {                       8  9
                logger(textAreaSummary, text: "Evaluar modelo en la iteración " + iter + " ....");   10

                if(model instanceof MultiLayerNetwork)          11
                    eval = ((MultiLayerNetwork)model).evaluate(testIter);   12
                else
                    eval = ((ComputationGraph)model).evaluate(testIter);    13

                logger(textAreaSummary, eval.stats());
                testIter.reset(); hasEval = true;               14
            }
            iter++;                                             15
        }
        trainIter.reset();
        other++;                                                16
    }                                                           17
}
```

Figura 11. Método evaluateModel().

Fuente: (Elaboración propia).

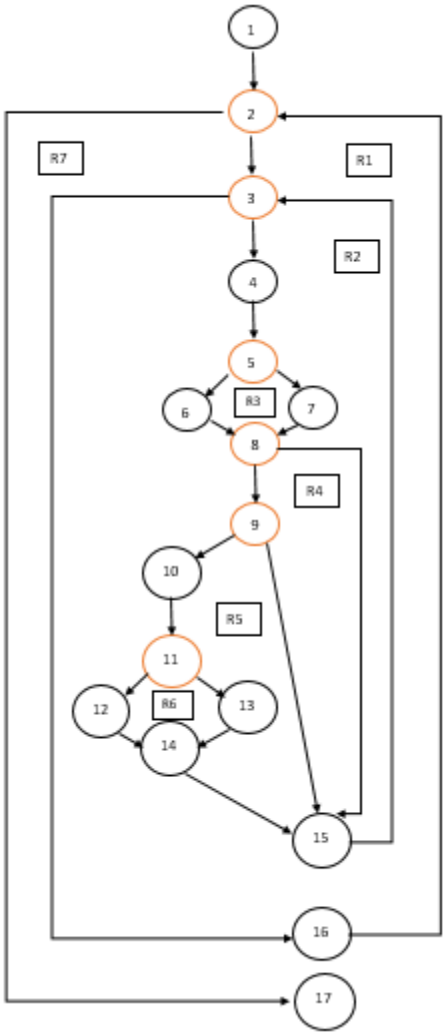


Figura 12. Grafo que representa el flujo del método evaluateModel().

Fuente: (Elaboración propia).

A partir del grafo de flujo del método evaluateModel(), la complejidad ciclomática es la siguiente:

- V(G) = 7
- V(G) = 22 - 17 + 2 = 7
- V(G) = 6 + 1 =7

Dado a que el cálculo de las fórmulas anteriormente expuestas arrojó el mismo resultado, se puede plantear que la complejidad ciclomática del método es 7. Este valor representa el número mínimo de casos de pruebas para el procedimiento tratado.

Rutas básicas identificadas:

- Camino #1: 1-2-17
- Camino #2: 1-2-3-16-2-17
- Camino #3: 1-2-3-4-5-6-8-9-10-11-12-14-15-3-16-2-17
- Camino #4: 1-2-3-4-5-7-8-15-3-16-2-17
- Camino #5: 1-2-3-4-5-6-8-9-10-11-13-14-15-3-16-2-17
- Camino #6: 1-2-3-4-5-6-8-9-10-11-13-14-15-3-16-2-17
- Camino #7: 1-2-3-4-5-7-8-9-10-11-13-14-15-3-16-2-17

Para cada ruta básica determinada se realiza un diseño de caso de prueba. A continuación, se muestra el caso de prueba para la ruta básica #2

Tabla 7. Caso de prueba para el camino #2.

Fuente: (Elaboración propia).

Caso de prueba para la ruta básica #2 (1-2-3-16-2-17)	
Descripción	Prueba si la iteración a atenderse ya ha sido entrenada.
Condición de ejecución	Necesita saber si la iteración a tratar ya ha sido entrenada.
Entrada	Mientras no haya sido evaluada y la cantidad de iteraciones sea menor que 5.
Resultado	Adicionar dicha iteración y reiniciar el entrenamiento.
Resultado de la prueba	Prueba satisfactoria

Resultados al aplicar la técnica de ruta básica

Esta técnica se aplicó a los métodos de las clases seleccionadas debido a que engloban las funcionalidades del sistema. Luego de haber aplicado esta práctica para comprobar la fiabilidad del código del algoritmo, así como también para elaborar los casos de prueba

necesarios, se logró asegurar que al instante de haber comprobado satisfactoriamente los casos de prueba, las rutas de este código se ejecutaron al menos una vez.

3.1.2: Pruebas funcionales.

En esta disciplina se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas (Rodríguez Sánchez, 2015).

Para la realización de estas pruebas se utilizó el método caja negra usando la técnica partición equivalente.

Método de caja negra

Las pruebas de comportamiento o también conocidas como prueba de caja negra, se centran en los requisitos funcionales del software, o sea, la prueba de caja negra permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Evalúa las funcionalidades del software (Pressman, 2010).

Este método se aplicó haciendo uso de la técnica partición equivalente. A continuación, se describen los resultados de esta técnica.

Técnica partición de equivalencia

Esta técnica divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. La partición de equivalencia se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada (Pressman, 2010).

En el Anexo 11, se especifica la descripción de las variables para el requisito Identificar imágenes de estructura ocular. Luego de definir el tipo de variables y las descripciones se procede a realizar el diseño de caso de pruebas para este requisito, donde se muestra su resultado en el Anexo 12.

Durante la ejecución de estas pruebas se realizaron tres iteraciones, donde los errores detectados fueron solucionados en su totalidad. Entre los errores detectados se encuentran:

errores ortográficos, de validación e interfaz. En la tabla 8 y en la figura 13, se presentan los resultados obtenidos.

Tabla 8. Resultado de las iteraciones al aplicar el método de caja negra.

Fuente: (Elaboración propia).

Iteraciones	No conformidades
Iteración 1	18
Iteración 2	8
Iteración 3	0

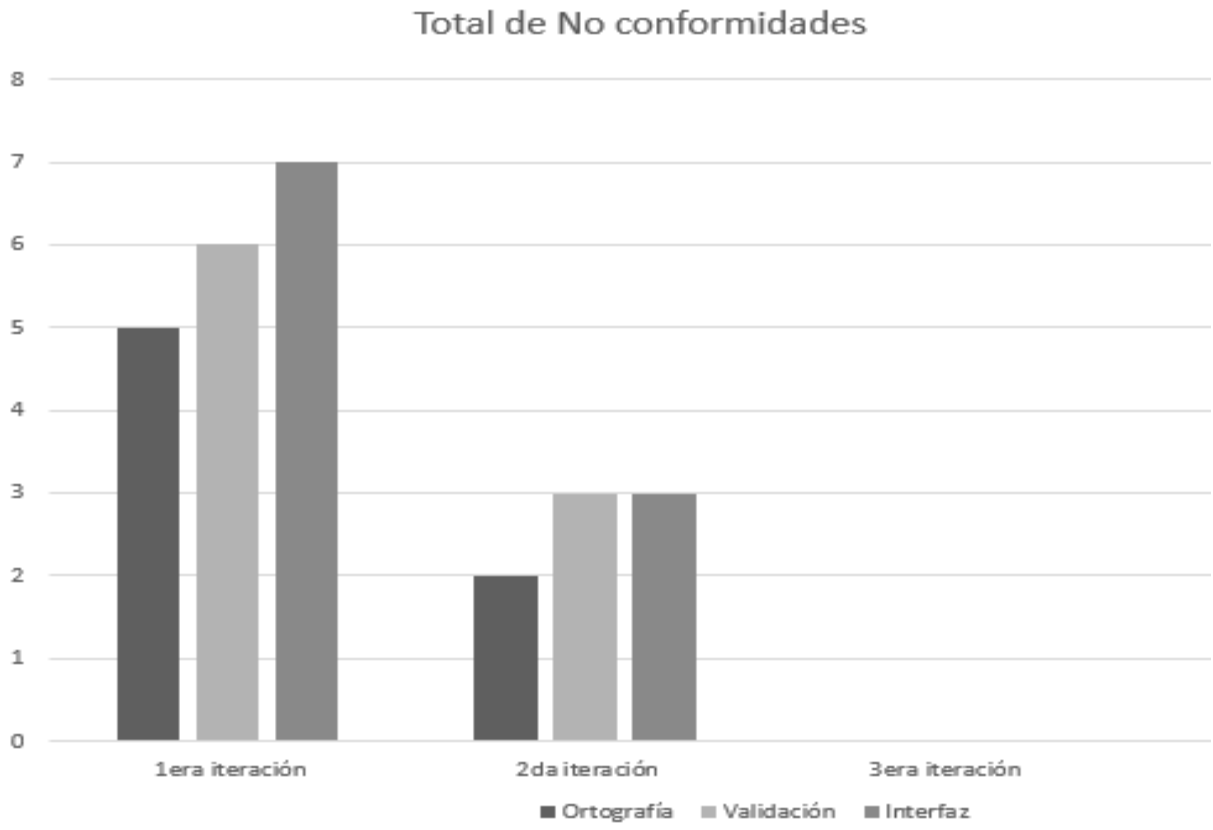


Figura 13. Resultados de la aplicación del método de caja negra.

Fuente: (Elaboración propia).

3.2: Validación de la solución.

Para el entrenamiento del algoritmo se utilizó un total de 161 imágenes de prueba, desglosándose en 41 oblicuas, 68 en retroiluminación y 52 desconocidas. Luego de haber sido entrenado, se realizó una ejecución para verificar el porciento de clasificación correcta con un conjunto de validación de 48 imágenes. Para ello se realizó la validación con el método ComputationGraph, ya que el trabajo con este se utiliza para medir la eficacia de un conjunto pequeño de imágenes. Para realizar dicha validación, se emplearon los niveles de predicción para la clasificación de estas, de los cuales se utilizó un umbral de 0.6 en un total de 5 iteraciones en todo el transcurso de desarrollo del algoritmo, estableciendo como efectividad la cantidad de imágenes correctamente clasificadas con respecto al total de ellas. Luego de haber realizado un total de 3 iteraciones, se pudo evidenciar que la clasificación logra un 67.71% de efectividad, debido a que del total de imágenes no se clasificaron correctamente las de clasificación “desconocidas”. En un segundo momento se realizaron 2 iteraciones, lo que arrojó como resultado que solo se clasificaron incorrectamente 6 imágenes desconocidas, mostrando un 96.37% de efectividad en la clasificación. En las figuras que se muestran a continuación, se evidencian los resultados obtenidos.

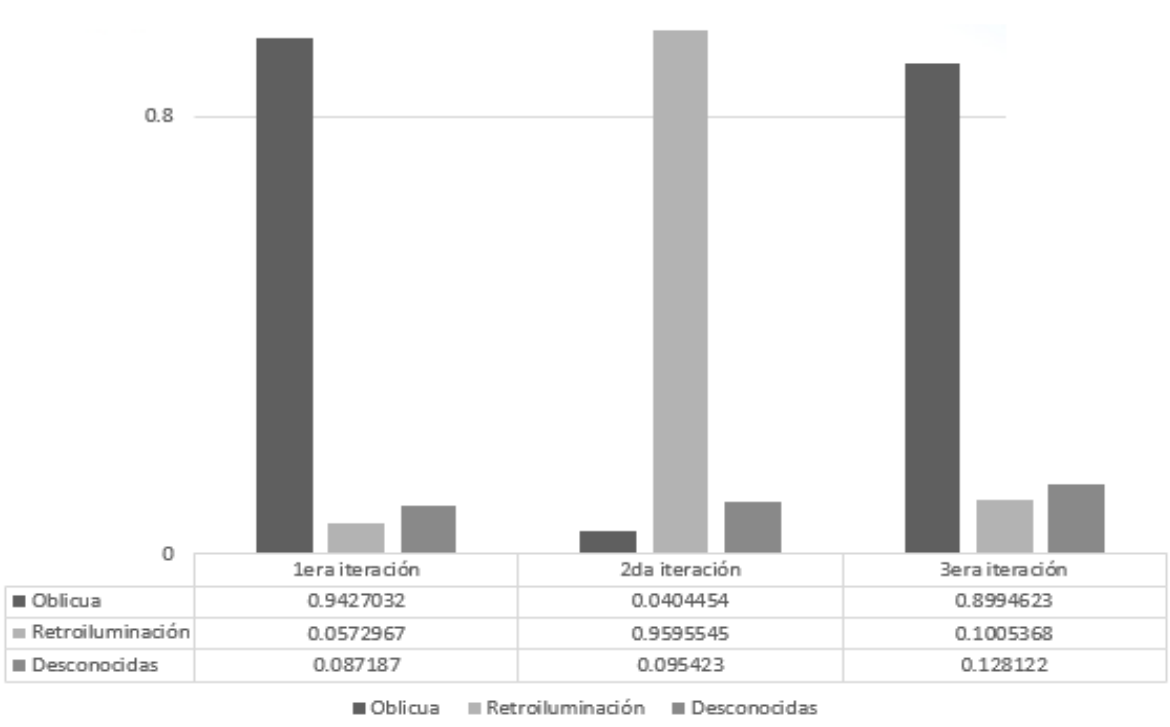


Figura 14. Primeros resultados arrojados de la prueba de validación.

Fuente: (Elaboración propia).

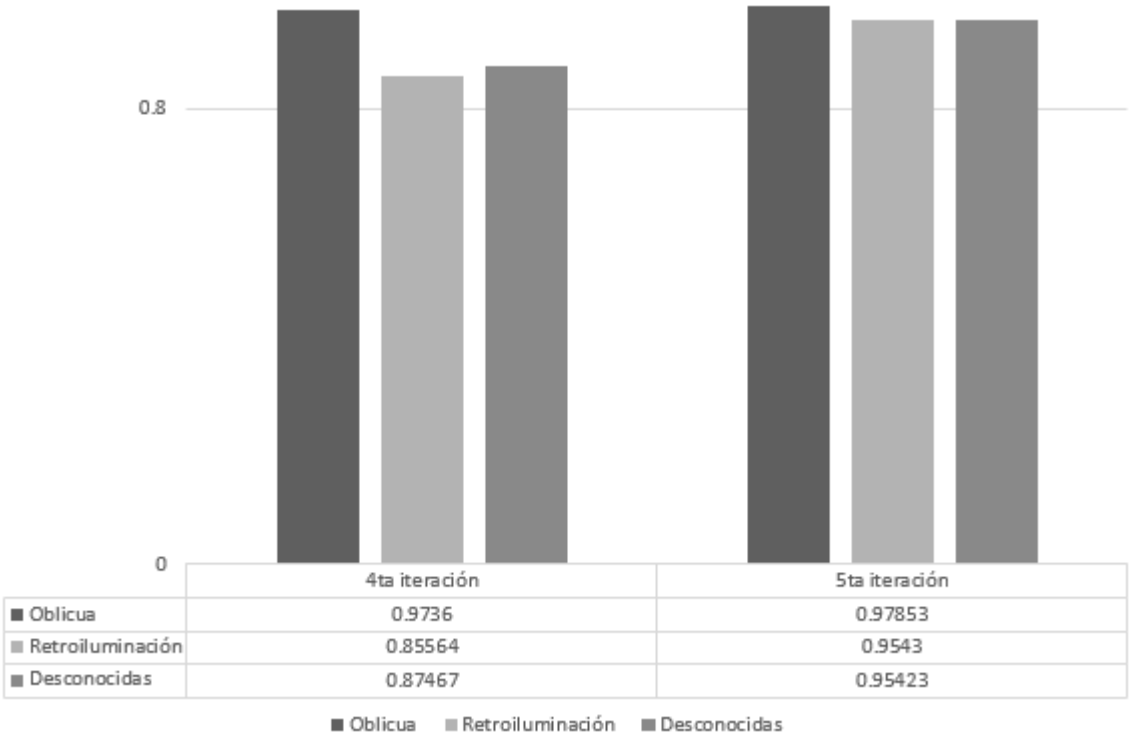


Figura 15. Últimos resultados arrojados de la prueba de validación.

Fuente: (Elaboración propia).

3.3: Pruebas de aceptación.

Es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido (Rodríguez Sánchez, 2015).

Para realizar estas pruebas se utilizó el mismo principio de las pruebas internas, permitiendo que sea el propio cliente el que las realice. Para esto, se realizaron 2 iteraciones y se obtuvo como salida el acta de aceptación. Los resultados alcanzados se evidencian en la siguiente figura.

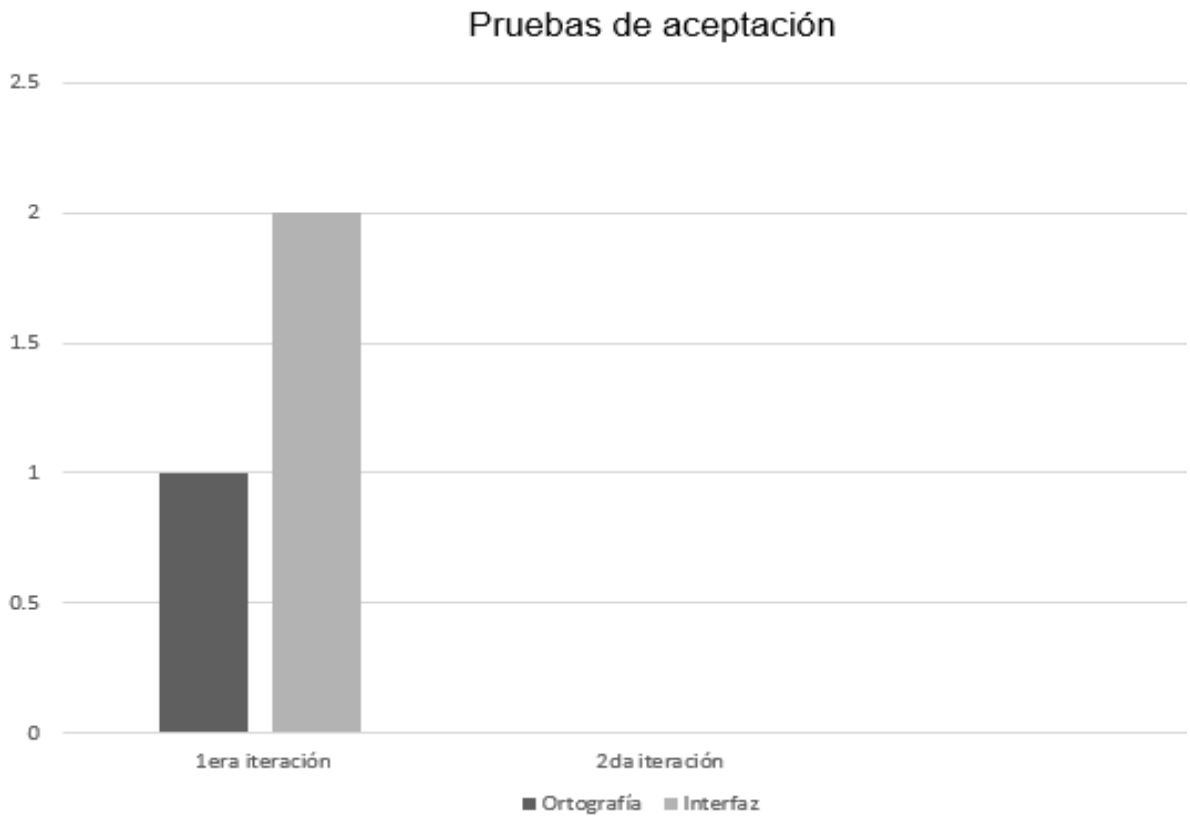


Figura 16. Resultados de las pruebas de aceptación.

Fuente: (Elaboración propia).

Conclusiones del capítulo

La ejecución de pruebas unitarias para la validación del código, reflejaron que las operaciones internas se ajustan correctamente y como resultado de las pruebas funcionales, se comprobó que las funciones son operativas a través de la interfaz del software, manteniendo así la integridad de la información externa. Después de entrenado el modelo de *deep learning* y utilizarlo para la clasificación de 48 imágenes, se obtuvieron excelentes resultados evidenciados en un 96.37% de efectividad.

CONCLUSIONES GENERALES

Al concluir la investigación para identificar imágenes de estructura ocular provenientes de la lámpara de hendidura se pudo arribar a lo siguiente:

- La identificación de los referentes teóricos en los que se sustenta la propuesta de solución sobre la identificación de imágenes médicas, permitió sentar las bases de la investigación.
- La especificación de los requisitos, así como el análisis y diseño de las funcionalidades correspondientes, permitieron obtener un punto de partida para garantizar los elementos necesarios para la implementación de las mismas.
- La implementación de las funcionalidades necesarias, favoreció la obtención de un software funcional, contribuyendo a la identificación de imágenes de estructura ocular provenientes de la lámpara de hendidura.
- El uso de *deep learning* para la identificación de estructuras oculares, arrojó un 96.37% de efectividad en el trabajo de clasificación en imágenes provenientes de la lámpara de hendidura.
- Con la validación de los resultados obtenidos, mediante las pruebas de software aplicadas, se pudo comprobar la calidad de los artefactos generados durante el desarrollo de la propuesta de solución.

RECOMENDACIONES

Se recomienda para versiones futuras de la propuesta de solución:

- Incorporar la opción de identificar tomogramas de Scheimpflug provenientes del PENTACAM.
- Incorporar la posibilidad de trabajar con los frames obtenidos de la lámpara de hendidura.
- Emplear otras de las arquitecturas para *deep learning.*

REFERENCIAS BIBLIOGRÁFICAS

Ackley D H, Hinton G E. 2014. *A learning Algorithm for Bolztmann Machines.* 2014.

Alonso, Carlos. 2015. *SVM : Máquinas de Vector Soporte.* 2015.

Alonso, M, Miranda C y Martin , N. 2016. *Chemical applications of neural networks : aromaticity of pyrimidine derivates.* 2016.

Alvarez Cancio, Michel, Hernández, Adrian y Rodríguez, Rafael. 2013. *PANDOC : Software para la cuantificación objetiva en la opacidad de la cápsula posterior mediante tomogramas de Scheimpflug.* 2013.

autores, Conjunto de. 2014. *Oracle. Java.* 2014.

Barr, Michael. 2015. *Getting to know the hardware. Programming Embedded System in C and C++.* s.l. : O'Reilly, 2015. 1565923545.

Beck, Kent, Martínez, Francisco Javier TR Zapata y MOLINA., Jesúz Rev. Téc García. 2015. Una explicación de la programación extrema: aceptar el cambio. 2015.

Bengio, Y, Courville, A y Vincent, P. 2013. *Representation Learning: A review and New Perspectives.* 2013.

Bernal de Lázaro, José, y otros. 2011. Estudio comparativo de clasificadores empleados en el diagnóstico de fallos de sistemas industriales. 2011, Vol. 14, 2.

Binstock, Andrew. 2010. *InfoWorld review: Top Java programming tools.* s.l. : InfoWorld, 2010.

Caparrini, Fernando Sancho. 2015. *Clasificación supervisada y no supervisada.* 2015.

Castro, Jose Luis Alba. 2015. *Máquinas de vector soporte.* 2015.

Chollet, Francois. 2018. *Deep Learning with Python.* s.l. : Manning Publications Co., 2018. ISBN 9781617294433.

Ciseran, Dan, y otros. 2015. Flexible, High Performance Convolutional Neural Networks for Image Classification. 2015.

Cisnero, Dalilis L. 2018. *Conjunto de métricas para la validación de la segmentación de imágenes médicas.* 2018.

Davydov, S y Efimov, Un. 2005. *IntelliJ IDEA. Professional'noe programmirovanie na Java (V podlinnike).* 2005. ISBN: 5-94157-607-2.

Duda, R. O y Heart, P. E. 1973. *Pattern Classification and Scence Analysis.* 1973.

Dumais, S, y otros. *Inductive learning algorithms and representations for text categorization. In Proceedings of the seventh international conference on Information and knowledge management (CIKM'98),.* s.l. : Bethesda. 15-002-864.

Elizondo, Manuel y Esqueda, José Jaime. 2012. *Fundamentos de Procesamiento de Imágenes.* 2012.

Friedman, N, Geiger, D y Cameron, Jones. 1997. *Bayesian network classifiers. Machine Learning.* 1997. ISBN: 29: 131-163.

Fukushima, Kunihiko. 2015. Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. 2015.

Giacamonte, Javier, Violini, Lucia y Lorenti, Luciano. 2015. *Reconocimiento Automático de Patrones, Análisis de imágenes y generación de características.* 2015.

Gollapudi, Sunila. 2016. *Practical Machine Learning.* s.l. : Packt Publishing, 2016. ISBN: 978-1-78439-968-4.

Gómez, María del Pilar. 2017. *Tópicos avanzados: Redes Neuronales Artificiales. Neurodinámica: Las redes de Hopfield.* 2017.

González, Raciel Ferrín. 2016. *Migración de los módulos Persona y Combustible del Sistema Orbita a la arquitectura de referencia en PHP Bosón.* s.l. : Universidad de La Habana, 2016.

Grewall, Daniel. 2008. *PENTACAM tomograms : A Novel Method for Quantification of Poste-.* 2008. Vol 49.

Hernández López, Iván, Hernández, Juan y Castro, Yadira. 2010. *Estrategias para la prevención de la opacidad de la cápsula posterior.* 2010. Vol 23.

Hernández, José Carlos y Letelier, Patricio. 2013. *Metodologías Ágiles en el desarrollo de software.* 2013.

Joskowicz, J. 2008. Reglas y prácticas en eXtreme Programming. 2008.

Kasthurirangan, Gopalakrishnan y Khaitan, S. K. 2017. *Deep Convolutional Neural Networks with transfer learning for computer vision-based data-driven pavement distress detection.* Evanston, Estaos Unidos : Department of Electrical Engineering & Computer Science, Northwestern University, 2017.

Ketkar, Nikhil. 2017. *Deep Learning with Python. A Hands-on Introduction.* s.l. : Apress, 2017.

Larman, Craig. 2003. UML y patrones, Introducción al análisis y diseño orientado a objetos. 2003.

Lecun, Yann. 1998. *Gradient-based learning applied to document recognition.* 1998.

Lerch, Daniel. 2018. *Neuron4.* [En línea] 2018. https://medium.com/.

Lippmann, R. P. 1989. *Pattern Classification using neural networks.* s.l. : IEEE Communications Magazine, 1989. ISBN: 052-687-27.

Madruga, Jean Carlos. 2016. *Algoritmo para la segmentación de la opacidad de la cápsula posterior en imágenes en retroiluminación provenientes de la lámpara de hendidura.* 2016.

Malfará, D, y otros. 2006. *Testing en eXtreme Programming.* 2006.

Márquez, Pedro y Ramiro, Joan. 2015. Sitio Web Oficial MySQL. *Sitio Web Oficial MySQL.* [En línea] 2015. http://www.mysql.com/.

Masero, Valentín. 2015. *Una nueva metodología de segmentación de imágenes basadas en Contornos Activos.* 2015.

Michie, D, Spiegelhalter, D J y Taylor, C C. 2016. *Machine Learning, Neural and Statistical Classification.* 2016.

Ortega, Dolgis Rainer y Iznaga, Arsenio. 2008. *Técnicas de Segmentación de Imágenes Médicas.* 2008.

Pressman, Roger S. 2010. *Ingeniería de Software un enfoque Práctico.* New York : s.n., 2010.

Quinlan, J. R. 1993. *C4.5: programs for machine learning.* s.l. : Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN: 1-55860-238-0..

RAI. 2016. *Perceptrón simple, Redes Neuronales Artificiales.* 2016.

Rodríguez Sánchez, Tamara. 2015. *Metodología de desarrollo para la Actividad Productiva de la UCI.* 2015.

Rosebrock, Adrian. 2017. *Deep Learning for Computer Vision with Python.* s.l. : PYImageSearch, 2017. ISBN: 487-301-592.

Schmidhuber, J. 2014. Deep Learning in Neural Networks: An Overview. 2014.

Sewak, Mohit, Rezaul, Karim y Pujari, Pradeep. 2018. *Practical Convolutional Neural Networks.* s.l. : Packt Publishing, 2018. ISBN: 410-968-357.

Shapiro, Linda y Stockman, George. 2000. *Computer Vision.* 2000.

Simonyan, Karen y Zisserman, Andrew. 2015. *Very Deep Convolutional Network for large-scale image recognition.* 2015.

Somerville, Ian. 2005. Ingeniería de software. 2005.

Started, G. 2013. *Software design tools for Agile Teams, with UML, BPMN and More.* 2013.

Sucar, Luis E. 2016. *Clasificadores Bayesianos: de Datos y Conceptos.* 2016.

Tetz, Manferd, Sperker, Martina y Blum, Marcus. 2013. *Photographic image analysis system of posterior capsule pacification.* 2013. Vol 23.

Urlocker, Zack. 2010. *The Open Force. .* 2010.

Varone, Marco, Mayer, Daniel y Melegari, Andrea. 2016. [En línea] 2016. https://www.expertsystem.com/machne-learning-definition/.

Verdarguer, Sergi Laencina. 2016. *Proyecto de fin de carrera- Descripción y clasificación de imágenes mediante su color.* 2016.

Wesley, Addison y Beck, Kent. 2002. *Una explicación de la programación extrema: aceptar el cambio.* 2002.