Universidad Autónoma de Yucatán

Facultad de Matemáticas

Algoritmos para diseñar plantillas de convolución que determinan características tipo Haar

Tesis presentada por
L.M. César Iván Cobos May

Para obtener el grado de
Maestro en Ciencias Matemáticas

Directores de tesis
Dr. Víctor Uc Cetina
Dr. Carlos Brito Loeza

2014

Resumen

En este trabajo se presentan dos métodos para generar, de forma automática, plantillas de convolución que determinen características tipo Haar a partir de una imagen dada, ya que, actualmente, no existe algún método para generarlas, y su diseño es de forma empírica. Los métodos se desarrollan a partir de una imagen en blanco y negro y, primeramente, para regiones conexas y convexas; luego se generaliza la idea para imágenes en RGB y para cualquier tipo de región en general. Se implementó cada uno de los métodos y se probaron las plantillas generadas en la detección de imágenes que contienen al nervio óptico, obteniendo resultados que, incluso, mejoran el desempeño de los clasificadores generados con las plantillas que usualmente se utilizan en el reconocimiento de patrones. Se presenta también el análisis computacional de los métodos propuestos, y las ventajas y desventajas de utilizar plantillas generadas con ellos.

Agradecimientos

Quiero empezar este trabajo, no sin antes dedicar unas palabras de agradecimiento a todos los que lo hicieron posible. Agradezco primeramente a Dios por la salud, paciencia y fortaleza brindados para realizar esta tesis; fue difícil el camino por momentos, pero siempre estuvo allí para recordarme que su yugo es suave y su carga ligera.

A mis papás y hermanita, por las consideraciones que tuvieron conmigo este tiempo; todas esas veces que falté a alguna reunión o no pude viajar en familia, rindieron frutos.

Agradezco también al Consejo Nacional de Ciencia y Tecnología por la beca otorgada estos dos años. A mis asesores, Dr. Víctor Uc Cetina y Dr. Carlos Brito Loeza, por todas sus enseñanzas, por sus llamadas de atención que me hicieron doblar esfuerzos, pero también por las veces que me sacaron una sonrisa; me llevo mucho de ellos. A la Dra. Anabel Martín González por sus valiosos consejos, observaciones y aportaciones al trabajo, y al Dr. Ramón Peniche Mena por su vasto apoyo en gran parte de mi estancia en la facultad.

A mis amigos Luis López, Julio Yerbes, Juan Ríos y Manuel Uc, que siempre mostraron interés, preocupación y apoyo hacia este trabajo; agradezco su amistad y sus palmadas de ánimo.

Finalmente, quiero agradecer a Luz María por ser una guía y una segunda madre para mí; agradezco su apoyo y sus consejos. A todos ellos, y a muchos más, muchas gracias.

A la memoria de mi abuelita.

Índice general

Capítulo 1

Introducción

Hoy en día, los algoritmos de aprendizaje automático, son muy usados en nuestra vida cotidiana. Y es que, desde que desbloqueamos nuestro celular con pantalla táctil, o cuando llegamos a trabajar y registramos nuestra llegada con huella digital (o hasta con nuestro rostro), estamos utilizando algoritmos de aprendizaje automático. Nada se salva de las matemáticas.

Un problema muy común del aprendizaje automático, es el problema de clasificación: consiste en asignar una etiqueta a un elemento dado [5]. El problema de clasificación es muy variado; se pueden clasificar desde objetos en imágenes, así como un correo electrónico (en deseado y no deseado), o hasta se puede clasificar el resultado de una pelea de box (en victoria o derrota), como se hace en las casas de apuestas. Para la clasificación de objetos en imágenes, los algoritmos de aprendizaje se pueden utilizar en conjunto con las características tipo Haar. Esta mancuerna se ha hecho muy popular en los últimos años, y se han detectado con ellas rostros [4], ademanes con las manos [13] y hasta peatones [2]. El problema es que, las plantillas que definen estas características, se diseñan empíricamente, es decir, no hay un método que las determine, dependiendo del problema que se quiera tratar. Es por eso que, en este trabajo, se sugieren dos métodos para el diseño de dichas plantillas, para dar más formalidad al uso de estas características.

1.1. Descripción del problema

Actualmente, las características de tipo Haar, se utilizan en la detección de rasgos en imágenes, como por ejemplo, la detección de ojos, nariz y boca

dentro de la imagen de un rostro, o parásitos dentro de imágenes de muestras de sangre. Estas características, en conjunto con el AdaBoost, forman una de las herramientas más poderosas que existen para clasificar objetos.

El diseño de las plantillas que definen las características tipo Haar depende en gran parte de la morfología del objeto, y una desventaja al utilizarlas es que, actualmente, su diseño es de forma empírica, es decir, el autor, con base en su experiencia, decide cuáles van a ser las características que va a utilizar; ésto hace pensar que se impide aprovechar toda la información que brinda la forma del objeto de estudio.

Por ello es que, en este trabajo, se implementan un par de métodos para diseñar, de forma automática, estas plantillas, a partir de la imagen de un objeto dado. Estos métodos se basan en la morfología del objeto y se deducen utilizando conceptos de topología básica y geometría computacional, dejando a un lado el diseño empírico que actualmente se utiliza.

Los métodos se desarrollan a partir de una imagen en blanco y negro, primeramente, para regiones conexas y convexas. Posteriormente, se extienden para ser aplicados a cualquier tipo de región, y para imágenes en RGB. El desempeño de los algoritmos es analizado a partir de experimentos con una tarea de detección de objetos, y se presentan evidencias de las ventajas y desventajas de cada uno de los métodos.

1.2. Objetivo general

Desarrollar un algoritmo capaz de encontrar el conjunto de plantillas que determinan características de tipo Haar, que mejor se adapten a la morfología de un conjunto de imágenes dado, para lograr un mejor desempeño en la clasificación de ejemplos nuevos de dichas imágenes.

1.3. Objetivos particulares

Para lograr el objetivo general, se plantean algunos objetivos particulares:

- Sentar las bases matemáticas para crear métodos que diseñen las plantillas de convolución.
- Implementar los métodos desarrollados, en Matlab.
- Analizar computacionalmente los métodos.

- Probar la efectividad de las plantillas diseñadas, en comparación con las plantillas que se usan comúnmente.

1.4. Estructura del documento

El documento se desarrolla como sigue: en el primer capítulo se habla acerca de la importancia del problema a abordar. En el capítulo dos, se da toda la teoría que sustenta al trabajo. En el capítulo tres, se desarrolla un primer método (en imágenes binarias) para generar plantillas (método simple) y se hace un análisis computacional de él. En el capítulo cuatro, se desarrolla un segundo método (en imágenes binarias) para diseñar plantillas (método optimizado) y se hace un análisis computacional de él. En el capítulo cinco, se generalizan ambos métodos para poder aplicarlos en imágenes en RGB. En el capítulo seis se compara el desempeño de las plantillas generadas con los métodos anteriores junto con el desempeño que tienen las plantillas usuales. En el capítulo siete se cierra el trabajo con las conclusiones y el trabajo futuro. Finalmente, en el capítulo ocho, se incluyen algunos análisis de algoritmos como apéndice.

Capítulo 2

Marco Teórico

2.1. Aprendizaje Automático

El aprendizaje automático es una de las ramas principales de la inteligencia artificial (IA) [1], que se centra en el desarrollo de técnicas para poder crear algoritmos capaces de extraer modelos o patrones de información no estructurada, a partir de datos o experiencias previas.

En ocasiones se define este conjunto de técnicas como aquellos procedimientos que permiten a una computadora aprender. Realmente, dicha máquina no aprende en sí, sino que se le dan pautas para poder transformar los datos de entrada en conocimiento, y así dotarla de capacidad de adaptación a circunstancias específicas. Esto hace que la aplicación de técnicas de aprendizaje automático a problemas que varían en el tiempo, o que necesitan un entorno particular, sea exitosa.

El esquema de cómo funciona un algoritmo basado en técnicas de aprendizaje automático es el siguiente: se tiene un conjunto de ejemplos sobre el cual se aplicarán estas técnicas, con el fin de generar un modelo. Este modelo será capaz de dar una respuesta satisfactoria sobre un nuevo ejemplo, no contemplado en el conjunto inicial, como se muestra en la Figura 2.1.

Las aplicaciones de los distintos tipos y técnicas de aprendizaje automático están en constante desarrollo, ya que se pueden aplicar a un rango muy amplio de tecnologías. Los sectores en los cuáles estas técnicas de la IA están teniendo un mayor auge y efectividad son los siguientes: banca, algoritmos para la predicción de la evolución del mercado, detección de operaciones anómalas en tarjetas de crédito, software, creación de nuevos algoritmos basados en

estrategias evolutivas, detección de la mejor ruta en una red, geolocalización, técnicas de minería de datos, bioinformática, clasificación de secuencias de ADN, entre otros.

Se han categorizado las distintas técnicas y algoritmos que pertenecen a esta rama de la IA según el tipo de aprendizaje que utilizan. Entre los tipos de aprendizaje que existen, están los siguientes: Aprendizaje supervisado, aprendizaje no supervisado, aprendizaje multi-tarea y aprendizaje por refuerzo.

Este trabajo se centra en el aprendizaje supervisado, que es el tipo de aprendizaje en el cual se engloban técnicas en las que, para poder deducir una función o un patrón en un conjunto de datos, es necesario proveer al algoritmo en desarrollo un conjunto de datos de entrenamiento, para así poder "entrenar" a dicho algoritmo y que éste pueda predecir el comportamiento de nuevos datos dados, ajenos al conjunto inicial.

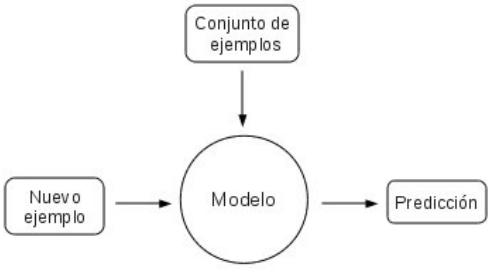

Figura 2.1: Metodología del aprendizaje supervisado. Un conjunto de ejemplos genera un modelo, el cual, evalúa nuevos elementos para generar predicciones acerca de ellos.

Un ejemplo de este tipo de aprendizaje es el algoritmo de clasificación: consiste en determinar una función $G(x)\colon X \to \{1, 2, \ldots, N\}$ que relaciona un conjunto de vectores $x \in X$ en N clases distintas posibles [5]. Un caso particular de este algoritmo, es el de clasificación binaria, en el que sólo hay dos opciones posibles ($N = 2$). Este tipo de algoritmo es utilizado, por ejemplo, para predecir el resultado de un equipo en un partido de béisbol, en el cual sólo se tienen las opciones de ganar o perder. Así, la función G predecirá el resultado del partido dependiendo del vector x que se evalúe, el cual puede contener datos como: porcentaje de victorias del equipo, porcentaje de derrotas, estado del campo, condiciones climáticas, etc. Por razones de

simplicidad, se considerarán esas dos opciones posibles como -1 y 1, así la función está definida como $G(x)\colon X \to \{-1, 1\}$.

Se conocen como algoritmos de clasificación débiles, a los algoritmos cuya tasa de error ϵ es apenas menor a 0.5 ($\epsilon < 0.5$), es decir, apenas son mejores que una elección al azar, ya que, para la elección al azar, se tiene que $\epsilon = 0.5$.

2.2. Adaptive Boosting

El algoritmo adaptativo de empuje (boosting) [16] es una de las ideas de aprendizaje automático más poderosa desarrollada en los últimos diez años. Es un algoritmo que fue diseñado originalmente para problemas de clasificación, y se basa en combinar las predicciones de varios clasificadores débiles para formar, con todas ellas, un "comité". Es decir, teniendo $G_1, \ldots, G_M$ clasificadores binarios débiles, el boosting determina un nuevo clasificador $G(x)_{final}$ como resultado de una combinación de los anteriores:

$$G(x)_{final} = \text{sign}[\sum_{m=1}^{M} \alpha_m G_m(x)]$$

donde $\alpha_m = \log(\frac{1-\epsilon_m}{\epsilon_m}) > 0$.

La manera como opera boosting es la siguiente: dado un conjunto de entrenamiento $(x_1, y_1), \ldots, (x_t, y_t)$ con $y_i \in \{-1, 1\}$, en cada iteración m, el algoritmo construye una distribución de probabilidad D_m, en la cual, se basa para encontrar un clasificador binario débil $G_m(x)$ con error $\epsilon_m = \Pr_{D_m}[G_m(x_i) \neq y_i]$, donde $\Pr_{D_m}$ es la función de probabilidad bajo la distribución D_m. Así, al final de M iteraciones, se construye el clasificador final $G(x)_{final}$ definido anteriormente.

Los diferentes tipos de boosting se dan dependiendo de la distribución de probabilidad D_m que se calcule en cada iteración. Uno de los algoritmos de boosting más utilizados es el algoritmo adaptativo de empuje (Adaptive Boosting) [16], mejor conocido como AdaBoost, el cual, recibe el adjetivo de "adaptativo"debido a que, cada clasificador nuevo que se crea, le da más importancia (peso) a los ejemplos que estuvieron mal clasificados por el anterior, es decir, se va adaptando conforme al desempeño que tenga cada clasificador. Para el AdaBoost, la distribución D_m se inicializa como una distribución de probabilidad uniforme, es decir, $D_1(i) = 1/t$, para $i \in \{1, \ldots, t\}$, y se va actualizando en cada iteración como sigue:

$$D_{m+1}(i) = \frac{D_m(i)}{Z_m} \times \begin{cases} e^{-\alpha_m} & \text{si } y_i = G_m(x_i) \\ e^{\alpha_m} & \text{si } y_i \neq G_m(x_i) \end{cases}$$

donde Z_m es una constante de normalización.

Para entender mejor cómo funciona el AdaBoost, considérese el siguiente ejemplo: se requiere clasificar los signos azules y los rojos, como se muestra en la Figura 2.2, utilizando líneas verticales y horizontales. Estas líneas serán los clasificadores débiles G_m. La región contemplada en la parte derecha de las líneas verticales y en la parte inferior de las horizontales, será tomada como región de color rojo (ahí pertenecen los signos rojos), y la región en la parte izquierda de las líneas verticales y en la superior de las horizontales, será tomada como región de color azul (ahí pertenecen los signos azules).

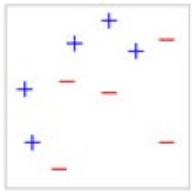

Figura 2.2: Problema de clasificación.

Considérese un número de tres iteraciones ($M = 3$). En este ejemplo, se tienen 10 signos por clasificar, así que, la distribución de probabilidad inicial será $D_1(i) = 1/10$. Una primera iteración del algoritmo arroja el siguiente resultado:

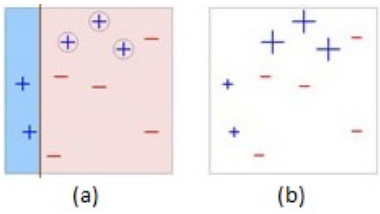

Figura 2.3: (a) Primer clasificador débil $G_1(x)$. (b) D_2, actualización de la distribución de probabilidad D_1. Los signos azules a la derecha están siendo mal clasificados por el clasificador débil $G_1(x)$, por lo tanto, su importancia aumenta.

Con la distribución uniforme, todos los signos tenían un peso igual, ahora, nótese en la Figura 2.3 que la nueva distribución aumenta el peso de los tres signos azules que estuvieron mal clasificados por G_1 y disminuye el peso de los que se clasificaron bien. En esta iteración $\epsilon_1 = 0.30$ y $\alpha_1 = 0.42$. El segundo clasificador G_2 le dará más importancia a los signos que estuvieron mal clasificados por G_1, como se puede apreciar en la siguiente figura:

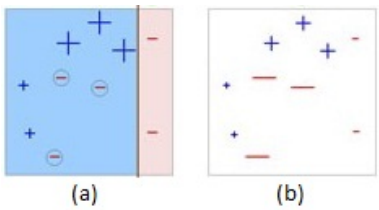

Figura 2.4: (a) Segundo clasificador débil $G_2(x)$. (b) D_3, actualización de la distribución de probabilidad D_2.

Nótese, en la Figura 2.4, que G_2 clasificó bien a todos los signos de mayor peso. Análogamente a la primer iteración, se reducen los pesos de los signos clasificados correctamente y se aumenta el peso de los clasificados incorrectamente. En esta segunda iteración $\epsilon_2 = 0.21$ y $\alpha_2 = 0.65$. Para la iteración final se obtiene el siguiente resultado:

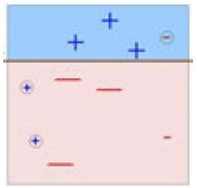

Figura 2.5: Tercer clasificador débil $G_3(x)$.

En esta iteración se tiene que $\epsilon_3 = 0.14$ y $\alpha_3 = 0.92$. Así, de las tres iteraciones obtenemos que el clasificador final es $G_{final}(x) = \text{sign}(0.42 * G_1(x) + 0.65 * G_2(x) + 0.92 * G_3(x))$, como se muestra en la Figura 2.6.

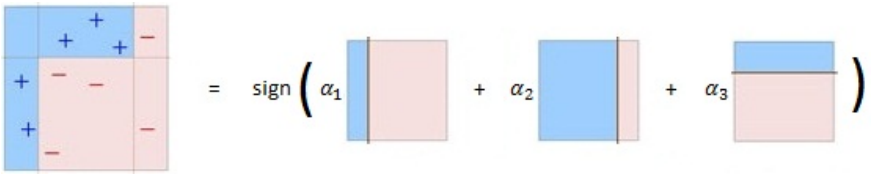

Figura 2.6: Clasificador final $G_{final}(x)$ como combinación de los clasificadores débiles G_1, G_2 y G_3.

A los clasificadores finales determinados con AdaBoost se les llamará clasificadores fuertes.

En este trabajo se abordará un problema de clasificación basándose en el método propuesto por Viola y Jones [4], utilizando características tipo Haar y el concepto de imagen integral. Se generará un clasificador con AdaBoost, y los elementos que se evaluarán en el clasificador serán imágenes.

2.3. Plantillas de convolución

Cada imagen en escala de grises de $m \times n$ pixeles se puede representar como una matriz, también de $m \times n$, asignando, a cada elemento de la matriz, la intensidad del pixel que le corresponde. A esta representación numérica de imágenes en matrices, se le conoce como imagen digital. Análogamente, las imágenes en formato RGB, al tener tres canales de datos (rojo, verde, y azul), se pueden representar por matrices cúbicas de $m \times n \times 3$, en donde cada entrada (i, j, k) de la matriz, contiene la intensidad del pixel en la posición (i, j) en el canal k.

En la imagen digital, las operaciones grupales [17], calculan el nuevo valor de un pixel a partir de los valores de pixeles vecinos. Las operaciones grupales, usualmente, se expresan en términos de una platilla de convolución, la cual es un conjunto de coeficientes ponderados, agrupados en una matriz. El nuevo valor del pixel se calcula colocando la plantilla en el pixel de interés de la imagen; los pixeles comprendidos en la plantilla son multiplicados por los correspondientes coeficientes ponderados y agregados a la suma total, Figura 2.7.

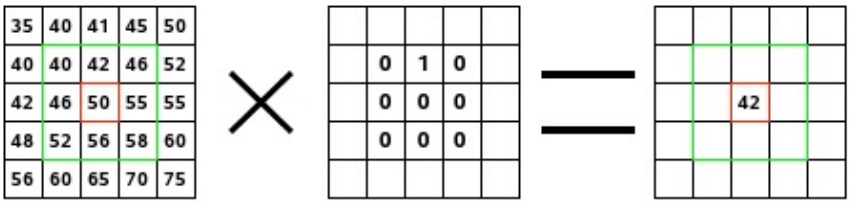

Figura 2.7: Aplicación de una plantilla de convolución de 3×3 a una matriz. El valor del nuevo pixel es $40 \times 0 + 42 \times 1 + 46 \times 0 + 46 \times 0 + 50 \times 0 + 55 \times 0 + 52 \times 0 + 56 \times 0 + 58 \times 0 = 42$.

El tamaño de la plantilla puede variar siempre y cuando esté contenido dentro del tamaño de la imagen; usualmente, se utilizan plantillas cuyas matrices son cuadradas, como 3×3, 5×5, etc., pero también pueden utilizarse plantillas cuyas matrices no lo sean, como 3×5.

2.4. Caraterísticas tipo Haar

La visión computacional [6] es un campo en el cuál se determinan métodos para procesar, analizar y entender imágenes para producir algún tipo de información, ya sea numérica o simbólica, es decir, en forma de decisiones. En visión computacional, el concepto de característica se usa para denotar una parte de información que es relevante para resolver una tarea computacional, relacionada a cierta aplicación. Específicamente, las características se pueden referir a:

- El resultado de una operación entre pixeles cercanos, en una región de la imagen.
- Estructuras específicas dentro de la imagen, que van, desde simples estructuras, como puntos o bordes, hasta estructuras más complejas como objetos.

El concepto de característica es muy general, y la elección de características en un sistema depende altamente del problema que se quiere tratar.

Las características tipo Haar [2], son características de imágenes digitales usadas para el reconocimiento de objetos y reciben su nombre por su similaridad con las onduletas de Haar [18].

Una característica de tipo Haar se define como el resultado de aplicar, sobre algún pixel de la imagen, una plantilla de convolución binaria, es decir, los coeficientes ponderados de la plantilla solamente pueden tomar dos valores; en el caso de las características tipo Haar, dichos valores serán 1 ó -1. Dado que la plantilla sólo toma dos valores en las características tipo Haar, estas plantillas se pueden representar como imágenes binarias (en blanco y negro), Figura 2.8. Cada característica de tipo Haar puede indicar la existencia (o ausencia) de ciertos rasgos en la imagen, como bordes o cambios de textura. Por ejemplo, una característica de dos rectángulos (Figura 1.8), puede indicar en dónde se encuentra un borde entre una región oscura y una más clara, dentro de la imagen.

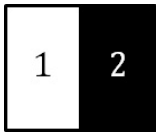

Figura 2.8: Plantilla que define una característica de dos rectángulos vertical. Los pixeles comprendidos en las regiones 1 y 2, toman el valor de 1 y -1, respectivamente.

2.5. Imagen integral

La ventaja de utilizar plantillas que definan características rectangulares sobre otro tipo de características es la velocidad del cálculo de éstas, dado el concepto de imagen integral, desarrollado por Viola y Jones [4]. La imagen integral en la ubicación (x, y) se define como:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

en donde $i(x, y)$ es la intensidad del pixel (x, y). El cálculo de la imagen integral, independientemente del tamaño de la plantilla, se hace en tiempo constante, ya que, se calcula $ii(x, y)$ una sola vez para todo pixel (x, y) en la imagen, y cada uno de estos valores se almacena en la memoria de la computadora en un arreglo bidimensional. Por lo tanto, para calcular el valor de la característica, bastará que se acceda a la matriz almacenada previamente,

e independientemente del tamaño de la plantilla que se utilice, se necesitarán, a lo más, tres operaciones para hacer el cálculo (Figura 2.9).

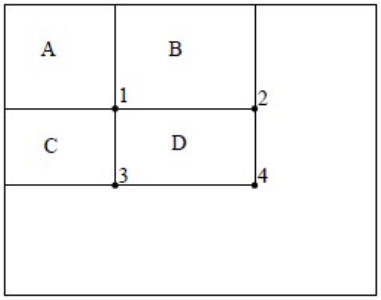

Figura 2.9: El valor de la imagen integral en la ubicación 1 es la suma de los pixeles comprendidos en el rectángulo A, es decir, $ii(1) = A$. Análogamente, $ii(2) = A + B$, $ii(3) = A + C$, $ii(4) = A + B + C + D$. De ésto se deduce que, la suma de los pixeles comprendidos en D es $ii(4) + ii(1) - [ii(2) + ii(3)]$, para lo cual se necesitan tres operaciones: dos sumas y una resta.

2.6. Trabajos relacionados

El concepto de característica de tipo Haar, fue introducido en 1997 por Papageorgiou et. al. [2]; se buscaba una representación de la imagen que capturara la relación entre las intensidades promedio en regiones cercanas. Las primeras características que se utilizaron, fueron características de tipo Haar de dos rectángulos horizontales y verticales, y la característica de cuatro rectángulos en diagonal, como se muestra en la Figura 2.10.

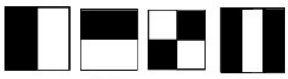

Figura 2.10: Plantilla que define una característica de tipo Haar vertical, horizontal, diagonal y de tres rectángulos, respectivamente.

Principalmente, fueron desarrolladas para la detección de peatones en imágenes, pero en un trabajo posterior [3], las mismas características, fueron utilizadas para la detección de rostros, y en general, para la detección de objetos.

En 2001, Viola y Jones [4] introducen las características de tres rectángulos, añadiéndolas al conjunto propuesto por Papageorgiou. Ésto, junto con la inclusión de la imagen integral y la cascada de clasificadores binarios entrenados con AdaBoost, hacen su sistema de detección de rostros, aproximadamente, 15 veces más rápido que cualquier otro sistema propuesto anteriormente.

Basándose en este último sistema, Lienhart y Maydt [7] añaden las características centrales de tipo Haar al conjunto utilizado por Viola y Jones, junto con un conjunto de estas mismas ca- racterísticas rotadas 45°, para la detección de rostros, Figura 2.11. A su vez, definen una imagen integral rotada para que el cálculo de las sumas de los valores de los pixeles en las características de tipo Haar rotadas sea tan rápido como el de la imagen integral de Viola y Jones (para características de tipo Haar sin rotar). Esta aportación redujo en un 10 % la falsa detección en imágenes de rostros.

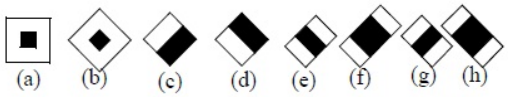

Figura 2.11: (a) Plantilla que define una característica de tipo Haar central. (b)-(h) Plantillas que definen características de tipo Haar rotadas 45°.

Como se ha mencionado, cada característica de tipo Haar es considerada como un clasificador débil que sirve para determinar al clasificador fuerte. Mita et. al. [10], en 2005, consideraron la posibilidad de incluir simultáneamente dos o más características de tipo Haar, basándose también en el método descrito por Viola y Jones. A esta nueva característica le llamaron la característica de tipo Haar conjunta y se basa en la co-ocurrencia de múltiples características de tipo Haar (Figura 2.12). Esta modificación del conjunto de características, aproximadamente, reduce en un 50 % el error de clasificación obtenido en [4].

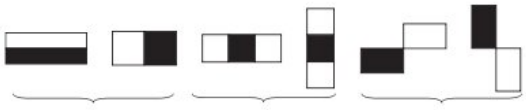

Figura 2.12: Plantillas que definen características de tipo Haar conjuntas.

En 2006, Barckzak et. al. [11] retoma la idea de Lienhart y Maydt, utilizando su mismo conjunto de características, pero ahora propone, además de las características rotadas 45°, la rotación de éstas en ángulos de 26.5° y 63.5°, Figura 2.13, y propone un método para que un clasificador ya entrenado funcione para cualquier ángulo (o uno aproximado a él), y así, los objetos rotados puedan ser detectados sin entrenar específicamente a ese clasificador para dicho ángulo en particular, es decir, no es necesario calcular todas las rotaciones posibles de las características de tipo Haar y entrenar al clasificador con ellas.

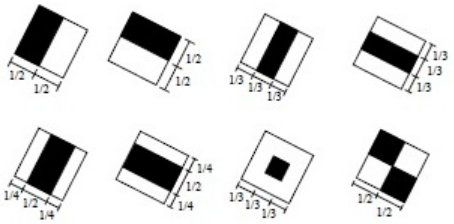

Figura 2.13: Plantillas que definen características de tipo Haar rotadas 26.5.

En 2008, Chen et. al. [13] utilizan la misma metodología y características de tipo Haar utilizadas en [7], para la clasificación de ademanes.

En 2010, Pavani et. al. propusieron pesos óptimos para las plantillas que determinan las características [24]. Utilizando tres algoritmos (búsqueda por fuerza bruta, algoritmos genéticos y análisis del discriminante lineal de Fisher), propusieron tres métodos diferentes para generar plantillas con pesos óptimos, esto hace que las plantillas generadas ya no sólo puedan tomar valores de 1 y -1 en las regiones negra y blanca, respectivamente, sino que, prácticamente, puedan tomar cualquier valor entre -1 y 1. Las plantillas con pesos óptimos fueron probadas con imágenes frontales de rostros y del

corazón humano, y obtuvieron mejores detectores de objetos tanto en precisión como en velocidad de detección, en comparación con las plantillas con pesos usuales.

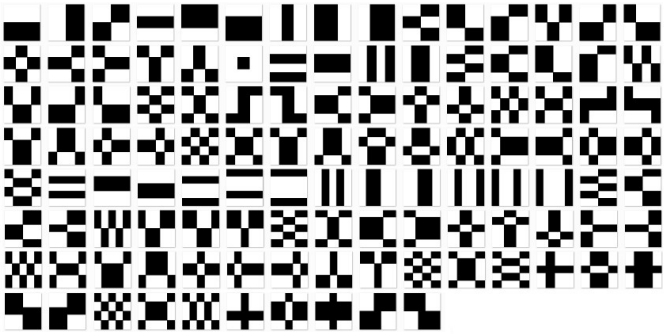

Figura 2.14: Plantillas utilizadas en el reconocimiento biométrico.

Las características tipo Haar no sólo se han utilizado en la detección de rostros humanos, sino también en la detección de rostros de representaciones gráficas humanas (avatar) [25]. Es muy común en videojuegos, y en internet en general, que un persona tenga una representación gráfica de sí misma, así que, D'Souza et. al. en 2012 utilizaron las características propuestas por Lienhart y Maydt (2002) para detectar rostros de avatares en imágenes, obteniendo buenos resultados para imágenes frontales de rostros, pero resultados del 74 % de efectividad en imágenes con poca luz, oclusión parcial y rotaciones de rostros.

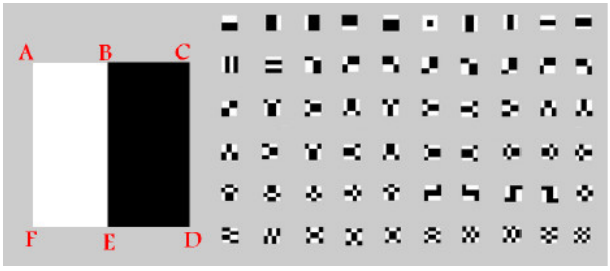

Figura 2.15: Plantillas utilizadas en la detección robusta de rostros en tiempo real.

En 2013, Wei et. al. utilizaron las mismas plantillas que Viola y Jones, y las implementaron, primero, en conjunto con AdaBoost para hacer una primera selección de características tipo Haar, para posteriormente extraer de ellas los descriptores del histograma de gradiente orientado (HOG) y con ellos entrenar a un clasificador de máquinas de vectores de soporte (SVM) [26], todo esto para mejorar el desempeño de la detección de peatones en imágenes. Esto produjo un 95.14 % de efectividad en la tasa de detección, mientras que un 2.36 % en la tasa de falsos positivos. En ese mismo año, Nasrollahi et. al. extendieron el conjunto de plantillas utilizadas, considerando las plantillas de la Figura 2.14, y las aplicaron en conjunto con una red neuronal probabilística (PNN), para el reconocimiento biométrico [27], considerando un total de 115 plantillas. Esto produjo resultados del 93.9 % de efectividad en los clasificadores, no obstante, es un número muy elevado de plantillas para extraer las características. Finalmente, meses después, Nasrollahi et. al., redujeron el conjunto de características utilizado en el trabajo anterior, considerando las plantillas de la Figura 2.15, y las aplicaron nuevamente en conjunto con una red neuronal probabilística para una detección robusta de rostros en tiempo real [28], logrando resultados hasta del 100 % de efectividad en una de las dos bases de datos utilizadas en la experimentación. Cabe mencionar que, en dicho trabajo, se utilizaron un total de 61 plantillas para determinar características.

Capítulo 3

Método simple para trazar rectángulos

3.1. Introducción

Como se mencionó anteriormente, una característica es una parte relevante de la imagen. Las características tipo Haar utilizan plantillas binarias, es decir, de dos valores: 1 ó -1; estas características se utilizan en el reconocimiento y detección de objetos.

Hasta ahora, el diseño de estas plantillas ha sido siempre rectangular, esto se debe al concepto de imagen integral utilizado por Viola y Jones en 2001. Si se tiene una región rectangular de $N \times M$ pixeles en una imagen, se necesitarían $NM - 1$ sumas para calcular la suma total de todos los valores de los pixeles comprendidos en dicho rectángulo fijo en la imagen. La imagen integral permite calcular esta suma total realizando, a lo más, tres sumas únicamente, esto reduce considerablemente el tiempo de cálculo de los valores de las características, ya que, cada plantilla que define a la característica no sólo está compuesta de uno, sino de varios rectángulos, y la imagen integral evita recorrer el área de cada uno para calcular la suma, y además, es independiente del tamaño de los mismos, ya que siempre se requieren tres sumas para calcular dicha suma total, sin importar las dimensiones de los rectángulos.

Tomando en cuenta lo anterior, en este capítulo se proponen métodos que aproximan la imagen dada por medio de rectángulos, esto para facilitar el cálculo del valor de la característica utilizando el concepto de la imagen

integral.

Se definirán primero algunos conceptos, antes de desarrollar el algoritmo. Las siguientes definiciones fueron tomadas de [20].

Definición 3.1. Definimos una *curva simple* Z como la imagen de una aplicación $z : I \to R^2$ que es continua e inyectiva en el intervalo I, es decir, si $z(t_1) = z(t_2)$ entonces $t_1 = t_2$, para todo $t_1, t_2 \in I$ (ver Figura 3.1).

La aplicación z recibe el nombre de *parametrización* de Z. Así, una curva simple es aquella que no se corta a sí misma. Si $I = [a, b]$, llamaremos a $z(a)$ y $z(b)$ los extremos de la curva.

Figura 3.1: Curva simple.

Definición 3.2. Se dice que una curva simple Z es *cerrada* si la aplicación $z : I \to R^2$ de la que es imagen, cumple que $z(a) = z(b)$, para $I = [a, b]$ (ver Figura 3.2).

Podemos también considerar a Z como el conjunto:

$$Z = \{(x(t), y(t)) | z(t) = (x(t), y(t)) \text{ para } t \in I\}$$

En esta sección sólo consideraremos curvas cerradas simples, cuyo dominio sea $I = [a, b]$.

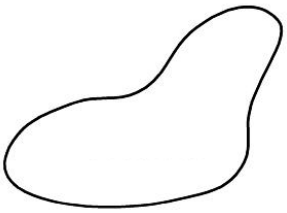

Figura 3.2: Curva cerrada simple.

Notación. Sea Z una curva simple, y $z = (x, y) \in Z$. Se denotará $[z]_x = x$ y $[z]_y = y$.

Definición 3.3. Sea Z una curva cerrada simple. Un punto $(x, y) \in R^2$ está en el *interior* de Z, si existen puntos $(x_a, y), (x_b, y), (x, y_a), (x, y_b)$ en Z, tales que $x \in [x_a, x_b]$ y $y \in [y_a, y_b]$. Se denotará este conjunto como $int(Z)$ (ver Figura 3.3).

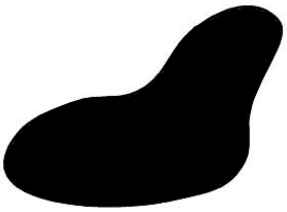

Figura 3.3: Interior de la curva cerrada simple de la Figura 3.2.

Definición 3.4. Sea A un conjunto no vacío, y sean x_1 y x_2 dos puntos en A. Se denotará como l_{x_1,x_2} al segmento de línea recta comprendida entre los puntos x_1 y x_2. Un conjunto A es convexo si, para cada $x_1, x_2 \in A$, $l_{x_1,x_2} \subset A$.

Afirmación 3.1. Sean A y B dos conjuntos convexos. $A \cap B$ es un conjunto convexo.

Prueba:

Sean dos puntos $x_1, x_2 \in A \cap B$, esto implica que $x_1, x_2 \in A$ y $x_1, x_2 \in B$. Ya que A y B son convexos, se tiene que $l_{x_1,x_2} \subset A$ y $l_{x_1,x_2} \subset B$. Así,

$l_{x_1,x_2} \subset A \cap B$ y por tanto $A \cap B$ es convexo. □

Definición 3.5. Se dice que un conjunto es *conexo*, si no puede expresarse como la unión de dos conjuntos abiertos disjuntos.

Definición 3.6. Sea un conjunto X no conexo, se llama *componente conexa* a todos los subconjuntos maximales conexos de X. Es decir, si $C \subset X$, C es una componente conexa si y sólo si C es conexo, y para cualquier $T \subset X$ tal que $C \subsetneq T$, T es no conexo.

El siguiente par de axiomas fueron tomados de [21].

Axioma del ínfimo. Todo subconjunto de números reales, no vacío y acotado inferiormente, tiene un ínfimo en R.

Axioma del supremo. Todo subconjunto de números reales, no vacío y acotado superiormente, tiene un supremo en R.

3.2. Desarrollo del algoritmo

Sea Z una curva cerrada simple tal que $int(Z)$ es una región conexa y acotada (Figura 3.4), y $\pi_x : R^2 \to R$ la función proyección al eje x, definida por $\pi[(x, y)] = x$, para todo $(x, y) \in R^2$. Se considerará el conjunto $\pi_x[int(Z)]$. Dado que $\pi_x[int(Z)]$ es un conjunto acotado en R y no vacío, (ya que $int(Z)$ es acotado y no vacío), por los axiomas del ínfimo y el supremo, existe un elemento mínimo y uno máximo de $\pi_x[int(Z)]$. Sean:

$$x_0 = \text{mín}\, \pi_x[int(Z)]$$

$$x_N = \text{máx}\, \pi_x[int(Z)]$$

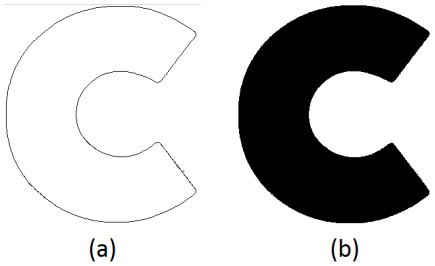

Figura 3.4: (a) Curva cerrada simple Z a considerar. (b) Interior de Z.

Se particionará el intervalo $[x_0, x_N]$ en N regiones de longitud $\frac{x_N - x_0}{N}$ cada una. Se considerarán los puntos $x_i = x_0 + \frac{i(x_N - x_0)}{N}$ y las rectas verticales v_i que pasan por los puntos x_i, con $i \in \{0, \ldots, N\}$. Para cada intervalo $[x_i, x_{i+1}]$, $(i \neq N)$, se considerará la recta vertical v_{m_i} que pasa por el punto medio del intervalo (Figura 3.5) y el conjunto:

$$W = \{w \in R^2 | w \text{ es un punto de corte entre } v_{m_i} \text{ y } Z\}$$

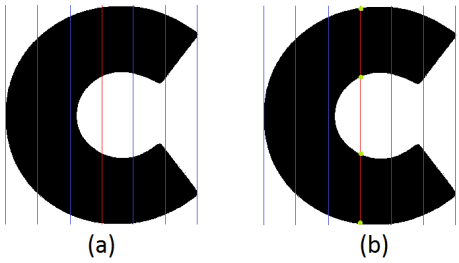

Figura 3.5: (a) Las rectas v_i y v_{m_i} en colores azul y rojo, respectivamente, con $N = 3$. (b) Puntos de corte entre v_{m_2} y Z en color verde.

Nótese que, para cada $w = (x, y) \in W$, $x = \frac{x_i + x_{i+1}}{2}$, pues w corta a la recta que pasa por la mitad del intervalo $[x_i, x_{i+1}]$, así que, tomemos todos los valores $y_j \in R$, con $j \in \{1, \ldots, |W|\}$, tales que $(\frac{x_i + x_{i+1}}{2}, y_j) \in W$ y $y_j < y_k$ para $j < k$, esta última restricción es para que los valores de y_j sean tomados de forma ordenada. Para cada valor j, si el segmento de recta $l_{y_j, y_{j+1}}$ está contenido en el interior de la curva Z, se trazará el rectángulo de lados v_j, v_{j+1}, y cuya base pase por los puntos y_j y y_{j+1} (Figura 3.6).

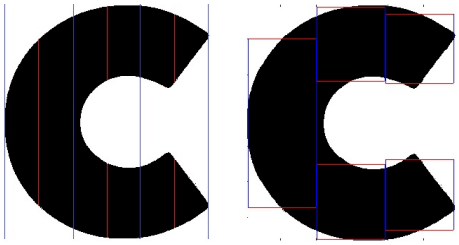

Figura 3.6: Se eliminan los fragmentos de recta que caen fuera del interior de Z, sólo se consideran los que están dentro. Se trazan los rectángulos cuya base sean los puntos de corte de la curva Z con las rectas que pasan por el punto medio

Así, con este algoritmo, se logró cubrir la región $int(Z)$ con N rectángulos de igual ancho, Figura 3.7. Para regiones no conexas, basta con aplicar este algoritmo en cada una de las componentes conexas de la región.

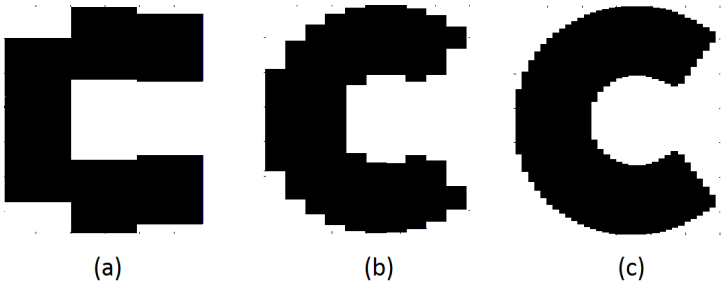

Figura 3.7: Aproximación final por rectángulos: (a) $N = 3$. (b) $N = 10$. (c) $N = 32$.

En resumen, la metodología del algoritmo es la siguiente:

1. Dividir el interior de la región Z a considerar, en N subregiones de igual ancho, utilizando rectas verticales v.

2. Considerar la recta vertical v_m que pasa por el punto medio del ancho de cada subregión, y todos sus puntos de corte y_j con la curva Z.

3. Si el segmento de recta entre dos puntos de corte $l_{y_j,y_{j+1}}$, está contenido en el interior de Z, trazar los rectángulos que tengan como lados las

rectas v que delimitan a cada subregión, y cuya base pase por los puntos y_j y y_{j+1}.

3.3. Implementación del método simple

Se implementó el método simple en un script de Matlab, utilizando el siguiente código:

```
% Inicializar variables
clear all
clc
close all
mypath='D:\MATLAB\';
I=imread(strcat(mypath,'Cbin.png'));
I=rgb2gray(I); [n,m]=size(I);
subplot(121)
imagesc(I); colormap(gray); axis image
color = 0; Feature = 255*ones(n,m);
proy = zeros(1,2); k = 0;
cont = 0; contResiduo = 0;
contRectangulos = 0; rect = 0;
Puntos = zeros(100,2);

% Definir el numero de intervalos que se utilizaran
N = 32;

% Proyeccion al eje x
for i=1:m
    suma = sum(I(:,i));
    if k==0 && suma < 255*n
        proy(1) = i;
        k = 1;
    end
    if k==1 && suma == 255*n
        proy(2) = i-1;
        k = 2;
    end
end

% Calcular la longitud del intervalo
longi = floor((proy(2)-proy(1))/N);
residuo = rem(proy(2)-proy(1),N);

```

```
36 % Ubicar los puntos medios de cada intervalo sobre la curva
37 if rem(longi,2) == 0
38     puntom = longi/2;
39     q = 1;
40 else
41     puntom = (longi+1)/2;
42     q = 0;
43 end
44
45 j = proy(1) + puntom - longi;
46
47 for k = 1:N
48     color = 0;
49
50     if contResiduo < residuo
51         j = j + longi + 1;
52     else
53         j = j + longi;
54     end
55
56     contResiduo = contResiduo + 1;
57
58     for i = 1:n
59         if I(i,j) == color
60             cont = cont + 1;
61             if color == 0;
62                 Puntos(cont,:) = [i,j];
63                 color = 255;
64             else
65                 Puntos(cont,:) = [i-1,j];
66                 color = 0;
67             end
68             contRectangulos = contRectangulos + 1;
69         end
70     end
71 end
72
73 % Construir el feature
74 contRectangulos = contRectangulos/2;
75
76 for k = 1:2:cont
77
78     cutoff = proy(1,1)+residuo*(longi+1);
79
80     if Puntos(k,2) < cutoff
```

```
81          y = Puntos(k,2)-puntom+1;
82          z = y + longi + 1;
83          Feature(Puntos(k,1):Puntos(k+1,1),y:z) = 0;
84      else
85          y = Puntos(k,2)-puntom+1;
86          z = y + longi;
87          Feature(Puntos(k,1):Puntos(k+1,1),y:z) = 0;
88      end
89      rect = rect + 1;
90  end
91
92  subplot(122)
93  imagesc(Feature); colormap(gray); axis image
```

Los resultados que se obtuvieron, para casos sencillos, fueron los siguientes: primero, para el caso donde la región de interés es un cuadrado, tomando $N = 1$, obtenemos que, la plantilla que determina la característica de tipo Haar, es la misma que la imagen del objeto de estudio, Figura 3.8, esto era de esperarse, ya que el método aproxima a la región por medio de rectángulos.

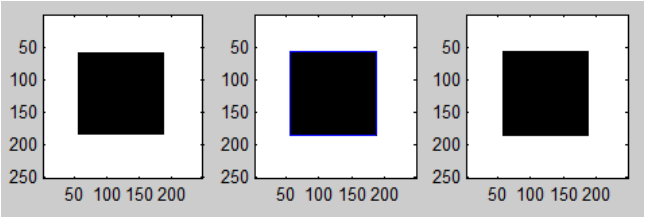

Figura 3.8: Resultado de aplicar el método simple a un cuadrado, con $N = 1$.

Para el caso en donde el objeto de estudio es un triángulo, tomando $N = 5$, obtenemos la plantilla de la Figura 3.9.

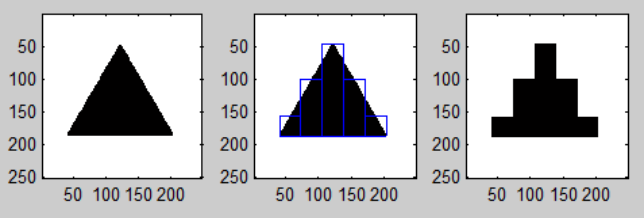

Figura 3.9: Resultado de aplicar el método simple a un triángulo, con $N = 5$.

Y por último, cuando el objeto de estudio es un círculo, tomando $N = 4$, obtenemos la plantilla de la Figura 3.10.

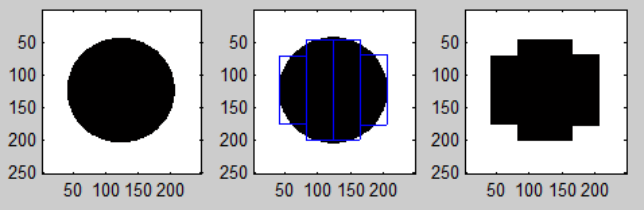

Figura 3.10: Resultado de aplicar el método simple a un círculo, con $N = 4$.

3.4. Análisis del algoritmo

En esta sección se desarrolla el análisis agregado del algoritmo, el cual, muestra que, para cada valor n, una secuencia de operaciones toma, en el peor de los casos, un tiempo total $T(n)$ [19]. En otras palabras, con el análisis agregado del algoritmo, se determina una cota superior $T(n)$ para el costo total de una secuencia de n operaciones.

Así, en el peor de los casos, el costo promedio de una operación (también conocido como costo amortiguado) es $\frac{T(n)}{n}$. El costo amortiguado del algoritmo se utiliza para mostrar que el costo promedio de una operación, sobre una secuencia de operaciones, es pequeño, a pesar de que, probablemente, existan algunas operaciones dentro de la secuencia cuyo costo de ejecución sea elevado.

Partiendo del código de la sección 3.3, y definiendo como t_i al tiempo que se requiere para ejecutar la línea i (costo), tenemos que el tiempo total de

ejecución es $C = \sum t_i s_i$, con $i \in \{1, \ldots, 93\}$, y donde s_i es el número de veces que se ejecuta la línea i. Considerando el peor de los casos:

- Los valores de $s_2, \ldots, s_{14}, s_{17}, s_{30}, s_{33}, s_{34}, s_{37}, \ldots, s_{43}, s_{45}, s_{71}, s_{74}, s_{90}, s_{92}$ y s_{93} están acotados superiormente por uno.
- Los valores de $s_{20}, s_{21}, \ldots, s_{29}$ y s_{70} están acotados superiormente por m.
- Los valores de $s_{47}, s_{48}, s_{50}, \ldots, s_{54}$ y s_{56} están acotados superiormente por N.
- Los valores de $s_{59}, \ldots, s_{69}$ están acotados superiormente por $N \cdot n$.
- Los valores de $s_{76}, s_{78}, s_{80} \ldots, s_{89}$ están acotados superiormente por $\frac{\text{cont}}{2}$.

Para cualquier i no mencionada anteriormente, el valor de s_i es cero. Teniendo estos valores, una primera cota superior para el costo está dada por:

$$C < a_0 + a_1 m + (a_2 + a_3 n)N + \frac{a_4}{2}\text{cont}$$

con:

- $a_0 = t_2 + \cdots + t_{14} + t_{17} + t_{30} + t_{33} + t_{34} + t_{37} + \cdots + t_{43} + t_{45} + t_{71} + t_{74} + t_{90} + t_{92} + t_{93}$.
- $a_1 = t_{20} + t_{21} + \cdots + t_{29} + t_{70}$.
- $a_2 = t_{47} + t_{48} + t_{50} + \cdots + t_{54} + t_{56}$.
- $a_3 = t_{59} + \cdots + t_{69}$.
- $a_4 = t_{76} + t_{78} + t_{80} + \cdots + t_{89}$.

Generalmente, los valores de N (número de intervalos en los que se divide la imagen) y $\frac{\text{cont}}{2}$ (número de rectángulos que se trazan dentro de la imagen), no crecen más que el valor de n (número de filas de la matriz asociada a la imagen), a menos que se tome una partición muy fina del intervalo, lo cual, es impráctico para la solución del problema, ya que mientras menos rectángulos contenga la plantilla obtenida, se calcula en menor tiempo la característica tipo Haar.

Así, para un cierto valor n de filas en la matriz asociada a la imagen, tanto N como $\frac{\text{cont}}{2}$, están acotados superiormente por n. Tomando $k = \text{máx}\{n, m\}$, una nueva cota superior es:

$$C < a_0 + a_1 m + (a_2 + a_3 n)N + \frac{a_4}{2}\text{cont} < a_0 + (a_1 + a_2 + a_4)k + a_3 k^2 = T$$

Al ser, el costo C, acotado superiormente por una ecuación cuadrática, el orden del algoritmo será $O(k^2)$.

Capítulo 4

Método optimizado para trazar rectángulos

4.1. Introducción

En la sección anterior, se aproximó $int(Z)$ por medio de un número fijo de rectángulos. En esta sección, se propone un método para aproximarlo por el menor número de rectángulos posibles, es decir, que cada rectángulo trazado dentro de la región, en cada iteración, sea el que ocupe la mayor área posible dentro de esta, ya que ésta aproximación se utilizará después para crear una plantilla que determine una característica tipo Haar, y mientras menos rectángulos contenga dicha plantilla, menos tiempo de cálculo se necesitará para hallar la suma de los pixeles comprendidos en ella.

4.2. Método para trazar rectángulos paralelos a los ejes

Sea Z una curva cerrada simple tal que $int(Z)$ es una región conexa, convexa y acotada, y sea h una línea horizontal que corta a Z en, al menos, dos puntos. Sean z_1^h y z_2^h los puntos de corte de dicha línea horizontal con la curva Z, con $[z_1^h]_x < [z_2^h]_x$ y tales que la distancia entre $[z_1^h]_x$ y $[z_2^h]_x$ sea máxima, es decir, que para cualquier otro par de puntos z_3^h y z_4^h en los que h corta a Z, se tenga que $|[z_2^h]_x - [z_1^h]_x| \geq |[z_4^h]_x - [z_3^h]_x|$. Tomemos ahora las líneas verticales v_1 y v_2 que pasan por los puntos z_1^h y z_2^h, respectivamente

(Figura 4.1) y consideremos los conjuntos:

$$A = \{a \in R | a \text{ es un punto de corte entre } v_1 \text{ y } Z\}$$

$$B = \{b \in R | b \text{ es un punto de corte entre } v_2 \text{ y } Z\}$$

Tenemos que $A \neq \emptyset \neq B$ pues, al menos, $z_1^h \in A$ y $z_2^h \in B$. Ahora, consideremos los conjuntos:

$$D_A = \{||z_1^h - a||, \text{ para todo } a \in A\}$$

$$D_B = \{||z_2^h - b||, \text{ para todo } b \in B\}$$

Y definamos:

$$\hat{a} = \max_a D_A$$

$$\hat{b} = \max_b D_B$$

$$c = \min\{\hat{a}, \hat{b}\}$$

Así, $c = (c_1, c_2)$ para $c_1 = [z_1^h]_x$ ó $c_1 = [z_2^h]_x$, y algún $c_2 \in R$.

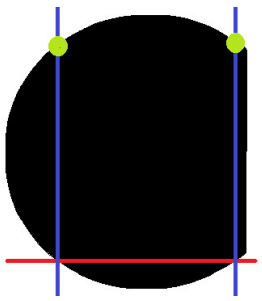

Figura 4.1: La recta h en color azul, las rectas v_1 y v_2 en color rojo, y los puntos $\hat{a}$, $\hat{b}$ en verde.

Para la línea horizontal h dada, formemos el rectángulo R_h cuyos vértices están en $z_1^h, z_2^h, ([z_1^h]_x, c_2)$ y $([z_2^h]_x, c_2)$, como se ilustra en la Figura 4.2. Notemos que el área de dicho rectángulo es $A(R_h) = ([z_2^h]_x - [z_1^h]_x)|c_2 - [z_1^h]_y|$.

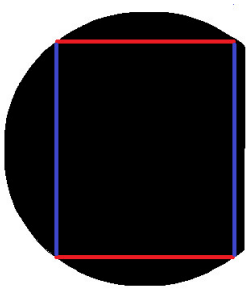

Figura 4.2: Rectángulo R_h.

Ahora, consideraremos todas las posibles líneas horizontales h que corten a Z en, al menos, dos puntos, y a los rectángulos R_h asociados a ellas, y denotaremos:

$$h_{\text{máx}} = \max_{h}\{A(R_h)|h \text{ línea horizontal, corta en, al menos, dos puntos a } Z\}$$

$$R_{\text{máx}} = R_{h_{\text{máx}}}$$

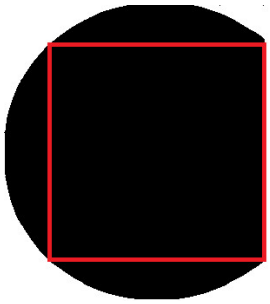

Figura 4.3: Rectángulo $R_{h_{\text{máx}}}$; subdivide al interior de Z en cuatro ubregiones convexas.

En resumen, la metodología del algoritmo es la siguiente:

1. Trazar una recta horizontal h que corte a la curva Z en, al menos, dos puntos.
2. Trazar las rectas verticales v_1 y v_2 que pasan por el primer y último punto de corte de h con Z, (ordenados de menor a mayor en referencia al valor de su coordenada x), respectivamente.

3. Elegir la recta vertical (v_1 ó v_2) que esté contenida, en menor longitud, en el interior de Z.

4. Completar el rectángulo que tiene como lados h y v_1, (ó h y v_2), según la recta que se haya elegido en 3.

5. Repetir los pasos del 1 al 4 para toda recta horizontal h que corte en, al menos, dos puntos a Z, y elegir, de todos los rectángulos que se generen, el que cubra la mayor área.

Afirmación 4.1. Sea Z una curva cerrada simple tal que $int(Z)$ es una región convexa, $R_{\text{máx}} \subset int(Z)$.

Prueba:

Dado que $int(Z)$ es una región convexa, basta con demostrar que los vértices de $R_{\text{máx}}$ están contenidos en $int(Z)$ para que todo el rectángulo esté contenido ahí. Sean $z_1^h, z_2^h, ([z_1^h]_x, c_2)$ y $([z_2^h]_x, c_2)$ los vértices de $R_{\text{máx}}$ y $\hat{b} = (b_1, b_2)$ como se construyeron en el algoritmo. Sin pérdida de la generalidad, supongamos que $c_1 = [z_1^h]_x$ (la demostración es análoga para el otro caso), entonces $c = ([z_1^h]_x, c_2)$. Por dicha construcción, tenemos que $z_1^h, z_2^h, c \in int(Z)$, así, falta demostrar que $([z_2^h]_x, c_2) \in int(Z)$, pero ésto es cierto ya que $[z_2^h]_x \in [[z_1^h]_x, [z_2^h]_x]$ y $c_2 \in [[z_1^h]_y, b_2]$ (si $[z_1^h]_y \leq b_2$), ó $c_2 \in [b_2, [z_1^h]_y]$ (si $b_2 \leq [z_1^h]_y$), con $b_2 \in Z$. Así, $R_{\text{máx}} \subset int(Z)$. □

Con este algoritmo hemos hallado, dada una recta horizontal h, el mayor rectángulo contenido en una región, que tenga como base la recta h. No obstante, dicho rectángulo no es el mayor contenido dentro de la región, en la siguiente sección se hablará un poco más a detalle de ello.

4.3. Método para trazar rectángulos rotados

Consideremos ahora la Figura 4.4, la cual se obtiene de rotar, una elipse vertical, 45° en sentido contrario de las manecillas del reloj. Al aplicar el método de la sección 4.2 a dicha figura, se obtiene el rectángulo enmarcado en azul, el cual, claramente podría ser más grande si se dibujara de otra forma.

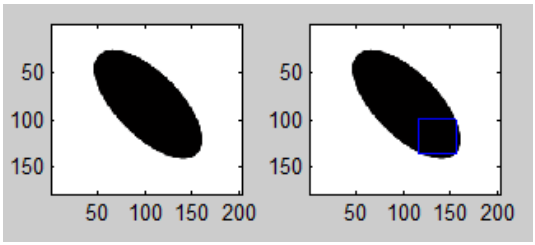

Figura 4.4: Resultado de aplicar el método de la sección 4.2 a una elipse vertical rotada 45° en sentido contrario de las manecillas del reloj.

Para solventar este detalle, procederemos de la siguiente forma: imitaremos el procedimiento de la sección anterior, pero ahora considerando una rotación θ aplicada a la imagen, para $\theta \in [0, 2\pi)$. Así, para cada ángulo θ habrá un rectángulo asociado $R^{\theta}_{\text{máx}}$, el cual, se calcula de la misma manera que $R_{\text{máx}}$ pero partiendo de la curva Z', la cual se obtiene de aplicar la rotación θ a la curva original Z (Figura 4.5) (notemos que si $\theta = 0$, entonces $Z' = Z$). Denotaremos:

$$\theta_{\text{máx}} = \max_{\theta}\{A(R^{\theta}_{\text{máx}})|\theta \in [0, 2\pi)\}$$

$$M_1 = R^{\theta_{\text{máx}}}_{\text{máx}}$$

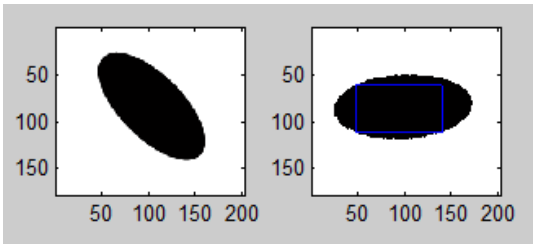

Figura 4.5: Resultado de aplicar el método de la sección 4.2 (considerando rotaciones) a la Figura 4.4. El rectángulo máximo se obtiene al rotar 48° la figura, es decir, $\theta_{\text{máx}} = 48°$.

Para regresar a la imagen original, basta con rotar $-\theta_{\text{máx}}$ la imagen con el rectángulo trazado.

Proposición 4.1. $R^{\theta+\frac{\pi}{2}}_{\text{máx}}$ se obtiene de rotar $\frac{\pi}{2}$ el rectángulo $R^{\theta}_{\text{máx}}$.

Nota: Se toma en cuenta como eje de rotación la intersección entre las rectas $h_{\text{máx}}$ y v_1, ó $h_{\text{máx}}$ y v_2 (Figura 4.1), según sea el caso. Ésto va a depender del valor de c; si $c = \hat{a}$ entonces se considera el corte con v_1, si $c = \hat{b}$, se considera el corte con v_2.

Prueba:

Sean $z_1, z_2, ([z_1]_x, c_2)$ y $([z_2]_x, c_2)$, los vértices del rectángulo $R^{\theta}_{\text{máx}}$ y $c = (c_1, c_2)$ como se construyeron en el algoritmo. Supongamos que $c_1 = [z_1]_x$, (en este caso el eje de rotación es el punto de corte de las rectas $h_{\text{máx}}$ y v_1, para el otro caso, la demostración es análoga, pero ahora considerando $c_1 = [z_2]_x$), apliquemos ahora el algoritmo para la recta horizontal que pasa por los puntos $(-[z_1]_y, [z_1]_x)$ y $(-c_2, [z_1]_x)$ de la curva Z' (la cual se obtiene de rotar $\frac{\pi}{2}$ la curva Z), esto producirá un nuevo punto $d = (d_1, d_2)$, con el cual, se construyen los otros dos vértices del rectángulo, $(-[z_1]_y, d_2)$ y $(-c_2, d_2)$. Por la construcción del algoritmo, tanto $[z_2]_x$ como d_2, son el máximo del mismo conjunto, así que, dada la unicidad del máximo, tenemos que $d_2 = [z_2]_x$ y los vértices del rectángulo serían $(-[z_1]_y, [z_1]_x), (-c_2, [z_1]_x), (-[z_1]_y, [z_2]_x)$ y $(-c_2, [z_2]_x)$. Ahora, apliquemos la matriz de rotación de $\frac{\pi}{2}$,

$$T_{\frac{\pi}{2}} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

a los vértices de $R^{\theta}_{\text{máx}}$, z_1, z_2, c y $([z_2]_x, c_2)$. De ésto, resultan los vértices $(-[z_1]_y, [z_1]_x), (-[z_2]_y, [z_2]_x), (-c_2, [z_1]_x$ y $(-c_2, [z_2]_x)$, respectivamente. Al estar z_1 y z_2 en la misma línea horizontal, $[z_1]_y = [z_2]_y$, así, podemos cambiar el segundo vértice por $(-[z_1]_y, [z_2]_x)$. Notemos que los dos conjuntos de vértices coinciden, de ésto se sigue el resultado. □

La Proposición 4.1 ahorra el 75 % del trabajo para calcular M_1, pues sólo basta con calcular $R^{\theta}_{\text{máx}}$ para $\theta \in [0, \frac{\pi}{2})$, ya que los demás rectángulos se obtienen de ir rotando $\frac{\pi}{2}$, consecutivamente, $R^{\theta}_{\text{máx}}$, con $\theta \in [0, \frac{\pi}{2})$, y dado que esta rotación no altera el área de dichos rectángulos, concluimos que:

$$\theta_{\text{máx}} = \max_{\theta}\{A(R^{\theta}_{\text{máx}}) | \theta \in [0, \frac{\pi}{2})\}$$

Después de hallar M_1, se puede dividir la región Z en, a lo más, cinco regiones convexas, delimitadas por los lados de M_1 (Figura 4.10). Por la

Afirmación 3.1, estas regiones son convexas, pues se pueden ver como la intersección de la curva Z (que es convexa), con algún rectángulo (el cual, también es convexo).

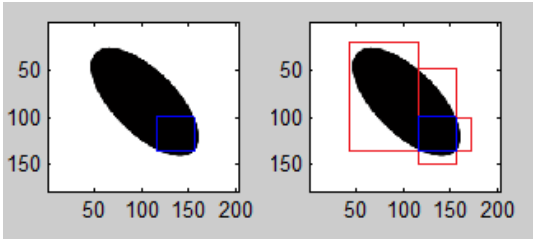

Figura 4.6: Después de trazar el primer rectángulo (azul), se puede dividir la región en, a lo más, cinco regiones convexas, las cuatro contenidas en los rectángulos rojos, y la contenida en el azul.

Ahora, procedamos a iterar este algoritmo. La manera correcta de hacerlo es, calcular el rectángulo M_1 para $int(Z)$, de ahí, considerar las regiones sobrantes no cubiertas por dicho rectángulo; calcular, para cada una de ellas, su respectivo rectángulo M_1, y así sucesivamente. Ésto producirá una serie de ángulos $\{\theta_1, ..., \theta_N\}$, (en donde N es el número de iteraciones), no necesariamente iguales, con los cuales sería difícil de utilizar el concepto de imagen integral, pues, dicho concepto, trabaja con un ángulo fijo. Por lo tanto, se ofrece una manera alterna de iterar:

1. Para cada $\theta \in [0, \frac{\pi}{2})$, calcular $R^{\theta}_{\text{máx}}$.
2. Calcular $M_1 = R^{\theta_0}_{\text{máx}}$, para algún $\theta_0 \in [0, \frac{\pi}{2})$.
3. Dividir la región no cubierta por rectángulos en regiones convexas.
4. Sea Γ el conjunto de regiones convexas contenidas en $int(Z)$ no cubiertas por rectángulos. Escoger la región que tenga área máxima $\gamma_{\text{máx}} = \text{máx}_{\gamma}\{A(\gamma)|\gamma \in \Gamma\}$.
5. Calcular el rectángulo $R^{\theta_0}_{\text{máx}}$ sobre la región $\gamma_{\text{máx}}$.
6. Si el área cubierta por todos los rectángulos $\{M_1, R^{\theta_0}_{\text{máx}_1}, ..., R^{\theta_0}_{\text{máx}_N}\}$, (en donde N es el número de veces que ya se repitió 5), es menor que la proporción p del área de $int(Z)$, con $p \in (0, 1]$, es decir, si

$$A(M_1) + \sum_{k=1}^{N} A(R^{\theta_0}_{\text{máx}_k}) < p * A(int(Z))$$

entonces, regresar a 3, de lo contrario, el algoritmo habrá concluido.

4.4. Implementación del método de la sección 4.2

Se implementó el método de la sección 4.2 en un script de Matlab, utilizando el siguiente código:

```
clear all
clc
close all
mypath='D:\MATLAB\';
I=imread(strcat(mypath,'Triangulo.png'));
I=rgb2gray(I);
[n,m]=size(I); regionmax = I;
subplot(121)
imagesc(I); colormap(gray); axis image
Feature = 255*ones(n,m);

%Definir el numero de rectangulos que se trazaran
N = 4;

for i = 1:N

    %Hallar el rectangulo mas grande contenido en la region
    [B, Feature] = optimo(n,m,regionmax,Feature);
    if i == 1
        MV = B;
    else
        MV = cat(2,MV,B);
    end
    copiaI = I;

    %Hallar las regiones convexas sobrantes
    regionmax = regiones(MV, copiaI,n,m,N);
end

```

```
30 subplot(122)
31 imagesc(Feature); colormap(gray); axis image
```

Los resultados que se obtuvieron, para casos sencillos, fueron los siguientes: primero, para el caso donde la región a considerar es un cuadrado, tomando $N = 1$, obtenemos que, la plantilla que determina la característica tipo Haar, es la misma que la imagen original (Figura 4.7), ésto era de esperarse, ya que, al igual que el método simple, este método también aproxima a la región por medio de rectángulos.

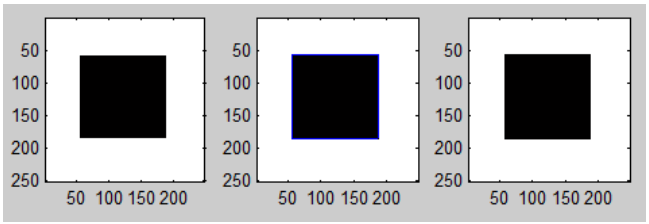

Figura 4.7: Resultado de aplicar el método optimizado a un cuadrado, con $N = 1$.

Para el caso en donde el objeto de interés es un triángulo, con $N = 4$ obtenemos la plantilla de la Figura 4.8.

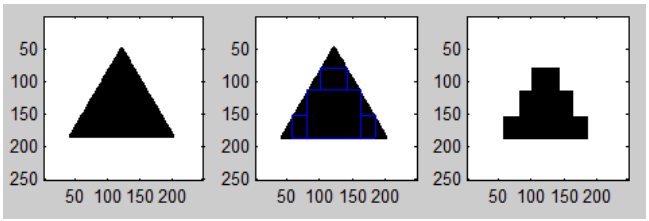

Figura 4.8: Resultado de aplicar el método optimizado a un triángulo, con $N = 4$.

Por último, cuando la región de interés es circular, con $N = 5$ obtenemos la plantilla de la Figura 4.9.

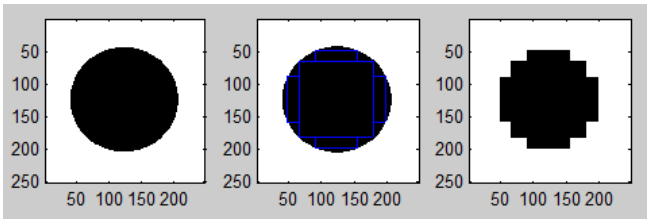

Figura 4.9: Resultado de aplicar el método optimizado a un círculo, con $N = 5$.

4.5. Implementación del método de la sección 4.3

Se implementó el método de la sección 4.3 en un script de Matlab, utilizando el siguiente código:

```
clear all
clc
close all
mypath='D:\MATLAB\';
I=imread(strcat(mypath,'Elipse.png'));
I=rgb2gray(I);
[n,m]=size(I);
subplot(131)
imagesc(I); colormap(gray); axis image
hold on;
Feature = 255*ones(n,m);
regionmax = I;
Imax = I;
AreaMaxima = 0;
anguloMax = 0;
Bmax = zeros(1,8);

%Definir el numero de rectangulos que se trazaran
N = 5;

for i = 1:N
    if i == 1
    for angulo = 0:89

```

```
25            % Rotar imagen
26            Rot = imrotate(I,angulo,'crop');
27            [n2,m2] = size(Rot);
28            Rot = limpiarImg(Rot,n2,m2);
29
30            % Calcular rectangulo de area maxima
31            [Feature,Imax,AreaMaxima,anguloMax,Bmax] = optimoRot(
                  n2,m2,Rot,Feature,Imax,AreaMaxima,angulo,anguloMax,
                  Bmax);
32        end
33        regionmax = Imax;
34        B = Bmax;
35        [n,m] = size(Imax);
36        end
37
38        if i > 1
39            [B,Feature] = optimo(n,m,regionmax,Feature);
40        end
41        if i == 1
42            MV = Bmax;
43        else
44            MV = cat(2,MV,B);
45        end
46        copiaI = Imax;
47        regionmax = regiones(MV,copiaI,n,m,iter);
48    end
49
50    % Regresar feature al angulo original de la imagen
51    Feature = imrotate(Feature,-anguloMax,'crop');
52    Feature = limpiarImg(Feature,n2,m2);
53    subplot(133)
54    imagesc(Feature); colormap(gray); axis image;
```

Para ejemplificar este método, primero se considerará un cuadrado rotado 60° en sentido de las manecillas del reloj. Si a este rectángulo se le aplica el método de la sección 4.2 (es decir, sin considerar rotaciones), con $N = 1$, la aproximación que resulta es la de la Figura 4.10, la cual, apenas cubre el 56 % de la imagen inicial.

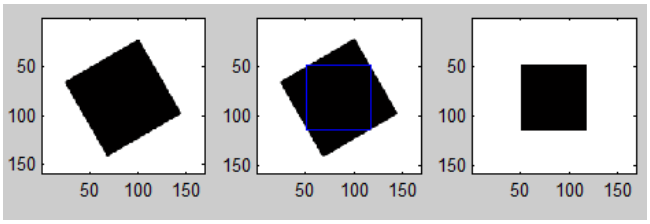

Figura 4.10: Resultado de aplicar el método de la sección 4.2, a un cuadrado rotado 60°.

Por otra parte, al aplicar el método de la sección 4.3, (es decir, considerando rotaciones), para el mismo valor de N, resulta la aproximación de la Figura 4.11, la cual cubre completamente la imagen. Para dicha aproximación, el valor de $\theta_{\text{máx}}$ es 60°.

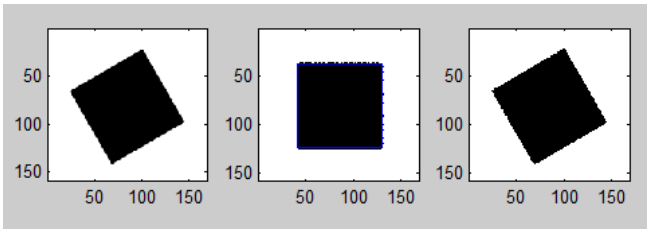

Figura 4.11: Resultado de aplicar el método de la sección 4.3, a un cuadrado rotado 60°. Para este ejemplo, $\theta_{\text{máx}} = 60°$.

Por último, consideraremos un triángulo rotado 50° en sentido de las manecillas del reloj. La aproximación que resulta, utilizando el método de la sección 4.3, es la de la Figura 4.12

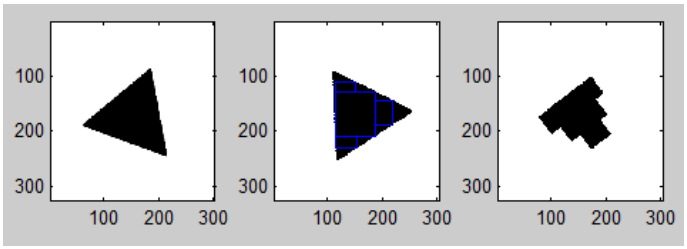

Figura 4.12: Resultado de aplicar el método de la sección 4.3, a un triángulo rotado 50°. Para este ejemplo, $\theta_{\text{máx}} = 53°$.

4.6. Análisis de los algoritmos

Partiendo del código de la sección 4.4, y definiendo como t_i al tiempo que se requiere para ejecutar la línea i (costo), tenemos que el tiempo total de ejecución es $C_1 = \sum t_i s_i$, con $i \in \{1, \ldots, 31\}$, y donde s_i es el número de veces que se ejecuta la línea i. Considerando el peor de los casos:

- Los valores de $s_1, \ldots, s_{10}, s_{13}, s_{28}, s_{30}$ y s_{31} están acotados superiormente por uno.
- Los valores de $s_{15}, s_{19}, \ldots, s_{24}$ están acotados superiormente por N.
- El valor de s_{18} está acotado superiormente por $N(a_0+(a_1+a_3)k+a_2k^2)$, (véase Apéndice 8.1.).
- El valor de s_{27} está acotado superiormente por $N(b_0+(b_1+b_3)k+(b_2+b_4 \cdot iter+b_5)k^2+b_6k^3)$, (véase Apéndice 8.2.).

En donde $k = \text{máx}\{n, m\}$. Para cualquier i no mencionada anteriormente, el valor de s_i es cero. Teniendo estos valores, una primera cota superior para el costo está dada por:

$$C_1 < c_0 + (c_1 + c_2 + c_3)N$$

con:

- $c_0 = t_1 + \ldots + t_{10} + t_{13} + t_{28} + t_{30} + t_{31}$.
- $c_1 = t_{15} + t_{19} + \ldots + t_{24}$.

- $c_2 = t_{18}(a_0 + (a_1 + a_3)k + a_2k^2)$.
- $c_3 = t_{27}(b_0 + (b_1 + b_3)k + (b_2 + b_4N + b_5)k^2 + b_6k^3)$.

Como el valor de N (número de iteraciones) no crece mucho cuando los valores de n y m (número de filas y columnas) incrementan, (ya que el objetivo es aproximar la región por el menor número de rectángulos posibles), se considerará como constante. Así, al ser N constante, el grado del polinomio que es cota de C_1, es tres, y por lo tanto, el orden del algoritmo será $O(k^3)$.

Análogamente, ahora partiendo del código de la sección 4.5, y definiendo como u_i al tiempo que se requiere para ejecutar la línea i (costo), tenemos que el tiempo total de ejecución es $C_2 = \sum u_i r_i$, con $i \in \{1, \ldots, 54\}$, y donde r_i es el número de veces que se ejecuta la línea i. Considerando el peor de los casos:

- Los valores de $r_1, \ldots, r_{16}, r_{19}, r_{48}, r_{51}, r_{53}$ y r_{54} están acotados superiormente por uno.
- Los valores de $r_{21}, r_{22}, r_{32}, \ldots, r_{36}, r_{38}, \ldots, r_{46}$ están acotados superiormente por N.
- El valor de r_{52} está acotado superiormente por $d_0 + d_1k + d_2k^2$, (véase Apéndice 8.4.).
- Los valores de r_{23}, r_{26} y r_{27} están acotados superiormente por $90 \cdot N$.
- El valor de r_{28} está acotado superiormente por $90 \cdot N(d_0 + d_1k + d_2k^2)$, (véase Apéndice 8.4.)
- El valor de r_{31} está acotado superiormente por $90 \cdot N(a_0 + (a_1 + a_3)k + a_2k^2)$, (véase Apéndice 8.3.).
- El valor de r_{47} está acotado superiormente por $N(b_0 + (b_1 + b_3)k + (b_2 + b_4 \cdot iter + b_5)k^2 + b_6k^3)$, (véase Apéndice 8.2.).

En donde $k = \text{máx}\{n, m\}$. Para cualquier i no mencionada anteriormente, el valor de r_i es cero. Teniendo estos valores, una primera cota superior para el costo está dada por:

$$C_2 < e_0 + e_2 + (e_1 + e_3 + e_4 + e_5 + e_6)N$$

con:

- $e_0 = u_1 + \ldots + u_{16} + u_{19} + u_{48} + u_{51} + u_{53} + u_{54}$.
- $e_1 = u_{21} + u_{22} + u_{32} + \ldots + u_{36} + u_{38} + \ldots + u_{46}$.
- $e_2 = u_{52}(d_0 + d_1k + d_2k^2)$.
- $e_3 = 90(u_{23} + u_{26} + u_{27})$.
- $e_4 = 90u_{28}(d_0 + d_1k + d_2k^2)$.
- $e_5 = 90u_{31}(a_0 + (a_1 + a_3)k + a_2k^2)$.
- $e_6 = u_{47}(b_0 + (b_1 + b_3)k + (b_2 + b_4N + b_5)k^2 + b_6k^3)$.

Similarmente al análisis anterior, consideraremos el valor de N como constante, así, el grado del polinomio que es cota de C_2, es tres, y por lo tanto, el orden del algoritmo será $O(k^3)$.

Capítulo 5

Construcción de las plantillas de convolución

5.1. Introducción

Hasta ahora, lo único que se ha determinado son dos métodos para trazar rectángulos a partir de una curva dada. El objetivo de la tesis se aborda en este capítulo: construir la plantilla de convolución que mejor se adapte a la morfología del objeto de estudio, y que mejore el funcionamiento de los algoritmos de aprendizaje automático en comparación con las plantillas tradicionales. Esta plantilla determinará una característica tipo Haar con la que se entrenará el AdaBoost.

Para hacer la construcción de la plantilla, se considerará el escenario más general: una imagen en RGB. Para ello se utilizará el algoritmo de $k-$medias [5], el cual, es un algoritmo de agrupamiento que tiene como objetivo particionar un conjunto en k grupos distintos. Al aplicar este algoritmo, la imagen va a quedar particionada en varias regiones; a cada una de estas regiones se le aplica alguno de los dos métodos desarrollados, no sin antes corroborar que cumplan los supuestos para aplicar cada método (ser región conexa en el simple, y convexa en el optimizado).

Hay que notar que, a pesar de que el algoritmo agrupa a los pixeles de una imágen en k diferentes categorías, el número de regiones en las que la imagen queda dividida, no necesariamente es k, puede variar, pero sí es, al menos, dicho número. Sabiendo ésto, se procederá a la construcción de las plantillas.

5.2. Construcción utilizando el método simple

Sea T un conjunto de entrenamiento, y sea I una imagen que pertenece a T, como se ilustra en la Figura 5.1. Se construirá la plantilla que determina la característica de tipo Haar asociada a I, utilizando el método simple para trazar rectángulos, como sigue:

1. Seleccionar $I \in T$.

2. Aplicar el algoritmo de k-medias a I (ver Figura 5.2).

3. Seleccionar las regiones $R_{i_1}, R_{i_2}, \ldots, R_{i_j}$ que describan a nuestro objeto de estudio.

4. Aplicar el método simple a cada región, para generar las aproximaciones $A_{\alpha_1}, A_{\alpha_2}, \ldots, A_{\alpha_j}$ por medio de rectángulos.

5. Sustituir cada R_{i_*} por su respectiva aproximación A_{i_*}.

6. Trazar un rectángulo, del tamaño de la imagen, que contenga a todas las aproximaciones, (ver Figura 5.3).

Figura 5.1: Imagen del *Trypanosoma cruzi* tomada de una muestra de sangre.

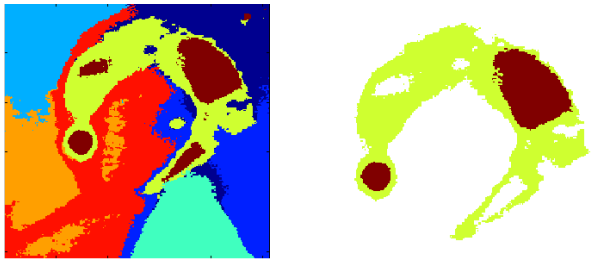

Figura 5.2: Aplicación del algoritmo $k-$medias, con $k = 8$, y selección de las regiones que determinan el objeto de estudio.

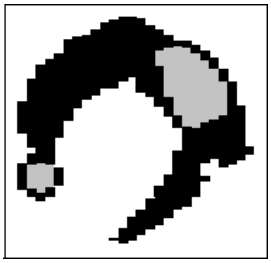

Figura 5.3: Plantilla asociada a las regiones de la Figura 5.2, obtenida con el método simple.

5.3. Construcción utilizando el método optimizado

Sea T un conjunto de entrenamiento, y sea I una imagen que pertenece a T. Se construirá la plantilla que determina la característica de tipo Haar asociada a I, utilizando el método optimizado para trazar rectángulos, como sigue:

1. Seleccionar $I \in T$.
2. Aplicar el algoritmo de k-medias a I.
3. Seleccionar las regiones $R_{i_1}, R_{i_2}, \ldots, R_{i_j}$ que describan a nuestro objeto de estudio.

4. Si alguna región R_{i_k} es no convexa, calcular su envolvente convexa $E(R_{i_k})$, (ver Figura 5.4).

5. Aplicar el método optimizado a cada región convexa, y a las envolventes convexas de las regiones no convexas, para generar las aproximaciones $A_{\alpha_1}, A_{\alpha_2}, \ldots, A_{\alpha_j}$ por medio de rectángulos.

6. Sustituir cada R_{i_*} por su respectiva aproximación A_{i_*}.

7. Trazar un rectángulo, del tamaño de la imagen, que contenga a todas las aproximaciones.

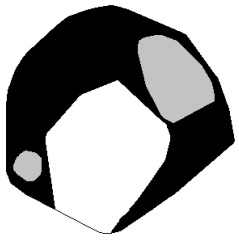

Figura 5.4: Cálculo de las envolventes convexas de las regiones a considerar.

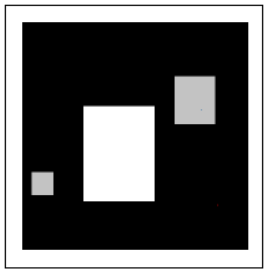

Figura 5.5: Plantilla asociada a las regiones de la Figura 5.2, obtenida con el método optimizado.

El valor de la característica asociada a la plantilla de la Figura 5.5 será un vector de tres entradas: e_1, e_2 y e_3. La primer entrada (e_1) se calcula aplicando la plantilla que toma por valores $1, -1$ y 0, en los pixeles comprendidos

en las regiones blanca, negra y gris, respectivamente. La segunda entrada (e_2) se calcula aplicando la plantilla que toma por valores $-1, 0$ y 1, en los pixeles comprendidos en las regiones blanca, negra y gris, respectivamente. Y la última entrada (e_3) se calcula aplicando la plantilla que toma por valores $0, 1$ y -1, en los pixeles comprendidos en las regiones blanca, negra y gris, respectivamente. Una vez que se tenga dicho vector para cada una de las imágenes del conjunto de entrenamiento, se genera, con AdaBoost, un clasificador fuerte para cada entrada, es decir, con el conjunto de escalares $\{e_i^1, \ldots, e_i^{|T|}\}$ (en donde $|T|$ es el número de elementos del conjunto de entrenamiento), se generan los clasificadores G_i para cada $i \in \{1, 2, 3\}$. Con estos tres clasificadores, se genera un nuevo clasificador $G_{(1,2,3)}$ para evaluar los vectores $v = (a, b, c)$ del conjunto de prueba. Este nuevo clasificador se define como sigue: $G_{(1,2,3)}(v) = 1$ si $|G_1(a) + G_2(b) + G_3(c)| > 0$, y $G_{(1,2,3)}(v) = -1$ en caso contrario.

Capítulo 6

Experimentación

6.1. Introducción

Para probar el desempeño de las plantillas que se generan con los métodos descritos anteriormente, se recortaron imágenes de la base de datos DIARETDBI - Standard Diabetic Retinopathy Database [22] de la Universidad Tecnológica de Lapperanta (Finlandia). Esta base de datos contiene imágenes tomadas del fondo del ojo humano, en donde aparece resaltado el nervio óptico; algunas de estas imágenes contienen manchas amarillas, las cuales indican la presencia de retinopatía diabética en el ojo del paciente (Figura 6.1), la cual, es una enfermedad causada por el daño a los vasos sanguíneos de la retina a causa de la diabetes. En la experimentación, para probar las plantillas diseñadas, se discriminó entre la parte de la imagen en donde se encuentra el nervio óptico y lo demás (venas, arterias, mácula, etc.), es decir, la experimentación no se centró en diferenciar los ojos infectados de retinopatía diabética, de los demás, sino en determinar cuál fragmento de la imagen es nervio óptico, y cuál no. Se escogió esta base de datos ya que la imagen del fondo del ojo contiene venas que pueden ser confundidas con las venas contenidas en el nervio óptico; esto agrega un poco más de dificultad al problema de clasificación, a pesar de que el color del nervio óptico contrasta con el resto del fondo del ojo. A continuación, se presentan los detalles de la experimentación, que van desde la selección del conjunto de entrenamiento y prueba, hasta las plantillas utilizadas para entrenar al clasificador, y el número de iteraciones que requiere éste para el entrenamiento.

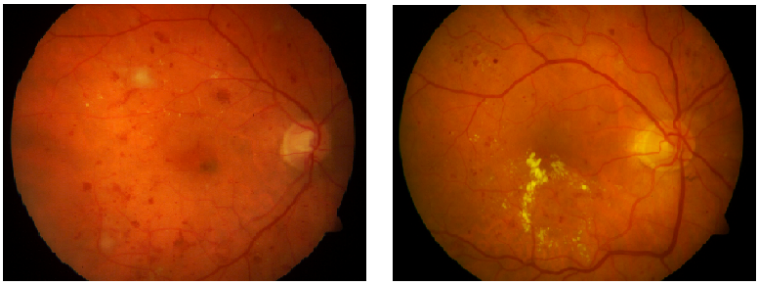

Figura 6.1: Ejemplos de las imágenes contenidas en la base de datos original. De izquierda a derecha: imagen de un ojo no infectado con retinopatía, y uno infectado, respectivamente.

De la base de datos mencionada anteriormente, se recortaron 50 imágenes de 174×192 pixeles, en donde se encuentra el nervio óptico, y 50 imágenes en donde no se encuentra, es decir, en donde aparecen venas y otras regiones del fondo del ojo, fuera del nervio (Figura 6.2).

Figura 6.2: Algunos recortes tomados de la base de datos mencionada, con los que se formaron los conjuntos de entrenamiento y de prueba.

El algoritmo de clasificación que se utilizó para los experimentos fue el clasificador clásico de AdaBoost [23], implementado en Matlab por Dirk-Jan Kroon de la Universidad de Twente (Holanda). Éste programa sólo requiere definir el número de iteraciones y, en este caso, fijamos dicho valor a 100 iteraciones para todos los experimentos realizados, es decir, se generaron 100 clasificadores débiles en cada experimento.

Así mismo, se utilizó la técnica de validación cruzada [5] para validar el modelo, dividiendo el conjunto inicial en $k = 10$ subconjuntos. Las plantillas que se utilizaron en la experimentación se consideraron del tamaño de las imágenes del conjunto de prueba (174×192), y sin escalamiento.

6.2. Experimentación utilizando el método simple

Se escogió arbitrariamente una imagen del conjunto de imágenes del nervio óptico y se generó, utilizando el método simple (con $N = 6$), la plantilla de convolución asociada a la imagen, que se ilustra en la Figura 6.3, y dado que el nervio óptico puede estar tanto del lado izquierdo, como del lado derecho del ojo, la plantilla diseñada se reflejó (de izquierda a derecha).

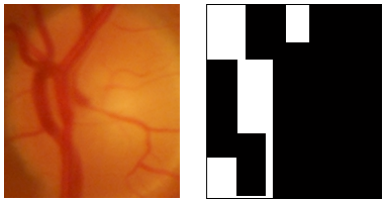

Figura 6.3: Imagen original y su plantilla asociada, diseñada con el método simple.

Por lo tanto, para este experimento se utilizaron dos plantillas: la generada por el método simple y la reflexión de dicha plantilla (de izquierda a derecha), (ver Figura 6.4).

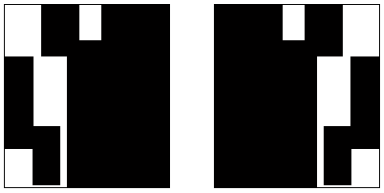

Figura 6.4: Plantillas que se utilizaron en la experimentación del método simple.

Los resultados que se obtuvieron fueron los siguientes:

k	Falsos negativos	Falsos Positivos	Error total
1	0/5	1/5	1/10
2	0/5	0/5	0/10
3	1/5	0/5	1/10
4	2/5	0/5	2/10
5	1/5	3/5	4/10
6	2/5	0/5	2/10
7	0/5	0/5	0/10
8	0/5	0/5	0/10
9	1/5	0/5	1/10
10	0/5	0/5	0/10
Error:	7/50	4/50	11/100

En donde k es el número del conjunto (de los 10 posibles) que se utiliza para validar el modelo (Figura 6.5), los falsos negativos son las imágenes del conjunto de prueba que contienen nervio óptico, pero que el modelo las clasificó erróneamente, y los falsos positivos son las imágenes del conjunto de prueba que no contienen nervio óptico, que fueron también clasificadas erróneamente por el modelo.

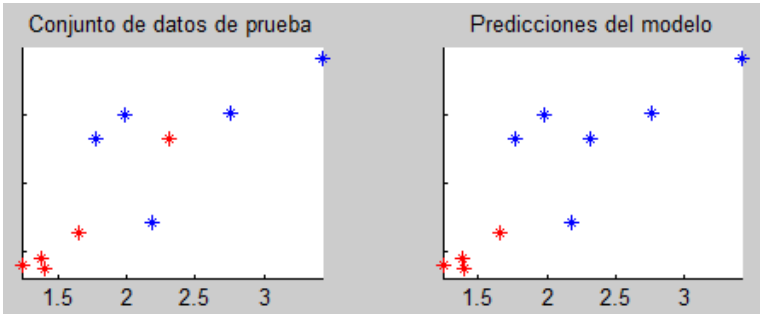

Figura 6.5: Experimento realizado con $k = 1$. En la primera imagen se muestra el conjunto de prueba que se utilizó y en la segunda las predicciones que arrojó el modelo, las viñetas azules son las imágenes que contienen nervio óptico, y las rojas las que no contienen. Se puede ver que el modelo clasificó bien todos los nervios ópticos, pero casificó mal una imagen que no era nervio óptico.

Se puede ver que el modelo arroja una buena clasificación del nervio óptico

respecto a los demás fragmentos del fondo del ojo: 89 % de las imágenes fueron bien clasificadas, mientras que el 86 % de las imágenes que contenían nervio óptico y el 92 % de las imágenes del fondo del ojo, las clasificó correctamente.

6.3. Experimentación utilizando el método optimizado

Ahora, para realizar esta experimentación, se utilizó la misma imagen escogida para generar la plantilla con el método simple, y se generó, ahora utilizando el método optimizado (sin iterar y sin rotar), la plantilla de convolución asociada a dicha imagen (Figura 6.6) y, bajo la misma justificación que en la sección anterior, reflejamos dicha plantilla (de izquierda a derecha), para utilizarla también en la experimentación.

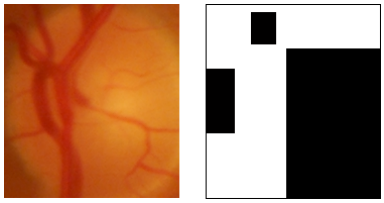

Figura 6.6: Imagen original y su plantilla asociada, diseñada con el método optimizado.

Así, para este experimento se utilizaron también dos plantillas: la generada por el método optimizado y la reflexión de dicha plantilla (de izquierda a derecha), cómo se ilustra en la Figura 6.7.

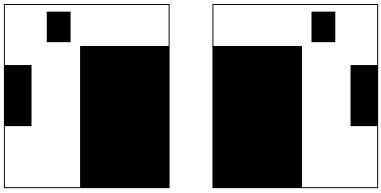

Figura 6.7: Plantillas que se utilizaron en la experimentación del método optimizado.

Los resultados que se obtuvieron, con este método, fueron los siguientes:

k	**Falsos negativos**	**Falsos Positivos**	**Error total**
1	0/5	0/5	0/10
2	0/5	0/5	0/10
3	0/5	0/5	0/10
4	0/5	0/5	0/10
5	0/5	0/5	0/10
6	0/5	0/5	0/10
7	1/5	0/5	1/10
8	0/5	1/5	1/10
9	0/5	2/5	2/10
10	0/5	0/5	0/10
Error:	1/50	3/50	4/100

En donde k es el número del conjunto (de los 10 posibles) que se utiliza para validar el modelo (ver Figura 6.8), los falsos negativos son las imágenes del conjunto de prueba que contienen nervio óptico, pero que el modelo las clasificó erróneamente, y los falsos positivos son las imágenes del conjunto de prueba que no contienen nervio óptico, que fueron también clasificadas erróneamente por el modelo.

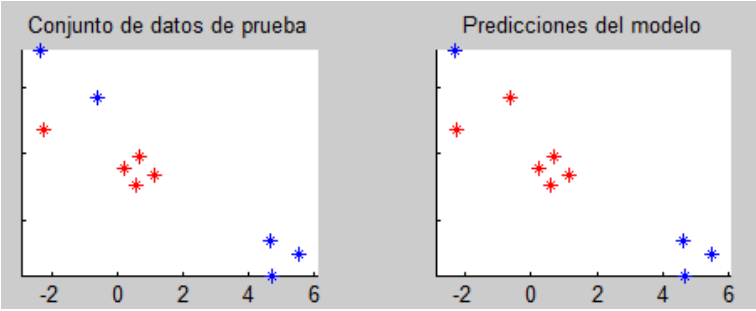

Figura 6.8: Experimento realizado con $k = 7$. En la primera imagen se muestra el conjunto de prueba que se utilizó y en la segunda las predicciones que arrojó el modelo, las viñetas azules son las imágenes que contienen nervio óptico, y las rojas las que no contienen. Se puede ver que el modelo clasificó bien todas las imágenes que no contenían nervio óptico, pero casificó mal una imagen que sí lo contenía.

Se puede ver que el modelo arroja una muy buena clasificación del nervio óptico (mejor que la obtenida por las plantillas generadas con el método simple) respecto a los demás fragmentos del fondo del ojo: 96 % de las imágenes fueron bien clasificadas, mientras que el 98 % de las imágenes que contenían nervio óptico y el 94 % de las imágenes del fondo del ojo, las clasificó correctamente.

6.4. Comparación con plantillas comúnmente utilizadas

Para realizar esta comparación, se utilizaron las plantillas que comúnmente se usan para la detección de objetos, las cuales se ilustran en la Figura 6.9. Éstas se utilizaron bajo las mismas condiciones que las anteriores: De tamaño 174×192 pixeles, y sin escalamiento.

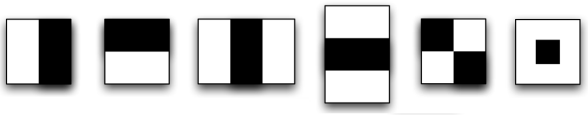

Figura 6.9: Plantillas que generalmente se utilizan en los problemas de detección de patrones en imágenes.

Los resultados obtenidos se observan en la tabla siguiente:

k	**Falsos negativos**	**Falsos Positivos**	**Error total**
1	0/5	0/5	0/10
2	0/5	0/5	0/10
3	0/5	1/5	1/10
4	2/5	0/5	2/10
5	0/5	0/5	0/10
6	0/5	1/5	1/10
7	1/5	0/5	1/10
8	2/5	1/5	3/10
9	0/5	0/5	0/10
10	0/5	0/5	0/10
Error:	5/50	3/50	8/100

En comparación con los resultados anteriores, este modelo tiene mejores resultados que el modelo de la plantilla generada con el método simple, pero no es tan bueno como el modelo de la plantilla generada con el método optimizado, para este conjunto de imágenes. En este modelo (ver Figura 6.10), el 92 % de las imágenes totales fueron bien clasificadas, de entre ellas, el 90 % de las imágenes que contenían nervio óptico y el 94 % de las imágenes que contenían fragmentos del fondo del ojo, fueron bien clasificadas.

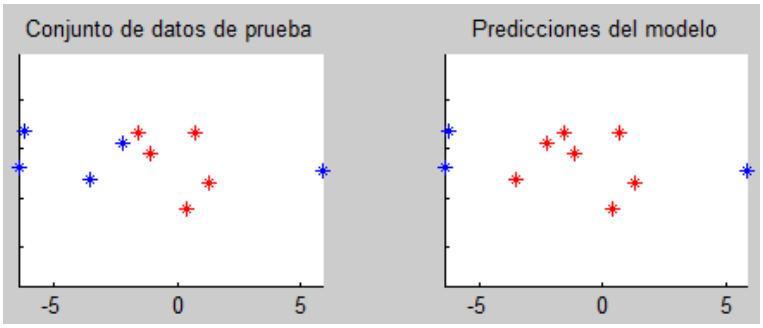

Figura 6.10: Experimento realizado con plantillas comunes y $k = 4$. Se puede ver que el modelo resultante de utilizar plantillas comunes clasificó bien todas las imágenes que no contenían nervio óptico, pero casificó mal dos imágenes que sí lo contenían.

La comparación se puede apreciar mejor en la siguiente tabla:

Plantilla	**Falsos negativos**	**Falsos Positivos**	**Error total**
Método simple	7/50	4/50	11/100
Método optimizado	1/50	3/50	4/100
Viola y Jones (2001)	5/50	3/50	8/100

Así podemos ver que, el mejor desempeño para el problema de clasificación, se logra con el método optimizado, el cual, sólo utilizó dos plantillas, en comparación con las seis plantillas usadas comúnmente. Ésto es favorable para el algoritmo AdaBoost ya que, para encontrar el mejor clasificador débil, debe de recorrer una matriz de tamaño 2×90 (número de plantillas $\times$ número de imágenes usadas para el entrenamiento), en comparación con la matriz de

6×90 que tendría que recorrer si se utilizaran las plantillas convencionales. Esto ahorra dos terceras partes del trabajo, y además, las plantillas utilizadas son generadas por un algoritmo que justifica su diseño, en comparación con las plantillas tradicionales que se diseñan de manera subjetiva.

6.5. Discusión sobre el uso de plantillas

Por último, en esta sección, se detallan algunos puntos a favor y en contra acerca de generar las plantillas de convolución para la extracción de características. Algunas ventajas que tiene el diseño de las plantillas son las siguientes:

- Dada una imagen ya segmentada, se elimina el diseño empírico: es importante, al momento de hacer investigación, el justificar el uso de métodos o instrumentos para el desarrollo de la misma. Los algoritmos desarrollados en este trabajo proveen ese uso justificado de las plantillas diseñadas.
- Uso de menos plantillas: para los problemas de detección en donde el objeto que se quiere detectar es un objeto rígido (es decir, no cambia de forma), basta con diseñar una plantilla para entrenar al algoritmo.

A su vez, también tiene desventajas este diseño, las cuales son:

- Las plantillas diseñadas contienen más rectángulos que las utilizadas comúnmente: esto, al momento de calcular el valor de la característica, necesita de un mayor trabajo computacional.
- Requiere un cómputo previo: muchos de los algoritmos implementados de AdaBoost, ya incluyen las plantillas comunes para hacer más práctica su utilización. El uso de las plantillas generadas, requiere un cómputo extra para calcularlas, y en ambos algoritmos, el orden mínimo es $O(k^2)$.

Capítulo 7

Conclusiones

Se concluye esta tesis con un resumen de los aportes de este documento, y los diversos caminos posteriores que puede seguir esta investigación

7.1. Conclusión

En este trabajo se propusieron las bases para generar plantillas de convolución, en vez de utilizar las plantillas que comúnmente se usan en el reconocimiento de objetos en imágenes. Ésta es un área a la que no se le había prestado mucho interés, ya que, los modelos generados por plantillas comunes, han tenido un buen desempeño de clasificación. Por varios años, desde el desarrollo del detector de rostros de Viola y Jones, los investigadores fueron utilizando las plantillas usadas por el detector sin preguntarse si eran las plantillas adecuadas para el problema que requerían tratar. Ésto, afortunadamente, produjo los resultados esperados por ellos, tanto así, que las mismas plantillas se siguen utilizando hasta ahora con muy buenos resultados, y no solamente para detectar rostros, sino en una gran variedad de problemas diversos. No obstante, se quizo dar el toque de formalidad en este tema, es por eso que se propusieron dos métodos: el simple y el optimizado.

El método simple, consiste en hacer una aproximación a una región por medio de rectángulos, similar a la de la aproximación de una curva cuando se integra. Fue desarrollado a partir de una región conexa, y luego generalizado a cualquier región posible, aplicando dicho método en cada una de las componentes conexas de la región. El método optimizado, por su parte, aproxima a la región por medio del rectángulo más grande que quepa en ella;

ésto para ahorrar tiempo en el cálculo de los valores de las características. Fue desarrollado, primeramente, para regiones convexas y generalizado después a cualquier región posible. El trabajo realizado con estos métodos fue con imágenes binarias, así que, después de su desarrollo, se generalizaron ambos métodos para su utilización en imágenes en RGB y se hicieron análisis de cada algoritmo.

Finalmente, se probó que en el problema de detectar el nervio óptico del ojo, las plantillas generadas por el método optimizado le ahorraron dos tercios de trabajo al AdaBoost, y no sólo eso, sino que tuvieron, en un 4 %, un mejor desempeño que las plantillas utilizadas comúnmente. Así que, al menos para este conjunto de imágenes de prueba, el método optimizado tuvo un trabajo redondo, no sólo disminuyendo el trabajo del AdaBoost, sino mejorando ligeramente el desempeño de dicho algoritmo. Esto podría hacer pensar que, para algunos problemas en particular, el diseño de estas plantillas podría tener un futuro alentador.

7.2. Trabajo futuro

En la experimentación, se probaron algunas cuestiones sobre el diseño de las plantillas, para ambos métodos, pero todavía quedaron abiertas otras cuantas. Algunos de los caminos que puede seguir esta investigación son los siguientes:

- En el método simple, ¿cuál sería el número óptimo N de intervalos en los que se debe de dividir la región para crear la plantilla? Ésto para lograr un mejor desempeño del AdaBoost.

- En el método optimizado, ¿cuál sería el número de veces óptimo que se debe de iterar el algoritmo para crear la plantilla? De igual manera, para tener un mejor desempeño del AdaBoost.

- Para ambos métodos, ¿es significativo el uso de más de dos colores para el diseño de la plantilla? o ¿es redundante su uso? Ya que si no aporta algo significativo, no vale la pena que el costo del cómputo sea mayor.

- En el diseño de las plantillas, por simplicidad tomamos, el cuadrado que contiene a las aproximaciones, del mismo tamaño que la imagen original. De ésto surge la interrogante, ¿cuál es el tamaño óptimo del

cuadrado que contiene a todas las aproximaciones? De nuevo, pensando en un mejor desempeño del AdaBoost.

- El diseño de las plantillas para imágenes en RGB depende mucho del resultado del algoritmo k-medias, ésto lo hace susceptible a errores que pueda tener dicho algoritmo en alguna imagen en particular. Habrá que mejorar la generalización de los algoritmos para imágenes en RGB, ya sea utilizando un algoritmo de agrupamiento diferente a k-medias, o encontrar la manera de que los métodos propuestos trabajen adecuadamente en conjunto con él.

De ésto se puede concluir que aún hay mucho camino por recorrer, y que no basta con generar plantillas que se aproximen bien a la forma del objeto, sino que estas plantillas deben de generar un buen desempeño de clasificación en conjunto con el AdaBoost y mientras menos rectángulos utilicen en su diseño, menos tiempo de cómputo se requiere para calcular el valor de la característica asociada a ellas. Sin duda, son muchos aspectos no triviales a considerar cuando se trata de abordar este tema, los cuales no se pudieron cubrir por completo en este trabajo pero quedan abiertos para futuras investigaciones. Como diría el escritor Frank Herbert: "No hay un final verdadero, es simplemente el momento en el que paras la historia".

Capítulo 8

Apéndice

8.1. Apéndice 1: Análisis de la función optimo()

```
1  function [B,Feature] = optimo(n,m,I,Feature)
2
3  A=zeros(n,8);
4  largo = 0;
5  maximo = 0;
6
7  %Trazar lineas horizontales
8  for i=1:n
9      suma = sum(I(i,:));
10     if suma < 255*(m-1)
11         for j=1:m
12             if I(i,j) == 0
13               largo = largo + 1;
14               A(largo,1:4) = [i,j,i,j+(255*m-suma)/255-1];
15               break
16             end
17         end
18     end
19 end
20
21 %Hallar rectangulo maximo
22 for i=1:largo
23     fila = A(i,1);
24     col1 = A(i,2);
25     col2 = A(i,4);
```

```
26      area1 = 0; area2 = 0;
27      lado1 = 0; lado2 = 0;
28      if I(fila-1,col1) == 0 && I(fila-1,col2) == 0
29          suma1 = sum(I(1:fila,col1));
30          suma2 = sum(I(1:fila,col2));
31          if suma2 > suma1
32              suma = suma2;
33          else
34              suma = suma1;
35          end
36          lado1 = (255*fila - suma)/255 - 1;
37          if col2 > col1
38              area1 = lado1*(col2 - col1);
39          end
40          if col1 > col2
41              area1 = lado1*(col1-col2);
42          end
43      end
44      if I(fila+1,col1) == 0 && I(fila+1,col2) == 0
45          suma1 = sum(I(fila:n,col1));
46          suma2 = sum(I(fila:n,col2));
47          if suma2 > suma1
48              suma = suma2;
49          else
50              suma = suma1;
51          end
52          lado2 = (255*(n-fila) - suma)/255;
53          if col2 > col1
54              area2 = lado2*(col2 - col1);
55          end
56          if col1 > col2
57              area2 = lado2*(col1-col2);
58          end
59      end
60      if area1 >= area2
61          A(i,5:6) = [fila-lado1, col1];
62          A(i,7:8) = [fila-lado1, col2];
63          area = area1;
64      else
65          A(i,5:6) = [fila+lado2, col1];
66          A(i,7:8) = [fila+lado2, col2];
67          area = area2;
68      end
69      if area > maximo
70          maximo = area;
```

```
71          indice = i;
72      end
73  end
74
75  %Construir el feature
76  i = indice;
77  B = A(i,:);
78  if A(i,1) < A(i,5)
79      Feature(A(i,1):A(i,5),A(i,2):A(i,4)) = 0;
80  else
81      Feature(A(i,5):A(i,1),A(i,2):A(i,4)) = 0;
82  end
```

Partiendo de este código, y definiendo como t_i al tiempo que se requiere para ejecutar la línea i (costo), tenemos que el tiempo total de ejecución es $C = \sum t_i s_i$, con $i \in \{1, \ldots, 82\}$, y donde s_i es el número de veces que se ejecuta la línea i. Considerando el peor de los casos:

- Los valores de $s_1, s_3, \ldots, s_5, s_{10}, s_{73}, s_{76}, \ldots, s_{82}$ están acotados superiormente por uno.
- Los valores de $s_8, \ldots, s_{10}, s_{17}$ y s_{18} están acotados superiormente por n.
- Los valores de $s_{11}, \ldots, s_{16}$ están acotados superiormente por $n \cdot m$.
- Los valores de $s_{22}, \ldots, s_{72}$ están acotados superiormente por $largo$.

Para cualquier i no mencionada anteriormente, el valor de s_i es cero. Teniendo estos valores, una primera cota superior para el costo está dada por:

$$C < a_0 + (a_1 + a_2 m)n + a_3 \cdot largo$$

con:

- $a_0 = t_1 + t_3 + \cdots + t_5 + t_{10} + t_{73} + t_{76} + \ldots + t_{82}$.
- $a_1 = t_8 + \cdots + t_{10} + t_{17} + t_{18}$.
- $a_2 = t_{11} + \cdots + t_{16}$.
- $a_3 = t_{22} + \cdots + t_{72}$.

Asímismo, el valor de *largo* (largo del cuadrado) está acotado por n (número de filas), ya que el cuadrado está contenido dentro de la imagen. Tomando $k = \text{máx}\{n, m\}$, una nueva cota superior es:

$$C < a_0 + (a_1 + a_3)k + a_2k^2$$

Al ser, el costo C, acotado superiormente por una ecuación cuadrática, el orden del algoritmo es $O(k^2)$.

8.2. Apéndice 2: Análisis de la función regiones()

```
function I = regiones(MB,I,dim,m,iter)

n = length(MB);
MV = zeros(2*iter,2*iter);
Orden1 = zeros(1,n/2);

% Copiar los valores de las columnas
for i = 2:2:n
    Orden1(1,i/2) = MB(i);
end

% Ordenarlos de menor a mayor
Orden1 = sort(Orden1);

% Contar cuantos elementos hay
col = 1;
for i = 2:(n/2)
    if Orden1(i) ~= Orden1(i-1)
        col = col + 1;
    end
end

% Eliminar los elementos repetidos
Orden = zeros(1,col);
cont = 1;
Orden(1) = Orden1(1);
for i = 2:(n/2)
    if Orden1(i) ~= Orden(cont)
        cont = cont + 1;
        Orden(cont) = Orden1(i);
```

```
31      end
32  end
33
34  % Ordenar los valores de la columna junto con todos los
        valores de la fila que tomen
35  cont = 0;
36  for i = 1:col
37      for j = 1:2:n
38          if MB(j+1) == Orden(i)
39              cont = cont + 1;
40              MV(i,cont) = MB(j);
41          end
42      end
43      cont = 0;
44  end
45
46  suma1 = 0;
47  suma2 = 0;
48  areamax = 0;
49  min1 = dim;
50  min2 = dim;
51
52  arriba = 0;
53  abajo = 0;
54  Elegidos1 = zeros(1,3);
55  Elegidos2 = zeros(1,3);
56  Elegidosf = zeros(1,3);
57  venga = 0;
58  dale = 0;
59
60  for i = 1:(col-1)
61      cota = col - i;
62      for k = 1:cota
63          if venga == 0;
64          % Calcular lado minimo
65          for j = 1:2*iter
66              if MV(i,j) > 0 && MV(i,j) < min1
67                  min1 = MV(i,j);
68              end
69              if MV(i+k,j) > 0 && MV(i+k,j) < min2
70                  min2 = MV(i+k,j);
71              end
72          end
73
74          maxmin = max(min1,min2);
```

```
75
76          % Verificar si no hay rectangulos arriba
77          for t = 1:4:n
78              if maxmin > MB(t) && MB(t+1) <= Orden(i) && MB(t
                    +3) >= Orden(i+k)
79                  arriba = 1;
80              end
81              if maxmin > MB(t) && MB(t+1) >= Orden(i) && MB(t
                    +3) <= Orden(i+k)
82                  arriba = 1;
83              end
84              if maxmin > MB(t) && MB(t+1) < Orden(i) && Orden(i
                    ) < MB(t+3)
85                  arriba = 1;
86              end
87              if maxmin > MB(t) && MB(t+1) < Orden(i+k) && Orden
                    (i+k) < MB(t+3)
88                  arriba = 1;
89              end
90          end
91
92          % Si no hay, calcular area
93          if arriba == 0
94              Elegidos1 = [maxmin,Orden(i),Orden(i+k)];
95
96              suma1 = sum( sum( I(1:maxmin, Orden(i):Orden(i+k))
                    ));
97              completo = (Orden(i+k) - Orden(i) + 1)*maxmin;
98              suma1 = suma1/255;
99
100             suma1 = completo - suma1;
101
102             venga = 1;
103         end
104         end
105         arriba = 0;
106     end
107
108     for k = 1:cota
109         if dale == 0
110
111         % Calcular lado maximo
112         max1 = max(MV(i,:));
113         max2 = max(MV(i+k,:));
114         minmax = min(max1,max2);
```

```
115
116         % Verificar si no hay rectangulos abajo
117         for t = 1:4:n
118             if minmax < MB(t) && MB(t+1) <= Orden(i) && MB(t
                    +3) >= Orden(i+k)
119                 abajo = 1;
120             end
121             if minmax < MB(t) && MB(t+1) >= Orden(i) && MB(t
                    +3) <= Orden(i+k)
122                 abajo = 1;
123             end
124             if minmax < MB(t) && MB(t+1) < Orden(i) && Orden(i
                    ) < MB(t+3)
125                 abajo = 1;
126             end
127             if minmax < MB(t) && MB(t+1) < Orden(i+k) && Orden
                    (i+k) < MB(t+3)
128                 abajo = 1;
129             end
130         end
131
132         % Si no hay rectangulos, sumar area
133         if abajo == 0
134             Elegidos2 = [minmax,Orden(i),Orden(i+k)];
135
136             suma2 = sum( sum( I(minmax:m, Orden(i):Orden(i+k))
                    ));
137             completo = (Orden(i+k) - Orden(i) + 1)*(m - minmax
                    + 1);
138             suma2 = suma2/255;
139
140             suma2 = completo - suma2;
141
142             dale = 1;
143         end
144         end
145         abajo = 0;
146     end
147
148     if suma1 > areamax
149         areamax = suma1;
150         Elegidosf = Elegidos1;
151         up = 1;
152     end
153
```

```
154     if suma2 > areamax
155         areamax = suma2;
156         Elegidosf = Elegidos2;
157         up = 0;
158     end
159
160     suma1 = 0;
161     suma2 = 0;
162     dale = 0;
163     venga = 0;
164     min1 = m;
165     min2 = m;
166 end
167
168 %Calculo del area de las regiones laterales
169 suma3 = 0;
170 suma4 = 0;
171 for i = 1:dim
172     for j = 1:Orden(1)
173         if I(i,j) == 0
174             suma3 = suma3 + 1;
175         end
176     end
177 end
178
179 for i = 1:dim
180     for j = Orden(col):m
181         if I(i,j) == 0
182             suma4 = suma4 + 1;
183         end
184     end
185 end
186
187 %Tomar el area maxima
188 above = 0;
189 if areamax > suma3 && areamax > suma4
190     above = 1;
191 end
192
193 if suma3 > areamax
194     areamax = suma3;
195     izquierda = 1;
196 end
197
198 if suma4 > areamax
```

```
199     izquierda = 0;
200 end
201
202 if above == 1
203 for i = 1:dim
204     for j = 1:m
205         if up == 1
206             if j < Elegidosf(2) || j > Elegidosf(3) || i >
                    Elegidosf(1)
207                 I(i,j) = 255;
208             end
209         end
210         if up == 0
211             if j < Elegidosf(2) || j > Elegidosf(3) || i <
                    Elegidosf(1)
212                 I(i,j) = 255;
213             end
214         end
215     end
216 end
217 end
218
219 if above == 0
220
221     if izquierda == 1
222         for i = 1:dim
223             for j = 1:m
224             if j > Orden(1)
225                 I(i,j) = 255;
226             end
227             end
228         end
229     end
230
231     if izquierda == 0
232         for i = 1:dim
233             for j = 1:m
234             if j < Orden(col)
235                 I(i,j) = 255;
236             end
237             end
238         end
239     end
240 end
```

Partiendo de este código, y definiendo como t_i al tiempo que se requiere para ejecutar la línea i (costo), tenemos que el tiempo total de ejecución es $C = \sum t_i s_i$, con $i \in \{1, \ldots, 240\}$, y donde s_i es el número de veces que se ejecuta la línea i. Considerando el peor de los casos:

- Los valores de $s_1, s_3, \ldots, s_5, s_{10}, s_{13}, s_{16}, s_{21}, s_{24}, \ldots, s_{26}, s_{32}, s_{35}, s_{44}, s_{46}, s_{47}, \ldots, s_{50}, s_{52}, \ldots, s_{58}, s_{166}, s_{169}, s_{170}, s_{177}, s_{185}, s_{188}, \ldots, s_{191}, s_{193}, s_{194}, s_{195}, s_{196}, s_{198}, \ldots, s_{200}, s_{202}, s_{216}, s_{217}, s_{219}, s_{221}, s_{228}, s_{229}, s_{231}, s_{238}, s_{239}$, y s_{240} están acotados superiormente por uno.

- Los valores de $s_8, s_9, s_{17}, \ldots, s_{20}, s_{27}, \ldots, s_{29}, s_{30}, s_{31}, s_{171}, s_{176}, s_{179}, s_{184}, s_{203}, s_{215}, s_{222}, s_{227}, s_{232}$ y s_{237} están acotados superiormente por n.

- Los valores de $s_{37}, \ldots, s_{40}, s_{41}, s_{172}, \ldots, s_{175}, s_{180}, \ldots, s_{183}, s_{204}, \ldots, s_{214}, s_{223}, \ldots, s_{226}, s_{233}, \ldots, s_{236}$ están acotados superiormente por $n \cdot m$.

- Los valores de $s_{36}, s_{42}, s_{43}, s_{60}, s_{61}, s_{106}, s_{146}, s_{148}, \ldots, s_{152}, s_{154}, \ldots, s_{158}, s_{160}, \ldots, s_{166}$ están acotados superiormente por m.

- Los valores de $s_{65}, \ldots, s_{71}$ están acotados superiormente por $iter \cdot m^2$.

- Los valores de $s_{62}, s_{63}, s_{72}, s_{74}, s_{90}, s_{93}, s_{94}, s_{96}, \ldots, s_{98}, s_{100}, s_{102}, s_{105}, s_{108}, \ldots, s_{114}, s_{130}, s_{133}, s_{134}, s_{136}, \ldots, s_{138}, s_{140}, s_{142}, \ldots, s_{145}$ están acotados superiormente por m^2.

- Los valores de $s_{77}, \ldots, s_{89}, s_{117}, \ldots, s_{129}$ están acotados superiormente por $n \cdot m^2$.

Para cualquier i no mencionada anteriormente, el valor de s_i es cero. Teniendo estos valores, una primer cota superior para el costo está dada por:

$$C < b_0 + b_1 n + b_2 n \cdot m + b_3 m + b_4 m^2 \cdot iter + b_5 m^2 + b_6 n m^2$$

con:

- $b_0 = t_1 + t_3 + \ldots + t_5 + t_{10} + t_{13} + t_{16} + t_{21} + t_{24} + \ldots + t_{26} + t_{32} + t_{35} + t_{44} + t_{46} + \ldots + t_{50} + t_{52} + \ldots + t_{58} + t_{166} + t_{169} + t_{170} + t_{177} + t_{185} + t_{188} + \ldots + t_{191} + t_{193} + \ldots + t_{106} + t_{108} + \ldots + t_{200} + t_{202} + t_{216} + t_{217} + t_{219} + t_{221} + t_{228} + t_{229} + t_{231} + t_{238} + \ldots + t_{240}$.

- $b_1 = t_8 + t_9 + t_{17} + \ldots + t_{20} + t_{27} + \ldots + t_{31} + t_{171} + t_{176} + t_{179} + t_{184} + t_{203} + t_{215} + t_{222} + t_{227} + t_{232} + t_{237}$.
- $b_2 = t_{37} + \ldots + t_{41} + t_{172} + \ldots + t_{175} + t_{180} + \ldots + t_{183} + t_{204} + \ldots + t_{214} + t_{223} + \ldots + t_{226} + t_{233} + \ldots + t_{236}$.
- $b_3 = t_{36} + t_{42} + t_{43} + t_{60} + t_{61} + t_{106} + t_{146} + t_{148} + \ldots + t_{152} + t_{154} + \ldots + t_{158} + t_{160} + \ldots + t_{166}$.
- $b_4 = t_{65} + \ldots + t_{71}$.
- $b_5 = t_{62} + t_{63} + t_{72} + t_{74} + t_{90} + t_{93} + t_{94} + t_{96} + \ldots + t_{98} + t_{100} + t_{102} + t_{105} + t_{108} + \ldots + t_{114} + t_{130} + t_{133} + t_{134} + t_{136} + \ldots + t_{138} + t_{140} + t_{142} + \ldots + t_{145}$.
- $b_6 = t_{77} + \ldots + t_{89} + t_{117} + \ldots + t_{129}$.

Como el valor de *iter* (número de iteraciones) no crece mucho cuando los valores de n y m (número de filas y columnas) incrementan, (ya que el objetivo es aproximar la región por el menor número de rectángulos posibles), se considerará como constante. Tomando $k = \text{máx}\{n, m\}$, una nueva cota superior es:

$$C < b_0 + (b_1 + b_3)k + (b_2 + b_4 \cdot iter + b_5)k^2 + b_6 k^3$$

Al ser, el costo C, acotado superiormente por una ecuación cúbica, el orden del algoritmo es $O(k^3)$.

8.3. Análisis de la función optimoRot()

```
1 function [Feature,Imax,AreaMaxima,anguloMax,Bmax] = optimoRot(
      n,m,I,Feature,Imax,AreaMaxima,angulo,anguloMax,Bmax)
2
3 A=zeros(n,8);
4 largo = 0;
5 maximo = 0;
6
7 %Trazar lineas horizontales
8 for i=1:n
9       suma = sum(I(i,:));
10      if suma < 255*(m-1)
11            for j=1:m
```

```
12              if I(i,j) == 0
13                  largo = largo + 1;
14                  A(largo,1:4) = [i,j,i,j+(255*m-suma)/255-1];
15                  break
16              end
17          end
18      end
19  end
20
21  % Trazar rectangulos
22  for i=1:largo
23      fila = A(i,1);
24      col1 = A(i,2);
25      col2 = A(i,4);
26      area1 = 0; area2 = 0;
27      lado1 = 0; lado2 = 0;
28      if I(fila-1,col1) == 0 && I(fila-1,col2) == 0
29          suma1 = sum(I(1:fila,col1));
30          suma2 = sum(I(1:fila,col2));
31          if suma2 > suma1
32              suma = suma2;
33          else
34              suma = suma1;
35          end
36          lado1 = (255*fila - suma)/255 - 1;
37          if col2 > col1
38              area1 = lado1*(col2 - col1);
39          end
40          if col1 > col2
41              area1 = lado1*(col1-col2);
42          end
43      end
44      if I(fila+1,col1) == 0 && I(fila+1,col2) == 0
45          suma1 = sum(I(fila:n,col1));
46          suma2 = sum(I(fila:n,col2));
47          if suma2 > suma1
48              suma = suma2;
49          else
50              suma = suma1;
51          end
52          lado2 = (255*(n-fila) - suma)/255;
53          if col2 > col1
54              area2 = lado2*(col2 - col1);
55          end
56          if col1 > col2
```

```
57              area2 = lado2*(col1-col2);
58          end
59      end
60      if area1 >= area2
61          A(i,5:6) = [fila-lado1, col1];
62          A(i,7:8) = [fila-lado1, col2];
63          area = area1;
64      else
65          A(i,5:6) = [fila+lado2, col1];
66          A(i,7:8) = [fila+lado2, col2];
67          area = area2;
68      end
69      if area > maximo
70          maximo = area;
71          indice = i;
72      end
73  end
74
75  i = indice;
76  B = A(i,:);
77
78  %Area maxima por angulo
79  if area > AreaMaxima
80    AreaMaxima = area;
81    Imax = I;
82    anguloMax = angulo;
83    subplot(132)
84    imagesc(Imax); colormap(gray); axis image;
85    hold on
86    Bmax = B;
87    Feature(:,:) = 255;
88    if A(i,1) < A(i,5)
89      Feature(A(i,1):A(i,5),A(i,2):A(i,4)) = 0;
90    else
91      Feature(A(i,5):A(i,1),A(i,2):A(i,4)) = 0;
92    end
93  end
```

Partiendo de este código, y definiendo como t_i al tiempo que se requiere para ejecutar la línea i (costo), tenemos que el tiempo total de ejecución es $C = \sum t_i s_i$, con $i \in \{1, \ldots, 93\}$, y donde s_i es el número de veces que se ejecuta la línea i. Considerando el peor de los casos:

- Los valores de $s_1, s_3, \ldots, s_5, s_{10}, s_{73}, s_{75}, s_{76}, s_{79}, \ldots, s_{93}$ están acotados

superiormente por uno.

- Los valores de $s_8, \ldots, s_{10}, s_{17}$ y s_{18} están acotados superiormente por n.
- Los valores de $s_{11}, \ldots, s_{16}$ están acotados superiormente por $n \cdot m$.
- Los valores de $s_{22}, \ldots, s_{72}$ están acotados superiormente por $largo$.

Para cualquier i no mencionada anteriormente, el valor de s_i es cero. Teniendo estos valores, una primer cota superior para el costo está dada por:

$$C < a_0 + (a_1 + a_2 m)n + a_3 \cdot largo$$

con:

- $a_0 = t_1 + t_3 + \cdots + t_5 + t_{10} + t_{73} + t_{75} + t_{76} + t_{79} + \ldots + t_{93}$.
- $a_1 = t_8 + \cdots + t_{10} + t_{17} + t_{18}$.
- $a_2 = t_{11} + \cdots + t_{16}$.
- $a_3 = t_{22} + \cdots + t_{72}$.

Por otra parte, el valor de $largo$ (largo del cuadrado) está acotado por n (número de filas), ya que el cuadrado está contenido dentro de la imagen. Tomando $k = \text{máx}\{n, m\}$, una nueva cota superior es:

$$C < a_0 + (a_1 + a_3)k + a_2 k^2$$

Al ser, el costo C, acotado superiormente por una ecuación cuadrática, el orden del algoritmo es $O(k^2)$.

8.4. Análisis de la función limpiarImg()

```
1 function Rot = limpiarImg(Rot,n,m)
2
3 keep = 1;
4 %Izquierda
5 for i = 1:n
6     for j = 1:m
7         if Rot(i,j) == 0 && keep == 1
8             Rot(i,j) = 255;
```

```
9           else
10              keep = 0;
11          end
12      end
13      keep = 1;
14  end
15  %Derecha
16  for i = 1:n
17      for j = 1:m
18          if Rot(n-i+1,m-j+1) == 0 && keep == 1
19              Rot(n-i+1,m-j+1) = 255;
20          else
21              keep = 0;
22          end
23      end
24      keep = 1;
25  end
```

Partiendo de este código, y definiendo como t_i al tiempo que se requiere para ejecutar la línea i (costo), tenemos que el tiempo total de ejecución es $C = \sum t_i s_i$, con $i \in \{1, \ldots, 25\}$, y donde s_i es el número de veces que se ejecuta la línea i. Considerando el peor de los casos:

- Los valores de s_1, s_3, s_{14} y s_{25} están acotados superiormente por uno.
- Los valores de $s_5, s_{12}, s_{13}, s_{16}, s_{23}$ y s_{24} están acotados superiormente por n.
- Los valores de $s_6, \ldots, s_{11}, s_{17}, \ldots, s_{22}$ están acotados superiormente por $n \cdot m$.

Para cualquier i no mencionada anteriormente, el valor de s_i es cero. Teniendo estos valores, una primer cota superior para el costo está dada por:

$$C < d_0 + (d_1 + d_2 m)n$$

con:

- $d_0 = t_1 + t_3 + t_{14} + t_{25}$.
- $d_1 = t_5 + t_{12} + t_{13} + t_{16} + t_{23} + t_{24}$.

- $d_2 = t_6 + \cdots + t_{11} + t_{17} + \cdots + t_{22}$.

Tomando $k = \text{máx}\{n, m\}$, una nueva cota superior es:

$$C < d_0 + d_1 k + d_2 k^2$$

Al estar, el costo C, acotado superiormente por una ecuación cuadrática, el orden del algoritmo es $O(k^2)$.

Bibliografía

[1] Mitchell T. (1997). *Machine Learning*. McGraw Hill.

[2] Oren M., Papageorgiou C., Sinha P., Osuna E. and Poggio T. (1997). *Pedestrian Detection Using Wavelet Templates*. Computer Vision and Pattern Recognition, vol. 1, pages 193-199.

[3] Papageorgiou C., Oren M. and Poggio T. (1998). *A General Framework for Object Detection*. International Conference on Computer Vision.

[4] Viola P. and Jones M. (2001). *Rapid object detection using a Boosted Cascade of Simple Features*. Conference on Computer Vision and Pattern Recognition, vol. 1, pages 511-518.

[5] Hastie T., Tibshirani R. and Friedman J. (2001). *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer.

[6] Shapiro L. and Stockman G. (2001). *Computer Vision*. Prentice Hall.

[7] Lienhart R., Maydt J. (2002). *An Extended Set of Haar-like Features for Rapid Object Detection*. International Conference in Image Processing, vol. 1, pages 900-903.

[8] Mendelson S. and Smola A. (2002). *Advanced Lectures on Machine Learning*. Machine Learning Summer School 2002, Canberra, Australia. Springer.

[9] MacKay D. (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.

[10] Mita T., Kaneko T. and Hori O. (2005). *A General Framework for Object Detection*. International Conference on Computer Vision, vol. 2, pages 1619-1626.

[11] Barczak A., Johnson M. and Messom C. (2006). *Real-time Computation of Haar-like features at generic angles for detection algorithms*. Research Letters in the Information and Mathematical Sciences, vol. 9, pages 98-111.

[12] Bishop C. (2006). *Patern Recognition and Machine Learning*. Springer.

[13] Chen Q. and Georganas N. (2008). *Hand Gesture Recognition Using Haar- like Features and Stochastic Context Free Grammar*. IEEE Transactions on Instrumentation and Measurement, vol. 57, pages 1562-1571.

[14] Mavaddat N., Kim T. and Cipolla R. (2009). *Desing and evaluation of features that best define text in complex scene images*. Conference on Machine Vision Applications.

[15] Alpaydi, E. (2010) *Introduction to Machine Learning*. Massachussets Institute of Technology.

[16] Schapire R. and Freund Y. (2012). *Boosting: Foundations and Algorithms*. Massachussets Institute of Technology.

[17] Nixon M. and Aguado A. (2002). *Feature Extraction and Image Processing*. First edition. Newnes.

[18] Chui C. (1992) *An Introduction to Wavelets*. Academic Press.

[19] Cormen T., Leiserson C., Rivest R. and Stein C. (2003). *Introduction to algorithms*. Second edition. The MIT Press.

[20] Marsden J., Tromba A. (2008). *Cálculo Vectorial*. Quinta edición. Pearson.

[21] Apostol T. (2006). *Análisis Matemático*. Segunda edición. Reverté.

[22] http://www2.it.lut.fi/project/imageret/diaretdb1/

[23] http://www.mathworks.com/matlabcentral/fileexchange/27813-classic-adaboost-classifier/content/adaboost.m

[24] Pavani S., Delgado D. and Frangi A. (2010). *Haar-like features with optimally weighted rectangles for rapid object detection*. Pattern Recognition, vol. 43 issue 1, pages 160-172.

[25] DSouza D. and Yampolskiy R. V. (2012). *Baseline Avatar Face Detection using an Extended Set of Haar-like Features*. Proceedings of the Twenty-third Midwest Artificial Intelligence and Cognitive Science Conference. Pages 72-77.

[26] Wei Y., Tian Q. and Guo T. (2013). *An Improved Pedestrian Detection Algorithm Integrating Haar Like Features and HOG Descriptors*. Advances in Mechanical Engineering, vol. 2013, article ID 546206, 8 pages.

[27] Nasrollahi K., Moeslund T. B. and Rashidi M. (2013). *Haar-like Rectangular Features for Biometric Recognition*. International Conference in Biometrics (ICB), pages 1-7.

[28] Nasrollahi K. and Moeslund T. (2013). *Haar-like features for robust real-time face recognition*. IEEE International Conference on Image Processing. IEEE Signal Processing Society, 2013.