Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

"Segmentación simplificada para etiquetado de imágenes utilizando *quadtrees*".

Tesis Profesional

Que para obtener el título de:
Ingeniero en Ciencias de la Computación

Presenta
Gerardo Rodmar Conde Márquez

Asesor
Dr. Luis Enrique Sucar Succar

Co-Asesor
Dr. Abraham Sánchez López

Puebla, Pue. Invierno 2010

Dedico y regalo este trabajo,

A los que nunca nadie les ha dedicado una tesis, sea por que en donde viven no se ha tenido el detalle de construir universidades, sea por que habiéndolas las oportunidades no han sido suficientes para ingresar y concluir.

A los compañeros que por diversas circunstancias tuvieron que abandonar las aulas.

A las personas honestas que leen, y escriben, para construir un mundo mejor.

A la comunidad académica y científica que insistentemente sigue entregando su vida por la educación.

A los grupos estudiantiles que trabajan para impulsar la ciencia.

Agradezco,

Al Dios de la Vida que me ha formado y enseñado a ver en los demás a hermanos y a ver en el mundo una sola casa común.

A mi madre que nunca dejó de apoyar.

A mi padre, por la inspiración.

A mis tíos Lilia y José Juan, por la ayuda brindada y la cercanía.

Al Dr. Luis Enrique Sucar Succar, por el siempre atento apoyo.

Al Dr. Abraham Sánchez López, por la formación recibida.

A mi ruidosa conciencia que siempre me advirtió la importancia de terminar este trabajo.

Resumen

Un sistema de anotación automática de imágenes capaz de conocer la ubicación, forma, tamaño y cantidad de los objetos a los que corresponde cada anotación necesita pasar las imágenes que anota por un proceso de segmentación. Segmentar una imagen consiste en dividirla en grupos de píxeles utilizando algún criterio particular de tal forma que la unión de todos estos grupos da como resultado la imagen original. Estos sistemas normalmente trabajan con bases de miles de imágenes por lo que el tiempo invertido en la segmentación se vuelve un factor crucial en su desempeño. Por lo anterior, para la anotación automática de imágenes se requiere un proceso de segmentación muy eficiente aunque no sea muy preciso.

La segmentación automática de imágenes es un problema inherentemente complejo que, a pesar de la gran variedad de soluciones propuestas, hoy en día sigue sin encontrar una respuesta definitiva.

En esta tesis presentamos un algoritmo de segmentación de imágenes basado en *quadtrees* capaz de segmentar una imagen en pocos segundos y producir segmentos con información suficiente para generar anotaciones correctas. Para lograr esto, proponemos una segmentación simplificada orientada más a la homogeneidad de los segmentos producidos que a la exactitud de los mismos. Realizamos esto siguiendo un enfoque de *quadtrees*, dividiendo y uniendo recursivamente regiones de la imagen. Durante el recorrido tomamos en cuenta información sobre el color y los bordes para ir formando los segmentos. Al terminar el recorrido aplicamos un proceso de unión de segmentos que mejora el resultado final.

Se realizaron pruebas de anotación de imágenes utilizando segmentaciones obtenidas con nuestro algoritmo y con otros dos utilizados en el estado del arte: *normalized cuts* y *grid segmentation*. Nuestro algoritmo tardó aproximadamente 2 segundos en segmentar cada imagen y obtuvo resultados de anotación 35% mejores que *normalized cuts* y 51% mejores que rejillas.

Índice General

Capítulo 1

Capítulo 2

Capítulo 3

Capítulo 4

Capítulo 5

Índice de Figuras

Índice de Tablas

Índice de Gráficas

Capítulo 1

Introducción

1.1 Motivación

Hoy en día la visión por computadora ha alcanzado niveles adecuados para proponerse identificar y clasificar el contenido de grandes bases de datos de imágenes. Para conseguir este objetivo, se han propuesto diversas metodologías. Una de ellas consiste en segmentar las imágenes, analizar cada segmento y asignar una anotación para el mismo. El inconveniente que encuentra este enfoque es que el proceso de la segmentación suele consumir mucho tiempo y en muchos casos los segmentos no son adecuados para producir anotaciones certeras. Ante este problema, proponemos un nuevo algoritmo de segmentación capaz de producir de forma rápida los segmentos que se requieren para anotar imágenes con mayor precisión.

Segmentar una imagen consiste en dividirla en grupos de píxeles utilizando algún criterio particular de tal forma que la unión de todos los grupos da como resultado la imagen original. Existen muchos algoritmos para poder segmentar una imagen. Cada uno de ellos se enfoca a un tipo particular de imágenes y sigue sus propios métodos y criterios para lograrlo. Existen algoritmos que utilizan métodos de agrupamiento de información (*clustering*), análisis estadístico global o local del contenido, separación por contornos, separación por texturas o correspondencia de plantillas, entre otros. Además, como veremos en el capítulo siguiente, cada uno de ellos utiliza diferentes estructuras de datos tales como árboles, grafos, redes neuronales, modelos bayesianos o *quadtrees*. En éstos últimos se basa nuestra propuesta. Segmentar en *quadtree* significa dividir la imagen en cuatro regiones y éstas a su vez en otras cuatro, de manera ciclíca, descartando aquéllas que no cumplen con algún criterio específico. Esto hace que grandes porciones de información puedan ser excluidas en cada paso del proceso. De esta forma, obtenemos una segmentación muy veloz. Precisamente, la segmentación en *quadtree* tiene como principal ventaja el poco tiempo que tarda en obtener resultados.

Etiquetar imágenes ha tomado gran importancia en los últimos años debido al potencial que implica poder conocer automáticamente el contenido visual de una imagen. Este potencial abarca desde reconocimiento y conteo de objetos hasta ilustración automática de textos, pasando por búsqueda e indexado semántico en bibliotecas de imágenes. La segmentación de imágenes puede utilizarse para generar etiquetas que describan el contenido de una imagen. En este caso, primero se segmentan las imágenes, después se analiza el contenido de cada segmento y, según el resultado de este análisis, se asignan una o más etiquetas a cada imagen y/o segmento. Una gran desventaja que existe hoy en

día para crear motores de anotación automática basados en segmentación es la gran cantidad de tiempo que se necesita para segmentar una imagen. Algunos de los algoritmos más utilizados tardan hasta un minuto para segmentar una sola imagen. Esto hace muy poco viable segmentar bases de datos de miles de imágenes. Si queremos poder anotar bases de miles de imágenes, es necesario contar con un algoritmo que ocupe muy poco tiempo en la segmentación y dé resultados suficientemente buenos para poder crear anotaciones basándose en ellos.

1.2 Propuesta de Solución

Partiendo de las necesidades aquí expuestas, proponemos un algoritmo rápido, capaz de encontrar segmentos grandes y homogéneos con información suficiente que permita crear anotaciones para la imagen que representan. Nuestro algoritmo está basado principalmente en un recorrido en forma de *quadtree* a través de la imagen que va uniendo y dividiendo cuadrantes basándose en su homogeneídad y en los contornos detectados dentro de ellos. Nuestro algoritmo aumenta la calidad de la segmentación en cada ciclo y podemos ajustar el número de ciclos deseados variando los parámetros de configuración. Después del recorrido de *quadtree* la segmentación obtenida pasa por un proceso que mejora el resultado uniendo segmentos que pertenecen a un mismo objeto y uniendo segmentos pequeños a otros similares de mayor tamaño. Asimismo, nuestro algoritmo permite parametrizar el nivel de homogeneidad deseado en los segmentos encontrado y el tamaño mínimo de los objetos que deseamos encontrar. La Figura 1.1 muestra un ejemplo de los resultados que produce nuestro algoritmo.

Figura 1.1: Ejemplo de resultado de segmentación con el algoritmo propuesto.

Nuestro principal objetivo es disminuir el tiempo para segmentar una imagen en comparación con otro algoritmo utilizado en anotación como *normalized cuts* [P. Duygulu, 2002]. También ocuparemos como referencia el algoritmo *gridsegmentation* por ser uno de los más rápidos que existen ya que aún cuando produce segmentos que no se asemejan a la forma de los objetos en la imagen, también es utilizado en trabajos de anotación. Evaluaremos numéricamente las segmentaciones y anotaciones obtenidas y compraremos estadísticamente los resultados.

1.3 Resultados

Realizamos pruebas con tres conjuntos distintos de imágenes reales y en todos ellos obtuvimos en pocos segundos segmentaciones que permitieron encontrar anotaciones correctas para las imágenes correspondientes. Nuestro análisis nos dice que estos resultados y tiempos son superiores a los obtenidos con *normalized cuts* y con *rejillas*. Estos resultados ubican a nuestro algoritmo como una opción factible para ser utilizado en sistemas de anotación que requieran dar una respuesta rápida. La Figura 1.2 muestra un ejemplo de los resultados de segmentación producidos por nuestro algoritmo y las anotaciones generadas.

Figura 1.2: Ejemplo de resultados de anotación obtenidos con nuestro algoritmo. Izquierda, imagen original. Derecha, imagen segmentada con nuestro algoritmo y las anotaciones obtenidas con ellos.

1.4 Descripción del documento

Iniciamos este trabajo en el capítulo dos, en el que presentamos la definición mas aceptada de segmentación de imágenes, definimos las características más comúnmente utilizadas por los algoritmos de segmentación. Describimos diversos métodos utilizados para segmentar imágenes y explicamos las limitaciones que enfrentan. Finalmente, comentamos algunas aplicaciones de la segmentación.

En el capítulo tres presentamos a los sistemas de anotación automática, qué son, cómo funcionan y cuales son sus aplicaciones. Explicamos los diferentes enfoques de anotación que existen y comentamos diversos métodos utilizados para extraer la información que se utiliza para generar anotaciones. Destacamos el método de segmentación simplificada por ser el algoritmo que inspiró este trabajo.

En el capítulo cuatro, describimos detalladamente nuestro algoritmo y cada una de sus etapas: obtención de gradiente, cálculo de la imagen de restricciones, discretización del

color, segmentación *quadtree* y postprocesamiento. Partimos del algoritmo de segmentación débil mencionado en el capítulo tres. Después, presentamos brevemente el modelo funcional de nuestro algoritmo y continuamos profundizando en todos los conceptos necesarias para entender cada etapa presentando las condiciones, algoritmos internos, y/o reglas presentes en cada una de ellas. A lo largo del capítulo, ilustramos paso a paso la segmentación con nuestro algoritmo de una imagen ejemplo.

En el capítulo cinco, describimos las pruebas realizadas con nuestro segmentador y analizamos detalladamente los resultados obtenidos en segmentación y en anotación. Describimos el ambiente de desarrollo utilizado y los conjuntos de imágenes de pruebas seleccionados. Explicamos por que elegimos cada conjunto de pruebas y cómo fuimos afinando nuestro algoritmo durante la etapa de pruebas. Analizamos con detalle los resultados, buenos y malos, de segmentación de cada conjunto e ilustramos y comentamos las etapas de segmentación de imágenes seleccionadas. Para algunas imágenes seleccionadas, ilustramos y comentamos cada una de sus etapas en el proceso de la segmentación y mostramos cómo varían estos resultados para valores diferentes de los parámetros de entrada. Comparamos nuestros resultados con los de *normalized cuts* y *grid segmentacion* y, finalmente, mostramos gráficas de los resultados obtenidos.

En el capítulo seis se resume la motivación y desarrollo de esta tesis, se presentan las principales conclusiones y se comenta el trabajo futuro que puede hacerse con nuestro algoritmo.

Finalmente, después del capítulo seis se incluyen una serie de anexos que incluyen mayor profundización en el tema de quadtrees, un manual para utilizar el software desarrollado y algunos listados de la ejecución del programa que permiten analizar con detalle cómo se va construyendo la segmentación.

Capítulo 2

Segmentación de imágenes

La segmentación de imágenes consiste en dividir una imagen en regiones que cumplen un criterio de homogeneidad. Las principales características que podemos utilizar para segmentar una imagen son el color, la textura y los contornos. Existe una gran cantidad de propuestas para segmentar una imagen, cada una con un enfoque distinto, como crecimiento de regiones, detección de contornos, teoría de grafos, *quadtrees* y plantillas, entre otros.

Debido a la gran variedad de imágenes que existen, ningún algoritmo de segmentación es bueno para segmentar todo tipo de imágenes. Algunos inconvenientes que hacen difícil segmentar una imagen son los cambios en la iluminación, el espacio de colores utilizado y texturas complejas.

Entre las aplicaciones de la segmentación de imágenes podemos encontrar conteo automático de objetos, detección de movimiento, edición de imágenes, análisis de imágenes médicas, búsqueda de imágenes y anotación de imágenes. En este capítulo presentaremos en detalle cada uno de estos temas.

2.1 Segmentación

Segmentación es la operación de particionar una imagen en una colección de grupos de píxeles conectados.

La definición formal más aceptada es la de Pavlidis[Url5; Url6; T. Pavlidis, 1972]:

1. $\cup S_i = S$
2. $S_i \cap S_j = \phi, \; i \neq j$
3. $\forall S_i, P(S_i) =$ verdadero
4. $P(S_i \cup S_j) =$ falso, $\; i \neq j$, S_i adyacente a S_j

Donde S_i es un segmento dentro de la imagen S y P() es un predicado que mide la homogeneidad en cada segmento.

El punto 1 de la lista nos dice que la unión de cada uno de los segmentos debe dar como resultado la imagen misma. El punto 2 señala que ningún segmento debe encimarse con otro. El punto 3 indica que todos los segmentos deben cumplir el predicado de homogeneidad y finalmente, el punto 4 nos dice que dos segmentos distintos adyacentes no deben cumplir el predicado de homogeneidad.

La Figura 2.1muestra un ejemplo de segmentación.

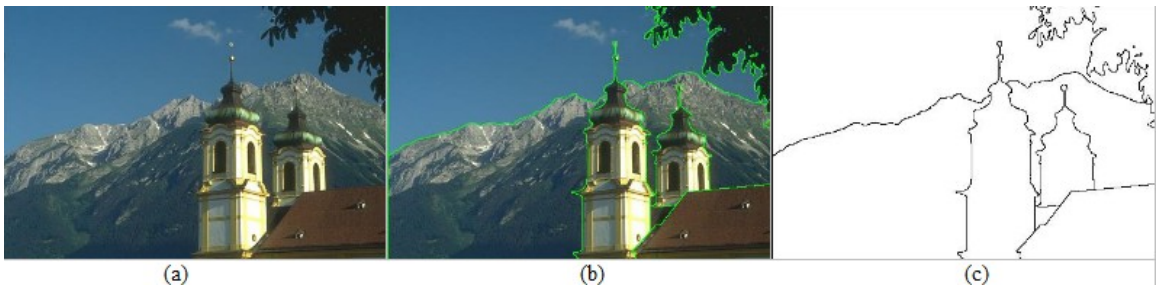


Figura 2.1: Ejemplo de segmentación de una imagen. (a) Imagen original (b) Segmentación sobrepuesta en la imagen original (c) Segmentación de la imagen.

Antes de describir algoritmos de segmentación específicos es importante conocer las principales características que éstos algoritmos analizan en las imágenes.

2.2 Características utilizadas en la segmentación

2.2.1 Color

El color de un píxel en una imagen es descrito por el valor de cada uno de sus componentes de color: rojo, verde y azul. El color que se proyecta en el monitor en cada píxel esta determinado por la mezcla de los valores de estos tres componentes, esto se

conoce como el modelo RGB (*Red, Green, Blue*) de representación de color. Hoy en día, es común utilizar un byte para cada componente, así tenemos que un píxel es representado por tres bytes. Algunos sistemas agregan un cuarto byte para almacenar el valor de transparencia del píxel. Este cuarto byte normalmente no es utilizado por los algoritmos de segmentación.

A pesar de que normalmente una computadora utiliza el modelo de colores RGB, se han propuesto otros modelos que tienen una mejor capacidad para medir la similitud entre los colores de cada píxel, por ejemplo el modelo HSV (*Hue, Saturation, Value*). Existen modelos que poseen mayor resistencia a la diferencia de iluminación como el modelo CIE (Comisión Internacional de Iluminación), y algunos que describen con mayor similitud la forma en que percibimos los humanos como el modelo *Opponent Color* [Dagan Feng, 2003].

Existen diversas características que podemos deducir de una imagen utilizando solamente la información del color. Una de las herramientas más utilizadas para medir el color en una imagen es el *histograma* el cual mide la cantidad de veces que aparece cada color en una imagen. Diversas mejoras se le han hecho al histograma para mejorar su efectividad en la descripción del contenido de una imagen. Tal es el caso del histograma CCV (vector de coherencia), el correlograma CCV, los descriptores regionales de colores dominantes, momentos de color, histogramas de uniones, y otros [Fuhui Long; M. J. Swain, 1991; Greg Pass, 1996, Jing Huang, 1997; Greg Pass, 1999].

2.2.2 Textura

La textura es otra caracteristica que puede ayudar a segmentar imágenes en regiones de interés y clasificar dichas regiones. La textura de una imagen es la información sobre el orden espacial de sus colores o intensidades [Linda Shapiro, 2003]. En una imagen es posible medir ciertas propiedades que sirven para describir la textura presente en ella tales como la granularidad, dirección, repetitividad, aleatoriedad y linealidad, entre otras. Para obtener esta información se aplican operadores matriciales a la matriz de píxeles de la imagen y luego se opera con esos resultados.

Diversos modelos matemáticos se han propuesto para describir texturas. Algunos ejemplos son: características de *Tamura*, características de *Wolf* , filtros de *Gabor* y transformadas *Wavelet* [Fuhui Long; Ani1 K. Jain, 1990]. La textura de la imagen es muy utilizada para la segmentación.

2.2.3 Contornos

En las imágenes muchas veces podemos encontrar puntos de alto contraste. Normalmente, los puntos de mayor contraste corresponden a los límites entre objetos distintos o partes de la escena, es decir conforman los contornos de las regiones de la imagen. Estos puntos se pueden encontrar calculando las diferencias locales de intensidad entre cada píxel y sus píxeles cercanos.

Muchos algoritmos utilizan información sobre los contornos existentes en la imagen que segmentan. Existen diversos métodos para localizar y extraer los contornos presentes en una imagen tales como el operador de *gradiente, Roberts, Canny o Sobel* [Linda Shapiro, 2003]. Éstos aplican operadores matriciales a la imagen y dan como resultado una imagen en escala de grises que contiene información sobre los cambios de intensidad de color. Esta información da lugar a la identificación de los bordes en aquellas regiones donde la variación de la intensidad es elevada.

Cabe mencionar que la localización de los contornos por esos operadores contiene ruido. Este ruido hace a los bordes más gruesos en algunas partes y discontinuos en otras, por lo que generalmente es necesario un post-procesamiento para mejorar los contornos encontrados. Diversos métodos se han propuesto para la eliminación del ruido en las imágenes de contornos y quedar sólo con contornos bien definidos y completos.

El color, la textura y los contornos son las características más utilizadas por los algoritmos de segmentación. Algunos algoritmos de segmentación combinan estas características para realizar la segmentación. También es común que se derive más información a partir de la mezcla de estas características y luego se utilicen todos los datos recopilados para realizar la segmentación. Una característica importante de un algoritmo de segmentación es la forma en que representa y procesa esta información. Las estructuras más utilizadas son: vectores, árboles, cadenas y matrices.

2.3 Métodos de Segmentación

Según las características que utilicen para segmentar, se suele agrupar a los algoritmos de segmentación en diferentes categorias como segmentación basada en píxeles, basada en regiones, basada en contornos o basada en grafos. A continuación trataremos con detalle algunos algoritmos clásicos y otros más modernos para mostrar un panorama amplio de los diferentes caminos que podemos seguir para segmentar una imagen

2.3.1 Métodos de Crecimiento de Regiones

Los algoritmos de crecimiento de regiones se caracterizan por iniciar con uno o varios píxeles *semilla* y luego comenzar a crecer estos píxeles agregándoles píxeles vecinos si se cumplen ciertas condiciones. Poco a poco las regiones van creciendo hasta formar segmentos completos dentro de la imagen

El algoritmo más simple de crecimiento de regiones es el que propuso Haralick, [Linda Shapiro, 2003, pag 315]. Este método toma un píxel en la imagen, revisa sus píxeles vecinos y, si cumplen con una condición de homogeneidad predefinida, los une para formar una sola región. Se repite este proceso para seguir añadiendo píxeles a la región.

El algoritmo calcula la media y la varianza de la región ya formada y evalúa la diferencia de intensidad de color con respecto a sus vecinos. Si este valor es menor de un umbral definido procede a unir los píxeles a esta región y recalcula los valores de la media y la

varianza antes revisar la similitud con los siguientes vecinos. Cuando ninguno de los vecinos de la región cumplan la condición de homogeneidad, se inicia una nueva región y se repite el proceso de crecimiento.

La Figura 2.2 ilustra el funcionamiento de este algoritmo. Los píxeles en azul indican una región que ya fue formada dentro de la imagen Los píxeles en verde son los *4-vecinos* de la región. Los píxeles en blanco son los píxeles de la imagen que no han sido analizados.

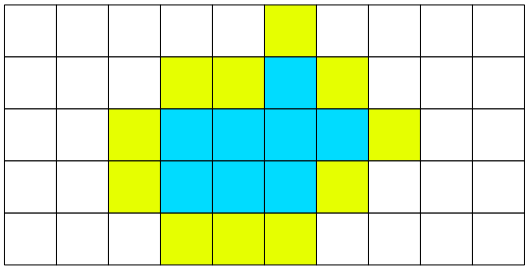


Figura 2.2: Azul.- Región ya formada. Verde.- Píxeles vecinos de la región que serán analizados

Los segmentos arrojados por este algoritmo son muy irregulares y no corresponden en gran parte a las diferentes regiones que incluye la imagen. La Figura 2.3 muestra un ejemplo de los resultados de este algoritmo. Podemos observar que algunos elementos visualmente grandes no son encontrados, como el trozo de tabla que atraviesa la parte superior de la imagen. Además este algoritmo tiende a la sobre segmentación, la cual consiste en devolver un gran número de segmentos pequeños que bien podrían ser unidos en otros más grandes.

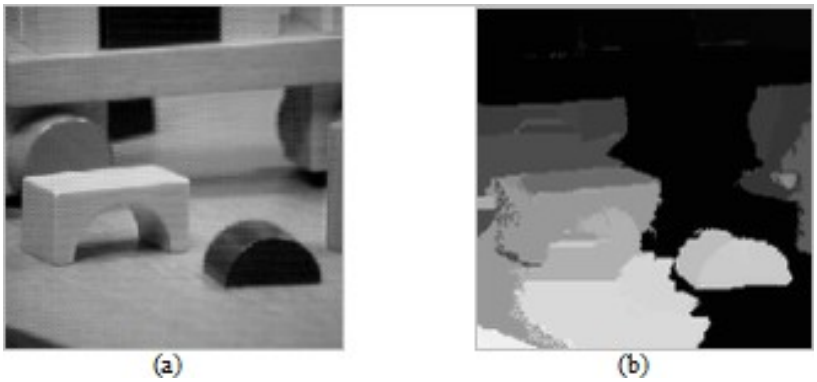


Figura 2.3: Segmentación utilizando el algoritmo de crecimiento de regiones. (a) Imagen original. (b) Imagen segmentada. Imagen tomada de [Linda Shapiro, 2003, pag 319].

El método de ***k-means*** [J. B. MacQueen, 1967; Anil K. Jain, 1988; Linda Shapiro, 2003; Url1; Url7] es muy parecido con la ventaja de que nos permite especificar el número de regiones *k* en que queremos dividir la imagen, evitando así la sobre segmentación. K-means toma *k* píxeles aleatorios dentro de la imagen, en principio estos píxeles son considerados regiones que contienen un solo píxel. Se calcula la media de cada una de estas regiones (al principio la media será igual al valor del único píxel que hay en la

región). A continuación, se toma un píxel cualquiera de la imagen y se agrega a la región cuya media sea más próxima al valor de este píxel. Conforme se van agregando píxeles a cada una de las regiones, la media es recalculada para esa región. Al terminar de revisar todos los píxeles se tiene una media más descriptiva de cada una de las regiones, así que se vuelve a revisar cada píxel de la imagen y se asigna a la nueva media más cercana. El proceso se repite hasta que ningún píxel cambie de región o hasta que se cumpla un cambio en las medias de las regiones que sea menor a un umbral previamente determinado. El valor que se da a cada uno de los píxeles depende de la implementación en particular que se hace de este algoritmo. Lo más común es tomar la intensidad del píxel. Se muestra un ejemplo de esta segmentación en la Figura 2.4.

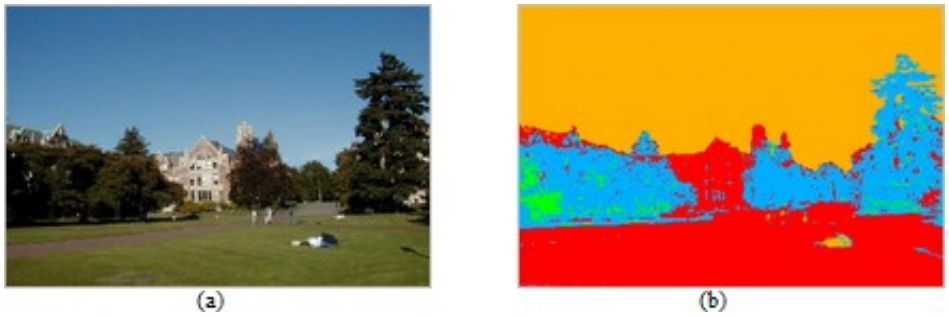


Figura 2.4: Segmentación utilizando el algoritmo k-means. (a) Imagen original. (b) Imagen segmentada.

El algoritmo ***Isodata Clustering*** [G. H. Ball, 1964; Linda Shapiro, 2003; Anil K. Jain, 1988; Url2] es una mejora a *k*-means que consiste en dividir la imagen sin tener que especificar el número de regiones que deseamos. Sigue un procedimiento similar a *k-means* dividiendo segmentos cuya varianza sea demasiado grande y agrupando las regiones que sean demasiado pequeñas. Esto lo hace de manera recursiva hasta que ya no pueda dividir ni agrupar ninguna región.

2.3.2 Métodos de División y Unión (Split and Merge)

Estos algoritmos siguen una filosofía opuesta a crecimiento de regiones. En lugar de comenzar con una semilla y hacerla crecer, comienzan con una región no homogénea muy grande dentro de la imagen y la dividen de manera recursiva hasta lograr segmentos homogéneos Después de esta etapa de división (*split*), ejecutan una etapa de unión (*merge*) para unir los segmentos encontrados y así refinar sus resultados.

Un método de segmentación de este tipo es el método de ***quadtrees*** [J. Shi, 2000] el cual consiste en dividir la imagen en cuatro cuadrantes de igual tamaño de manera recursiva hasta quedarse únicamente con cuadrantes homogéneos y unir posteriormente aquellos cuadrantes vecinos que resulten similares. Esta técnica la describiremos con mayor detalle en el Capítulos 4 y en e Anexo A. Una variación al método de *quadtrees* consiste en dividir la imagen en triángulos en lugar de cuadrantes.

Estos métodos normalmente terminan con algunos segmentos muy pequeños en sus resultados. Por esto, después de la etapa de división se ejecuta una etapa de unión en la cuál estos segmentos pequeños son añadidos a regiones grandes cercanas e incluso dos regiones grandes pueden ser combinadas en una sola si cumplen determinadas condiciones.

2.3.3 Métodos Basados en Detección de Contornos

Los algoritmos basados en detección de contornos suponen que todo objeto tiene un borde y por lo tanto, si podemos hallar los bordes, habremos hallado el objeto. Concentran toda su atención en la correcta detección de los bordes de los objetos. Normalmente inician detectando contornos con operadores clásicos como S***obel*** o C***anny*** y posteriormente refinan los bordes encontrados.

El algoritmo en [Baris Sumengen, 2004] basa su segmentación en los contornos que puede encontrar en una imagen. Para una mejor localización de los contornos, analiza la imagen en diferentes escalas. Así, determina cuales son los contornos más significativos -que aparecen en todas las escalas- y también encuentra contornos muy detallados -cuando utiliza escalas muy grandes-.

Un ejemplo más elaborado de este tipo de algoritmos lo tenemos en [W.Y. Ma, 1997]. El algoritmo primero detecta los contornos de la imagen, después les aplica un proceso de refinamientos y seguimiento y da como resultado las regiones que quedan dentro de los contornos mejorados.

Hay métodos que una vez que han refinado los contornos de una imagen examinan las áreas dentro de dichos contornos buscando homogeneidad en su interior. Si encuentran que esta región es homogénea deducen que han descubierto un objeto en ese lugar, en caso contrario tratan buscar a qué objeto "cercano" pertenece.

Otros métodos como [Thomas Leung, 1998] también consideran los contornos subjetivos dentro de la imagen, completando todos aquellos contornos que aunque no se ven, son formados por la mente humana como lo muestra la Figura 2.5.

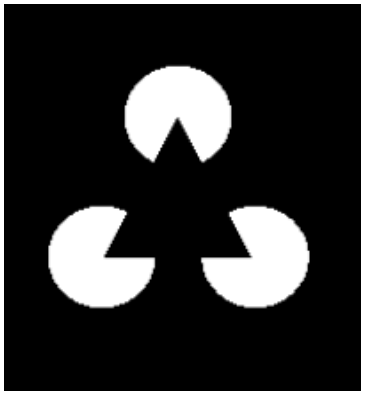


Figura 2.5: Ejemplo de contornos subjetivos en una imagen.La mente humana percibe un triángulo al centro de la imagen aún cuando sus lados no son continuos. Imagen tomada de [Thomas Leung, 1998].

Algunos otros, convierten los contornos encontrados en líneas geométricas llamadas víboras (*snakes*) [M. Kass, 1987]. Los *snakes* son figuras sobrepuestas en una imagen descritas por puntos que utilizan la información alrededor de ellas para amoldarse y adaptarse a la forma original del objeto que contienen. Un ejemplo de algoritmos que utilizan s*nakes* en su proceso de segmentación lo podemos encontrar en [Tony F. Chan, 2001].

Los *snakes* inician con un contorno aproximado de los objetos y luego se van moldeando hasta que este contorno se ajuste lo más posible a la forma real del objeto. Esto lo hacen analizando la textura de la región a ambos lados del contorno y haciéndolo dirigirse hacia donde la región interior sea más homogénea. El método de *snakes* es ampliamente utilizado para buscar malformaciones en imágenes médicas. Un ejemplo del procesamiento de un algoritmo de *snakes* se observa en la Figura 2.6.

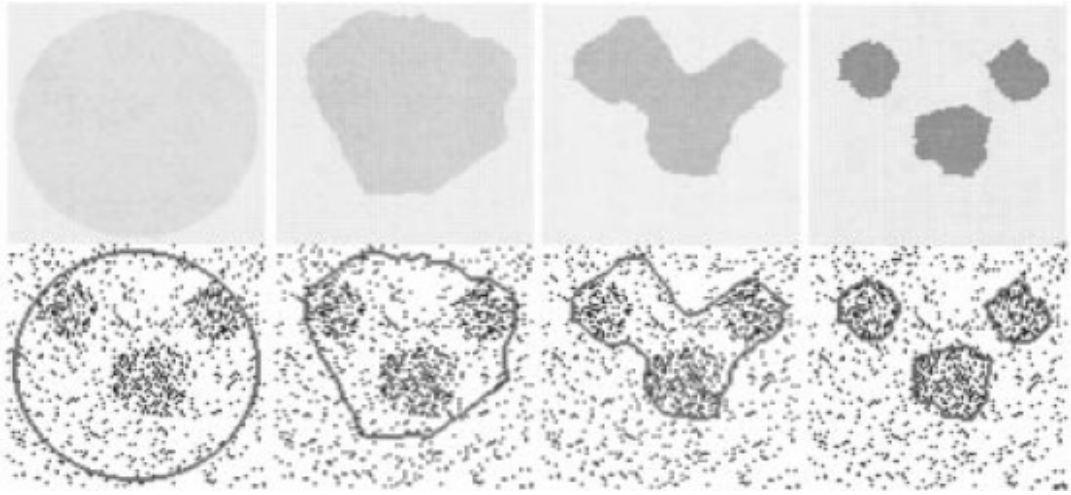


Figura 2.6: Ejemplo de segmentación utilizando el método de snakes. Arriba, segmentación producida por el snake en diferentes etapas. Abajo: Proceso de adaptación del snake en las diferentes etapas de la segmentación. Imangen tomada de [Tony F. Chan, 2001].

2.3.4 Segmentación Basada en Teoría de Grafos

Estos algoritmos segmentan imágenes de manera similar a como se segmenta un grafo. Forman un grafo a partir de una imagen de la siguiente manera. Cada píxel se toma como un nodo del grafo. Se crean arcos que enlacen a todos los nodos entre sí. Los valores para los arcos entre cada par de nodos son calculados por una función que mide el grado de similitud entre dichos nodos. Se han proupesto muchas variantes de esta función [J. Shi, 2000; Serge Belongie, 1998; Thomas Leung, 1998]. Después de este proceso, terminamos con un grafo con pesos no dirigido que representa toda la imagen. Una vez que tenemos este grafo, sigue buscar en qué nodos podemos eliminar cuáles arcos de tal manera que nos queden dos grafos en lugar de uno. Cada uno de estos grafos debe cumplir la propiedad de ser homogéneo y distinto al otro.

El mejor corte a realizar es aquel en el que la similitud de los nodos pertenecientes al mismo sub-grafo sea muy alta y la similitud entre nodos pertenecientes a distinto sub-grafo sea muy baja. Existen diversas propuestas para lograr el mejor corte antes mencionado. Una de las más usadas es llamada Corte Normalizado o *normalized cuts* [J. Shi, 2000]. Consiste en realizar un *corte normalizado del grafo* para el cual se toma en cuenta el número de arcos que se eliminarán y la fuerza con que estos arcos están ligados a los demás nodos de la región. En la actualidad, es un método ampliamente utilizado por las comunidades que realizan recuperación de imágenes y anotaciones. Este método de segmentación se ilustra en la Figura 2.7. Otros métodos de segmentación basada en grafos se pueden encontrar en [Zhen Wu, 1993].

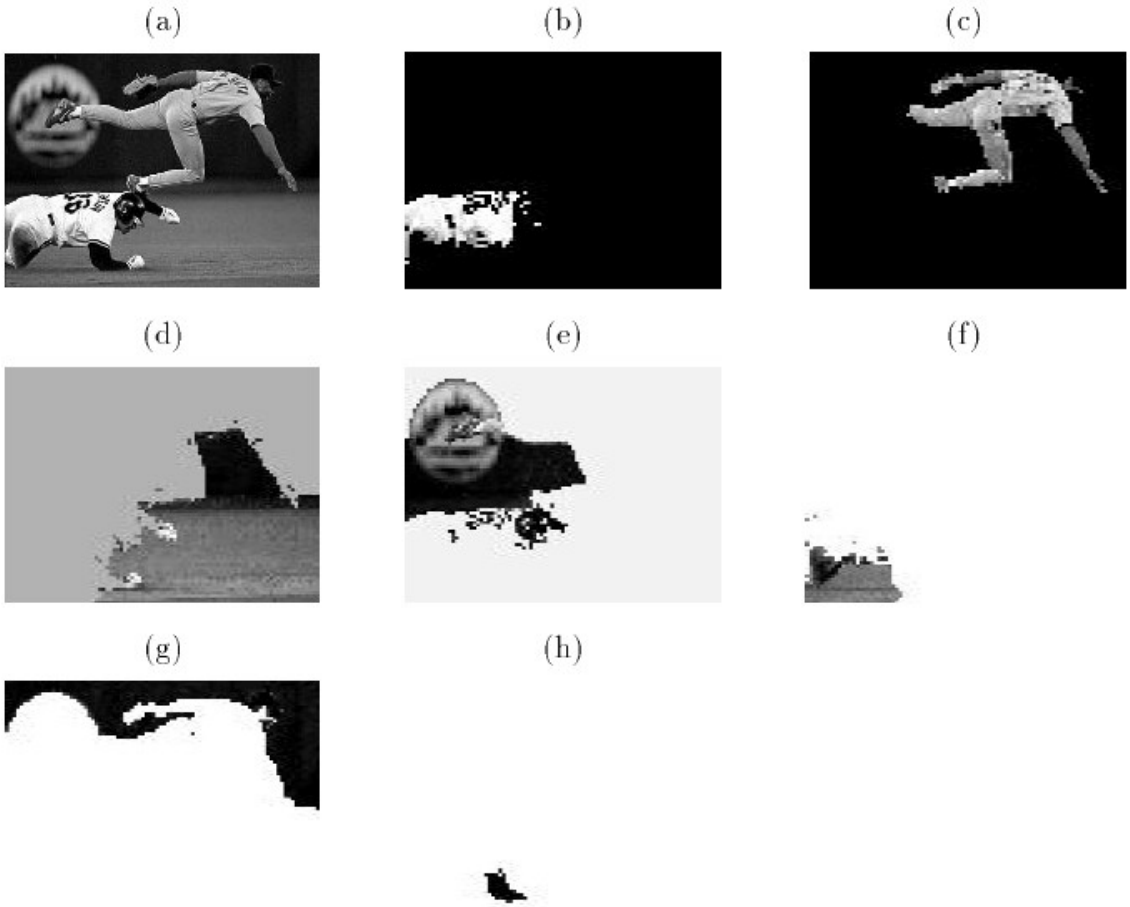


Figura 2.7: Resultados de segmentación de normalized cuts. (a) imagen original, (b – h) regiones encontradas en la imagen. Imagen tomada de [J. Shi, 2000].

2.3.5 Segmentación de Arriba a Abajo (Top-Down)

La segmentación de arriba a abajo se basa en información adquirida de clases específicas. Esta información se recopila en plantillas que después son buscadas dentro de la imagen para saber si la imagen contiene un objeto de cierto tipo. Borestein [Eran Borestein, 2004; Eran Borestein, 2002; Eran Borestein, 2003] y otros [A. Yuille, 1992, Anat Levin, 2006]

han estudiado desde hace unos años la relevancia que tiene la forma en la psicología humana para poder reconocer un objeto y han creado modelos de segmentación basados en esto. El humano desarrolla la capacidad de reconocer objetos con el paso del tiempo a través de un aprendizaje constante. Ullman y sus colaboradores piensan que también una computadora puede *aprender* a distinguir objetos si es *entrenada*. Si la computadora es entrenada lo suficientemente bien, será capaz de distinguir objetos aún cuando presente variaciones significativas, al igual que los humanos. A esto, Ullman le llama ***Class Specific Segmentation*** (Segmentación de Clases Específicas).

Existen dos enfoques principales en segmentación *top-down*. El primero de ellos trabaja con plantillas elaboradas a mano, el segundo incluye dentro de su proceso una forma de encontrar estas plantillas automáticamente. Explicaremos brevemente un enfoque del segundo tipo publicado en [Eran Borestein, 2003].

Primero recolectamos diversas y variadas imágenes de un mismo tipo de objeto en diferentes escenarios y circunstancias, por ejemplo caballos. Damos este conjunto de imágenes a la computadora y le indicamos que todos los objetos mayores, presentes en las imágenes son caballos. La computadora procede a identificar cuales son los rasgos más significativos de la clase de objeto introducido (caballo), busca las partes más representativas y las almacena en pequeños fragmentos. En el ejemplo de los caballos, estos fragmentos pueden pertenecer a la cola, orejas, boca, pezuñas o cabeza, entre otros.

Una vez que ha encontrado los fragmentos más significativos, podemos darle una nueva imagen –*de cualquier tipo de caballos*- para que la *segmente*. La computadora buscará correspondencias entre los fragmentos almacenados para cada una de las clases que conoce dentro de la imagen. Si encuentra suficientes correspondencias de los fragmentos de un determinado objeto dentro de una imagen, entonces lo *rodea* basándose en la información de los fragmentos y lo *separa* –segmenta- del resto de la imagen.

Debido a que esta segmentación posee información previa sobre la forma de los objetos, es capaz de dar una segmentación más aproximada a la realidad y con bordes más finos. La Figura 2.8 ilustra esta segmentación.

Figura 2.8: Segmentación basada en plantillas. (a) imagen a segmentar, (b) fragmentos almacenados en la base de conocimiento sobre la clase caballo, (c) búsqueda y correspondencia de los fragmentos en la imagen original, (d) imagen segmentada, (e) contorno final. Imagen tomada de [Eran Borestein, 2004].

Una mejora a este tipo de algoritmos consiste en primero realizar una segmentación basada en plantillas y después refinarla basándose en características de bajo nivel de la

imagen como el color o la textura recurriendo a técnicas de segmentación tradicional [Anat Levin, 2006].

2.3.6 Métodos híbridos

Los métodos híbridos son aquellos que mezclan en su proceso más de un algoritmo de los mencionados anteriormente; ya sea que uno se ejecute después del otro o que lo hagan en sincronía. También se puede dar el caso de que uno ayude o supervise el funcionamiento del otro. Los métodos híbridos pueden mezclar más de un algoritmo de cada clasificación.

2.4 Mejoras a los algoritmos de segmentación

Se han desarrollado muchos algoritmos de segmentación diferentes. Cada uno con su propio enfoque. Sin embargo, muchas veces podemos mejorar los resultados de un algoritmo de segmentación refinando el criterio de homogéneidad o uniendo lo mejor de los dos o más algoritmos en un sólo.

2.4.1 El criterio de homogeneidad

El criterio de homogeneidad juega un papel determinante en todos los métodos de segmentación ya que la solución al problema de segmentar una imagen –agrupar los píxeles de una imagen en regiones– dependerá de que criterio utilicemos para decidir cuando un píxel o región es *similar* a otro [Jaume Vergés].

Una forma de mejorar los resultados de los algoritmos de segmentación consiste no tanto en proponer un *nuevo algoritmo* sino en *modificar sus criterios de homogeneidad*. Hoy en día tenemos muchas variantes de cada método. Algunos añaden a su criterio de homogeneidad relaciones espaciales, contornos subjetivos, cambios de iluminación, cambio de escalas, texturas, histogramas extendidos, o puntos sobresalientes, entre otros.

2.4.2 Algoritmos híbridos

Después de mejorar el criterio de homogeneidad, la manera más común de mejorar los resultados de un método propuesto es mezclar algoritmos.

Como ejemplo de este tipo de métodos está Competición de Regiones o *Region Competition* [Song Chun Zhu, 1996] que combina *Snakes*, Redes Bayesianas y Crecimiento de Regiones (Figura 2.9). Otro ejemplo lo podemos encontrar en [Sophie Liu Xiao Fan, 2004], primero procesa la imagen con una mezcla de los operadores de gradiente *Sobel* y *Canny* para detectar contornos y luego hace crecimiento de regiones restringida por esos contornos. Estos dos ejemplos, son solo una pequeña muestra de la gran variedad de combinaciones que se pueden realizar.

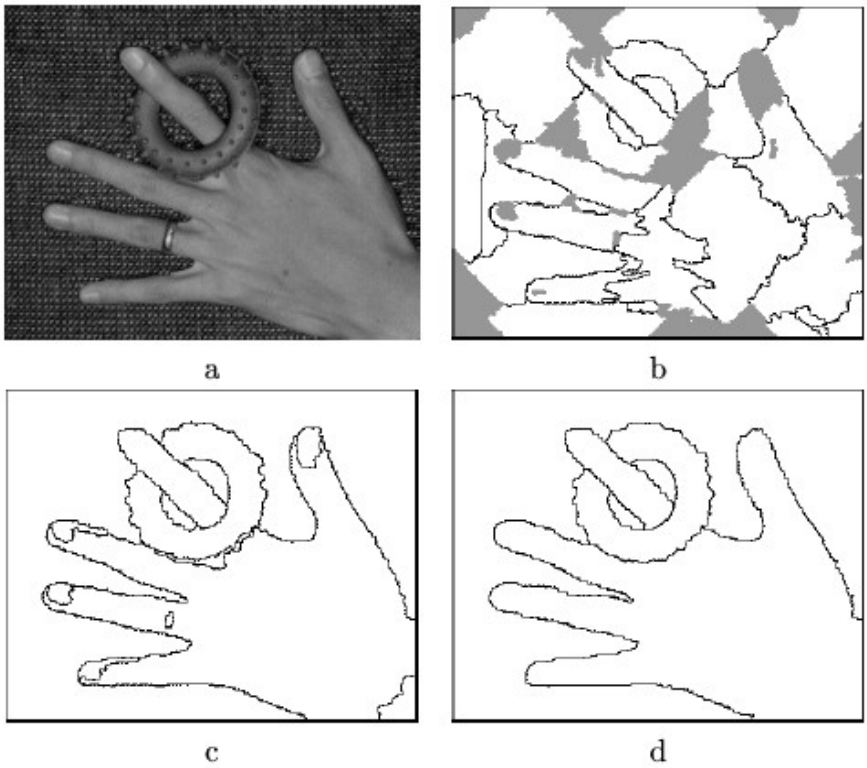


Figura 2.9: Ejemplo de segmentación utilizando el método de competición de regiones. a) imagen original, b) y c) etapas intermedias en el crecimiento de regiones, d) segmentación final.Imagen tomada de [Song Chun Zhu, 1996].

2.5 Limitaciones de los algoritmos de segmentación

A continuación comentamos algunos de los problemas más comunes a los que se enfrentan los algoritmos de segmentación.

2.5.1 Contornos

Ciertos objetos como una casa de madera, un camino empedrado o un ramo de flores, suelen presentar infinidad de contornos (cambios de intensidad) y, a pesar de ello, ser considerados por el humano como un solo objeto. Debido a esto, una vez que se han hallado los contornos dentro de una imagen, es necesario revisar si distintos contornos pertenecen al *mismo objeto* en lugar de indicar la presencia de un nuevo objeto. Esto lo hacemos analizando 1) la intensidad del contorno (mientras menor sea la intensidad más probabilidades de que no pertenezca a otro objeto) y 2) la textura presente en toda la región. Puede ser que, a pesar de presentar muchos contornos, como en el caso de un conjunto de piedras, la región siga siendo homogénea y entonces dichos contornos deberán no tomarse en cuenta. El problema de determinar si un contorno indica los límites de un objeto no es trivial de resolver. La Figura 2.10 ilustra este problema.

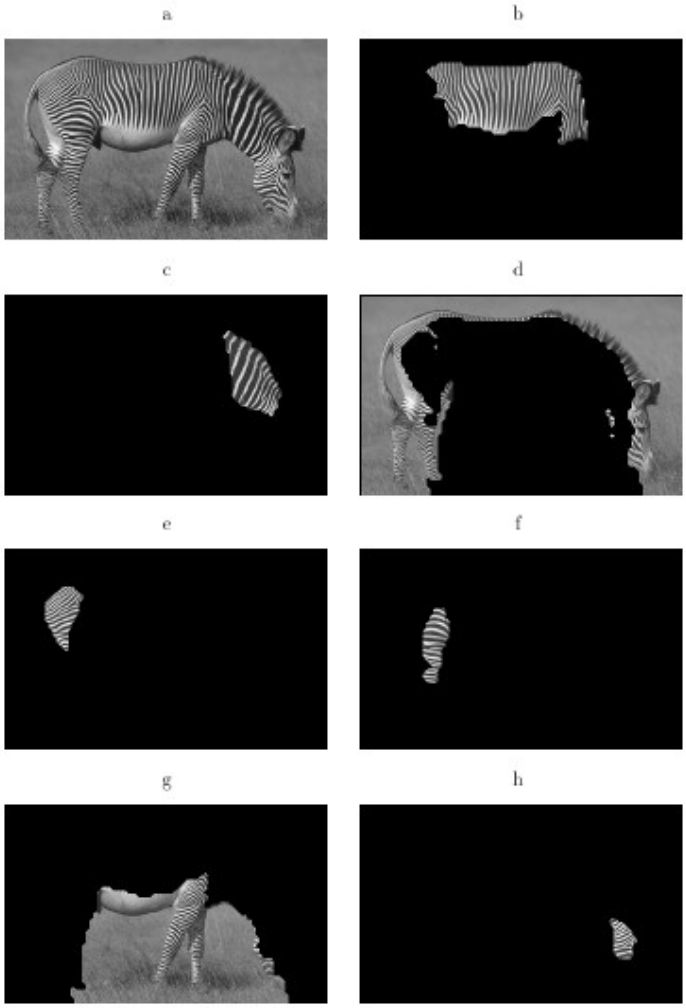


Figura 2.10: Dificultades para segmentar una imagen que presenta demasiados contornos. La cebra de la imagen no es reconocida como un solo objeto. Algoritmo utilizado: normalized cuts. Imagen tomada de [J. Shi, 2000].

2.5.2 Textura

Es difícil reconocer objetos que están cerca o sobre otros objetos con textura similar. Incluso los humanos tenemos problema con esto. Por ejemplo, en la imagen de un bosque, es imposible distinguir los límites de cada árbol. En este caso una segmentación de objetos es imposible. Solo podemos aproximarnos a una segmentación por regiones. Este problema tampoco es trivial.

2.5.3 Iluminación y Modelos de Color.

Es difícil saber que dos regiones pertenecen al mismo objeto cuando no todo el objeto esta iluminado con la misma intensidad. Por ejemplo una casa puede estar iluminada por el sol desde el oriente pero estar completamente oscura y con sombras en el poniente.

En algunos modelos de color, una pequeña variación en la intensidad se convierte en una diferencia de color muy grande. En respuesta a este problema, existen modelos de color que toman en cuenta la iluminación al momento de idear la posición de cada color en su espacio de colores, haciendo que los píxeles de diferentes intensidades permanezcan cerca. De cualquier manera, trabajar en un espacio de colores distinto al RGB siempre le agregará una significativa carga de trabajo al algoritmo de segmentación debido al tiempo requerido para realizar la conversión de todos los píxeles de la imagen.

También se da el caso contrario. Imágenes con colores distintos pueden tener distancias muy pequeñas en el espacio de colores RGB. Esta da como resultado una alta similitud al evaluar el criterio de homogeneidad aún cuando las regiones son diferentes.

En ambos casos, este problema ocasiona que la segmentación sea imprecisa y los segmentos generados rebasen al objeto que contienen o no lo abarquen completamente.

2.5.4 Percepción Humana

El problema de la segmentación perfecta se deriva tanto de las limitaciones técnicas para medir y discernir los diferentes elementos de un grupo como de la interpretación que cada humano puede hacer de una misma imagen. Estudios psicológicos de la visión humana nos muestran que la forma en que un humano aprende a segmentar una imagen es un proceso largo, complejo y hasta ahora no es completamente comprendido. La Figura 2.11 muestra un ejemplo de diferentes propuestas de segmentación manual realizadas en la Universidad de Berkeley [Url3]. Ninguno de los resultados satisface a todos los gustos.

2.6 Aplicaciones

La segmentación de imágenes se utiliza para múltiples fines en el área de visión por computadora. Un uso común es el seguimiento y detección de objetos en secuencias de video. Otros usos son reconocimiento, conteo, clasificación e identificación de objetos dentro de una imagen. La segmentación es una tarea clave en el proceso de emular la vista humana. El objetivo final de la segmentación es lograr que la computadora pueda reconocer lo que esta viendo. Un uso común es el conteo y análisis de objetos en una línea de producción. Se utiliza en las grandes fabricas que cuentan con procesos de producción y calidad automatizados. Cámaras observan constantemente la línea de producción, contando los objetos y buscando imperfecciones en la forma de los mismos.

Otro uso importante es el seguimiento de objetos. En un sistema de control de tráfico automatizado, existen cámaras que deben analizar la velocidad de los autos con miras a detectar aquellos que se están moviendo más rápido de lo permitido. También es utilizado para detectar intrusos en sistemas de seguridad o para seguimiento de objetos voladores.
Un uso muy frecuente de la segmentación de imágenes es la detección de tumores en análisis médicos.

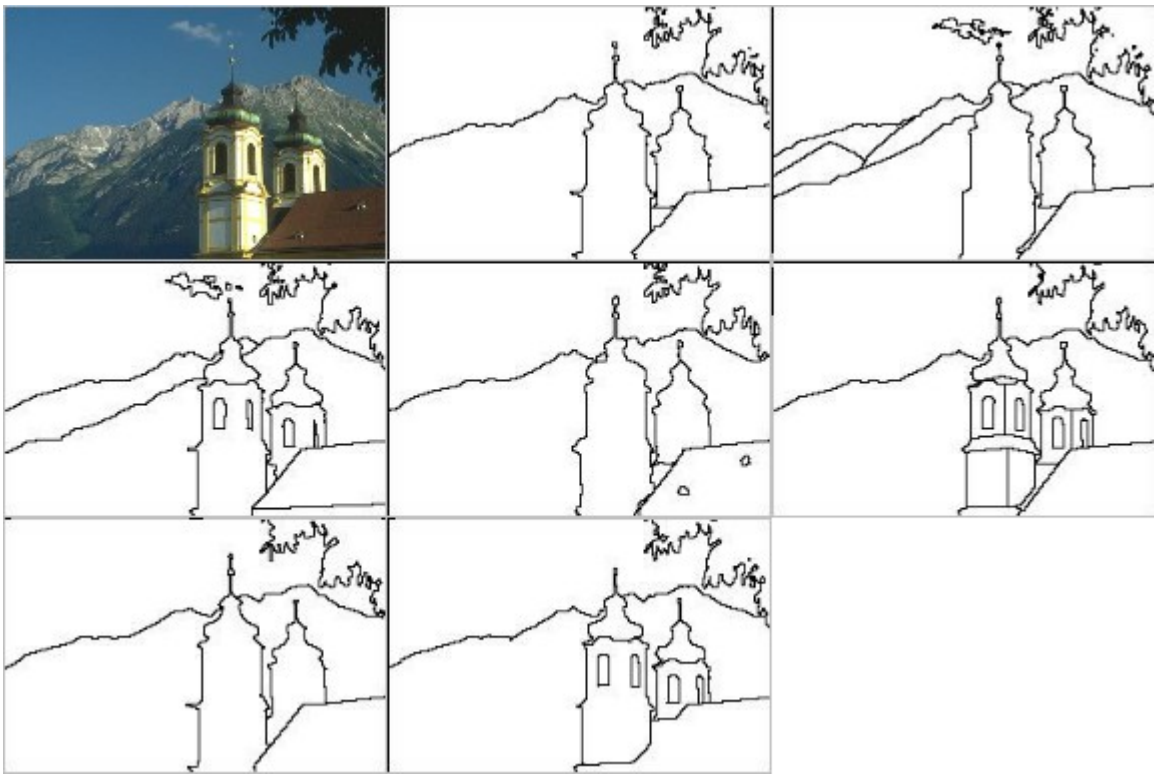


Figura 2.11: Distintos resultados de segmentación para una misma imagen. ¿Cuál es el mejor?

La segmentación también se utiliza en comparación de imágenes. Si queremos comparar una imagen con otra, sin necesidad de tener los mismos objetos en las mismas posiciones, podemos primero segmentar cada una de las imágenes y luego comparar uno a uno sus segmentos en busca de similitudes entre ellos y concluir con una posible similitud en los contenidos. Tal es el caso de los sistemas de recuperación de imágenes basados en contenido o CBIR por sus siglas en inglés *Content Based Image Retrieval* de los que hablaremos en el capítulo siguiente.

Otra aplicación importante es la anotación de imágenes. Para poder agregar etiquetas textuales a una imagen que describan su contenido, primer segmentamos la imagen, analizamos el contenido de cada segmento y luego asignamos una etiqueta adecuada utilizando conocimiento previamente adquirido de otros segmentos. Esta aplicación de la segmentación es la que nos interesa principalmente en esta tesis. La abordaremos con mayor detalle en el siguiente capítulo.

2.7 Conclusión

Tenemos muchos caminos para segmentar una imagen. El método que escogamos depende de la aplicación que le demos a los resultados ya que cada algoritmo va enfocado a un tipo de imágenes particular. A pesar de todas las opciones que tenemos para segmentar debemos estar conscientes de que no podemos lograr una segmentación

perfecta ni segmentar de la misma manera todo tipo de imágenes. Lo más importante de un algoritmo de segmentación son los criterios de homogeneidad que utiliza para comparar las regiones y los píxeles. En el siguiente capítulo, comentaremos como puede ser utilizada la segmentación como paso previo a la anotación automática de imágenes.

Capítulo 3

Sistemas de Anotación Automática de Imágenes

Las anotaciones son pequeños textos que se asignan a una imagen y describen su contenido. En lugar de anotar las imágenes manualmente, se suelen utilizar programas de anotación automática que parten de bases de imágenes previamente anotadas para construir un modelo que permita asignar anotaciones a imágenes no anotadas.

El punto más importante de un modelo de anotación automática es su manera de relacionar el contenido de una imagen con sus anotaciones (aprendizaje). La anotación se puede dar en diferentes niveles según el nivel de detalle de las anotaciones. Las técnicas de segmentación pueden ayudar a extraer y agrupar la información sobre el contenido de una imagen con miras a relacionarla con sus anotaciones.

Existen técnicas de segmentación orientados específicamente a apoyar la anotación de imágenes aunque no todos los sistemas de anotación automática pasan por una etapa de segmentación. Algunos autores opinan que no es necesario segmentar una imagen para anotarla, sin embargo esto limita el nivel de anotación que podemos lograr (global o local). Mostraremos diversos enfoques del estado del arte de estos sistemas y sus aplicaciones.

3.1 Anotación de imágenes

Las **anotaciones** son pequeños textos ligados a una imagen para describir su contenido. Anotar imágenes es una práctica común en las bibliotecas y museos. Estas anotaciones sirven para crear índices que faciliten su búsqueda y recuperación. Hace muchos años los sistemas de búsqueda de imágenes por computadora imitaban este comportamiento asignando palabras clave de manera manual a las imágenes que formaban su base de datos y la recuperación de imágenes se lograba buscando en estas anotaciones [Dagan Feng, 2003]. Estos sistemas son llamados sistemas ABIR (*Annotation Based Image Retrieval*). El mayor inconveniente de estos sistemas era la enorme cantidad de tiempo que se requería para anotar manualmente bibliotecas de miles de imágenes. Este inconveniente dio lugar a la *anotación automática de imágenes*.

La anotación automática de imágenes se esta desarrollando a ritmos acelerados. Los enfoques para anotar imágenes automáticamente son diversos. Algunos métodos enfocan sus anotaciones a describir el contenido global de la imagen, como edificios, personas, naturaleza, etc., mientras que otros se enfocan al contenido local, como tigre, cielo, pasto, agua, etc.En la Figura 3.1 vemos algunos ejemplos de imágenes con anotaciones.

Figura 3.1: Ejemplos de imágenes anotadas.

3.2 Aplicaciones de la anotación de imágenes

Existen diversas aplicaciones que se derivan de contar con imágenes anotadas, entre ellas:

1. Búsqueda de imágenes utilizando consulta textual.- Contar con imágenes anotadas es muy útil para realizar búsquedas en bibliotecas de imágenes. Por ejemplo,

podemos introducir la palabra *tigre* y obtener como resultado diversas imágenes en las que haya un tigre sin importar en donde se encuentre, ya sea en un circo o en la selva, este de frente o de perfil. Utilizando técnicas de lenguaje natural y procesamiento de texto, podemos extender estas búsquedas a conceptos compuestos como "*tigre en la selva*", [J. Jeon, 2003; Alexei Yavlinsky, 2004; Rong Jin, 2004; Jianping Fan, 2004; Masashi Inoue, 2004]

2. Indexado y navegación.- Al contar con anotaciones para cada imagen de nuestra base de datos, podemos crear índices que permitan navegar fácilmente a través de toda nuestra biblioteca de imágenes[Michela Lecca, 2007; Meng Zhu, 2007].
3. Ilustración automática.- Consiste en realizar una búsqueda que nos sugiera con que imágenes podemos ilustrar un texto dado. Utilizando técnicas de procesamiento de lenguaje podemos extraer las palabras más significativas de un texto y a partir de ellas buscar imágenes que contengan anotaciones alusivas al mismo [Meng Zhu, 2007; Kobus Barnard, 2001].

3.3 Modelos de anotación automática de imágenes

Se han propuesto diferentes métodos de anotación automática de imágenes. Los modelos de anotación automática parten de un conjunto grande de imágenes ya anotadas. Primero, extraen las características más descriptivas de cada imagen Después, agrupan todas las imágenes (o regiones de imágenes) que poseen las mismas características y las mismas anotaciones. Una vez hecho esto, proceden a construir un modelo que analice la relación entre estas características y las anotaciones de cada imagen Este modelo es usado posteriormente para anotar nuevas imágenes basándose en las características de la misma. La Figura 3.2 muestra un ejemplo resumido de este proceso. Partimos de imágenes anotadas, segmentamos algunas imágenes y agrupamos todos los segmentos parecidos. Una vez hecho esto, procedemos a buscar información que relacione los grupos de segmentos con las etiquetas de las imágenes a las que corresponden.

Lo más importante de un modelo de anotación automática de imágenes es como adquiere las características representativas de la imagen, es decir la información que posteriormente relacionará con las anotaciones de la imagen. De esto depende en gran parte la precisión de las anotaciones generadas por cada método ya que en esta información basan su aprendizaje. El otro punto vital de estos algoritmos es la forma en que relacionan esta información con las anotaciones. Una forma de extraer dicha información es utilizando un algoritmo de segmentación de imágenes; aunque no es la única.

A continuación exponemos brevemente algunos métodos utilizados actualmente para anotación automática de imágenes. Debido a que nuestro trabajo consiste en un algoritmo de segmentación orientado a anotaciones vamos a poner especial énfasis en la forma en que cada uno de ellos extrae la información necesaria para construir su modelo.

Figura 3.2: Ejemplo de los pasos que sigue un motor de anotación automática para aprender a relacionar imágenes y etiquetas. Izquierda.- Conjunto de imágenes. Centro.- Imágenes segmentadas. Derecha.- Segmentos agrupados por categorias. Imagen tomada de [J. Jeon, 2003].

3.3.1 Modelos de anotación global

Los modelos de anotación global han sido los métodos más estudiados hasta el momento. Sus técnicas están normalmente basadas en análisis de la estadística global de la imagen. Crean firmas o estructuras vectoriales para cada una de las imágenes del conjunto de entrenamiento y después buscan la relación entre dichas estructuras y las anotaciones de cada imagen No son capaces de indicar la posición espacial ni la cantidad o tamaño de los conceptos que describen con sus anotaciones [James Z. Wang, 2003; J. Li, 2000; S. L. Feng, 2004; Aude Oliva, 2001; Alexei Yavlinsky, 2004; Jiayu Tang, 2006].

3.3.2 Modelos de anotación local

Los modeles de anotación local buscan crear anotaciones con mayor precisión. Debido a esto, es necesario dar un paso más allá del análisis global de la imagen y buscar regiones para las cuales podemos crear una anotación. Esto implica realizar algún tipo de segmentación de la imagen como paso previo a la anotación. Una vez que se ha segmentado la imagen procedemos a analizar las características que hay en cada segmento y a partir de ahí deducimos una anotación para el mismo. Este enfoque permite conocer también el número, el tamaño, la forma y la cantidad de los conceptos que anota [Rong Jin, 2004; Le Thi, 2004].

3.4 Obtención de la información necesaria para anotar una imagen.

Los motores de anotación de imágenes necesitan extraer información de las imágenes tal que pueda relacionarse con sus anotaciones para poder aprender a anotar imágenes que no están anotadas.

Uno de los enfoques consiste en segmentar las imágenes y luego tratar de buscar las relaciones entre cada segmento y sus anotaciones. Con esto se logra conocer la ubicación

específica de la anotación en la imagen, sabemos qué hay y en donde está. También, tenemos una aproximación a la forma y un número de apariciones. Este proceso ideal de las anotaciones aún es difícil. Lamentablemente, los algoritmos de segmentación todavía son incapaces de dar como resultado regiones con significados específicos, son muy frágiles y producen errores. Todo esto ocasiona que los segmentos resultantes no correspondan única y totalmente al objeto que contienen. La falta de exactitud en los algoritmos de segmentación aunada al largo tiempo de ejecución que requieren, ha ocasionado que muchos investigadores busquen la forma de extraer información que les permita anotar la imagen sin tener que pasar por el proceso de la segmentación.

A continuación comentaremos algunos de los métodos actuales que los sistemas de anotación utilizan para extraer información de una imagen con la finalidad de utilizarla para relacionarla con sus anotaciones y, de esta manera, aprender a anotar automáticamente. Esta información pueden ser encontrada con la ayuda de un algoritmo de segmentación que encuentre las regiones más homogéneas -en color o textura- dentro de la imagen. Después, toda la información de la imagen, delimitada por cada segmento, será interpretada como información directamente relacionada al objeto que encierra dicho segmento y analizada como tal para predecir la anotación más probable. Para anotar imágenes exitosamente, necesitamos una segmentación que produzca segmentos bien delineados o con la menor cantidad de ruido posible.

Como se menciono en el capítulo anterior, la segmentación fina y perfecta esta muy lejos aún de nuestras posibilidades. Todos los algoritmos de segmentación terminan siempre con segmentos imperfectos que contienen información de otras regiones y no abarcan completamente la suya. Esto afecta la predicción de anotaciones. Debido a esto, muchos autores han tratado de evitar el proceso formal de segmentación de la imagen sustituyéndolo por otros métodos o simplificándolo para poder encontrar, de otra forma, información que les permitan anotar imágenes con mayor precisión. A continuación comentamos algunos métodos que se utilizan para extraer información que permita anotar una imagen.

3.4.1 Modelos Ocultos de Markov

En [James Z. Wang, 2002; James Z. Wang, 2003] utilizan Modelos Ocultos de Markov Multiresolución en 2-D (2DMHMM [J. Li, 2000]) para modelar la imagen estadísticamente tomando en cuenta el color y la textura. Un 2DMHMM explora la dependencia estadística entre píxeles o bloques de la imagen tomando una o múltiples resoluciones. Permite diferentes niveles de énfasis entre las dependencias entre una resolución y otra y las dependencias en una misma resolución. Un 2DMHMM resume dos tipos de información: conjuntos de vectores de características de múltiples resoluciones y la relación espacial entre ellos. Estos vectores normalmente reflejan el color y la textura de la imagen

Para crear el modelo 2DMHMM de una imagen, primero se obtienen versiones de la imagen en diferentes resoluciones utilizando transformadas wavelet. Después, se extraen las características representativas de la imagen en cada una de las resoluciones. Se forma una pirámide con todas las resoluciones de la imagen donde la imagen de mayor resolución esta en la base de la pirámide y la imagen de menor resolución esta en la punta. A cada píxel de una resolución menor le corresponden N píxeles en una resolución mayor. Podemos interpretar esta estructura jerárquica como un modelo de *quadtrees*. Un ejemplo de esta estructura se muestra en la Figura 3.3. Esta pirámide se utiliza para indicar las relaciones entre los vectores de características entre una resolución y otra.

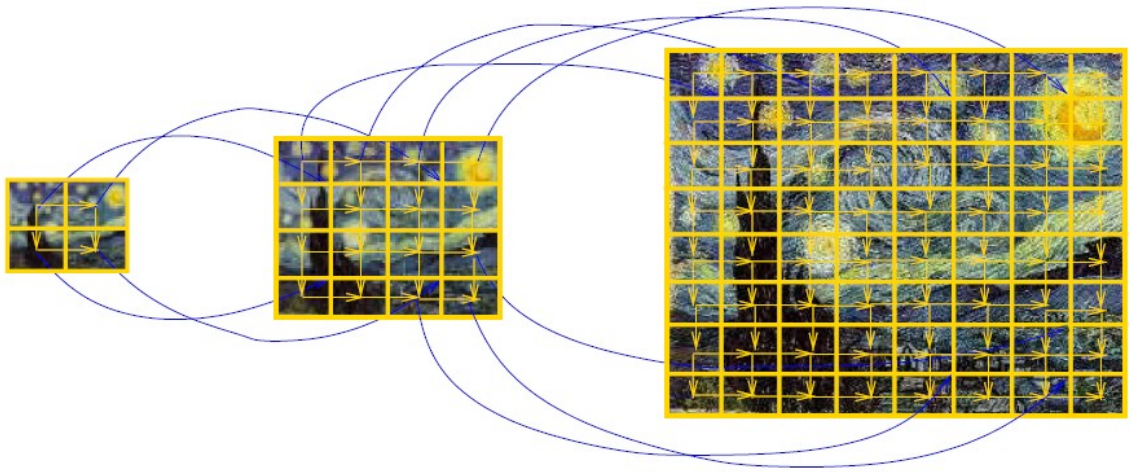


Figura 3.3: Pirámide jerárquica que indica las relaciones entre las distintas resoluciones de la imagen Este modelo evoca una estructura de quadtrees. Imagen tomada de [James Z. Wang, 2003].

Se construye un modelo para cada categoría en el cual la información gráfica de cada imagen es resumida en una colección de vectores de características extraídas de múltiples resoluciones y después acomodada en una cuadrícula piramidal Se construye un modelo por cada categoría basándose en un conjunto de imágenes representativo de la categoría. Para anotar una imagen nueva, se extraen sus vectores de características y se forma la cuadricula piramidal, luego se busca el modelo 2D MHMM más parecido y se asignan a la imagen las anotaciones de la categoría correspondiente. En la Figura 3.4 vemos un pequeño ejemplo de imágenes utilizadas para construir el modelo de la categoría: París/Francia.

3.4.2 Regiones Rectangulares

El modelo propuesto en [S. L. Feng, 2004; Jiwoon Jeon, 2004], sobrepone una cuadricula rectangular de tamaño fijo en cada imagen. La imagen es entonces representada por el conjunto de cuadros que forman la cuadricula y la información de la imagen es extraída en cada uno de estos cuadros. Esta superposición de una cuadricula sobre la imagen hace las veces de segmentación. Es una forma de realizar análisis local utilizando técnicas de análisis global. Los autores plantean que utilizar una cuadricula provee una serie de ventajas como reducción muy significativa en el tiempo computacional requerido para

Figura 3.4: Imágenes utilizadas para construir el modelo de la categoría París/Francia. Imagen tomada de [James Z. Wang, 2003].

extraer la información y que cada imagen contiene el mismo número de regiones, lo que simplifica la estimación de parámetros para el algoritmo.

Las características representativas como color, textura y posición son calculadas a partir -y en cada una- de estas regiones rectangulares. Después, todas las regiones obtenidas de todas las imágenes son agrupadas utilizando el algoritmo de *k-means* (el valor de k es elegido arbitrariamente). Los grupos encontrados son denominados *visterms* aludiendo al hecho de que estos grupos son como los *términos visuales* de un lenguaje. Después de construir este modelo, cualquier imagen de la base de datos puede ser reconstruida a partir de estos *visterms*.

Dada una nueva imagen, ésta es dividida en regiones utilizando la misma cuadrícula y luego sus regiones son analizadas. Posteriormente, se busca a que *visterms* pertenecen los cuadros de esta imagen. Las etiquetas de los *visterms* más cercanos a cada región de la cuadrícula son asignadas a esta imagen.

En [Yuli Gao, 2006] aseguran que utilizar cuadrículas de tamaño fijo para extraer características de la imagen puede dejar de detectar efectivamente varias propiedades visuales. Propone mejorar el modelo de cuadrículas aplicándolo en múltiples resoluciones para lograr una anotación *multinivel* (anotación específica y general). En este modelo, primero, las imágenes son particionadas utilizando un conjunto de cuadrículas de diferentes tamaños. Varias características son analizadas en cada rectángulo de cada tamaño de la cuadrícula. Las cuadrículas son clasificadas de acuerdo a las clases de objetos encontrados más relevantes. Los rectángulos adyacentes de las cuadrículas, que hayan sido asignados a la misma clase, se unen para formar una sola región. La Figura 3.5 nos muestra un ejemplo de anotación utilizando este modelo.

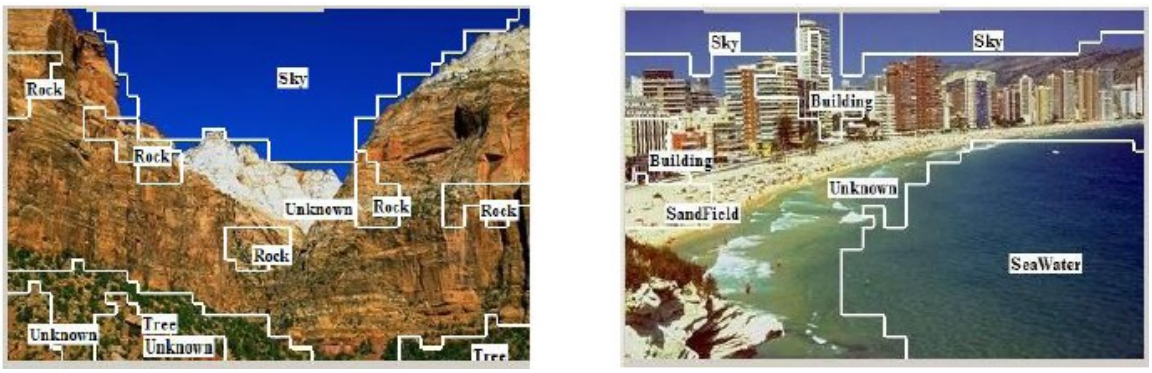


Figura 3.5: Modelo de anotación multinivel basado en análisis con cuadriculas de diferentes resoluciones. Los cuadros adyacentes de las cuadriculas que se encuentra que pertenecen a la misma clase (anotación), son unidos para formar una sola región. El análisis de las etiquetas de cada región nos permite asignar nuevas etiquetas sobre el contenido semántico de la imagen. Imagen tomada de [Yuli Gao, 2006].

3.4.3 Composición de Escena (Scene Envelope)

En [Aude Oliva, 2001] toman un enfoque inspirado por la arquitectura. Aseguran que las escenas que pertenecen a la misma categoría comparten una estructura y orden espacial similares que pueden ser extraídas sin necesidad de segmentar la imagen. Existen propiedades perceptuales tales como naturalidad (*naturalness*), ambientes creados por el hombre o naturales, apertura (openness), existencia de un horizonte sin muchos objetos de referencia, aspereza (*roughness*), tamaño de sus componentes principales, expansión (*expansion*), convergencia de líneas paralelas y rudeza (ruggedness), desvíación del suelo respecto al horizonte, y pueden ser obtenidas a partir de cálculos simples basados en transformadas discretas de Fourier y que pueden ayudar a construir descripciones representativos del espacio de la escena. Estos descriptores son llamados Plantillas Espectrales Discriminantes o DST (por sus siglas en inglés, *Discriminant Spectral Templates*). En las Figura 3.6 y Figura 3.7 podemos observar ejemplos de escenas de diferentes categorías con su respectivo espectro de energía y la representación visual de algunos de los componentes mencionados, utilizados para calcular el DST.

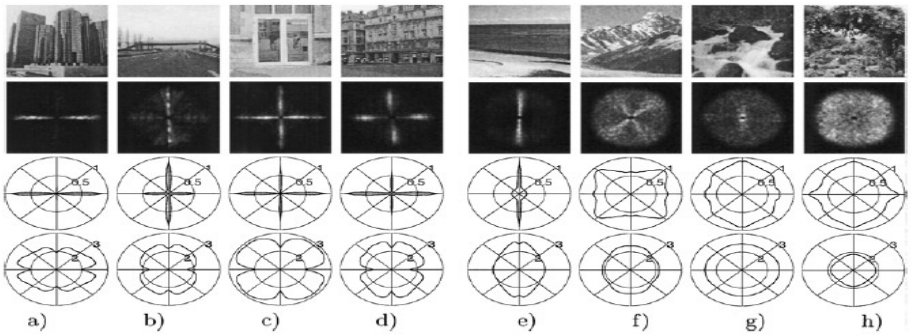


Figura 3.6: Ejemplos de escenas de diferentes categorias con su respectivo espectro de energía, obtenido a partir de transformadas de Fourier. Imagen tomada de [Aude Oliva, 2001].

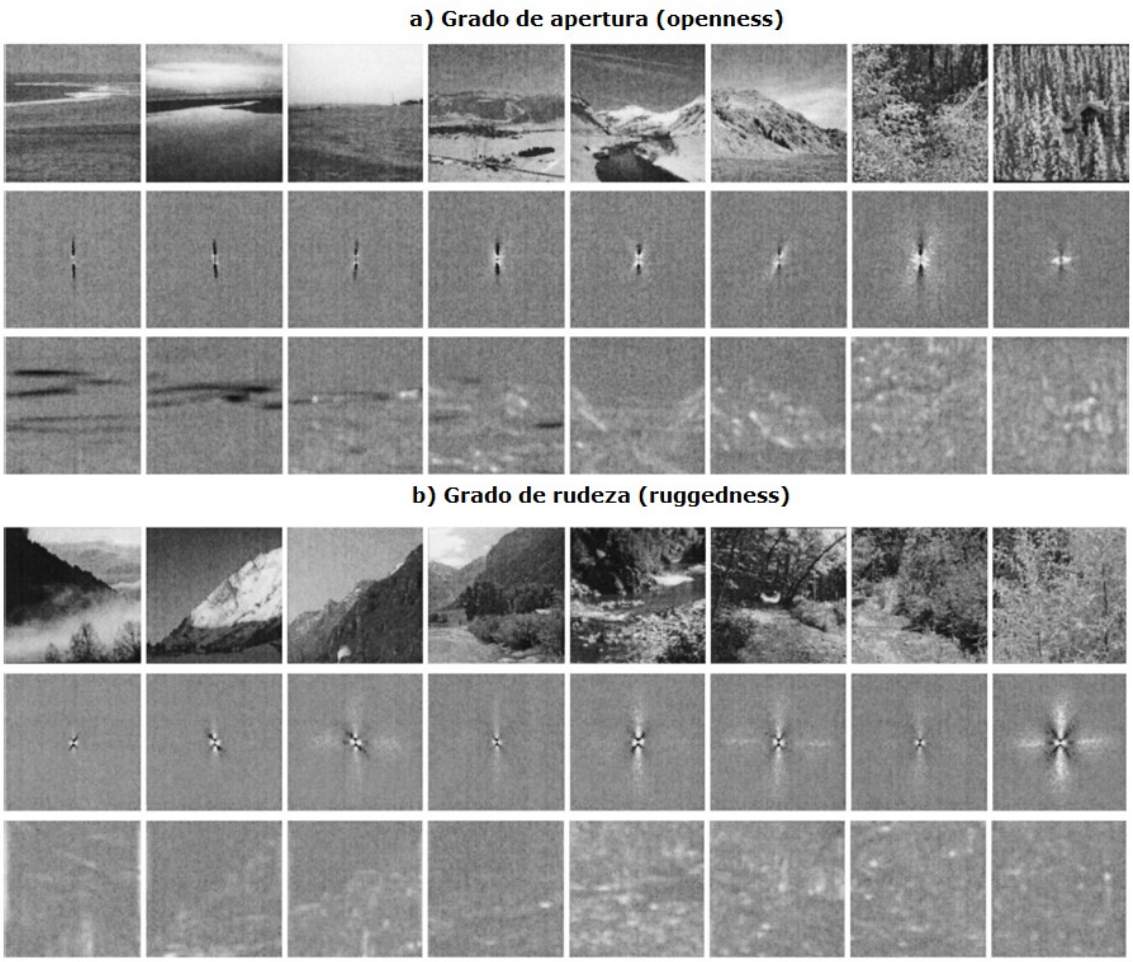


Figura 3.7: Representación visual de algunos componentes de Scene Envelope. a) Apertura (openness) mide el espacio abierto en la imagen y b) Rudeza (ruggedness) mide la variación entre el suelo y el horizonte. Un reglón abajo se muestra información del DST y otro renglón abajo se muestra el oponente de energía, éstos contribuyen al cálculo de las características mencionadas. Imagen tomada de [Aude Oliva, 2001].

3.4.4 Modelos Bayesianos

En [Alexei Yavlinsky, 2004] utilizan modelos bayesianos para construir una estimación de densidad no paramétrica utilizando la técnica de *kernel smoothing* [E Parzen, 1962]. Utiliza diferentes estructuras para extraer la información característica de las imágenes. Éstas son descritas a continuación:

Análisis de características globales.- Construye un modelo de dos vectores para describir el contenido de la imagen El primer vector contiene información del color y el segundo contiene información de la textura. Para cada píxel de la imagen calcula su valor de color CIELab, la rugosidad, el contraste y direccionalidad de textura utilizando las técnicas propuestas por Tamura [H. Tamura, 1978]. Esto da como resultado 6 dimensiones para cada píxel. Después de calcular estos valores para cada píxel, procede a

calcular la media, el segundo, tercero y cuarto momento central, para cada una de las 6 dimensiones dando como resultado vectores de 24 dimensiones. Este último vector es dividido en dos vectores: uno conteniendo información del color y el otro de la textura. En el proceso de análisis de la información estos 2 vectores son analizados de forma independiente.

Características sensitivas.- Cada imagen es dividida en 9 rectángulos de igual tamaño. Dentro de cada rectángulo, calculamos la media y el segundo momento para cada uno de las 6 dimensiones (canales) del punto anterior, dando como resultado un vector de 108 características. Notemos que esta extracción de información no intenta detectar ninguna presencia de contornos de los objetos por lo que conforma un descriptor global de la imagen

Firma de la imagen.- Convierte todos los píxeles a sus valores CIELab y después aplica el algoritmo de *k-means* con k = 16. Esto da 16 grupos de píxeles únicos para cada imagen

Los vectores generados por estas estructuras son insertados en las funciones de densidad del modelo bayesiano que predice las anotaciones en base a las probabilidades aprendidas con algún conjunto de pruebas previo.

3.4.5 Normalized Cuts

En [Rong Jin, 2004; P. Duygulu, 2002; Florent Monay, 2004; J. Jeon, 2003; Herve Glotin, 2005] utilizan el algoritmo *normalized cuts* para segmentar las imágenes. Primero segmentan todas las imágenes de la base de datos tomando en cuenta 33 características, incluyendo color, desviación estándar, orientación promedio de la energía (12 filtros), tamaño de la región, localización, convexidad, primer momento, entre otras. Una vez que se ha segmentado la imagen, se realiza una selección de únicamente las 10 regiones encontradas más grandes. Una vez que se han seleccionado los segmentos más grandes, se procede a agruparlos entre sí utilizando el algoritmo de *k-means* (a los grupos de segmentos de una misma clase se les denomina *Blobs*). Después de esto, se utiliza un método basado en Expansión-Maximización para analizar la relación entre los *blobs* y las anotaciones de las imágenes a las que pertenecen los segmentos de cada *blob*. Este método calcula la probabilidad de que una anotación *a* corresponda al *blob b* y al final le asigna la anotación que obtenga la probabilidad más alta. De esta forma, es posible anotar imágenes nuevas segmentándolas, buscando a que *blob* pertenece cada uno de sus segmentos y asignando las etiquetas correspondientes a estos *blobs*.

3.4.6 Regiones Sobresalientes

En [Jiayu Tang, 2006; Jianping Fan, 2004] se busca permitir una representación más expresiva del contenido de las imágenes utilizando para ello *regiones sobresalientes* de la imagen. Esta metodología de anotación logra crear anotaciones específicas sin necesidad

de segmentar la imagen. El proceso de segmentación es sustituido por un proceso de búsqueda y localización de regiones sobresalientes.

Las regiones sobresalientes de una imagen son encontradas en base a los picos de escala y espacio que son detectadas en una pirámide de diferencia múlti-escala del gausiano [D. G Lowe, 2004]. Una vez encontradas las regiones salientes, sus características son definidas utilizando el descriptor *SIFT* (*Scale Invariant Feature Transform*). Este descriptor esta compuesto por un histograma tridimensional de la intensidad y orientación del gradiente [D. G Lowe, 2004].

Una vez que se tienen identificadas las regiones salientes de un conjunto de imágenes anotadas de prueba, se procede a crear modelos probabilísticos de intersección distributiva, CMRM, (*Cross-Media Relevance Models*) [J Jeon, 2003] que permiten calcular la probabilidad de que una anotación corresponda a cierta region saliente basándose en sus características (*SIFT*). Una vez que se ha construido este modelo, es posible anotar nuevas imágenes detectando sus regiones salientes y consultando sus SIFT's en el modelo creado para predecir las anotaciones que corresponden.

3.4.7 Segmentación Simplificada o Débil (Weak Segmentation)

La segmentación simplificada es un enfoque de segmentación en el cual separar *perfectamente* todos los objetos que se encuentran en la imagen no es necesario. Basta encontrar ***regiones*** características, homogéneas y sin mucho ruido que proporcionen información **suficiente** para deducir el contenido de una imagen

En [Le Thi, 2004] utilizan este modelo para formar regiones homogéneas utilizando técnicas de unión y división (*split and merge*) sobre la imagen. Cada región es dividida mientras no sea homogénea y las regiones que se hacen muy pequeñas se agregan a regiones cercanas parecidas. El resultado final es suficiente para determinar el contenido principal y general de la imagen. En el siguiente capítulo se comentará más ampliamente sobre este tipo de segmentación.

3.5 Conclusión

Hemos descrito diferentes métodos para extraer información de las imágenes que nos sirva como base para poder realizar anotaciones. Todos estos métodos varían en complejidad y tiempo de ejecución. En este trabajo queremos presentar un nuevo algoritmo capaz de segmentar imágenes de manera rápida y sencilla. Una vez que nuestro algoritmo ha sementado una imagen, los segmentos resultantes contendrán información suficiente para poder anotarla. El algoritmo está diseñado para descubrir las regiones más grandes y homogéneas lo más rápido posible y después concentrarse progresivamente en las regiones más chicas. Esto se logra utilizando un enfoque basado en *quadtrees*.

Capítulo 4

Segmentador de imágenes orientado a anotaciones basado en *quadtrees*

En este capítulo describiremos el algoritmo que proponemos para segmentar imágenes. Nuestro algoritmo utiliza un recorrido en *quadtree* para segmentar la imagen, funciona para imágenes en color de cualesquiera dimensiones y se ejecuta rápidamente. Es posible especificar la resolución del *quadtree* y el tamaño mínimo de los objetos que deseamos encontrar. El funcionamiento general de nuestro algoritmo consiste en discretizar la imagen, encontrar los contornos, procesarlos para obtener la imagen de restricciones, hacer el recorrido de *quadtree* determinando cuáles cuadrantes unir y, finalmente, refinar los resultados obtenidos con este recorrido. A lo largo de este capítulo explicaremos detalladamente cada uno de estos pasos.

4.1 Antecedentes

Hoy en día contamos con algoritmos de segmentación muy buenos. No obstante, aún estamos muy lejos de un algoritmo que de como resultado segmentos que delineen finamente los objetos que representan. Además, generalmente, la segmentación es un proceso complejo que consume muchos recursos (memoria y tiempo). Uno de los algoritmos más usados por las comunidades de anotación de imágenes, *normalized cuts*, tarda aproximadamente 2 minutos para segmentar una imagen de 100 x 120 píxeles [J. Shi, 2000]. Los sistemas de anotación automática normalmente trabajan con bases de datos de cientos de miles de imágenes por lo que resulta inviable segmentar toda la base de datos, tardaría mucho tiempo y los segmentos que obtendríamos no serían lo suficientemente precisos como para justificar todo ese tiempo invertido. En sistemas que interactúan con el usuario la espera debe ser lo más corta posible. También es importante mencionar que normalmente las imágenes de la base de datos no son procesadas en su tamaño original ya que son reducidas para poder hacerlo con mayor velocidad lo cual ocasiona que se pierdan algunos detalles. Debido a todos estos inconvenientes la mayoría de los investigadores han intentado evitar la segmentación o al menos simplificarla.

Es en este contexto que nace la **segmentación débil** (*weak segmentation),* la cual se basa en el hecho de que es imposible obtener segmentos finamente delineados y apegados estrictamente a los objetos que representan. Con esta imposibilidad en mente, la segmentación débil busca reducir esfuerzos en logar esta precisión y se limita a buscar regiones generales que indiquen a grandes rasgos donde están y cuales son los principales objetos de la imagen. Diversos trabajos demuestran que este tipo de segmentación es suficiente para poder anotar bases de imágenes [Le Thi Lan, 2004]. En la Figura 4.1 podemos observar un ejemplo de segmentación débil.

Hay también algunos trabajos que cambian completamente el paradigma de "encontrar objetos en una imagen" basándose en metodologías tales como la búsqueda de regiones

Figura 4.1: Ejemplo de segmentación débil. Izquierda.- Imagen original. Derecha.- Resultado de segmentar la imagen original utilizando un algoritmo de segmentación débil. Podemos observar que aunque los segmentos no están finamente delineados, sí esbozan una segmentación general del contenido de la imagen.

salientes o patrones estadísticos en el contenido, sustituyendo complemente la segmentación. De estos métodos ya se hablo en el Capítulo 3.

4.2 Nuestra Propuesta

Para solucionar los inconvenientes mencionados en la sección anterior presentamos un *algoritmo de segmentación débil* basado en *quadtrees*, capaz de segmentar imágenes de cualquier tamaño en tiempos muy cortos. Más aún, es posible especificar la resolución de la segmentación, permitiendo así mayor control sobre el tiempo dedicado a este proceso. También se permite especificar el tamaño mínimo de los objetos que deseamos encontrar.

El algoritmo que proponemos parte de que una imagen normalmente consta de regiones *homogéneas,* en color y/o textura, como cielo, pasto, paredes, pavimento, ropa, montañas, edificios, ropa, mar, superficies, piel, etc. Por lo tanto podemos segmentar una imagen si logramos separar todas las regiones homogéneas que contiene. Para que la segmentación sea rápida, necesitamos encontrar estas regiones en la menor cantidad de pasos posibles.

Utilizando una estructura de *quadtree* para dividir la imagen de forma recursiva, podemos -por la misma naturaleza del *quadtree*- encontrar regiones homogéneas en poco tiempo mientras más grandes sean y dedicar mayor esfuerzo y tiempo sólo a las regiones menores. Con este método, ahorramos el mayor tiempo posible y además producimos segmentaciones más finas conforme avanza el tiempo de ejecución del algoritmo.

La idea principal consiste en dividir la imagen en cuatro cuadrantes de igual tamaño, trazando dos ejes perpendiculares que se intersecten en el centro de la imagen. Una vez dividida, compararemos cada uno de los cuadrantes con sus cuadrantes vecinos (utilizando el enfoque de conectividad *4-vecinos*) y procederemos a unir aquellos cuadrantes que consideremos pueden formar una sola región homogénea. Aquellos cuadrantes que no puedan ser unidos con ninguno de sus vecinos serán divididos a su vez en 4 sub-cuadrantes. Compararemos estos sub-cuadrantes con sus sub-cuadrantes vecinos y uniremos los que puedan ser considerados como una sola región homogénea revisando sí esta nueva región puede ser unida a alguna de las regiones previamente formadas. Los cuadrantes que no sean unidos con ninguno de sus vecinos volverán a dividirse. Este proceso se repite de manera recursiva hasta que alcancemos el tamaño mínimo (indicado por el usuario) para los cuadrantes o cuando ya hayan sido unidos entre sí todos los cuadrantes generados y, por lo tanto, no haya más cuadrantes que dividir. Podemos observar fácilmente que esta propuesta nos conduce a una implementación basada en *quadtrees*. Para decidir si dos cuadrantes pueden ser unidos en una sola región, utilizaremos un operador que tome en cuenta los contornos presentes en la región y la similitud entre los histogramas de ambas regiones.

Para llevar a cabo el proceso de segmentación descrito anteriormente son necesarios varios pasos. La Figura 4.2 muestra el diagrama funcional que describe el funcionamiento

global del algoritmo. El número dentro de los bloques internos indica su orden de ejecución y las flechas indican sus respectivas entradas y salidas. Podemos observar que son necesarias cuatro entradas para el funcionamiento del segmentador. La primera entrada es la imagen que se desea segmentar. El formato de imágenes permitidas es JPEG. La segunda entrada es el tamaño mínimo de los objetos que deseamos que encuentre el segmentador. Todos los objetos encontrados menores a este parámetro serán unidos al objeto adyacente más similar que sí cumpla esta condición, en caso de no encontrarse ningún objeto para dicha unión, el objeto en cuestión será ignorado en el resultado final de la segmentación. El tamaño mínimo de los objetos es un porcentaje del tamaño de la imagen indicado en valores del 0 al 1, donde 0 equivale a 0% y 1 equivale a 100%. El tercer parámetro permite especificar el tamaño de los cuadrantes del último nivel del *quadtree* que se generará. Gracias a este parámetro es posible indicar hasta que nivel de descomposición deseamos desarrollar el *quadtree*. Mientras menor sea este parámetro, más fina será la segmentación resultante. Finalmente, el cuarto parámetro permite especificar el valor límite que utilizará el algoritmo para decidir si dos cuadrantes del *quadtree* son suficientemente similares y, por lo tanto, deben ser unidos. Este parámetro toma como valor un porcentaje indicado en valores del 0 al 1, de manera análoga al parámetro del tamaño mínimo del objeto.

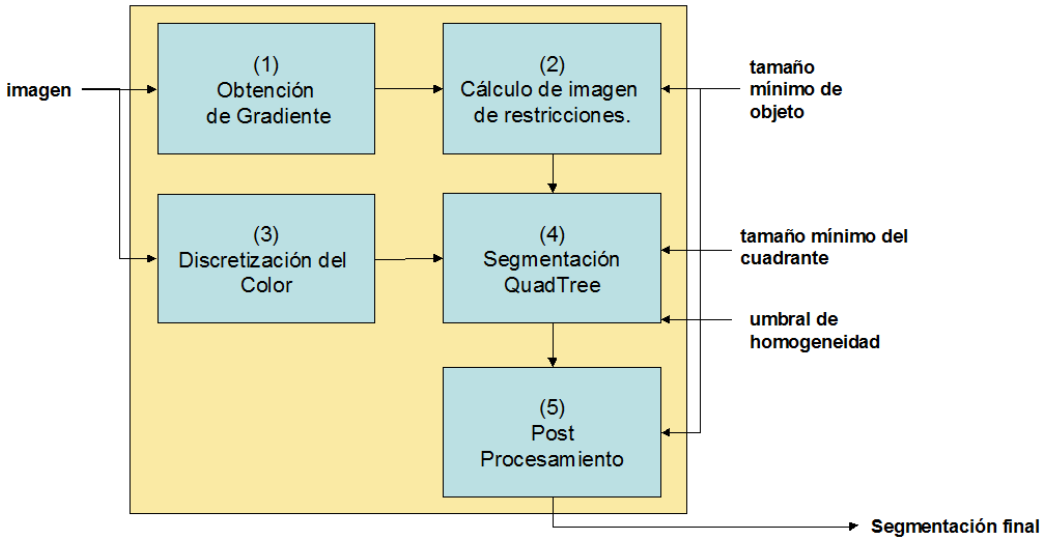


Figura 4.2: Modelo Funcional del Segmentador. Primero obtenemos el gradiente y en base a él construimos una imagen de restricciones que guiará la segmentación. Después discretizamos la imagen e iniciamos el recorrido quadtree uniendo y dividiendo regiones adyacentes de la imagen. Finalmente, refinamos la segmentación a través de la unión de segmentos pequeños a unos más grandes.

De manera general, el proceso que sigue el algoritmo para producir una imagen segmentada es el siguiente.

1. Obtenemos el gradiente de la imagen utilizando el operador *sobel(*1). Trataremos este punto en la Sección 4.3.
2. Creamos una *imagen de restricciones* ejecutando el algoritmo de componentes conectados sobre el gradiente encontrado para eliminar píxeles sueltos y

quedarnos únicamente con los bordes de mayor tamaño (2). Trataremos este punto en la Sección 4.4.

3. *Discretizamos* la imagen de entrada para simplificar la creación y las operaciones entre histogramas (3). Trataremos este punto en la sección 4.5.
4. Iniciamos el recorrido en *quadtree* de la imagen, realizando el proceso de división, comparación y unión descrito anteriormente. Trataremos este punto en la sección 4.6.
5. Mejoramos la segmentación producida en el paso 4, propagando algunas etiquetas entre segmentos y uniendo regiones que son demasiado pequeñas. Trataremos este punto en la sección 4.6.4 y 4.6.5.

A continuación explicaremos detalladamente el proceso y justificación de cada uno de los módulos del algoritmo.

4.3 Obtención de Gradiente

El núcleo de nuestro proceso de segmentación radica en comparar cuadrantes de la imagen revisando si son suficientemente similares para ser convertidos en un solo segmento. A lo largo de este proceso, podemos encontrarnos con algunas situaciones en las que dos cuadrantes estadísticamente similares, incluso idénticos, no deben ser unidos. Un ejemplo de esto lo observamos en la Figura 4.3. Al comparar estadísticamente el contenido de los cuadrantes 1 y 2 o 3 y 4 podemos obtener diferencias muy pequeñas, lo que llevaría a nuestro algoritmo a unirlas y terminar la segmentación con únicamente 2 segmentos. Este problema se presenta constantemente en las imágenes que poseen regiones simétricas.

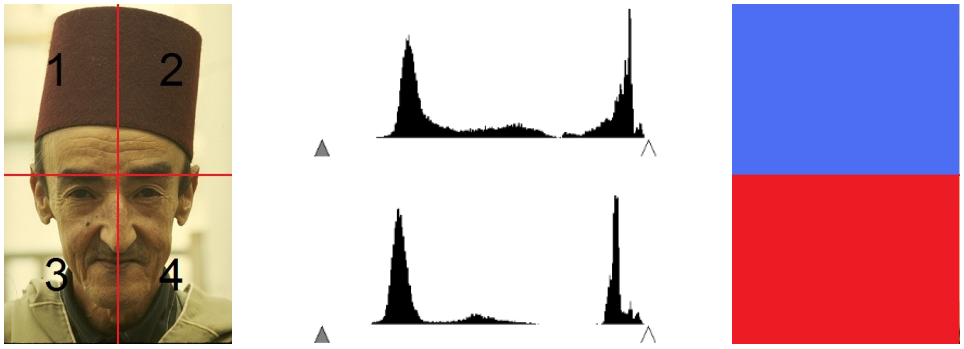


Figura 4.3: a) Imagen en la primera etapa de división quadtree, b) Histogramas del cuadrante 1 (superior) y del cuandrante 2 (inferior). Como podemos observar, al comparar los histogramas del cuadrante 1 y 2 la diferencia será mínima y ambos cuadrantes serán unidos en una sola región, lo mismo pasará al comparar las regiones 3 y 4. En este caso el resultado final de segmentación´sería la imagen c).

Para solucionar este problema, necesitamos una guía que nos ayude a saber en que lugar de la imagen existen bordes que nos indiquen la presencia de los límites de un objeto. De

esta manera, antes de comparar las cuadrantes de la imagen, primero revisaremos que dentro de dicho cuadrante no haya bordes que indiquen que en ese cuadrante hay más de un objeto.

Existen diversos métodos para encontrar los bordes de una imagen. Uno de los métodos más fáciles consiste en aplicar el operador ***sobel*** a la imagen. El operador sobel es una máscara de 9 píxeles acomodados en forma de matriz de 3 columnas por 3 filas. Para encontrar los bordes de una imagen, debemos colocar el centro de esta máscara sobre cada uno de los píxeles de la imagen original y multiplicar los valores de cada píxel por el valor de la máscara que quede sobre él, la suma de todas estas multiplicaciones, dividida entre el total de píxeles multiplicados, será el valor del gradiente para ese punto. Este valor de gradiente indica el cambio de intensidad de color que existe en la región donde se coloco la máscara. Este valor es útil debido a que los bordes de una imagen normalmente se encuentran en donde hay mayor variación de intensidad. Existen dos versiones del operador sobel, una para detectar los cambios de intensidad horizontales y otra para detectar los cambios de intensidad verticales (Figura 4.4). Esto quiere decir que si deseamos encontrar los bordes de la imagen en ambas direcciones, debemos aplicar dos veces este procedimiento, una vez con el operador horizontal y otra vez con el operador vertical. Esto nos dará como resultado dos valores de cambio de intensidad para cada píxel. Las propuestas para combinar este valor son variadas, una de ellas consiste en quedarse con el valor más alto y es el que utilizamos en este trabajo. Una vez que se ha calculado el gradiente para todos los píxeles de la imagen, procedemos a elegir solo aquellos que tengan un valor de intensidad muy alto dado que son éstos los que nos indican la presencia de bordes más notables. En este trabajo el proceso de selección consiste en quedarnos únicamente con los píxeles que estén por encima del 80% del valor más alto de intensidad encontrado. Es decir, si el valor más alto de intensidad fue 230, nos quedaremos con todos aquellos píxeles que sean mayores a 230 * 0.80 = 184.

Una práctica común que normalmente genera mejores resultados consiste en aplicar el operador S*obel* a una versión en escala de grises de la imagen original en lugar de aplicarlo directamente sobre la imagen a color. Así lo hacemos en este trabajo. La conversión de una imagen a color a escala de grises se realiza asignando a cada píxel el valor promedio de sus componentes RGB.

Sobel: $M_x =$

-1	0	1
-2	0	2
-1	0	1

; $M_y =$

1	2	1
0	0	0
-1	-2	-1

Figura 4.4: Operador Sobel. Mx - Máscara para detección de cambios de intensidad verticales. My - Máscara para detección de cambios de intensidad horizontales. Imagen tomada de [Computer Vision, Linda Shapiro, pag. 64]

La Figura 4.5 muestra el resultado de aplicar el operador sobel a la imagen de la Figura 4.3. Una vez encontrados los bordes en la imagen que deseamos segmentar, podemos

usarlos como guía para saber cuando no debemos tomar un cuadrante como sí fuera una sola región. Por ejemplo, en la Figura 4.6 vemos que el cuadrante 3 y 4, ambos, contienen bordes lo cual nos indica que estos cuadrantes seguramente pertenecen a un área donde inicia una región y comienza otra siendo claro que no debemos considerarlo como una sola región, por lo tanto los cuadrante 3 y 4 no pueden ser unidos. Lo mismo aplica al comparar los cuadrantes 6 y 8, 11 y 12 o 2 y 4, por ejemplo. Tal no es el caso de los cuadrantes 9 y 11, los cuales sí pueden ser unidos debido a que no hay dentro de ellos ningún borde que nos indique el fin de una región y principio de otra. Cabe mencionar que esta comprobación de la presencia de bordes dentro de dos cuadrantes adyacentes es sólo la primera condición para que éstos sean unidos. Una segunda condición consiste en que dichos cuadrantes sean similares.

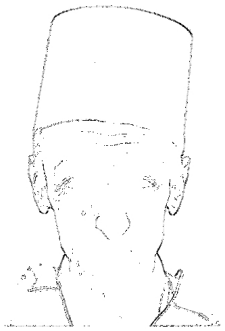


Figura 4.5: Gradiente de la imagen de la Figura 4.3

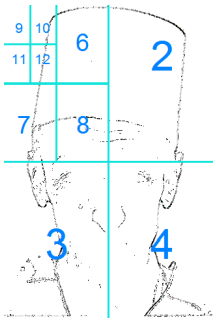


Figura 4.6: Cuadrantes superpuestos sobre el gradiente.

4.4 Cálculo de la Imagen de Restricciones

La presencia o ausencia de bordes es una forma simple de decidir si un cuadrante es homogéneo o no, es decir, si todo él pertenece a una única región. Sin embargo, el operador de gradiente puede detectar muchos cambios de intensidad en la imagen que no necesariamente indican el termino de una región y principio de otra. Por ejemplo, un lunar en el rostro de una persona ocasiona un cambio de intensidad en el perímetro del lunar pero un lunar es un objeto demasiado pequeño como para que deseemos localizarlo en un segmento propio, preferiríamos que fuera incluido dentro del segmento que pertenece al rostro de la persona.

Debido a lo anterior, es importante definir cuando interpretaremos la información que nos proporciona el gradiente como un borde y cuando debemos ignorarla. En la Figura 4.7 señalamos 3 diferentes áreas del gradiente de la Figura 4.3. El fragmento 1 muestra un borde largo y continuo, el fragmento 2 muestra un borde discontinuo y el fragmento 3 muestra bordes dispersos. De estos tres tipos de bordes, el que mejor nos indica la presencia de un cambio de región es el borde continuo del fragmento 1. Los bordes discontinuos del fragmento 2 también *podrían* indicar la presencia de un cambio de región pero no son lo suficientemente largos y sus discontinuidades nos hacen dudar que indiquen los límites de un objeto independiente. Si observamos la imagen original en la Figura 4.3 veremos que tales bordes pertenecen a las arrugas de la frente del hombre de la imagen, confirmando con esto nuestras suposiciones al respecto de ellos. Finalmente, el fragmento 3 nos muestra bordes que son muy pequeños y que se encuentran separados de otros bordes lo que nos hace pensar que no indican la presencia de otra región. Aún en caso de que realmente indicaran la presencia de otra región, su pequeño tamaño nos sugiere no tomarla en cuenta como una región independiente. Al observar la imagen de la Figura 4.3 vemos que estos bordes, en efecto, no pertenecen a ninguna región representativa de la imagen.

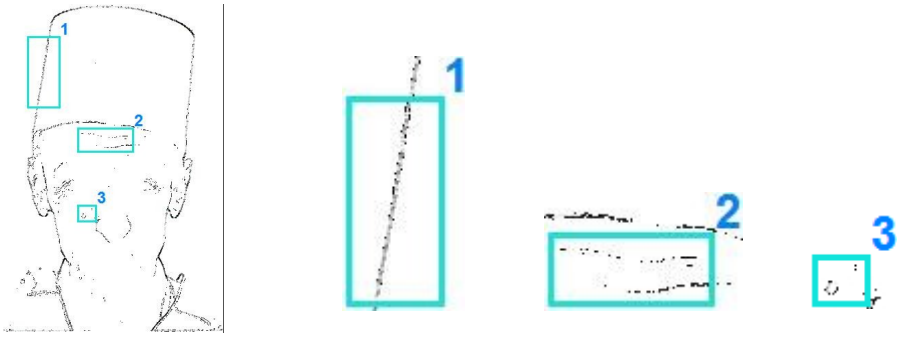


Figura 4.7: Tres diferentes patrones de bordes que se encuentran en el gradiente de la Figura 4.3. 1) Bordes continuos, 2) Bordes no continuos y 3) Bordes dispersos.

Ahora que hemos definido las propiedades que tomaremos en cuenta para decidir si un borde nos indica o no la presencia de los límites de una región, debemos idear un proceso que nos ayude a identificar automáticamente los bordes que las cumplen y quedarnos sólo con ellos.

Para quedarnos sólo con los bordes largos y continuos debemos primero definir exactamente cuando un borde es considerado **largo.** Dada la relatividad del término, no podemos fijar un valor exacto que separe a los bordes *largos* de los bordes *cortos*. Sin embargo, hemos ideado un método para aproximarnos a este valor basándonos en el *tamaño mínimo de los objetos que deseamos encontrar,* el cual es un parámetro de entrada para la ejecución del algoritmo.

Supongamos que deseamos encontrar objetos de al menos el 10% del tamaño de la imagen. Para esto, el perímetro de los objetos encontrados debe ser al menos del 10% del

perímetro de toda la imagen. Por lo tanto, todos aquellos bordes que sean más largos del 10% del perímetro de la imagen seguramente corresponderán a un objeto que ocupa al menos 10% de la imagen. De esta forma, la tarea de identificar que un borde indica la presencia de los límites de una región de al menos 10% del tamaño de la imagen es revisar si su longitud es mayor o igual al 10% del perímetro de la imagen.

El valor por omisión para este parámetro es de 1.25%, dicho valor fue hallado a prueba y error de manera exhaustiva durante las pruebas del algoritmo. Cabe mencionar que este es el mejor valor encontrado para obtener segmentos de tamaño aceptable según la visión del autor, sin embargo esta visión es subjetiva. El valor del parámetro puede ajustarse según las necesidades de la aplicación. Hacerlo más grande ocasionará encontrar menos segmentos (objetos) en la imagen. Hacerlo más pequeño ocasiona lo contrario.

Para poder medir la longitud de los bordes del gradiente de la imagen, es necesario identificar cuales son los grupos de píxeles que componen cada uno de ellos. Esto lo haremos con ayuda del algoritmo de *componentes conectados* [Linda Shapiro, 2003, p69]. El algoritmo de componentes conectados agrupa todos los píxeles adyacentes que pertenecen a un mismo objeto para cada uno de los objetos de una imagen. Este algoritmo supone que el grupo de objetos que desea encontrar están colocados sobre un fondo. La imagen de entrada para este algoritmo sólo puede tener píxeles con valor 0 o 1. Los píxeles con valor 0 pertenecen al fondo de la imagen y los píxeles con valor 1 pertenecen a los objetos. En nuestro caso, todos los píxeles pertenecientes al gradiente tomarán el valor 1 y todos los demás píxeles tomarán el valor 0. El algoritmo recorre toda la imagen, de izquierda a derecha y de arriba a abajo, comenzando por la esquina superior izquierda. Durante este recorrido, cada que encuentra un píxel blanco (1) revisa a su derecha y debajo si tiene vecinos, en caso afirmativo los une a él para formar un solo componente asignándoles una misma etiqueta. Se puede dar el caso de que encuentre píxeles que ya hayan sido etiquetados, en este caso toma la etiqueta más antigua y la propaga hacia los otros píxeles, llevando un registro de las etiquetas reemplazadas. Al final de su recorrido, evalúa los reemplazos de etiquetas que realizó para encontrar etiquetas equivalentes y actualizar esta información en los componentes construidos. Cuando termina su ejecución tenemos una imagen *coloreada* donde cada componente tiene un color (etiqueta) distinto. De esta manera, para contar el número de píxeles pertenecientes a cada uno de los bordes, solo debemos contar cuantos píxeles hay de cada una de las etiquetas encontradas por el algoritmo. Finalmente, procedemos a eliminar todos aquellos bordes (objetos) que no tienen un número suficiente de píxeles según el valor calculado en el paso anterior. La Figura 4.8 ilustra el proceso de selección de bordes. La imagen de la izquierda muestra el resultado original del gradiente. La imagen del centro es el resultado de aplicar el algoritmo de componentes conectados a la imagen de la izquierda. Podemos observar que el algoritmo ha unido y coloreado todos los píxeles que forman un solo componente basándose en su adyacencia. La imagen de la derecha resulta después de eliminar todos los componentes que no tienen la longitud requerida.

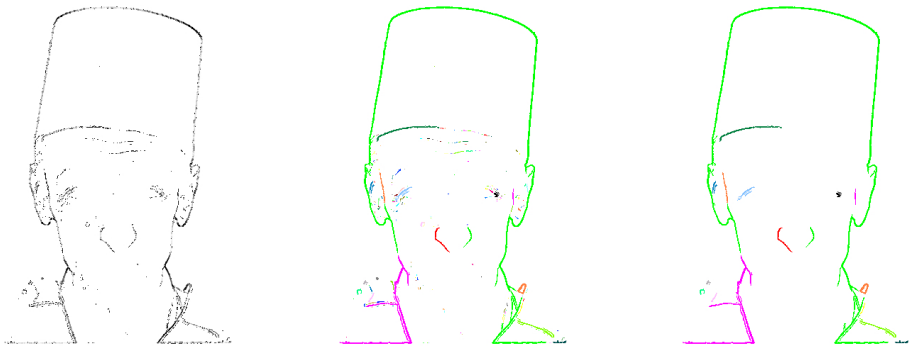


Figura 4.8: Proceso de selección de bordes largos y continuos. Izquierda.- Gradiente Original, Centro.- Resultado de aplicar el algoritmo de componentes conectados a la imagen de la izquierda. Derecha.- Eliminación de los componentes que no tienen la longitud requerida.

La imagen obtenida con este proceso de selección de bordes es la que será utilizada por nuestro algoritmo para consultar si un cuadrante contiene bordes. En principio, un solo píxel coloreado dentro del área correspondiente a un cuadrante del *quadtree* será suficiente para indicar que dicho cuadrante no puede ser unido con ningún otro, es por esto que llamamos a esta imagen: *imagen de restricciones.* Sin embargo, en la Sección 4.6.2 veremos una forma más práctica de interpretar estos píxeles.

Este método de selección de bordes es sencillo y únicamente requiere recorrer cuatro veces la imagen para concluir su trabajo. Una para colorear los componentes conectados, otra para sustituir etiquetas equivalentes, una para calcular el área de cada componente y otra para eliminar los componentes que no tengan la longitud suficiente. Sin embargo, existe un pequeño inconveniente con este método, lo mencionamos a continuación.

Algunas imágenes tienen una gran cantidad de cambios de intensidad que no necesariamente significan presencia de bordes. En la Figura 4.9 se ilustra un ejemplo de esta situación, b) es el gradiente de la imagen a). Al ejecutar el algoritmo de componentes conectados obtenemos c). La numerosa cantidad de cambios de intensidad a lo largo de toda la imagen ocasiona que la mayoría de los píxeles blancos en b) sean adyacentes con lo que el algoritmo de componentes conectados los toma como un solo objeto. Debido a esto, la gran mayoría de objetos encontrados en c) constan de muchos más píxeles de los necesarios para ser considerados "largos" y por eso no son eliminados cuando ejecutamos el proceso de selección de bordes. Después de eliminar los objetos pequeños obtenemos la imagen d), que es prácticamente igual a c). Recordemos que esta *imagen de restricciones* será utilizada como guía de la segmentación, es decir, la unión de cuadrantes no será posible donde la imagen d) contenga píxeles distintos de 0, con lo que la segmentación queda exageradamente limitada por esta imagen debido a la gran cantidad de píxeles restrictivos. El método utilizado para superar este inconveniente se explica en la Sección 4.6.2 cuando veamos las reglas del comparador de regiones que dirige el recorrido en *quadtree*.

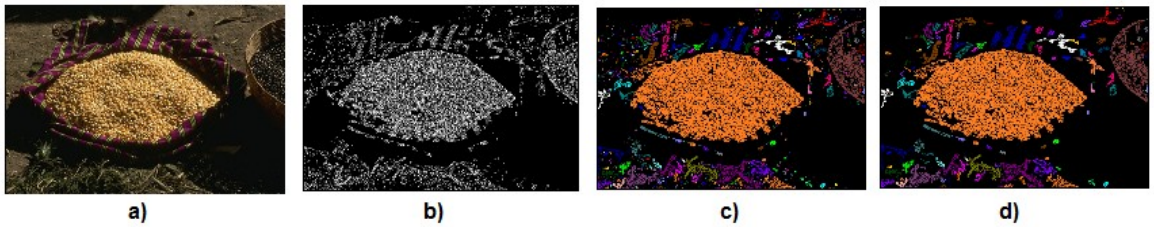


Figura 4.9: Imagen con muchos cambios de intensidad. Caso peculiar en que la imagen de restricciones calculada para guiar la segmentación es demasiado restrictiva. a) Imagen original, b) gradiente, c) componentes conectados, d) selección de bordes. Esta imagen de restricciones evitará encontrar segmentos en la mayor parte de la imagen y no encontrará el obejto principal. En la Sección 4.6.2 veremos como superar este inconveniente.

4.5 Discretización del Color

Normalmente los píxeles de una imagen son representados utilizando el modelo de color RGB, aunque no es el único modelo (Sección 2.2.1). El modelo RGB utiliza combinaciones de un valor de rojo (RED), un valor de verde (GREEN) y un valor de azul (BLUE) para representar cualquier otro color. Se utiliza un byte para almacenar el valor de cada color, con lo que los valores que puede tomar cada componente van del 0 al 255. Esto significa que tenemos 256^3 = 16,777,216 de combinaciones distintas, cada una produciendo un color distinto.

Cuando deseamos procesar imágenes a color, normalmente operaremos con los valores de cada uno de sus píxeles, calcularemos la media, la varianza, el histograma, la textura y un sin fin de propiedades. Al realizar estos cálculos siempre es aconsejable reducir el número de colores que vamos a trabajar, para simplicidad y velocidad en los cálculos. Esto se logra discretizando el espacio de colores del modelo RGB, haciendo que reduzca el número de colores posibles.

La forma más simple de discretizar el espacio de colores es reduciendo el rango de valores posibles para cada componente del modelo. Es decir, en lugar de otorgar un byte para almacenar el valor de cada uno de los componentes, podemos asignar solo ½ byte con lo que reducimos el rango [0, 255] a [0, 127] y eso nos da 128^3 = 2,097,152 combinaciones de colores diferentes. Este número es aún muy grande. En la práctica, es común utilizar un único byte para almacenar los tres componentes de color R, G y B. Dado que 8 / 3 no es un número entero, se suele utilizar la siguiente combinación: 3 bits para el rojo, 3 bits para el verde y 2 bits para el azul. Se otorgan menos bits al componente azul, debido a que es el que menos distingue la vista humana. De esta manera, se logra reducir el espacio de colores a solamente 255 colores diferentes, lo cual es una reducción que ahorra mucho tiempo en la ejecución de procesos relacionados con el color. Para este trabajo utilizamos el siguiente esquema de discretización: 2 bits para el rojo, 2 bits para el verde y 2 bits para el azul. Con esto, obtenemos un total de 2^3 = 64

colores distintos en el espacio de colores. Elegimos este valor porque fue el que nos dio mejores resultados durante la etapa de pruebas.

Para reducir el espacio de colores de la imagen, se recorre toda la imagen y para cada uno de sus píxeles tomamos los *n* bits más significativos de cada uno de sus componentes RGB. Armamos con ellos un nuevo byte colocando estos bits en la posición adecuada según el componente que representen. Finalmente, reemplazamos el valor original del píxel por el byte que armamos. También es importante modificar la cabecera del archivo JPEG para indicar que ahora la imagen se trata de una imagen en escala de grises. Esto último es para eliminar definitivamente los bytes ocupados por los demás componentes y evitar que ocupen espacio en memoria, innecesariamente. La Figura 4.10 ilustra el proceso de discretización de un color. El color utilizado para el bloque a) esta formado por los valores RGB que se observan en la imagen. Para discretizar este color, tomamos únicamente los 2 bits más significativos de cada componente y armamos con ellos un nuevo color colocándolos en un byte de la siguiente forma: bits 6 y 5: rojo, bits 4 y 3: verde, 2 y 1: azul. Los bits 8 y 7 quedan siempre con valor 0. La Figura 4.11 muestra el resultado de discretizar la imagen de la Figura 4.3.

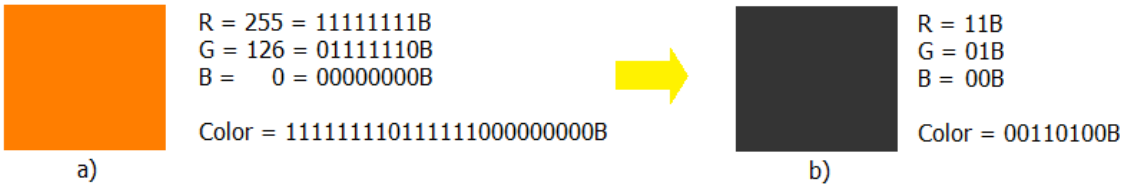


Figura 4.10: Ejemplo de discretización de color. a) Color original en modelo RGB convencional. Cada componente utiliza 1 byte para almacenar su valor. El color en a) utiliza 3 bytes en total. En b) se muestra el mismo color discretizado. Se toman únicamente los 2 bits más significativos de cada componente. El color discretizado utiliza sólo 1 byte para almacenar su valor.

Figura 4.11: Resultado de la discretización de la imagen de la Figura 4.3. Izquierda.- imagen discretizada original. Derecho.- imagen discretizada con brillo aumentado para permitir una mejor visualización al lector.

4.6 Segmentación *quadtree*

Los *quadtrees* [Hanan Samet, 1984; Wu A. Y., 1982, Tucker L.W., 1984; Ranade S., 1980, Ranade S., 1981; Hunter G. M., 1979] son estructuras de datos basadas en el principio de descomposición recursiva del espacio, inspirado en los métodos de *divide y vencerás*. Los *quadtrees* son útiles debido a que permiten concentrarse en subconjuntos de datos específicos y descartan rápidamente grandes volúmenes de datos que no son de interés. Esta característica la proporciona su habilidad para trabajar con grupos maximales de información. El hecho de poder concentrarnos rápidamente en la información que nos interesa hace que los algoritmos basados en *quadtrees* generalmente sean más veloces que sus análogos o, en el peor de los casos, se ejecuten en la misma cantidad de tiempo. Además de esto, los *quadtrees* son una estructura atractiva para trabajar debido a la simplicidad del concepto y a su fácil implementación. Los *quadtrees* son utilizados para procesar puntos, regiones, curvas, superficies y volúmenes. En esta tesis nos enfocaremos al procesamiento de regiones.

Un algoritmo de *quadtrees* debe descomponer un objeto siempre en partes del mismo tamaño y de la misma forma en cada nivel de la descomposición. El tamaño de la estructura base va disminuyendo conforme avanza la descomposición. Llamamos resolución al número de veces que se realiza el proceso de descomposición. La resolución de un algoritmo de *quadtree* puede ser fija o puede ser determinada en tiempo de ejecución por las propiedades de la información sobre la que se opera.

Durante el proceso de la descomposición se crea un árbol en donde el nodo raíz corresponde a la imagen completa. Debido a que dividimos la imagen en cuatro cuadrantes, el nodo raíz tendrá cuatro nodos hijo. Éstos a su vez deberán engendrar otros 4 nodos en caso de que la región que representen sea subdividida nuevamente. En este árbol, cada rama corresponde a una región no homogénea y cada hoja corresponde a una región homogénea de la imagen. El árbol resultante de ejecutar el proceso de subdivisión es un árbol de grado 4 debido a que cada nodo del árbol, que no es una hoja, tiene 4 nodos hijo. En la Figura 4.12 podemos observar un ejemplo de *quadtree*.

En el Anexo A se pone a disposición del lector mayor información sobre los *quadtrees.* Ahora veremos como podemos aplicar esta técnica para segmentar una imagen. Partimos del hecho de que una imagen esta formada por un conjunto de regiones homogéneas. Nuestra finalidad es encontrar y separar esas regiones. El tiempo requerido para segmentar una imagen es uno de los principales factores a considerar, ya que la mayoría de las bases de datos constan de miles de imágenes y un algoritmo que tarde mucho tiempo se vuelve inviable.

Para segmentar la imagen lo más rápido posible, nos conviene encontrar primero los segmentos más grandes de la imagen, ya que mientras más grandes sean las regiones que vayamos encontrando, más cerca estaremos de haber terminado de segmentar toda la imagen. Para encontrar los segmentos más grandes de la imagen, debemos revisar

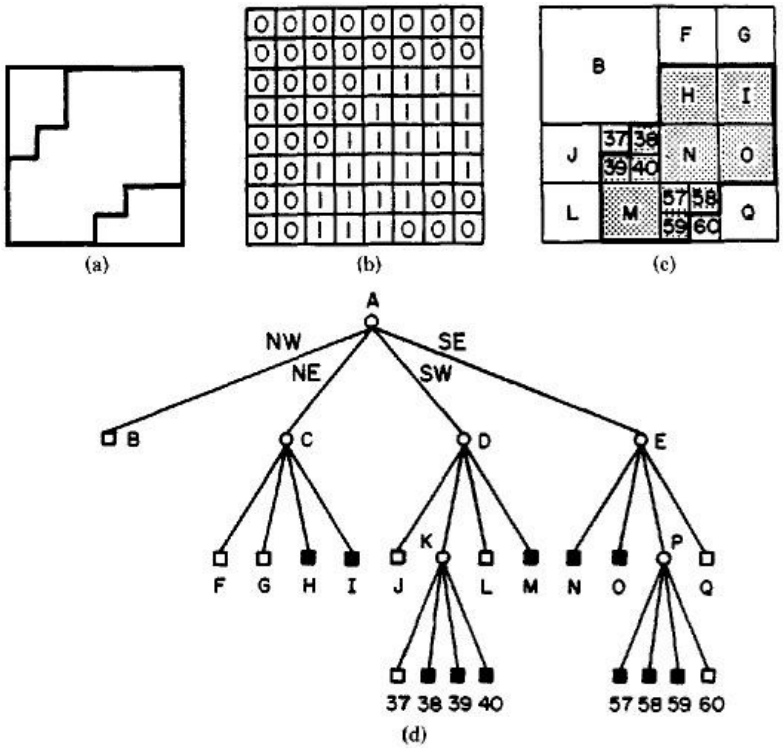


Figura 4.12: Ejemplo de un quadtree. a) Imagen original. b) Arreglo binario correspondiente a a). c) División recursiva en quadtree de las regiones homogéneas. d) Árbol de quadtree de a).

regiones grandes, lo más grande posible, y ver si forman un conjunto homogéneo. Si estas regiones grandes no forman un conjunto homogéneo, buscamos regiones más pequeñas en la región y volvemos a revisar si resultan en conjuntos homogéneos. Repetimos este proceso, disminuyendo cada vez más el tamaño de las regiones para poder encontrar regiones homogéneas. En el peor de los casos, terminaremos con regiones de 1x1 píxel, donde necesariamente serán homogéneas, todas del mismo color, pues contienen un único píxel. Conforme vayamos encontrando regiones homogéneas, marcamos el espacio que ocupan para no seguir revisando en esa área.

4.6.1 Recorrido en quadtree

En el proceso descrito anteriormente podemos observar claramente un proceso de descomposición recursiva. La forma en que proponemos esta descomposición evoca una estructura de *quadtrees*. Supongamos que deseamos segmentar la imagen *I*. El primer paso es dividir *I* en cuatro cuadrantes del mismo tamaño. Estos cuadrantes los obtenemos al trazar dos ejes perpendiculares, uno horizontal y otro vertical que se intersecten en el centro de la imagen. Llamaremos a estos cuadrantes R1, R2, R3 y R4 o noroeste, noreste, suroeste y sureste, respectivamente. R1 corresponde al cuadrante superior izquierdo, R2 al cuadrante superior derecho, R3 al cuadrante inferior izquierdo y R4 al cuadrante inferior derecho. La Figura 4.13 ilustra esta división.

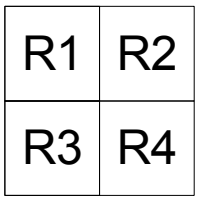


Figura 4.13: División de una imagen en cuatro cuadrantes, R1, R2, R3 y R4.

El primer paso consistirá en comparar estas cuatro regiones entre sí y unir aquellas que sean similares. Utilizaremos la relación de vecindad de cuatro vecinos para decidir que regiones deben ser comparadas. Elegimos la relación 4 vecinos para conservar la simplicidad del algoritmo y aumentar su velocidad. Sin embargo, esto no compromete la calidad del resultado debido a que en la etapa de "mejora de la segmentación (Sección 5.7) sí se toman en cuenta los vecinos diagonales. En este caso, R1 debe ser comparada con R2 y R3, R2 debe ser comparada con R1 y R4, y así sucesivamente. Para no duplicar comparaciones, las únicas comparaciones que haremos son:

1. R1 se compara con R2 y R3.
2. R2 se compara con R4.
3. R3 se compara con R4.

Aquellas regiones que resulten similares, serán unidas en una sola región. Supongamos que R2 y R4 resultan similares, la imagen de la Figura 4.13, después de la unión, se vería como lo ilustra la Figura 4.14. Podemos observar que, en este caso, el proceso de segmentación se ha reducido rápidamente a sólo la mitad izquierda de la imagen, es aquí donde radica el poder de nuestra propuesta y es este el beneficio de los enfoques de descomposición progresiva: *"descartar rápidamente grandes volúmenes de datos"* .

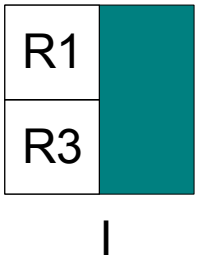


Figura 4.14: Resultado de unir las regiones R2 y R4 de la Figura 4.13

Dado que R1 y R3 no son similares, procederemos a repetir el proceso de división en cada una de ellas. Dividimos R1 en 1, 2, 3 y 4 y procedemos a comparar estos cuadrantes con sus cuadrantes vecinos tal como hicimos con la región R.

Antes de proseguir, notemos un detalle importante. Cuando dividimos R, comparamos sus cuatro cuadrantes entre sí y solamente entre sí. Sin embargo, en este nivel de la segmentación ya existen más posibilidades de comparación debido a los cuadrantes adyacentes a R1, que también tienen sus propios cuadrantes. En la Figura 4.16 *a)*

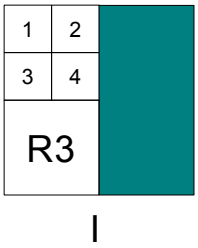


Figura 4.15: Siguiente paso en la segmentación: división recursiva de la región R1 de la Figura 4.14.

ilustramos los vecinos adicionales que deberán ser tomados en cuenta. Según sus relaciones 4-vecinos, debemos comparar:

1. 1 con 2 y 1 con 3.
2. 2 con 1 y 2 con a.
3. 3 con 4 y 3 con d.
4. 4 con c y 4 con b.

Con este enfoque cada uno de los cuadrantes de R1 puede ser comparado con otros cuadrantes en todas sus adyacencias posibles sin importar si pertenecen o no al mismo cuadrante. De esta manera, el cuadrante 4 tienen la posibilidad de buscar si tiene un cuadrante vecino inferior similar a él en el cuadrante R3 y también tiene la posibilidad de compararse con el cuadrante *b*, que ya pertenece a una región. En caso de que *4* y *c* resulten similares, formarían una nueva región. En caso de que *4* y *b* resulten similares, *4* se uniría a la región previamente formada que contiene *b*. La Figura 4.16 en *b)* y *c)* ilustra ambas posibilidades.

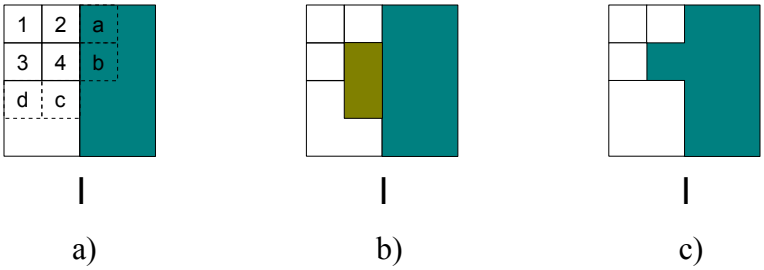


Figura 4.16: Segundo nivel de división de la imagen. a) El número de vecinos posibles para comparar ha crecido. b) Resultado después de unir 4 y c. c) Resultado después de unir 4 y b.

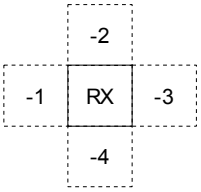


Figura 4.17: Diagrama de la convención utilizada para referirse a los extra-vecinos.

De manera similar a la Figura 4.13, ideamos una convención para referirnos a los vecinos extra-cuadrante que puede tener un subcuadrante. Esta convención se ilustra en la Figura 4.17. Debemos tomar en cuenta que nunca existen todos estos vecinos. En particular, basándonos en la Figura 4.13 y la Figura 4.17 podemos enunciar las siguientes afirmaciones:

1. R1 sólo puede tener los extra-vecinos -1 y -2.
2. R2 sólo puede tener los extra-vecinos -2 y -3.
3. R3 sólo puede tener los extra-vecinos -1 y -4.
4. R4 sólo puede tener los extra-vecinos -3 y -4.

Para aclarar estas afirmaciones, señalemos que, por ejemplo, R1 nunca puede tener el extra-vecino -3 debido a que su vecino de la derecha será siempre el cuadrante R2, el cual se genera al mismo tiempo que R1. La Figura 4.18 ilustra lo anterior.

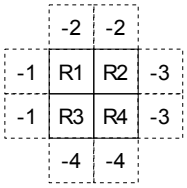


Figura 4.18: No todos los cuadrantes tienen todos los extra-vecinos.

Una vez que se ha dividido, comparado y probablemente unido alguno o todos los cuadrantes de la región R1, debemos proseguir con el proceso de la segmentación. Observemos nuevamente la Figura 4.14, después del análisis de R1 podemos tomar dos caminos. Nuestra primera opción es analizar la región R3. La segunda opción es analizar la región 1, cuadrante noroeste de R1 (Figura 4.15). En el primer caso, estaríamos recorriendo la imagen *en anchura*, en el segundo caso estaríamos recorriendo la imagen *en profundidad*. Para nuestro propósito de encontrar las regiones más grandes primero, debemos optar por un recorrido en anchura. Este tipo de recorrido, además, nos brinda la posibilidad de terminar la segmentación en cualquier momento y tener siempre una aproximación general al resultado final de la imagen.

El proceso de descomposición recursiva que ejecutamos sobre la imagen termina cuando una región llega a un tamaño que ya no puede dividirse, sea porque la región contiene un solo píxel o sea porque se ha llegado al tamaño límite de los cuadrantes solicitado por el usuario. Por ejemplo, si el tamaño mínimo límite de los cuadrantes es 3, significa que solamente podrán dividirse regiones cuyo tamaño mínimo sea de 6 x 6 píxeles. Todas aquellas regiones de tamaño menor, no podrán ser divididas nuevamente, por lo que cuando solamente queden regiones menores a este tamaño, el algoritmo terminará su ejecución. Este valor es introducido por el usuario. El valor por omisión es 2.

Ahora que hemos explicado la forma en que se va realizando el recorrido en *quadtree* de la imagen, pasemos a detallar la segunda parte del núcleo de nuestro algoritmo de

segmentación: el *operador de comparación*, que es el que evalúa la similitud de una región con otra y determina si pueden ser unidas.

4.6.2 Operador de Comparación de Regiones

Después del recorrido en *quadtrees,* el componente más importante para el éxito de nuestra propuesta es el operador de comparación de regiones. Este operador es el encargado de decidir cuando dos regiones deberán ser unidas. Para tomar esta decisión, el operador toma en cuenta dos elementos: los bordes existentes dentro de las regiones, tal como lo mencionamos en la Sección 4.3 y 4.4, y la distancia entre histogramas de las regiones. Para cada uno de ellos existen reglas y refinamientos especiales, los cuales describiremos a continuación.

A) Reglas para considerar los bordes de restricción

Supongamos que tenemos dos regiones A y B, el primer paso antes de compararlas consiste en analizar si dentro de ellas existen bordes de restricción (Sección 4.3), y en caso afirmativo, contar cuántos hay.

Para determinar el número de bordes que hay en un cuadrante C, es necesario contar cuantos píxeles de la *imagen de restricciones* (Figura 4.8) se encuentran dentro del área que abarca C. El número total de píxeles encontrados será expresado por las palabras **ninguno**, **algunos** y **muchos**, obedeciendo las siguientes reglas:

1. 0 píxeles encontrados: **ninguno**.
2. 1 hasta 50% del total de píxeles del cuadrante: **algunos**.
3. Más del 50% del total de píxeles del cuadrante: **muchos**.

Una vez que hemos contado el número de bordes de los cuadrantes A y B, procederemos a construir y comparar sus histogramas sí y solo sí:

1. A y B no tienen *ningún* borde de restricción o
2. A y B tienen, ambos, *muchos* bordes de restricción.

El punto 1 se justifica debido a que sí no existe ningún borde dentro del cuadrante, significa que todo el cuadrante pertenece a una única región. Entonces, si tanto A, como B pertenecen a una única región cada uno y, debido a la naturaleza de nuestro proceso, son adyacentes, existen altas probabilidades de que A y B pertenezcan a la misma región. Por lo tanto, es posible pasar al siguiente paso.

El punto 2 se justifica debido a que cuando hay muchos bordes (cambios de intensidad) dentro de una región, lo más probable es que dichos bordes no indiquen un límite de región, sino el patrón de una textura (Figura 4.9). Es por esto que también cuando A y B tienen muchos bordes, continuamos revisando si podemos unir estas regiones.

No así cuando sólo uno de ellos tiene muchos bordes y el otro pocos. Es claro que en este caso los cuadrantes corresponden a regiones diferentes y no deben ser unidos.

B) Reglas para comparar histogramas

Cuando el resultado del análisis de los bordes de los cuadrantes A y B es satisfactorio (cero bordes o mucho bordes, en ambos), procedemos a construir y comparar sus histogramas.

El histograma de una región es un mapeo que cuenta el número de observaciones que caen entre distintas categorías disjuntas y se representa por la siguiente función:

$$n = \sum_{i=1}^{k} m_i .$$

Ecuación 4.1

n es el histograma, k es el número de categorías y m_i es el número de elementos en cada categoría. Tratándose de imágenes, el histograma de una región cuenta el número de píxeles que existen de cada color dentro de esa región. Debido a esto, si la imagen tiene muchos colores tendrá un histograma muy grande. En una imagen que utiliza el modelo de color RGB, el histograma podría tener hasta 16,777,216 índices, pues tal es el número de colores posibles que podemos representar en dicho modelo, esto hace que la construcción del histograma consuma mucha memoria. Mas aún, para poder comparar este histograma con otro, necesitamos igual número de operaciones de comparación, esto hace que la comparación del histograma con otro, consuma mucho tiempo.

En este trabajo, utilizamos una discretización de 2 bits por cada componente (Sección 4.5), esto nos da un total de 64 colores diferentes y por lo tanto un histograma con sólo 64 índices, el cual no ocupa mucho espacio y es rápido de comparar. En la Figura 4.19 observamos la diferencia entre el histograma que se obtiene de la imagen original (a y b) y el histograma que se obtiene de la versión discretizada de la imagen (c y d). Esta imagen ilustra claramente toda la información que omitimos y por lo tanto, todo el esfuerzo que ahorramos. Es importante recalcar que a pesar de esta pérdida de información, el proceso de segmentación da resultados muy favorables. Con esto, nuestro algoritmo refleja su gran austeridad y simplificación en todos los aspectos del proceso de la segmentación.

La comparación entre dos histogramas es posible debido a que cada histograma puede verse como un vector de N-dimensiones (en este caso N = 64). De aquí que, para comparar dos histogramas lo único que necesitamos es obtener la distancia entre ambos vectores. Sin embargo, existen algunos detalles que debemos tratar antes de proceder a comparar los histogramas.

Observemos el histograma de la Figura 4.19 d). Sus barras grandes nos indican la presencia alta de un color mientras que las barras pequeñas nos indican una presencia baja. En esta tesis, consideramos que las barras pequeñas no ayudan a la comparación con otro histograma, al contrario, esa pequeña presencia de un cierto color puede evitar que dos histogramas muy similares acaben siendo diagnosticados como diferentes; por eso llamamos a las barras pequeñas del histograma: *ruido*. Por lo tanto, antes de comparar dos histogramas, es necesario eliminar las barras pequeñas que indican una presencia baja de sus respectivos colores en la región. El valor límite que utilizamos para considerar como *ruido* a un color en el histograma corresponde al 1% del tamaño de la región. Es decir, eliminamos todas aquellas barras que correspondan a colores que no ocupan más del 1% del área de la cual se calculo el histograma. Durante la fase de pruebas descubrimos que este pequeño arreglo en los histogramas mejora el resultado final de la segmentación.

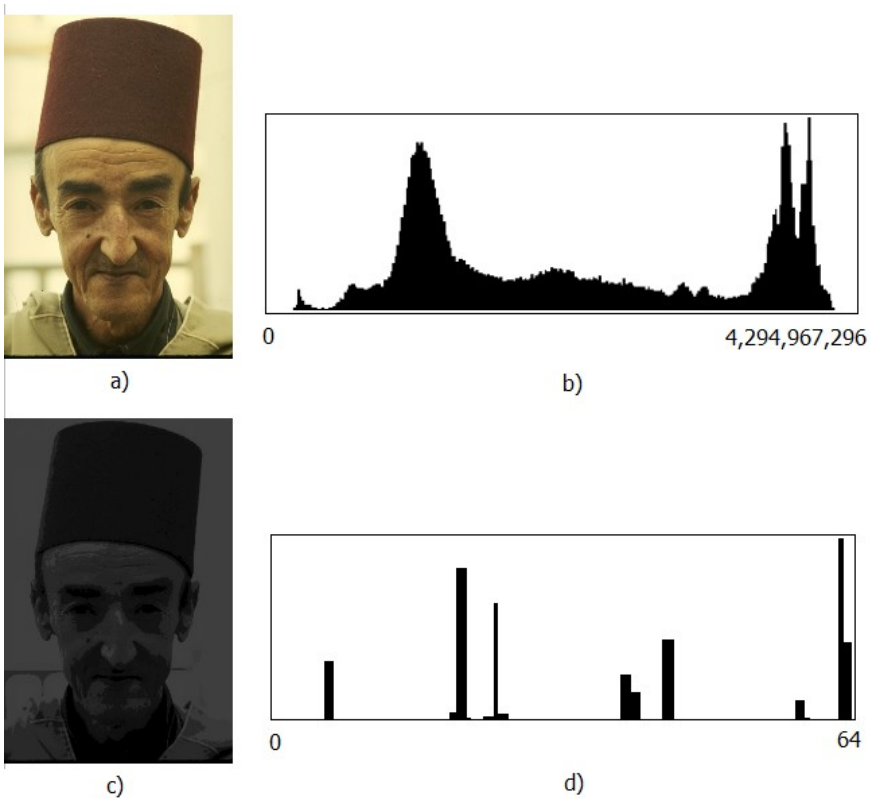
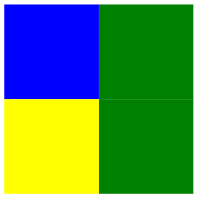


Figura 4.19: a) Imagen original, b) histograma correspondiente a la imagen a), c) imagen discretizada y d) histograma correspondiente a la imagen c).

El segundo arreglo que debemos hacer en los histogramas se conoce como: *normalización*. Este proceso consiste en cambiar los valores absolutos que cuentan el número de apariciones de cada color por valores relativos que indiquen el porcentaje del área ocupada por el color. Para normalizar un histograma calculamos el área de la región que representa y dividimos cada uno de sus valores entre el valor del área. La Figura 4.20 muestra un ejemplo de este cambio. Los valores que antes eran absolutos ahora son expresados en porcentajes. De esta manera es mucho más fácil comparar histogramas

correspondientes a regiones de distinto tamaño. En nuestro caso, siempre comparamos regiones del mismo tamaño, por lo que la normalización de los histogramas no sería necesaria. Sin embargo, normalizamos los histogramas debido a que es más fácil operar con números pequeños y un histograma normalizado siempre tiene valores entre 0 y 1 para cada una de sus barras. De otra forma, tendríamos que operar con valores muy grandes y corremos el riesgo de desbordar los límites permitidos por los lenguajes de programación al momento de implementar nuestro algoritmo.



Área: 65536px
Histograma[azul: 16384px, amarillo: 16384px, verde: 32768px]
Histograma Normalizado [azul: 25%, amarillo: 25%, verde: 50%]

Figura 4.20: Histograma e Histograma Normalizado.

Una vez que hemos eliminado el ruido y normalizado los histogramas, procedemos a compararlos. Debido a la similitud con vectores n-dimensionales, como lo mencionamos anteriormente, podemos utilizar para calcular la distancia entre dos histogramas la *norma euclidiana* que se muestra en la Ecuación 4.2 . Cuando la distancia obtenida está por debajo de un umbral preestablecido, el operador de comparación indica que las regiones sí pueden ser unidas. En caso contrario, el operador de comparación indica que A y B no pueden ser unidas y deben volver a dividirse. El umbral para esta diferencia entre histogramas es introducido por el usuario bajo el concepto de *umbral de homogeneidad*. El valor por omisión para este parámetro es 0.50.

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad \textit{Ecuación 4.2}$$

4.6.3 Unión de cuadrantes

Una vez que el operador de comparación ha indicado que dos cuadrantes pueden unirse, debemos realizar una serie de verificaciones antes de unirlos. El Algoritmo 4.1 muestra este punto:

```
1 Revisar si alguno o ambos cuadrantes ya fueron etiquetados
  previamente.
  1.1 En caso de que ninguno de ellos tenga etiqueta, generamos una
      nueva etiqueta y se la asignamos a ambos.
  1.2 En caso de que uno sólo tenga etiqueta, propagamos esa
      etiqueta sobre aquel que no tiene asignada ninguna.
  1.3 En caso de que ambos posean una etiqueta previa:
    1.3.1 Revisar si es la misma etiqueta
```

```
1.3.1.1 En caso de que sea la misma etiqueta no hacemos
        nada, ambos cuadrantes ya pertenecen a la misma
        región.
1.3.1.2 En caso de tener etiquetas diferentes, debemos
        propagar la etiqueta que fue generada primero.
1.3.1.3 Mantener una lista de todas las etiquetas que han
        sido sustituidas por otra.
```

Algoritmo 4.1 Pasos para unir cuadrantes.

La Figura 4.21 ilustra el proceso descrito en esta sección. Podemos observar el recorrido que realiza el *quadtree* sobre la imagen y cómo se va construyendo la segmentación.

Vimos como recorrer la imagen utilizando técnicas de *quadtrees* para obtener, comparar y unir regiones lo más grande posible, reduciendo el tamaño de la región iterativamente hasta terminar de segmentar la imagen. Sin embargo, este proceso nos arrojará una imagen con muchos segmentos, debido a la imagen de restricciones, al umbral de homogeneidad y a la forma en que se unen las regiones (daremos más detalles en la siguiente sección). Recordemos además que el algoritmo debe devolver únicamente objetos que ocupen más de un porcentaje de la imagen especificado por el usuario. Por todos esto, es necesario realizar unos pasos más antes de tener lista la segmentación final.

4.6.4 Verificación del reemplazo de etiquetas

A continuación describimos los pasos que se siguen para afinar la segmentación. Con la finalidad de mejorar la legibilidad de este texto, a lo largo del del capítulo se muestran únicamente las primeras líneas de cada listado. Sin embargo, la versión completa de los listados mencionados en este capítulo pueden encontrarse en el Anexo C.

En el punto 1.3.1.3 del Algoritmo 4.1 mencionamos que el algoritmo debe llevar una lista de todas las etiquetas que han sido reemplazadas por otra. Procederemos a explicar con mayor detalle este punto. Primero, retomemos la imagen de la cuarta fila y tercera columna de la Figura 4.21. Este resultado contiene la clave de lo que queremos explicar. Notemos que tanto el rostro (amarillo), la mejilla derecha del hombre (gris), el labio superior derecho del hombre (azul vivo) y la barbilla (azul opaco) son regiones que son similares entre sí, son adyacentes y además no hay ningún borde de restricción dentro de ellas. Surge la pregunta: ¿por qué no se han unido estas regiones en una sola? La respuesta es sencilla. Las regiones nacieron de forma aislada, cuando crecieron lo suficiente y se hicieron adyacentes llegó el momento de compararse entre sí. El comparador de regiones (Sección 4.6.2) las reconoció como similares y procedió a unir dichos CUADRANTES. Sin embargo, solamente se unieron los cuadrantes que en ese momento se estaban comparando y no los segmentos completos a los que pertenecían.

Por esta razón se lleva el registro de las etiquetas reemplazadas mencionado en la Sección 4.6.3. Gracias a esta lista, podemos conocer todos los puntos de la imagen donde ocurrió

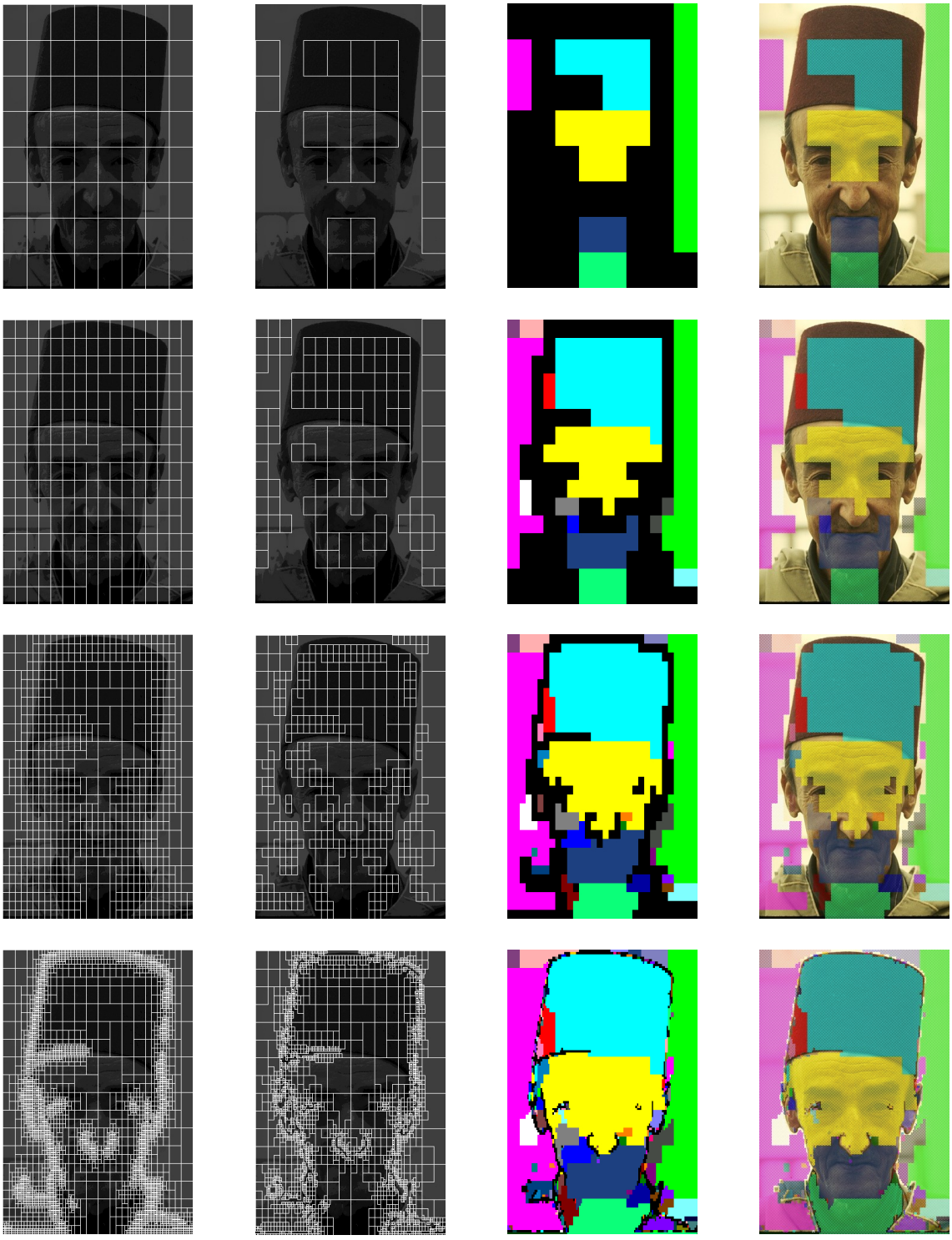


Figura 4.21: Recorrido de la imagen en quadtree para generar la segmentación. En la fila 1 observamos el proceso cuando el tamaño de los cuadrantes es 32 píxeles, en la fila 2 el tamaño es 16, en la fila 3 es 4 y en la fila 4 es 2. a) muestra todos los cuadrantes generados. b) muestra sólo los cuadrantes seleccionados para ser unidos. c) muestra las regiones que se han formado hasta el momento. d) muestra una superposición del resultado actual sobre la imagen original.

esta situación y revisar si el reemplazo de etiqueta en un cuadrante debe extenderse a todo el segmento del que dicho cuadrante forma parte.

El Listado 4.1 muestra la lista de etiquetas reemplazadas que se generó durante la segmentación de la imagen de la Figura 4.21.

```
La etiqueta (región)  7 fue sustituida por 1
La etiqueta (región) 12 fue sustituida por 5
La etiqueta (región) 14 fue sustituida por 4
La etiqueta (región) 13 fue sustituida por 5
La etiqueta (región) 13 fue sustituida por 5
La etiqueta (región)  9 fue sustituida por 2
La etiqueta (región) 16 fue sustituida por 2
```

Listado 4.1: Reemplazos de etiquetas realizados durante la etapa de recorrido en quadtree.

Lo primero que haremos es obtener una relación de todas las etiquetas que fueron reemplazadas por una misma etiqueta. Por ejemplo, podemos ver que la etiqueta 1 reemplazó a las etiquetas 7, 18, 10 y 35, entre otras, entonces crearemos una lista que indique todas las etiquetas reemplazadas por la etiqueta 1. La relación completa de estos reemplazos se muestra en el Listado 6.2 del Anexo D.

```
El segmento 1 se unió con los segmentos: [ 7 18 10 24 26 27 35 25 61 63 8 146 158 159
58 ]
El segmento 2 se unió con los segmentos: [ 9 16 36 37 38 3 80 86 111 115 ]
El segmento 3 se unió con los segmentos: [ 11 19 53 54 55 29 65 130 138 143 142 66 ]
El segmento 4 se unió con los segmentos: [ 14 47 21 51 195 ]
El segmento 5 se unió con los segmentos: [ 12 13 30 56 6 72 144 28 176 ]
El segmento 6 se unió con los segmentos: [ 28 31 155 78 ]
El segmento 9 se unió con los segmentos: [ 17 39 ]
```

Listado 4.2: Relación de todas las etiquetas que fueron reemplazadas por una misma etiqueta.

El hecho de que estas etiquetas hayan sido reemplazadas significa, que los cuadrantes que se comparaban en ese momento, fueron diagnosticados suficientemente similares por el operador de comparación de regiones. Sin embargo, esto no es suficiente para que podamos garantizar que la totalidad de las regiones a las cuales pertenece cada uno de estos cuadrantes sean igualmente similares. Por lo tanto, el primer paso de nuestra mejora consiste en revisar si *todo* el segmento correspondiente a una etiqueta reemplazada es similar a *todo* el segmento correspondiente a la etiqueta reemplazante. Por ejemplo, a través Listado 6.2 sabemos que algunos cuadrantes de las regiones con etiquetas 55, 140 y 12 resultaron similares a algunos cuadrantes de la región con etiqueta 11, ahora toca revisar si *toda* la región 55 es similar a *toda* la región 11, si *toda* la región 140 es similar a *toda* la región 11 y si *toda* la región 12 es igual a *toda* la región 11. En caso de que alguna de estas comparaciones resulte verdadera procederemos reemplazar la etiqueta de todo el segmento correspondiente con la etiqueta 11.

Para evaluar si dos segmentos son similares, utilizamos el mismo operador de comparación que se utilizó durante la etapa de recorrido de *quadtree*. En esta ocasión, las regiones a comparar serán de distinto tamaño, pero no tendremos ninguna dificultad para compararlas debido a la normalización de histogramas que mencionamos en la Sección 4.6.2.

El Listado 4.3 muestra la relación de los segmentos completamente similares entre sí después de comparar todos los segmentos del Listado 6.2 con sus respectivos segmentos adyacentes. Una vez que se tiene esta lista es posible realizar el reemplazo de etiquetas. Al momento de hacer el reemplazo debemos tomar en cuenta la transitividad de etiquetas. Por transitividad de etiquetas entendemos cuando una etiqueta X debe ser reemplazada por una etiqueta Y que a su vez será reemplazada por Z. En este caso, no tiene sentido realizar 2 reemplazos sino que es mejor realizar únicamente uno: reemplazar X por Z. Por ejemplo, en el Listado 6.3 podemos observar que la etiqueta 62 será reemplazada por la etiqueta 28, y ésta a su vez será reemplazada por la etiqueta 6, entonces reemplazaremos desde el principio la etiqueta 62 con la etiqueta 6. La Figura 4.22 muestra el resultado de aplicar las mejoras mencionadas en esta sección al resultado de segmentación de la imagen de la Figura 4.3. Las mejoras más visibles ocurrieron en la región de la barbilla y del cuello.

```
El segmento  3 es similar a los segmentos: [ 11 19 29 ]
El segmento  5 es similar a los segmentos: [ 12 ]
El segmento  6 es similar a los segmentos: [ 28 31 ]
El segmento 20 es similar a los segmentos: [ 45 ]
El segmento 28 es similar a los segmentos: [ 62 ]
El segmento 73 es similar a los segmentos: [ 75 ]
```

Listado 4.3: Resultado de revisar que segmentos son completamente similares obtenida a partir del Listado 4.2.

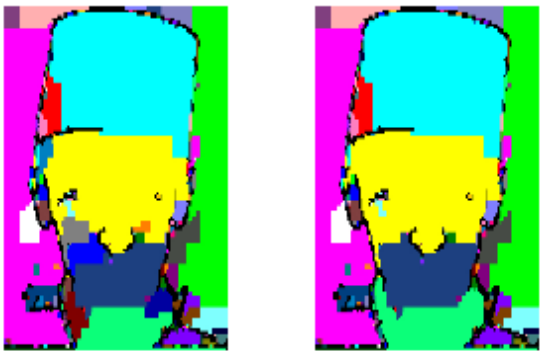


Figura 4.22: Resultado de aplicar el primer post-procesamiento a la segmentación obtenida con el recorrido en quadtree. a) resultado original, b) resultado después del post-proceso.

4.6.5 Eliminación de regiones pequeñas

Recordemos que una de las características de nuestro algoritmo consiste en poder indicar el *tamaño mínimo* de los objetos que deseamos encontrar. Sin embargo, hasta el momento nuestro proceso de segmentación no ha hecho nada por cumplir con este requisito. Los resultados que obtenemos son satisfactorios pero cuentan con segmentos muy pequeños que normalmente no aportan suficiente información para generar una anotación propia. El siguiente pos-proceso de la segmentación consistirá en deshacernos de todos los segmentos pequeños uniéndolos a segmentos adyacentes similares mayores. Para saber que segmentos debemos eliminar es necesario definir el umbral que utilizaremos para

distinguir segmentos pequeños de segmentos grandes. Este valor es proporcionado por el usuario por medio del parámetro: *tamaño mínimo de objetos*. El mejor valor por omisión para este parámetro, encontrado durante la etapa de pruebas, es 0.0125, lo cual significa que deseamos encontrar objetos que utilicen al menos el 1.25% del área total de la imagen. Para saber que segmentos debemos eliminar en la segmentación final, simplemente calculamos el área de cada segmento encontrado y revisamos sí el área que ocupa es menor al umbral especificado.

Una vez que se conocen todos los segmentos que serán eliminados, el siguiente paso consiste en revisar cuales son sus segmentos adyacentes para decidir a cual de ellos puede unirse. Debido a la extrema irregularidad que puede presentar cada uno de los segmentos resultantes de una segmentación, la única forma de conocer todos sus segmentos adyacentes es calculando el perímetro y después, evaluar el valor de cada uno de los píxeles que son adyacentes a ese perímetro. Este proceso es significativamente costoso por lo que nos dimos a la tarea de simplificarlo lo más posible. Durante la simplificación que proponemos dejamos de detectar algunos vecinos pero agilizamos mucho el proceso de búsqueda de los mismos, manteniendo la simplicidad y velocidad del algoritmo. En el cuadro del Algoritmo 2 detallamos los pases que se siguen para para esto.

Muchas veces sucede que uno o más segmentos pequeños se unen entre sí y forman un segmento de tamaño mayor que cumple con las dimensiones requeridas para no ser eliminado. Sin embargo, también puede suceder lo contrario: segmentos pequeños que a pesar de haber sido unidos con otros siguen sin tener el tamaño suficiente para ser incluidos en el resultado final de la segmentación. Debido a esto, el proceso de refinamiento que proponemos debe ser realizado iterativamente hasta terminar con todos los segmentos que no cumplan con las dimensiones especificadas por el usuario. En general, dos pasadas son suficientes para obtener una resultado sin segmentos pequeños.

A continuación, examinaremos detalladamente este proceso para el resultado de Figura 4.22 b). El Listado 4.4 muestra detalladamente el resultado de analizar cada uno de sus segmentos (paso 1 del Algoritmo 4.2). Dado que las medidas de la imagen son 321 de ancho por 481 píxeles de alto, el 1.25% del área total de la imagen es: 1930 píxeles. Por lo tanto debemos eliminar todos aquellos segmentos que no tengan un área de al menos este tamaño (paso 2 del Algoritmo 4.2). El Listado 4.5 muestra los segmentos que deberán ser eliminados.

1. Recorrer toda la imagen una sola vez. Durante este recorrido obtener el área, las cuatro esquinas mas externas y el histograma de cada uno de los segmentos. Las cuatro esquinas más externas de un segmento se entienden como sigue:
    1. El punto más arriba y más a la izquierda del segmento.
    2. El punto más arriba y más a la derecha del segmento.
    3. El punto más abajo y más a la izquierda del segmento.
    4. El punto más abajo y más a la derecha del segmento.

2. En base al área calculada y al umbral especificado por el usuario procedemos a generar la lista de todos los segmentos que no cumplan con el área requerida.
3. Revisar la etiqueta de los segmentos que son adyacentes a cada una de las 4 esquinas de cada segmento. Dado que conocemos el punto (x, y) de cada esquina, acceder a sus vecinos adyacentes es trivial. En este punto sí tomamos en cuenta a los vecinos diagonales (Sección 5.6.1).
4. Tomamos los segmentos agregados la lista del punto 2 y para cada uno de ellos:
   1. Comparamos su histograma con el histograma de todos los segmentos que sean adyacentes a sus esquinas.
   2. Haremos una unión con el segmento con el cual la diferencia entre histogramas sea la menor. Cabe mencionar que aún no los uniremos, por ahora, sólo almacenaremos esta información en una lista de reemplazo de etiquetas. La lista debe indicar que la etiqueta X correspondiente al segmento Xi será sustituida por la etiqueta Y correspondiente al segmento Xj.
5. Tomamos la lista de reemplazos creada en el punto anterior, evaluamos la transitividad de reemplazos y resolvemos los ciclos que pudieran existir. [Sección 4.7.1].
6. Una vez que hemos eliminado la transitividad de la lista de reemplazos, recorremos la imagen que contiene el resultado actual de la segmentación y efectuamos el reemplazado de etiqueta para cada uno los píxeles que pertenecen a los segmentos que deseamos eliminar.

Algoritmo 4.2 Pasos para unir segmentos pequeños al segmento grande más cercano.

```
Objeto 1:       área = 8,      e1(304,466),   e2( 305,466),  e3( 305,469),  e4( 304,469]
Objeto 2:       área = 12,     e1(301,459),   e2( 303,459),  e3( 303,462),  e4( 301,462]
Objeto 3:       área = 20,     e1(289,459),   e2( 293,459),  e3( 293,462),  e4( 289,462]
Objeto 4:       área = 32,     e1(271,474),   e2( 278,474),  e3( 278,477),  e4( 271,477]
Objeto 5:       área = 20,     e1(264,474),   e2( 268,474),  e3( 268,477),  e4( 264,477]
Objeto 6:       área = 32,     e1(246,474),   e2( 253,474),  e3( 253,477),  e4( 246,477]
Objeto 7:       área = 15,     e1(261,448),   e2( 265,448),  e3( 265,450),  e4( 261,450]
```

Listado 4.4: Etiqueta, área y extremos de cada segmento de la imagen de la Figura 4.21

```
Objeto 1:       área = 8
Objeto 2:       área = 12
Objeto 3:       área = 20
Objeto 4:       área = 32
Objeto 5:       área = 20
Objeto 6:       área = 32
Objeto 7:       área = 15
Objeto 8:       área = 20
```

Listado 4.5: Objetos que serán eliminados debido a que su área es menor al 1.25% de la imagen.

Según los puntos 3 y 4 del algoritmo uniremos cada uno de estos segmentos al segmento adyacente más parecido. Para evaluar la similitud entre segmentos utilizaremos el mismo operador de comparación que utilizamos para comparar cuadrantes en la Sección 4.6. Sin embargo, esta vez no utilizaremos un umbral que defina cuando unirlos y cuando no, esta vez simplemente mediremos la distancia L2 entre cada segmento y todos sus segmentos

adyacentes y lo uniremos con el segmento con el que tenga la distancia menor. El Cuadro 4.1 nos muestra un pequeño ejemplo de esto.

```
Vecinos del Objeto con Etiqueta 8: 158, 7, 9
Comparando histograma de 8 con 158     d2 -> 0.8851211987917359
Comparando histograma de 8 con 7       d2 -> 0.7382411530116699
Comparando histograma de 8 con 9       d2 -> 0.7919736247011148
Reemplazo para 8: 7

Vecinos del Objeto con Etiqueta 9: 158, 8, 7, 129
Comparando histograma de 9 con 158     d2 -> 0.5927982470264292
Comparando histograma de 9 con 8       d2 -> 0.7919736247011148
Comparando histograma de 9 con 7       d2 -> 0.5416025603090641
Comparando histograma de 9 con 129     d2 -> 0.6864025259484361
Reemplazo para 9: 7
```

Cuadro 4.1: Ejemplo de evaluación del segmento más similar utilizando la norma euclidiana de la Figura 4.21.

Una vez que tenemos la lista de reemplazos de etiquetas es importante revisar que ésta no contenga ciclos, paso 5 del algoritmo. Es decir que no suceda que X sea reemplazada por Y y Y, a su vez, sufra una serie de reemplazos que terminen nuevamente en X. Para eliminar los ciclos de una lista de reemplazos, realizamos los siguientes pasos.

1. Tomar cada una de las etiquetas Ei de la lista y su respectiva etiqueta de reemplazo ERi.
2. Buscamos todas las etiquetas Ej, j <> i, de la lista para las cuales Ei es su respectiva ERj y hacemos que su nueva ERj sea ERi.

En el Listado 4.6 mostramos como queda la lista de reemplazos antes y después de la eliminación de ciclos. La lista está en la forma Ei: ERi. ERi = -1 significa que Ei no será reemplazada. Una vez eliminados los ciclos de la lista, procedemos a realizar el reemplazo de etiquetas, paso 6 del algoritmo. La Figura 4.23 ilustra el resultado después de eliminar todos los segmentos del Listado 4.5. El número de regiones ahora es 35.

```
0: -1, 1: 124, 2: 124, 3: 168, 4: 174, 5: 129, 6: 129, 7: 9, 8: 7, 9: 7, 10: 18, 11: 13, 12: 13, 13: 176, 14: 13, 15: 129, 16: 0, 17: 174, 18: 10, 19: 174, 20: 131, 21: 22, 22: 21, 23: 169, 24: 176, 25: 177, 26: 136, 27: 175, 28: 177, 29: 134, 30: 177, 31: 30, 32: 161, 33: 177, 34: 177, 35: 36, 36: 137, 37: 179, 38: 175, 39: 40, 40: 39, 41: 0, 42: 56, 43: 42, 44: 174, 45: 138, 46: 179, 47: 179, 48: 179, 49: 179, 50: 179,

0: -1, 1: 124, 2: 124, 3: 168, 4: 174, 5: 129, 6: 129, 7: 9, 8: 9, 9: 9, 10: 18, 11: 176, 12: 176, 13: 176, 14: 176, 15: 129, 16: 0, 17: 174, 18: 18, 19: 174, 20: 174, 21: 22, 22: 22, 23: 160, 24: 176, 25: 177, 26: 136, 27: 175, 28: 177, 29: 134, 30: 177, 31: 177, 32: 161, 33: 177, 34: 177, 35: 137, 36: 137, 37: 179, 38: 175, 39: 40, 40: 40, 41: 0, 42: 56, 43: 56, 44: 174, 45: 138, 46: 179, 47: 179, 48: 179, 49: 179, 50:
```

Listado 4.6: Izquierda.- Lista de reemplazos con ciclos. Derecha.- Lista con ciclos corregidos (rojo).

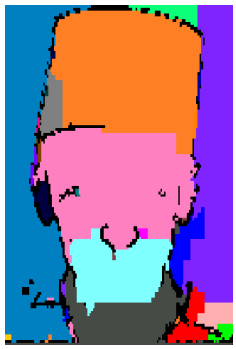


Figura 4.23: Resultado de la segmentación después de realizar la primera eliminación de segmentos pequeños.

Podemos observar que a pesar de que muchos segmentos fueron eliminados, aún quedan unos cuantos segmentos pequeños, sobre todo del lado inferior izquierdo de la imagen. Para eliminarlos es necesario volver a ejecutar el algoritmo descrito en esta sección. La Figura 4.24 muestra el resultado de segmentación después de esta segunda ejecución. El número final de segmentos encontrados en la imagen es 9. La Figura 4.25 se muestra como ayuda para una mejor valoración del resultado final.

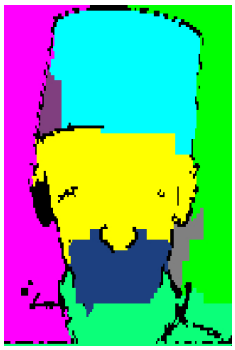


Figura 4.24: Imagen después de la segunda eliminación de segmentos pequeños. Resultado Final.

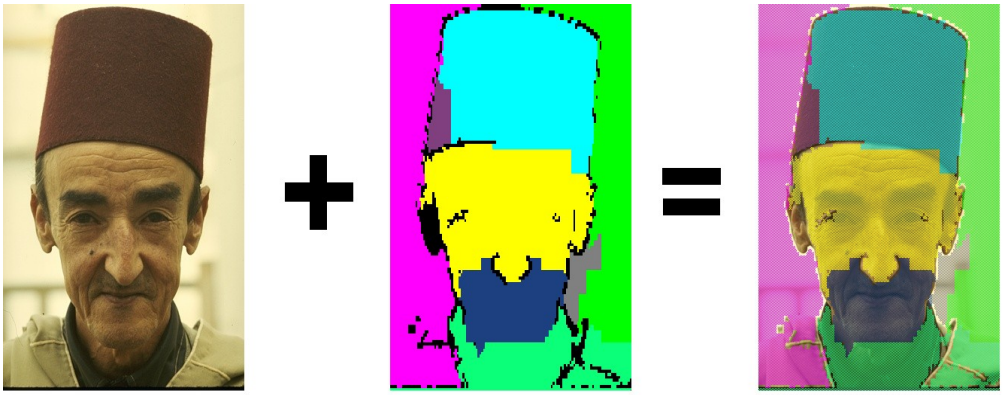


Figura 4.25: Resultado final sobrepuesto en la imagen original.

4.7 Análisis de complejidad del algoritmo

Después de analizar la complejidad de cada etapa de nuestro algoritmo tenemos los siguientes resultados.

Sea $N = w * h$ con w = ancho de la imagen y h = alto de la imagen.

Etapa 1, obtención de gradiente.

$$O(N) = 7\mathrm{N}$$

Etapa 2, crear imagen de restricciones.

$$O(N) = 4\mathrm{N}$$

Etapa 3, discretizar imagen.

$$O(N) = N$$

Etapa 4, Recorrido en *quadtree*.

$$O(N) = 2^{(log4(N)-1)}((4\mathrm{N}+C)+2\mathrm{N})$$ C, número de operaciones para dividir un *quadtree*.

Etapa 5, Mejorar la segmentación

Mejora 1

$$O(N) = 2\mathrm{K} + K^2 + N$$ K, número de segmentos encontrados

Mejora 2

$$O(N) = 2(N + 2\mathrm{L} + 3\mathrm{N} + 4\mathrm{M})$$ L, número de etiquetas producidas
M, número de segmentos después de la Mejora 1

Sumando todas las etapas, el orden total nos queda como sigue:

$$O(N) = 21\mathrm{N} + 2^{(log4(N)-1)}(6\mathrm{N} + 4\mathrm{C}) + K$$

Resumiendo y simplificando el orden del algoritmo nos queda:

$$O(N)=N2^{(log4(N))}$$

$$O(N)=N2^{(\frac{1}{2}log2(N))}$$

$$O(N)=N\sqrt{2\log2(N)}$$

$$O(N)=N\sqrt{N}$$

$$O(N)=N^{\frac{3}{2}}$$

Este análisis de complejidad nos muestra que contamos con un algoritmo sencillo capaz de procesar imágenes de gran tamaño sin elevar mucho su tiempo de respuesta. Esta es una ventaja sustancial.

4.8 Conclusiones

En este capítulo describimos de forma detallada el proceso que sigue nuestro algoritmo para segmentar una imagen. Vimos que, a pesar, de que el recorrido en *quadtree* es el núcleo de nuestra segmentación, son necesarios algunos pasos previos y algunos pasos posteriores. Son, precisamente, estos pasos adicionales, en conjunto, los que hacen que el algoritmo dé buenos resultados. El pre-procesamiento es importante para preparar la imagen y determinar las restricciones que serán tomadas en cuenta. El post-procesamiento es necesario para afinar los resultados que se obtienen en primera instancia. El análisis de complejidad reveló que contamos con un algoritmo muy competitivo ya que puede segmentar imágenes muy grandes sin aumentar relevantemente su tiempo de ejecución. A continuación realizaremos una evaluación experimental del algoritmo, revisaremos su desempeño en general (con distintos tipos de imágenes) analizaremos la utilidad de sus resultados y lo compararemos con otros algoritmos de segmentación.

Capítulo 5

Pruebas y Resultados Experimentales

En este capítulo analizaremos con detalle los resultados de nuestro segmentador. Mencionaremos la plataforma de desarrollo y pruebas del algoritmo. Comentaremos nuestros conjuntos de imágenes de prueba. Ilustraremos y explicaremos ejemplos de resultados satisfactorios y no satisfactorios en cada una de sus etapas de segmentación de imágenes seleccionadas de cada uno de los conjuntos de prueba. Mencionaremos los tiempos de ejecución. Compararemos los resultados de segmentación de nuestro algoritmo con los de *normalized cuts* y *Grid Segmentation*. Finalmente, utilizaremos los resultados de cada uno de los tres algoritmos para producir anotaciones automáticamente y veremos con cuál de ellos se obtienen anotaciones más precisas.

5.1 Ambiente de trabajo

Antes de comentar los resultados es importante mencionar que el algoritmo fue desarrollado en java 1.6 utilizando el entorno de desarrollo Netbeans de Sun Microsystems. Todas las pruebas fueron ejecutadas en una computadora con procesador AMD Turion de 64 bits a 1.6 GHz con doble núcleo y 1GB de RAM.

En el paquete desarrollado existe una clase principal llamada *QuadTreeSegmentation* la cual segmenta una sola imagen y una clase llamada *BatchSegmentator* la cual segmenta todas las imágenes en formato JPEG de una carpeta[1]. La forma en que se invocan estas clases desde la línea de comandos se detalla en el Anexo B.

5.2 Conjuntos de pruebas

El desarrollo del algoritmo se llevo a cabo utilizando como base inicial un conjunto heterogéneo de 18 imágenes en color seleccionadas tomando en cuenta su aparente facilidad o complejidad. Llamaremos a este conjunto de imágenes conjunto A, el cual puede observarse en la Figura 5.1. En él predominan las imágenes de paisajes con regiones homogéneas grandes (cielo, vegetación) debido a que nuestro algoritmo esta especialmente orientado a segmentar una imagen a través de la identificación y separación de este tipo de regiones. Sin embargo, el conjunto también contiene imágenes con pocas y pequeñas regiones homogéneas, difícilmente identificables y saturadas de contornos. Tal es el caso de las imágenes 12, 13, y 14. Éstas imágenes se incluyeron para cuidar que nuestro algoritmo fuera capaz de dar resultados satisfactorios en imágenes que no pertenecen a su dominio principal, por ser de naturaleza no homogénea. La imagen 15 se incluyó para calibrar el algoritmo de tal forma que pudiera identificar como una sola región texturas compuestas por regiones contrastantes, como es el caso de la piel del tigre. La imagen 16 se incluyó por considerarla un ejemplo interesante en el artículo [Song Chun Zhu, 1996] dada su composición de texturas y objetos superpuestos. Algunas imágenes contienen una marca de agua lo cual se considero conveniente para observar hasta que punto ésta marca podía afectar los resultados de nuestro algoritmo. El tamaño promedio de todas éstas imágenes es de 300 x 250 píxeles aunque los tamaños varían desde 116 x 261 hasta 800 x 433. En el Anexo C se lista el tamaño de cada imagen y el tiempo que tomó segmentarla.

Una vez que el algoritmo logró un desempeño satisfactorio en el conjunto anteriormente citado se busco perfeccionarlo utilizando conjuntos de prueba con otras características. Se eligieron 3 conjuntos de pruebas adicionales. El primero de ellos, el conjunto B, fue

1 Por el momento no se soporta ningún otro formato de imagen. Dejamos esa implementación para un trabajo futuro.

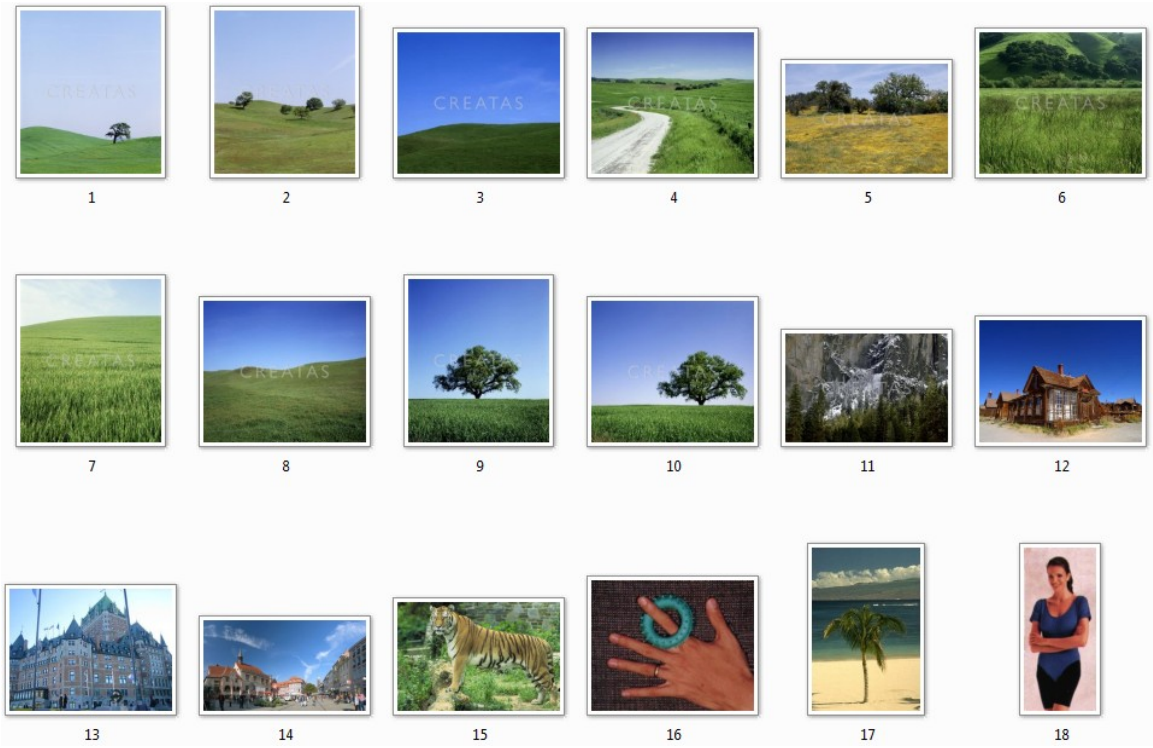


Figura 5.1: Conjunto A: 18 imágenes seleccionas desde el inicio para apoyar el desarrollo del algoritmo.

obtenido de una página especializada en el tema de segmentación de imágenes perteneciente a la Universidad de Berkeley, consta de 100 imágenes en color de 481 x 321 píxeles [Url3]. En busca de un conjunto C de imágenes más comunes y complejas recurrimos a la sección "*imagen del día*" de Wikipedia y tomamos 14 imágenes [Url4]. Éste último conjunto incluye imágenes de muy alto nivel de complejidad para la segmentación por su gran saturación de texturas y contornos, contiene imágenes en tonos de gris e imágenes en color y algunas de ellas han sido editadas o generadas por medios electrónicos.

5.3 Resultados

A continuación mostraremos y comentaremos algunos resultados seleccionados de los conjuntos utilizados durante el desarrollo de nuestro algoritmo.

5.3.1 Conjunto A

La Figura 5.2 muestra los resultados que arroja nuestro algoritmo después de segmentar el conjunto A utilizando los parámetros por omisión explicados en el Capítulo 4. La segmentación de todo el conjunto (18 imágenes) tomó **15.871 segundos**. A continuación explicaremos algunos de estos resultados.

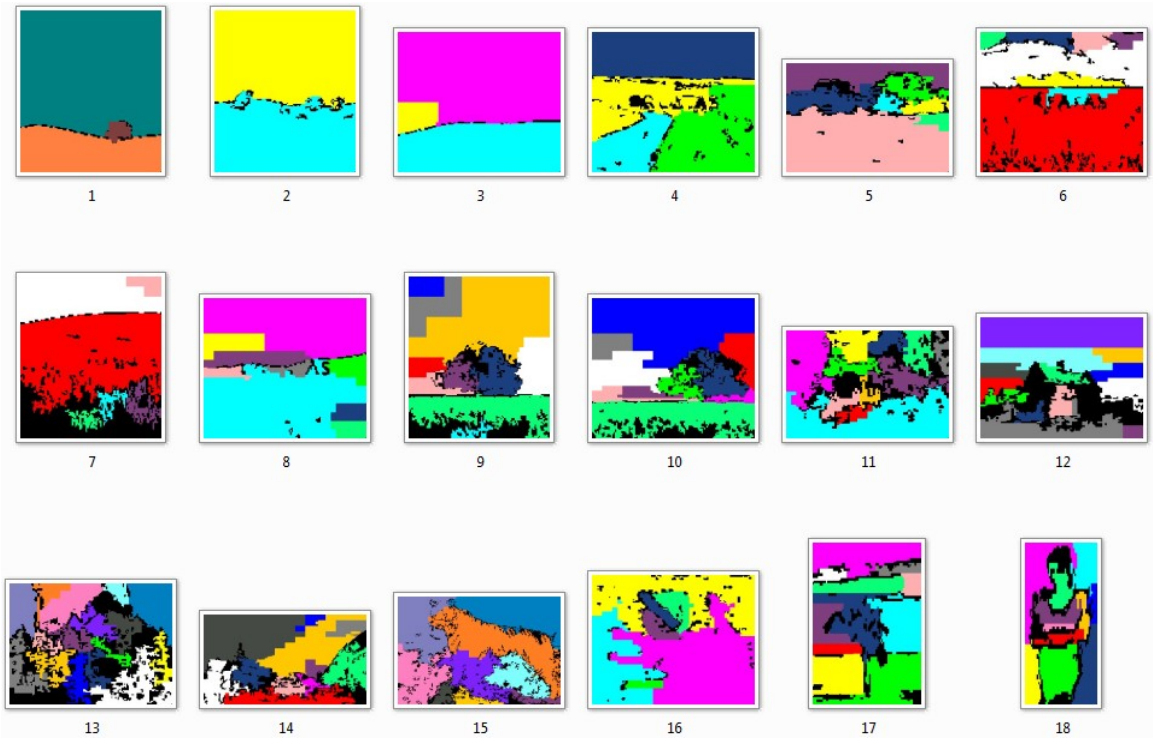


Figura 5.2: Resultado de segmentar el conjunto (a) de la Figura 5.1 utilizando los parámetros por omisión del algoritmo. En todos los resultados puede distinguirse cuál es el objeto central de cada imagen; asimismo mientras más homogéneas sean las regiones en una imagen, mejor es la segmentación que produce nuestro algoritmo.

La imagen 1 es la imagen más sencilla del conjunto A y presenta el caso ideal para nuestro segmentador. La segmentación de esta imagen tomó **2620** milisegundos. En la Figura 5.3 podemos observar todos los detalles del proceso realizado para realizar su segmentación. De arriba a abajo y de izquierda a derecha tenemos: el gradiente de la imagen, la imagen de restricciones, la imagen discretizada, el recorrido en *quadtree*, la segmentación original y la segmentación después de realizar el post-procesamiento. Esta imagen pudo segmentarse muy bien debido a que su gradiente es muy bueno, casi por sí sólo separa las principales regiones de la imagen, lo cual origina una muy buena imagen de restricciones. Además el resultado de discretizar la imagen da regiones muy homogéneas, lo cual facilita una comparación entre histogramas exitosa. La imagen que muestra el recorrido de *quadtree* que se generó para esta imagen ilustra y confirma perfectamente la idea de fondo que motiva este trabajo de tesis: las regiones similares más grandes son detectadas rápidamente y poco a poco se va reduciendo el tamaño de las regiones encontradas hasta llegar a las más pequeñas, todo esto de manera iterativa y con un esfuerzo progresivo. En las regiones cercanas a los bordes de restricción es evidente como la densidad del *quadtree* aumenta. En el resultado de segmentación original podemos notar las tres principales regiones de la imagen notoriamente delimitadas. Es importante señalar que este resultado posee regiones negras, también llamadas “región CERO”. Esta región corresponde a píxeles que no pudieron ser identificados como parte

de algún segmento dada su presencia en la imagen de restricciones ya que esto nos indica, por convención, que pertenecen a un contorno. Recordemos que la utilidad y justificación de la región CERO es que es preferible eliminar completamente de la segmentación aquellos píxeles que se ubican entre los límites de las regiones ya que introducirán ruido a los segmentos debido a la natural incapacidad de determinar a cual de los segmentos corresponde cada píxel (segmentación perfecta). Por lo tanto, es preferible excluir dichos píxeles de la segmentación a incluirlos en el segmento equivocado. Además, **eliminar píxeles de un segmento homogéneo no afecta su homogeneidad.** En la imagen pos-procesada de la segmentación observamos que los segmentos que no cumplían el tamaño mínimo requerido fueron unidos a la región adyacente similar más cercana. Un proceso muy similar sigue la segmentación de las imágenes 2-6 del conjunto (a).

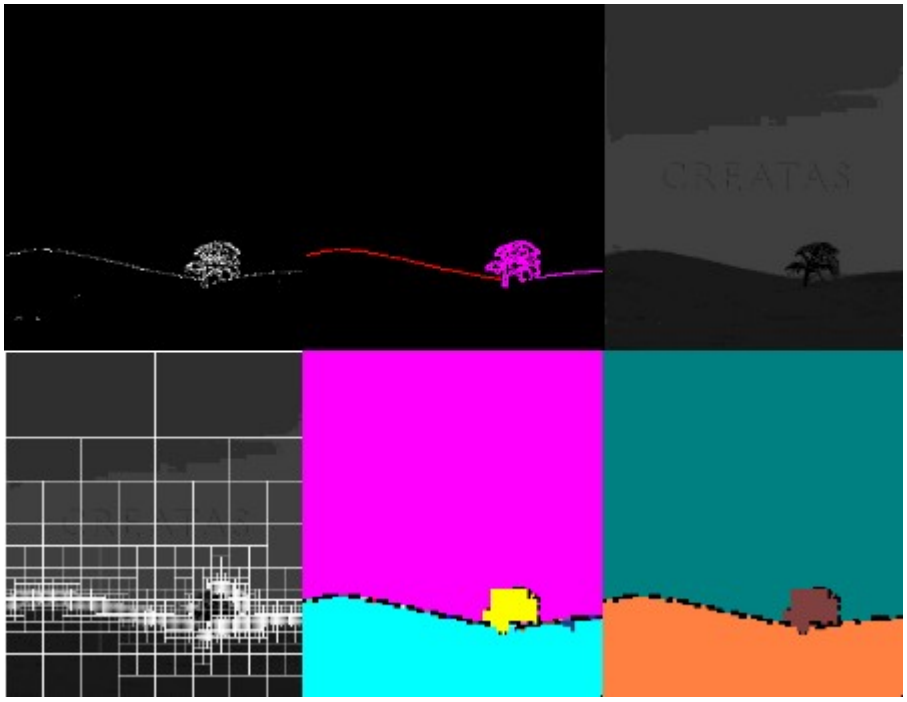


Figura 5.3: Etapas de segmentación de la imagen 1 del conjunto de la Figura 5.1. De izquierda a derecha y de arriba a abajo tenemos: gradiente, imagen de restricciones, discretización, recorrido en quadtree, segmentación original y segmentación después del post procesamiento.

El primer resultado de segmentación considerablemente diferente es el que muestra la Figura 5.4 correspondiente a la imagen 7 del conjunto A. Esta diferencia consiste en la gran dimensión de la región CERO. Procederemos a explicarla. El gradiente de la imagen tiene una densidad muy fuerte, esto genera muchos objetos en la imagen de restricciones. Observemos que en la imagen discretizada no se obtienen tonos de color homogéneos en el área inferior de la imagen. Debido a estos dos factores, numerosos píxeles en la imagen de restricciones y poca homogeneidad en la imagen discretizada, se genera un *quadtree* con densidad muy alta, sobretodo en la región inferior de la imagen. Esto ocasiona la sobre segmentación de toda esa zona. Muchos segmentos pequeños logran adherirse para

formar segmentos mayores en la etapa de post procesamiento. Sin embargo, muchos de estos pequeños segmentos no tienen algún vecino adyacente con quien unirse lo que ocasiona su desaparición (debido a nuestra restricción de tamaño mínimo para la detección de una región). Un detalle curioso de esta segmentación es el pequeño segmento detectado en la zona superior derecha de la imagen. Aunque también forma parte del cielo, podemos observar que en la imagen discretizada sus píxeles correspondieron a unos valores muy diferentes del resto de píxeles correspondientes al cielo, lo cual ocasiona que sea detectado como un segmento aparte. Consideramos este comportamiento correcto ya que si observamos detenidamente la imagen original notaremos que esa región del cielo tiene tonos diferentes. La segmentación de esta imagen tomó **483 milisegundos**. Este mismo comportamiento, en mayor o menor escala, se produce en las imágenes 8-14 del conjunto A.

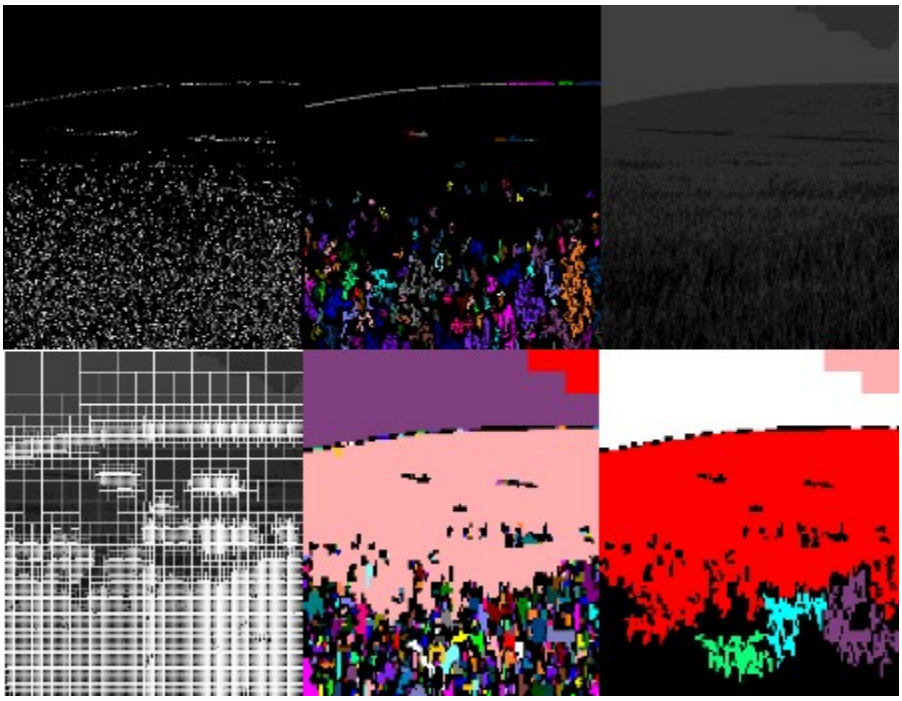


Figura 5.4: Etapas de la segmentación de la imagen 7 del conjunto de la Figura 5.1. De izquierda a derecha y de arriba a abajo tenemos: gradiente, imagen de restricciones, discretización, recorrido en quadtree, segmentación original y segmentación después del post procesamiento.

El siguiente resultado interesante que queremos tratar es el de la imagen del tigre, imagen 15 del conjunto A, mostrado en la Figura 5.5. Recordemos que la razón para incluir esta imagen en el conjunto de pruebas fue probar la capacidad de nuestro algoritmo para detectar como un sólo segmento regiones con texturas compuestas de múltiples tonos con cambios de intensidad bruscos, tal como la piel del tigre. Podemos observar que la imagen del gradiente posee un gran número de cambios de intensidad en la cabeza del tigre, en el cuerpo del tigre y en la vegetación de la zona inferior. Gran parte de toda esta densidad es eliminada en la imagen de restricciones, quedando únicamente las líneas más sobresalientes. Sin embargo, aún permanecen varias líneas que no corresponden

precisamente a límites de regiones, por el contrario, están dentro de regiones que deseamos considerar una sola. Por otro lado, la imagen discretizada presenta también fuertes variaciones de color dentro del cuerpo del tigre. Aparentemente tenemos un escenario muy desalentador para el algoritmo pues sabemos que su operador de unión de regiones se basa precisamente en las líneas de la imagen de restricciones y en la similitud de tonalidades de la imagen discretizada. En la imagen de recorrido de *quadtree* vemos el gran esfuerzo que se realiza para generar la segmentación original. A pesar de las dificultades mencionadas, nuestro algoritmo es capaz de detectar las regiones homogéneas de la imagen donde no encuentra limitaciones en la imagen de restricciones. De esta manera, logra crear el segmento color *cyan* que corresponde al cuerpo del tigre. Después de esto, el módulo que perfecciona la segmentación dejándola en un estado aceptable es el post procesamiento ya que logra agregar al cuerpo del tigre numerosos segmentos pequeños que habían quedado aislados debido a los bordes de la imagen de restricciones. Es satisfactorio notar que el segmento creció hasta el punto de abarcar gran parte de la cara del tigre que, recordemos, estaba completamente saturada de contornos. El proceso de segmentación de esta imagen tardó **4690 milisegundos**. El tiempo es más alto que en los ejemplos anteriores debido a que la imagen es más grande. Los tiempos de segmentación del resto de las imágenes del conjunto A se puede observar en el Anexo B.

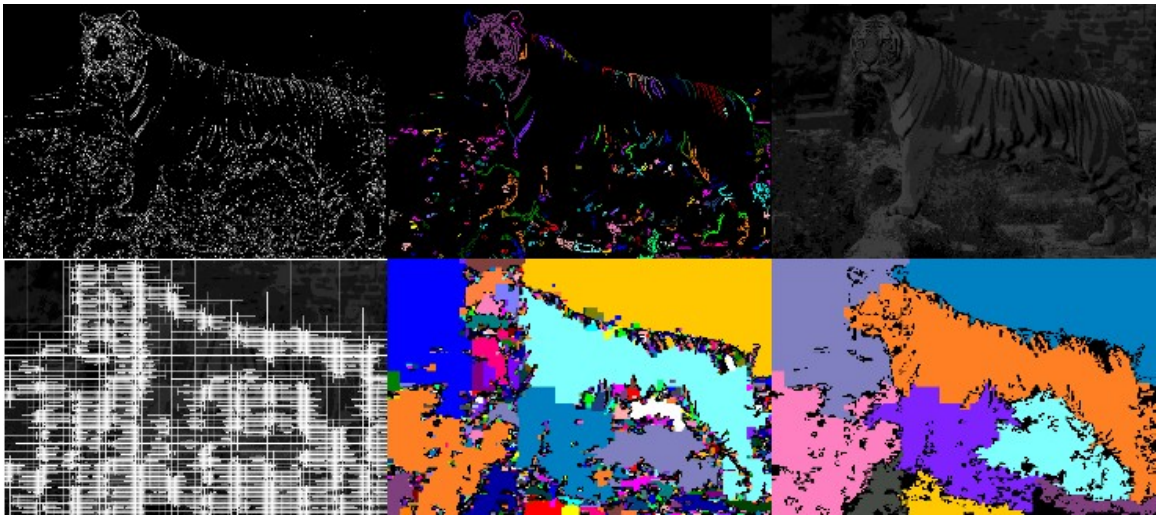


Figura 5.5: Etapas de la segmentación de la imagen 15 del conjunto de la Figura 5.1. De izquierda a derecha y de arriba a abajo tenemos: gradiente, imagen de restricciones, discretización, recorrido en quadtree, segmentación original y segmentación después del post procesamiento.

Es importante recordar que nuestro algoritmo no pretende ser un algoritmo de segmentación detallada, es decir obtener segmentos finos y bien delineados abarcando la totalidad de la imagen. Por el contrario, nuestro algoritmo únicamente pretende localizar regiones homogéneas y considerablemente grandes dentro de una imagen ya que creemos que la información de dichas regiones es útil para analizar el contenido visual de una imagen y construir anotaciones que lo describan. De esta forma, aunque no encontramos un segmento que delinee completa y finamente al tigre, **encontramos un segmento grande que contiene mucha información útil** que otras herramientas pueden analizar

para diagnosticar la presencia de un tigre en la imagen y generar la anotación “tigre” para dicha imagen.

5.3.2 Conjunto B

Ahora nos concentraremos en algunos resultados del conjunto B, obtenido de la Universidad de Berkeley[2], el cual contiene imágenes más complejas. Recordemos que el algoritmo fue afinado para poder dar resultados satisfactorios para todos los conjuntos de prueba, lo que limita su calidad. Sin embargo, si afinamos el algoritmo para un conjunto de imágenes específico, se incrementará la calidad de los resultados. A continuación mostraremos resultados de segmentación del conjunto B tanto con los parámetros por omisión del algoritmo, como por parámetros especificados manualmente, ajustados para este conjunto.

La Figura 5.6 muestra el resultado de segmentar la imagen 101085 del conjunto B con los parámetros por omisión del algoritmo. Se muestra el detalle de los pasos que ya han venido comentándose en este capítulo. En dicho detalle podemos observar que el gradiente es muy intenso en gran parte de la imagen. Dicho gradiente es atenuado por el cálculo de la imagen de restricciones aunque gran parte es conservada en la zona inferior. La discretización muestra que la imagen es muy variante en sus colores, lo que podría hacer difícil detectar regiones similares. Sin embargo, después de un esforzado recorrido en *quadtree*, la segmentación producida es satisfactoria. En la imagen del post procesamiento se perfecciona la segmentación inicial a través de la eliminación de las regiones que no cumplen nuestro tamaño mínimo requerido, uniendo algunas de ellas a una región mayor y eliminando aquellas que no pueden ser unidas a ninguna. La Figura

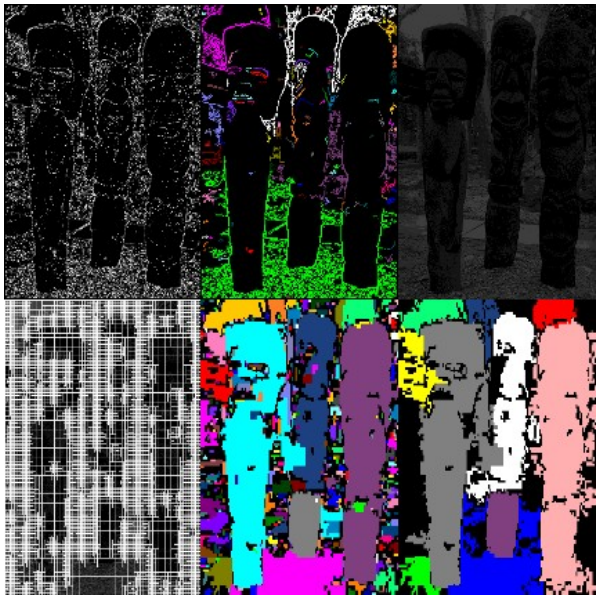


Figura 5.6: Etapas de la segmentación de la imagen 101085 del conjunto B.De izquierda a derecha y de arriba a abajo tenemos: gradiente, imagen de restricciones, discretización, recorrido en quadtree, segmentación original y segmentación después del post procesamiento.

2 http://berkeley.edu/

5.7 muestra más ejemplos de buenos resultados obtenidos con los parámetros por omisión del algoritmo.

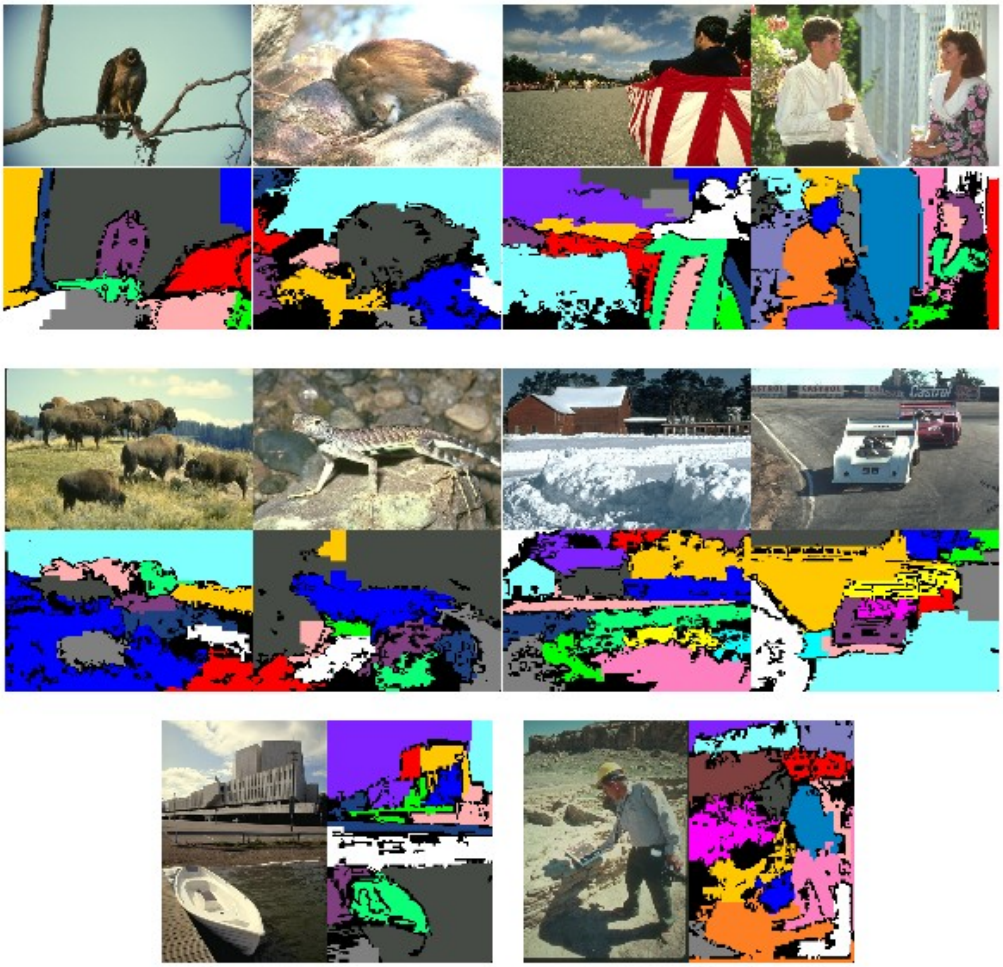


Figura 5.7: Resultados satisfactorios del conjunto B de la Universidad de Berkeley utilizando los parámetros por omisión del algoritmo. Cada bloque muestra la imagen original y el resultado de segmentación debajo (o a un lado) de ella.

Las figuras 5.10, 5.11, 5.9, 5.8 y 5.12 ilustran resultados de segmentación no suficientemente satisfactorios obtenidos con los parámetros por omisión del algoritmo y cómo podemos elevar la calidad de estos resultados afinando los parámetros del algoritmo. Recordemos: el parámetro *s* se refiere al tamaño mínimo de los objetos que deseamos detectar (utilizado para formar la imagen de restricciones, Sección 4.4) mientras que el parámetro *h* se refiere al umbral de homogeneidad (utilizado por el comparador de regiones para unir los cuadrantes del *quadtree* Sección 4.6.2). Una guía para modificar éstos parámetros es que mientras más grande sea el valor de *s*, menos restricciones habrá para unir regiones. Por lo que si al segmentar obtenemos muchos segmentos, es aconsejable aumentar el valor de *s*, por el contrario si obtenemos

segmentos muy grandes que abarcan más de una región, es aconsejable disminuir el valor de *s*. Por otro lado, conforme *h* tiende a cero, nuestros resultados tenderán a la sobre segmentación y mientras *h* tienda a uno, nuestros resultados tendrán menos regiones. Durante el proceso de afinar el algoritmo es necesario jugar con éstos dos parámetros.

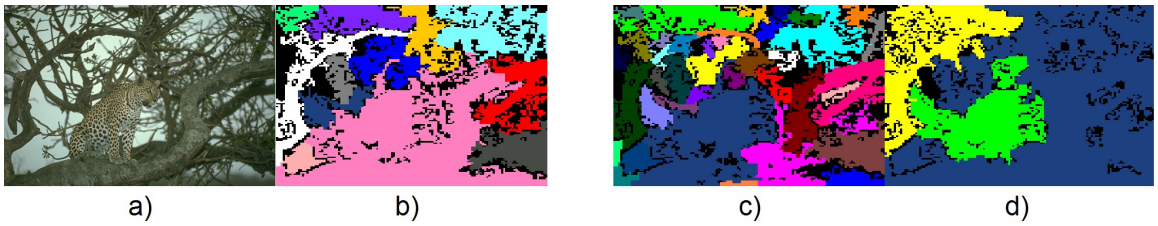


Figura 5.10: a) Imagen original, b) segmentación con parámetros por omisión, c) segmentación con s=0.003 y h=0.5, d) segmentación con s=0.1 y h=0.6.

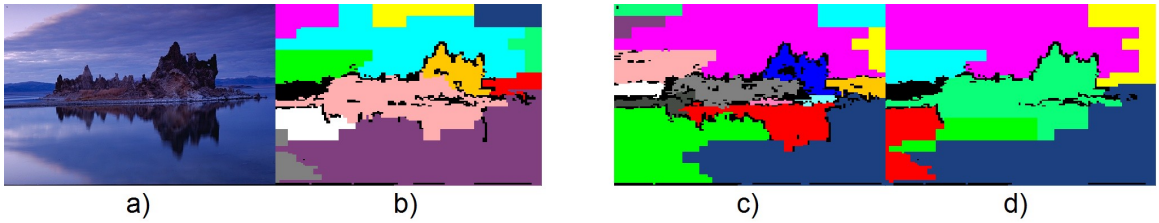


Figura 5.9: a) Imagen original, b) segmentación con parámetros por omisión, c) segmentación con s=0.00250 y h=0.65, d) segmentación con s=0.05 y h=0.5.

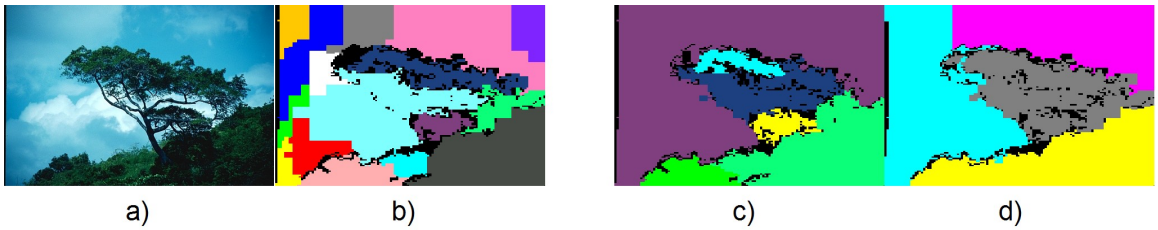


Figura 5.11: a) Imagen original, b) segmentación con parámetros por omisión, c) segmentación con s=0.02 y h=0.75, d) segmentación con s=0.1 y h=0.6.

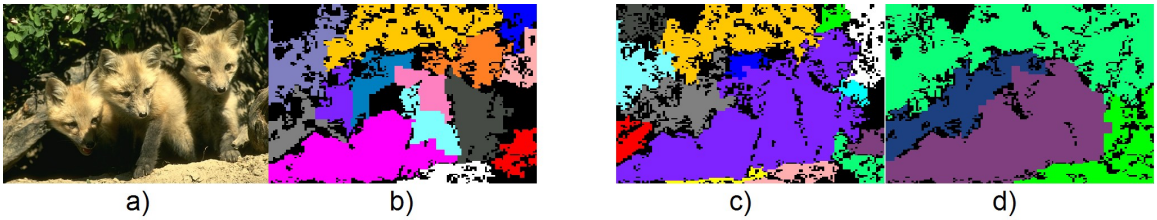


Figura 5.8: a) Imagen original, b) segmentación con parámetros por omisión, c) segmentación con h=0.9 y s=0.00625, d) segmentación con s=0.05 y h=0.65.

Para hacer una valoración justa de nuestro algoritmo, también presentaremos casos de imágenes que no pueden ser segmentadas satisfactoriamente sin importar cuanto modifiquemos los parámetros del algoritmo. Hemos detectado que los casos en que esto

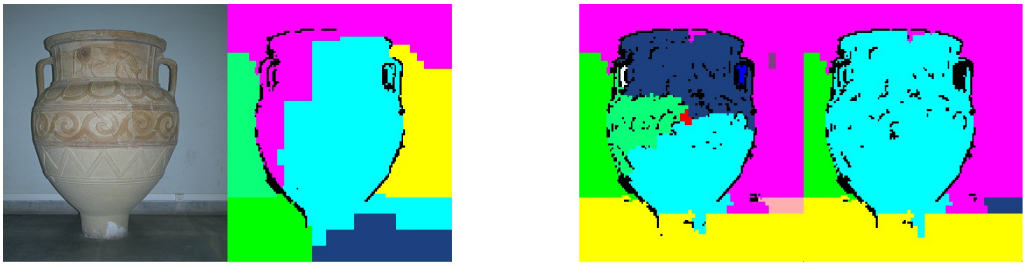


Figura 5.12: a) Imagen original, b) segmentación con parámetros por omisión, c) segmentación con s=0.00125 y h=0.7, d) segmentación con s=0.003 y h=0.675.

sucede es cuando la imagen no contiene o contiene muy escasos o/y sutiles delimitadores entre cada una de sus diferentes regiones (contornos). En estas imágenes es difícil detectar los límites de sus regiones dando como resultado segmentos que abarcan más de una sola región. También es muy difícil de segmentar el caso contrario: cuando las imágenes están sobresaturadas de bordes. De ambos casos nos muestra ejemplos la Figura 5.13.

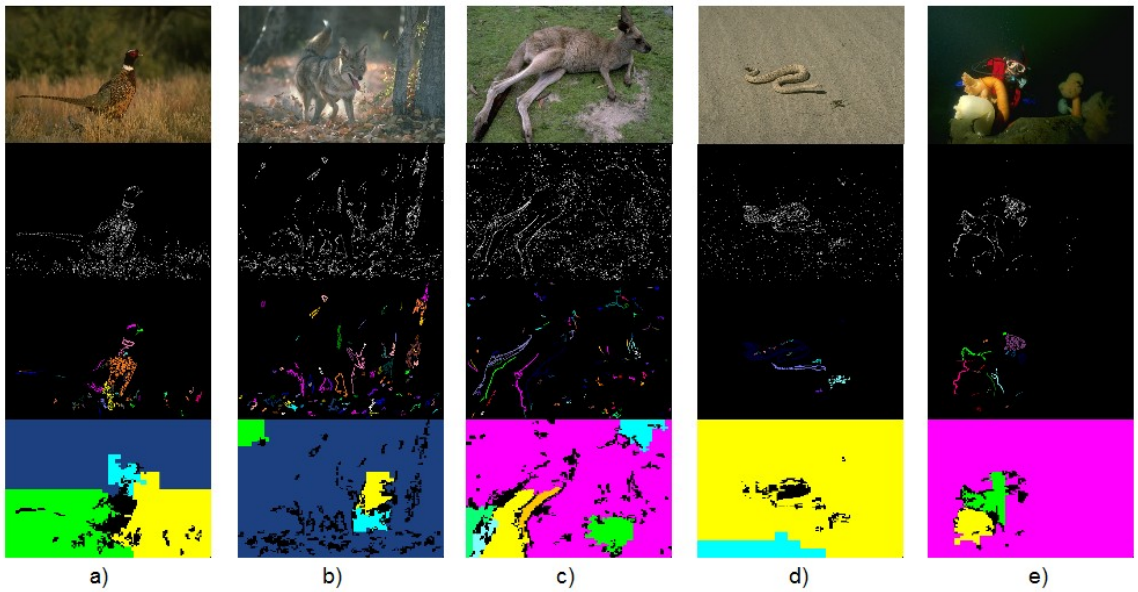


Figura 5.13: Imágenes del conjunto B que no son segmentadas satisfactoriamente debido a la dificultad para hallar los contornos de los objetos de las imágenes. Fila 1: imagen original, fila 2: gradiente de la imágen, fila 3: imagen de restricciones, fila 4: resultado de la segmentación.

5.3.3 Conjunto C

Finalmente, mostraremos los resultados que obtuvimos con el conjunto obtenido de la wikipedia, conjunto C (Figura 5.14). Estos resultados se pueden observan en la Figura 5.15. Recordemos que este conjunto de imágenes es el más complejo, por lo real de

algunas de sus imágenes y por lo artificial de otras. Podemos notar que los resultados de segmentar este conjunto son, en general, aceptables. El conjunto C completo fue segmentado en **38.828 segundos**.

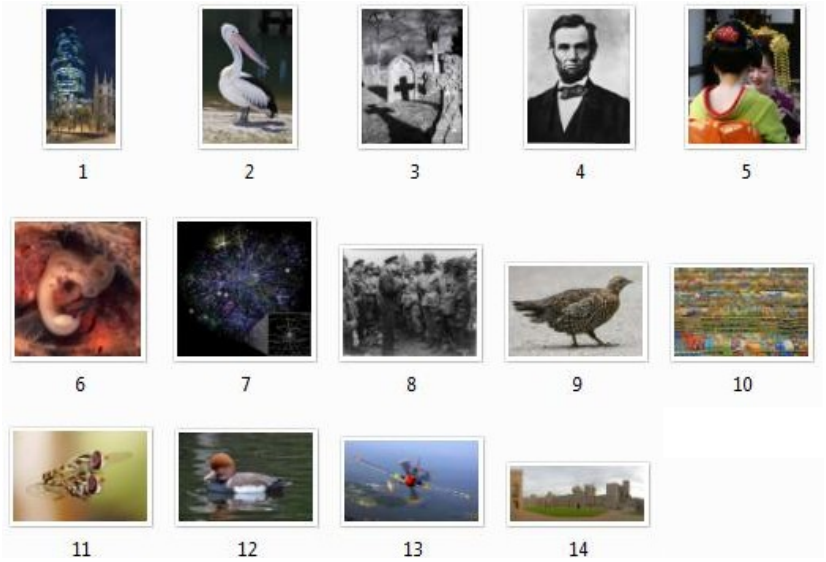


Figura 5.14: Conjunto C, el conjunto más heterogéneo con el que se probó el algoritmo.

Figura 5.15: Resultados de segmentación del conjunto C

5.4 Comparación Cualitativa con Normalized Cuts

En el Capítulo 3 pudimos observar que un algoritmo ampliamente utilizado para segmentar imágenes y luego anotarlas es *normalized cuts* [J. Shi, 2000]. Precisamente, el primer artículo que leímos y que inspiró este trabajo: *"Object annotacion as machine translation",* [P. Duygulu, 2002] esta basado en él. Decidimos comparar los resultados de nuestro algoritmo con los resultados que genera *normalized cuts*, por ser un algoritmo generalizado y comúnmente conocido en el área de segmentación de imágenes. Para esta tarea, utilizamos una versión de *normalized cuts* implementado en matlab con la que cuenta el grupo de procesamiento de texto e imágenes (TIA) al que también esta vinculado este trabajo de tesis. Cabe recordar que *normalized cuts* necesita como parámetro de entrada el número de regiones que queremos que encuentre y el algoritmo forzosamente encontrará ese número de regiones, no más, no menos.

La Figura 5.17 muestra los resultados de segmentación de *normalized cuts* para nuestro conjunto A. Volvemos a mostrar los resultados de la Figura 5.2 en la Figura 5.16 para poder comparar más fácilmente los resultados de ambos algoritmos. Desde nuestro punto de vista, señalamos lo siguiente: A partir de la imagen 1 hasta la imagen 7 nuestra ventaja es notable. A partir de la imagen 8 en adelante, parece que nuestro algoritmo encuentra más segmentos que *normalized cuts* pero esto, debemos recordar, se debe a que *normalized cuts* **forza** el número de regiones de sus resultados. Sin embargo, recordemos que una de nuestras principales hipótesis y guías consiste en encontrar regiones **homogéneas** y no segmentos perfectamente delineados. De esta manera, podemos considerar muy similares nuestros resultados y los resultados de *normalized cuts* en tanto cuanto, los nuestros, a pesar de contener más segmentos, **apoyan igualmente la anotación de imágenes**. Observando detenidamente las imágenes podemos ver que existen algunos detalles que hacen mejores nuestros resultados, como en el caso de la imagen 15, donde podemos encontrar una región más grande que abarca al tigre. En la imagen 14 y 16 también obtenemos resultados ligeramente mejores. Finalmente, mencionaremos que *normalized cuts* tomó **3613.6926** segundos para segmentar todo el conjunto A mientras que nuestro algoritmo tomó solamente **15.871** segundos, esto es **228** veces más rápido. Consiguiendo con esto nuestra principal meta: desarrollar un algoritmo que encuentre en un **tiempo reducido** regiones homogéneas grandes que permitan construir anotaciones para la imagen que segmentan.

5.5 Comparación Cuantitativa

5.5.1 Conjunto de datos de prueba

Para la prueba de comparación cuantitativa se eligió la base de datos IAPR TC-12 por ser esta la base de imágenes de trabajo del grupo de investigación TIA[3], del que esta tesis forma parte. Esta colección consiste de 20,000 imágenes tomadas en localidades

3 http://ccc.inaoep.mx/~tia/TIA/

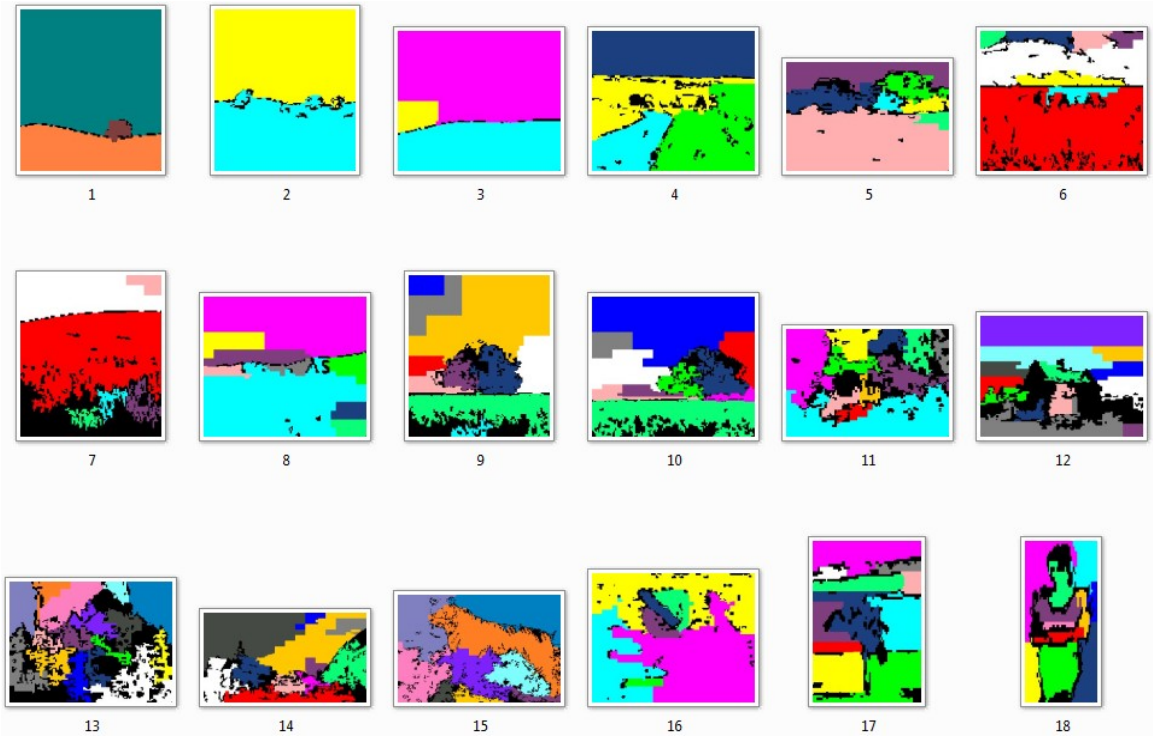


Figura 5.16: Resultados de segmentar el conjunto A con nuestro algoritmo: quadtree segmentation

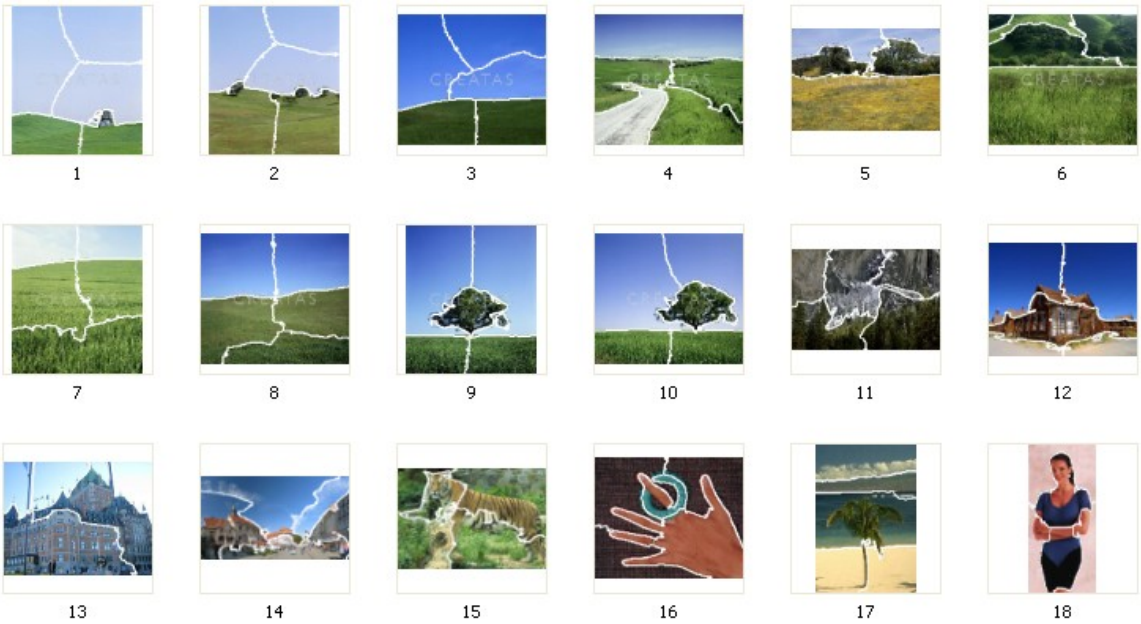


Figura 5.17: Resultados de segmentar el conjunto A con normalized cuts.

repartidas en todo el mundo. Incluye imágenes de diferentes deportes, acciones, personas, animales, ciudades, paisajes y muchos otros aspectos de la vida contemporánea

5.5.2 Descripción de la prueba

Actualmente el grupo TIA posee 20,000 imágenes de la base IAPRTC-12 segmentadas y anotadas manualmente. Asimismo, posee un motor de anotación automática que ha sido entrenado con esta base. La prueba para evaluar la precisión de segmentación y de anotación de nuestro algoritmo consistió en tomar 500 imágenes aleatorias de esta base y segmentarlas con tres algoritmos diferentes:

1. Segmentación de cuadros o rejillas
2. Segmentación *normalized cuts*
3. Segmentación *quadtree*

Una vez segmentadas las 500 imágenes con los tres algoritmos, utilizamos el motor de anotación automática para producir anotaciones para cada segmento. Así, obtuvimos 1500 imágenes anotadas; repartidas en 3 grupos de 500 correspondiendo cada grupo a uno de los algoritmos de segmentación utilizados. Luego, procedimos a comparar los resultados de segmentación y de anotación de cada grupo contra los resultados manuales.

5.5.3. Algoritmos de evaluación

Los algoritmos que se utilizaron para evaluar la segmentación y la anotación, comparan una imagen segmentada automáticamente con la misma imagen segmentada manualmente. Básicamente lo que hacen es que para cada segmento producido de forma automática buscan con qué segmento producido manualmente tiene más píxeles en común. Una vez localizada esta pareja de sementos, evaluán el segmento producido automáticamente en base a cuántos píxeles tiene de más y cuánto píxeles le faltan respecto al segmento producido manualmente. El algoritmo de evaluación de la anotación realiza el mismo proceso comentado para seleccionar los segmentos a comparar y luego revisa si las anotaciones coinciden o no.

Los Cuadro 5.1y Cuadro 5.2 muestran los algoritmos utilizados para calificar y comparar los resultados de los tres grupos mencionados en la sección anterior. El algoritmo de evaluación de la segmentación evalúa la correcta agrupación de píxeles en sus segmentos correspondientes, incluye una penalización por píxeles asignados incorrectamente y también toma en cuenta la relevancia de cada segmento, dando mayor puntaje a los segmentos más grandes. No toma en cuenta la geometría del segmento ya que la intención de esta tesis no es un algoritmo que proporcione segmentos perfectamente delineados sino segmentos homogéneos. Por la misma razón, este algoritmo no penaliza la sobresegmentación. El algoritmo de evaluación de la anotación evalúa la correcta asignación de etiqueta a cada segmento; al hacerlo, toma en cuenta la relevancia del segmento, dando mayor puntaje a los segmentos más grandes.

5.5.4. Resultados de segmentación

Las Figuras 5.18 y 5.19 muestran algunas de las imágenes que obtuvieron calificaciones más altas con cada uno de los segmentadores. La Tabla 5.1 muestra datos estadísticos de

las calificaciones que obtuvieron las 500 imágenes de cada grupo y la Tabla 5.2 muestra los tiempos de ejecución que tomó segmentar cada grupo. Dado que el algoritmo de evaluación sólo mide homogeneidad de los segmentos y correspondencia de los píxeles, y no mide la geometría de los segmentos, rejillas tiene una ligera ventaja sobre nosotros; ya que sus rejillas son muy pequeñas y por consecuencia el contenido de cada una es normalmente homogéneo. Sin embargo,la ventaja de nuestro algoritmo se verá en la siguiente sección cuando se muestre que nuestros segmentos son mejor anotados. En referencia a los tiempos, nuestro algoritmo tardó 6% del tiempo que tarda *normalized cuts* y sólo 15 veces más del tiempo que tarda rejillas. Las calificaciones de segmentación de cada una de las imágenes del conjunto de pruebas se muestran en la Gráfica 5.1.

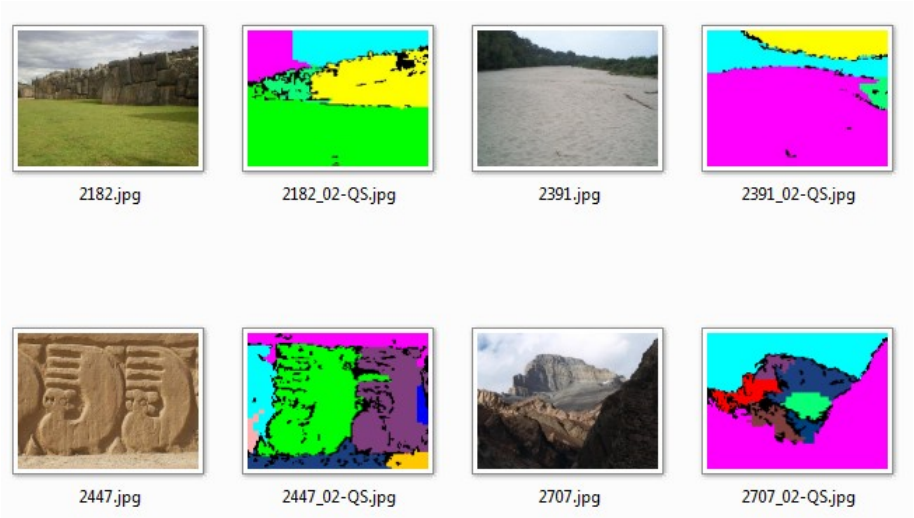


Figura 5.18: Ejemplo de segmentaciones realizadas con nuestro algoritmo de quadtrees y que obtuvieron alta calificación.

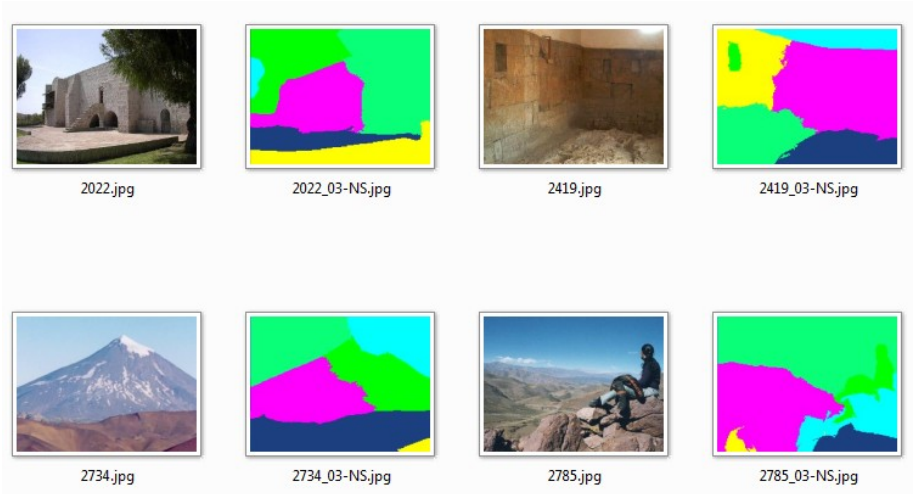


Figura 5.19: Ejemplo de segmentaciones realizadas con el algoritmo normalized cuts y que obtuvieron alta calificación.

Sea I una imagen cualquiera
Sea $I_m = (Sm_1 \cup Sm_2 \cup ... \cup Sm_p)$ *la segmentación manual de I*
Sea $I_a = (Sa_1 \cup Sa_2 \cup ... \cup Sa_q)$ *la segmentación automática de I*
Sea $dim(x) =$ *número de píxeles en x*
La calificación C de la segmentación I_a *con respecto a* I_m
se obtiene de la siguiente forma :

$\forall\, Sa_i \in I_a$
Encontrar Sm_j *tal que* $Sa_i \cap Sm_j > Sa_i \cap Sm_k \;\forall\, Sm_k$ *con* $k \neq j$; $Sm_j, Sm_k \in I_m$
Sea $a = \dfrac{dim(Sa_i \cap Sm_j)}{dim(Sa_i)}$
Sea $b = \dfrac{dim(Sa_i \cap Sm_j)}{dim(Sm_j)}$
Sea $c = \dfrac{dim(Sm_j)}{dim(I_m)}$
$c_i = a * b * c$

$C = \sum c_i$

Cuadro 5.1: Algoritmo utilizado para evaluar la segmentación de imágenes.

Sea I una imagen cualquiera
Sea $I_m = (Sm_1 \cup Sm_2 \cup ... \cup Sm_p)$ *la segmentación manual de I*
Sea $I_a = (Sa_1 \cup Sa_2 \cup ... \cup Sa_q)$ *la segmentación automática de I*
Sea $dim(x) =$ *número de píxeles en x*
Sea $etq(x) =$ *etiqueta asignada al segmento x*
La calificación C de la anotación de I_a *con respecto a* I_m
se obtiene de la siguiente forma :

$\forall\, Sa_i \in I_a$
Encontrar Sm_j *tal que* $Sa_i \cap Sm_j > Sa_i \cap Sm_k \;\forall\, Sm_k$ *con* $k \neq j$; $Sm_j, Sm_k \in I_m$
Sea $L_i = etq(Sa_i)$
Sea $L_j = etq(Sm_j)$
Sea $a = \dfrac{dim(Sm_j)}{dim(I_m)}$
Si $L_i = L_j \Rightarrow c_i = \dfrac{a}{p}$
Si $L_i \neq L_j \Rightarrow c_i = 0$

$C = \sum c_i$

Cuadro 5.2: Algoritmo utilizado para evaluar la anotación de imágenes.

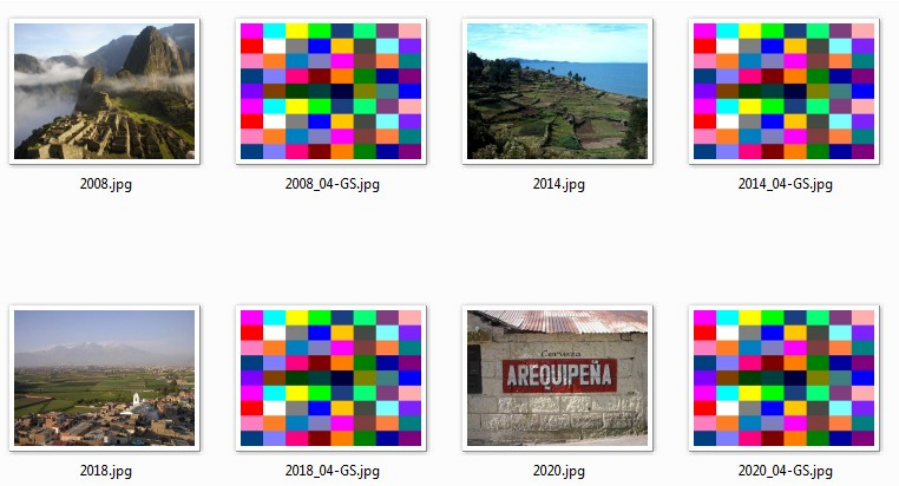


Figura 5.20: Ejemplo de segmentaciones realizadas con el algoritmo de rejillas y que obtuvieron alta calificación. Cabe mencionar que dicha calificación se debe a que sus segmentos son homogéneos y no a que su forma sea muy similar a la de los segmentos reales.

	QUADTREE	NORMALIZED CUTS	REJILLAS
Suma de calificaciones:	252.6800	271.5442	**328.14857**
Promedio:	0.5063	0.5452	**0.6589**
Desviación Estándar:	**0.2023**	0.2285	0.2683

Tabla 5.1: Resultados de evaluación de la segmentación.

	QUADTREE	NORMALIZED CUTS	REJILLAS
Segs * 500 Imágenes	1,126.83	20,000.00	74.40

Tabla 5.2: Tiempo de ejecución de la segmentación.

5.5.5. Resultados de Anotación

La Figura 5.21 muestra ejemplos de anotaciones que obtuvieron alta calificación y que se realizaron utilizando los segmentos generados por nuestro algoritmo de *quadtrees*. La Tabla 5.3 muestra estadísticas sobre los resultados de anotación de las 500 imágenes. Nuestra propuesta de algoritmo produjo resultados 50% mejores que rejillas y 35% mejores que *normalized cuts*. Los resultados de anotación obtenidos son muy interesantes. Tal es el caso de la imagen 2292, la cual originalmente tiene asignada sólo la etiqueta *vegetación* mientras que con los segmentos encontrados por nuestro algoritmo se obtienen también las etiquetas *pasto* y *árboles*. Estas etiquetas están dentro de la misma clasificación por lo cual pueden relacionarse y ser útiles en búsqueda de imágenes por contenido. Asimismo, es interesante observar que se cumple lo esperado con nuestro algoritmo de evaluación de anotación: "se da una mejor calificación a la imagen en relación directa a la relevancia de los segmentos". Tal es el caso de la imagen 2608, la

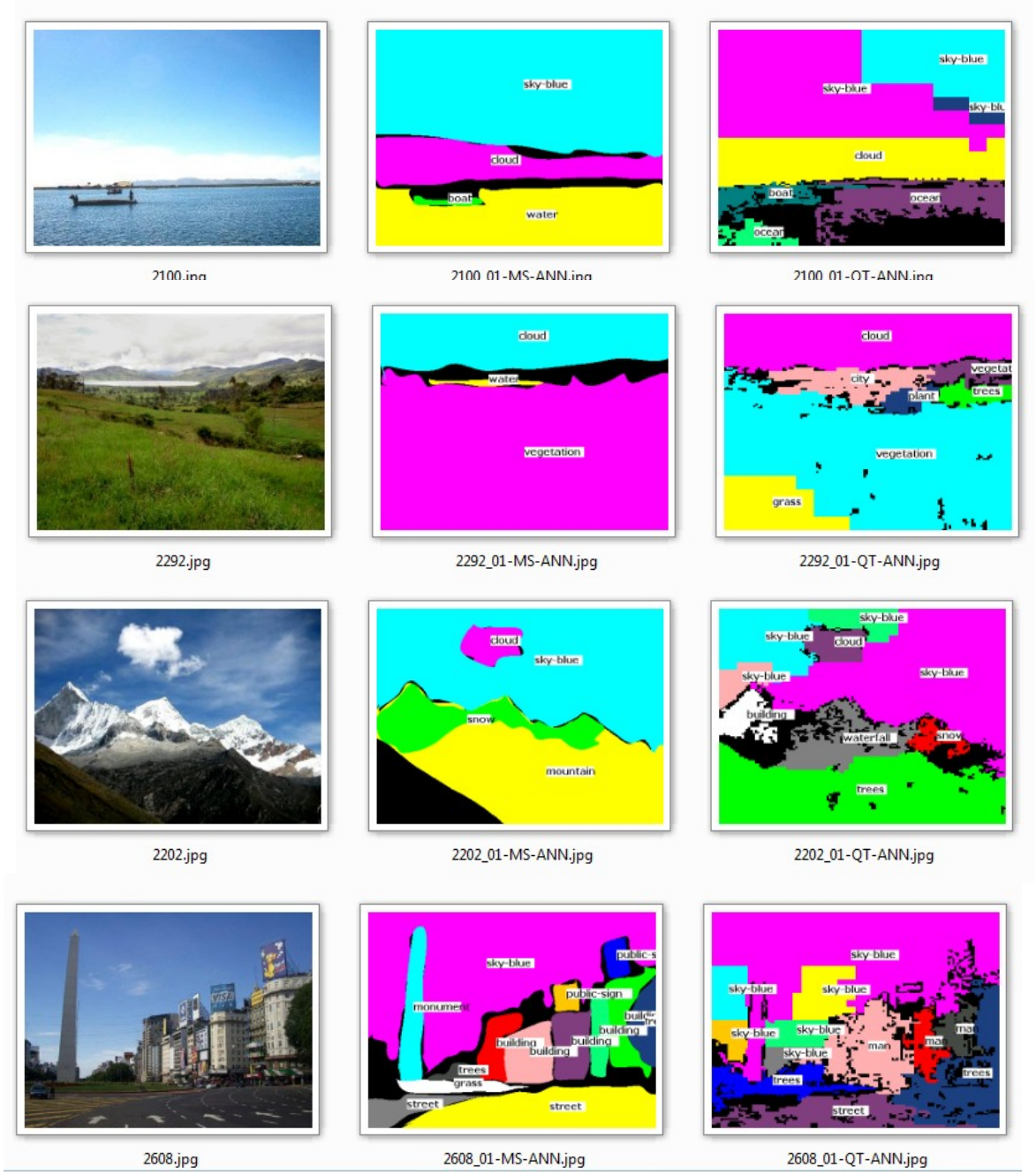


Figura 5.21: Ejemplo de anotaciones realizadas utilizando segmentos obtenidos con nuestro algoritmo de quadtrees y que obtuvieron alta calificación. En la primera columna vemos la imagen original , en la segunda vemos las anotaciones realizadas manualmente y en la tercera las anotaciones realizadas automáticamente con ayuda de nuestro algoritmo de segmentación.

cual, a pesar de tener muchos más segmentos de los esperados (los cuales son anotados erróneamente), tiene una alta calificación gracias a que obtiene correctamente las anotaciones más relevantes para la imagen: “cielo” y “calle”. Las calificaciones de anotación de las imágenes del conjunto de pruebas se muestran en la Gráfica 5.2.

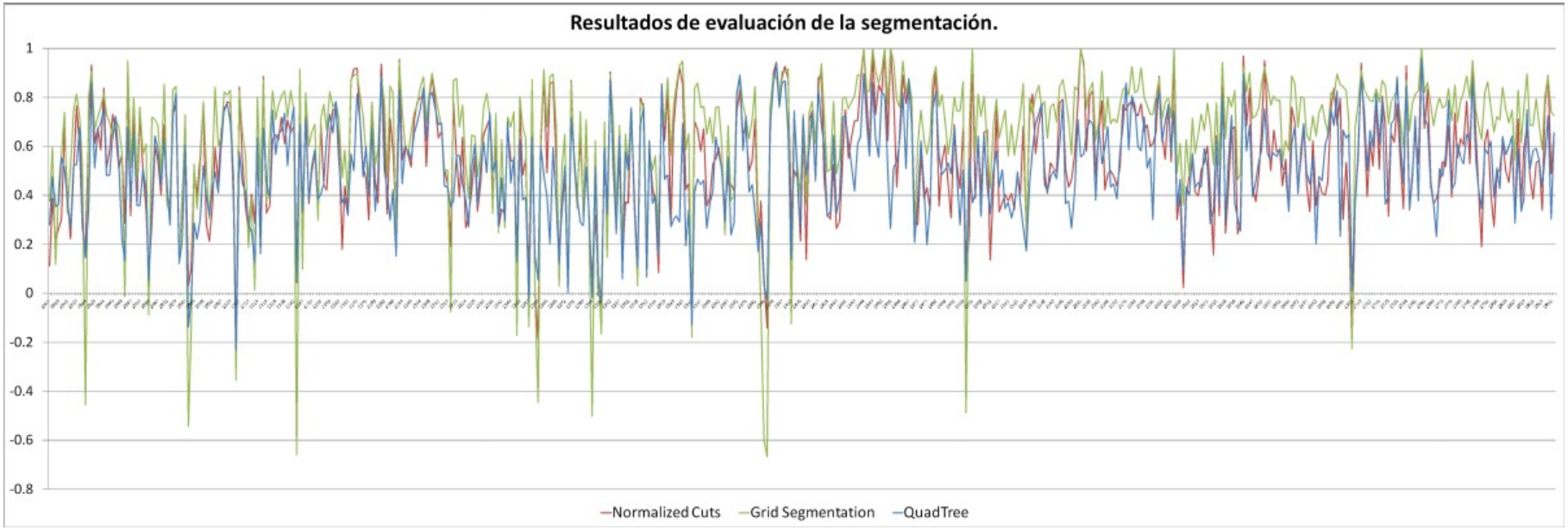
	QUADTREE	NORMALIZED CUTS	REJILLAS
Suma de calificaciones:	**30.8258**	22.7632	20.3786
Promedio:	**0.06177**	0.04570	0.04100
Desviación Estándar:	0.08465	**0.06054**	0.07216

Tabla 5.3: Resultados de evaluación de la anotación.

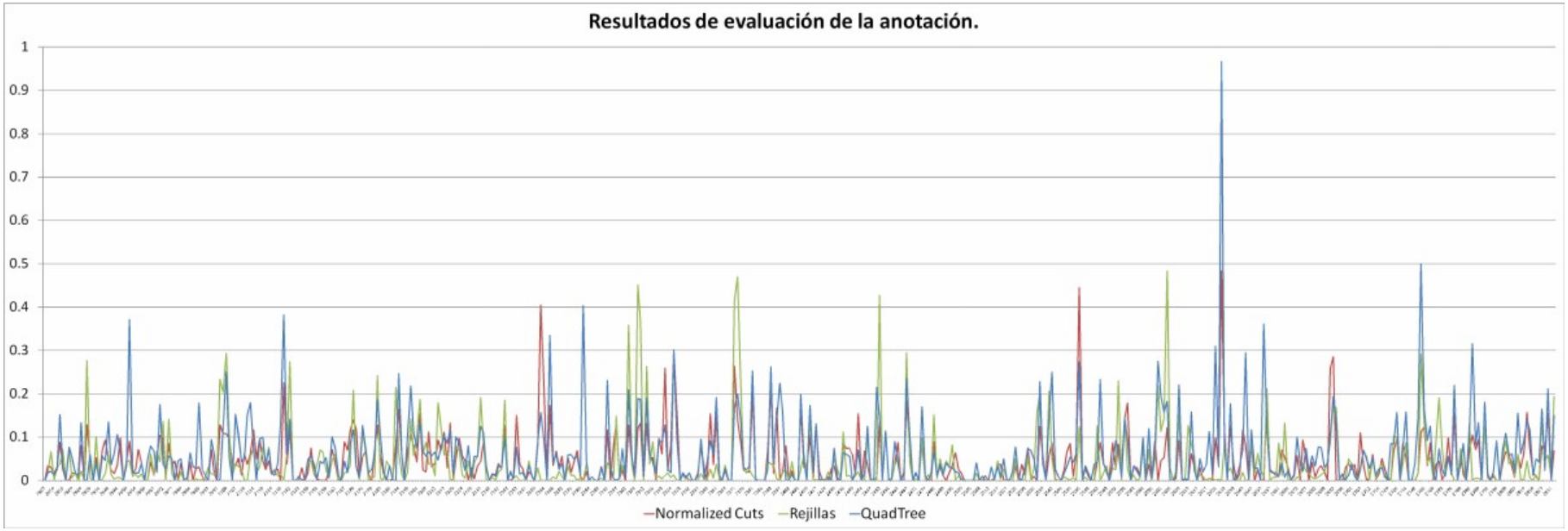
5.6 Conclusiones

En conclusión el algoritmo que proponemos resulta útil para segmentar imágenes de diferentes tipos. Es ajustable a diferentes tipos de imágenes gracias a sus parámetros de entrada aunque en general sus valores por omisión son suficientes para obtener buenos resultados de segmentación lo cual facilita su utilización. Da resultados en un tiempo muy corto en comparación con normalized cuts. Obtiene segmentos grandes y homogéneos. Todo su rendimiento está sustentado por sus 5 etapas y aún cuando todas son importantes, la imagen de restricciones resulta determinante en el resultado de la segmentación; no se obtienen buenos resultados si no se cuenta con una buena imagen de restricciones. Este punto debe considerarse para mejorar su desempeño.

Nuestro segmentador *quadtree* tardó en promedio 2.25 segundos para segmentar una imagen. Esto es menos tiempo que *normalized cuts* (37.5 segundos) y más que el algoritmo de rejillas (0.15 segundos). En la evaluación de la homogeneidad y correcta agrupación de píxeles, sin tomar en cuenta geometría de segmentos, rejillas y *normalized cuts* fueron ligeramente mejor calificados que nuestro algoritmo. Y finalmente, en la evaluación de las anotaciones nuestro algoritmo tuvo una ventaja considerable sobre los otros dos (35% mejor que *normalized cuts* y 51% mejor que rejillas). La razón de nuestra ventaja sobre rejillas es que los segmentos que produce dicho algoritmo para poder ser homogéneos necesitan ser pequeños y esto ocasiona que no provean la información suficiente para inducir correctamente las anotaciones. La razón de nuestra ventaja sobre *normalized cuts* es que éste se ve obligado a producir para todas las imágenes la misma cantidad de segmentos y además éstos al intentar corresponder a segmentos bien delineados introducen mucho ruido en los contornos. Nuestro algoritmo logra obtener la homogeneidad de rejillas llevando los segmentos al tamaño máximo en que pueden seguir siendo homogéneos y estar libres de ruido. De hecho, nuestro algoritmo puede verse como la extensión de rejillas, uniendo las rejas que son similares y adyacentes. Tal método de segmentación equivaldría a nuestro mismo recorrido de *quadtrees* hecho en profundidad en lugar de anchura. No obstante, no debemos olvidar que nuestro recorrido en anchura es lo que le da a nuestro algoritmo la potencia de poder terminar la segmentación en cualquier momento teniendo siempre una aproximación al resultado final. Con estos resultados vemos concluida la motivación de esta tesis: “desarrollar un algoritmo rápido, capaz de encontrar segmentos grandes y homogéneos con información suficiente que permita crear anotaciones para la imagen que representan”. Esta conclusión sustenta su viabilidad para ser implementado dentro de sistemas que realicen búsquedas de imágenes por texto en tiempo real.



Gráfica 5.1: Calificaciones de segmentación obtenidas para cada imagen del conjunto de pruebas.



Gráfica 5.2: Calificaciones de anotación obtenidas para cada imagen del conjunto de pruebas.

Capítulo 6

Conclusión y Trabajo Futuro

Existen muchos métodos para segmentar una imagen. Se eligen diferentes enfoques según la aplicación que se vaya a dar a los resultados de la segmentación. Para anotar imágenes, se necesitan segmentos homogéneos grandes que puedan ser encontrados en poco tiempo. Las *quadtrees* son estructuras de datos que permiten separar el contenido de una imagen rápidamente. Por este motivo, un algoritmo de segmentación basado en *quadtrees* es una buena opción como base para un sistema de anotación automática de imágenes. Para guiar eficazmente la creación del *quadtree* sobre la imagen que se segmenta es necesario contar con información sobre los límites de las regiones de la imagen y contar con un comparador de regiones confiable. El resultado de la segmentación en *quadtree* puede mejorarse uniendo algunas de sus regiones y eliminando otras. El algoritmo aquí expuesto ofrece buenos resultados en poco tiempo en una computadora de características comunes. Está construido con una arquitectura sólida, configurable, extensible y puede ser utilizado en cualquier plataforma.

6.1 Conclusiones

Hoy en día segmentar una imagen con precisión sigue siendo una tarea a la que no se le ha encontrado una respuesta definitiva que sea satisfactoria para todo tipo de necesidades. Podemos decir que la segmentación de imágenes sigue siendo un problema abierto al que sólo podemos proponer soluciones aproximadas.

Los métodos de segmentación que existen actualmente son costosos en tiempo y producen segmentos que no corresponden de forma precisa a los objetos de la imagen que representan. Por esta razón, muchos motores de anotación automática evitan el proceso de segmentación. Al hacer esto pierden la capacidad de conocer la posición, el tamaño y el número de apariciones de los objetos que describen con sus anotaciones.

En este trabajo describimos un método de segmentación basado en *quadtrees,* que divide la imagen recursivamente basándose en la información del color y de los bordes, capaz de dar resultados en poco tiempo y con segmentos grandes y homogéneos que proporcionan información suficiente para producir anotaciones correctas.

Nuestra propuesta de algoritmo produce segmentos homogéneos que en general son mejor anotados que los que producen *normalized cuts* y *rejillas*. Las razones de nuestra ventaja sobre éstos radican en que *normalized cuts* se obliga a producir un número fijo de segmentos y eso afecta mucho sus resultados. Además *normalized cuts* se obliga también a asignar todos los píxeles a un segmento mientras que nuestro algoritmo tiene la libertad de descartar del resultado de segmentación los píxeles que considere que no aportarán información útil para la anotación. Respecto a *rejillas* nuestra ventaja es que nuestros segmentos son más grandes y por tanto proporcionan más información al motor de anotación. Cabe también mencionar que nuestro algoritmo depende mucho de la homogeneidad y bordes bien definidos en las imágenes. Cuando falta alguno de estos dos elementos nuestros resultados disminuyen su precisión. Por otro lado, debido a la rapidez con que segmenta nuestro algoritmo es un candidato viable para ser utilizado en sistemas que requieren anotar imágenes rápidamente.

6.2 Trabajo Futuro

Hemos mostrado y explicado los resultados de nuestro algoritmo, los cuales son en términos generales buenos. Sin embargo, existen varios puntos sobre los que se puede seguir trabajando para perfeccionar sus resultados. Sin limitarse sólo a estos puntos, queremos trazar lo que podrían ser las principales líneas de trabajo:

a) Hacer más efectiva la detección de contornos en la imagen. Adaptando los umbrales de varianza y selección de píxeles al contenido de la imagen. Esto permitiría perfeccionar la imagen de restricciones que guía la segmentación.

b) Efectuar un post procesamiento, tipo *Canny*, a los contornos encontrados en la imagen para poder unir aquellos contornos que son uno sólo pero han quedado separados en 2 o más contornos. Esto también perfeccionaría la imagen de restricciones.
c) Refinar la forma en que son interpretados los bordes de restricciones dentro de cada par de *quads* que se desee unir.
d) Tomar en cuenta propiedades de textura (que sean de cómputo rápido) en el comparador de regiones. (Sección 2.2.2 y Sección 4.6.2).
e) Hacer un auto-evaluador que, en base a la homogeneidad, de los segmentos encontrados, determine sí es posible mejorar la segmentación cambiando el valor de alguno de los parámetros que se utilizaron en la ejecución del algoritmo.
f) Estudiar si existe alguna relación formal entre los parámetros *s* y *h* del algoritmo. (Figura 5.8 a Figura 5.12).

Referencias

[A. Yuille, 1992] A. Yuille and P. Hallinan, “Deformable templates”. In A. Blake and A. Yuille, editors, Active Vision, pages 21–38, MIT press, 1992.

[Alexei Yavlinsky, 2004] Alexei Yavlinsky, Edward Schoeld1;2 and Stefan Ruger1, “Automated Image
Annotation Using Global Features and Robust Nonparametric Density Estimation”, 2004.

[Anat Levin , 2006] Anat Levin and Yair Weiss, “Learning to Combine Bottom-Up and Top-Down Segmentation”, 2006.

[Anil K. Jain, 1988] Anil K. Jain and Richard C. Dubes, “Algorithms for Clustering Data”, 1988.

[Aude Oliva, 2001] Aude Oliva and Antonio Torralba, “Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope”, 2001.

[Baris Sumengen, 2004] Baris Sumengen and B.S. Manjunath, “Multi-Scale Edge Detection and Image Segmentation”, 2004.

[D. G Lowe, 2004] D. G Lowe, “Distinctive image features from scaleinvariant keypoints,” Int. J. Comput. Vision, vol. 60, no. 2, pp. 91–110, 2004.

[Eran Borestein, 2002] Eran Borestein and Shimon Ullman, “Class Specific, Top-Down Segmentation”, 2002.

[Eran Borestein, 2003] Eran Borestein and Shimon Ullman, “Learning to segment”, 2003.

[Eran Borestein, 2004] Eran Borenstein, Eitan Sharon and Shimon Ullman, “Combining Top-down and Bottom-up Segmentation”, 2004.

[Florent Monay, 2004] Florent Monay and Daniel Gatica-Perez, “PLSAbased Image AutoAnnotation: Constraining the Latent Space”, 2004.

[Dagan Feng, 2003] David Dagan Feng,Wan-Chi Siu and HongJiang Zhang, “Multimedia information retrieval and management: technological fundamentals.”, chapter 1: “Fundamentals of content-based image retrieval”, 2003.

[G. H. Ball, 1964] G. H. Ball and D. I. Hall, “Some fundamental concepts and synthesis procedures for pattern recognition preprocessors”, 1964.

[Greg Pass , 1996] Greg Pass and Ramin Zabih, “Histogram Refinement for Content-Based Image Retrieval”, 1996.

[Greg Pass, 1999] Greg Pass and Ramin Zabih, “Comparing Images Using Joint Histograms”, 1999.

[H. Tamura, 1978] H. Tamura, S. Mori, and T. Yamawaki, "Texture features corresponding to visual perception," IEEE Trans. On Systems, Man, and Cybernetics, vol. Smc-8, No. 6, June 1978.

[Hanan Samet, 1984] Hanan Samet, “The *quadtree* and Related Hierarchical Data Structures”, 1984.

[Herve Glotin, 2005] Herve Glotin and Sabrina Tollari, “Fast Image Auto-Annotation With Visual Vector Approximation Clusters”, 2005.

[Hunter G. M., 1979] Hunter G. M., Steiglitz K., “Operations on images using *quadtrees*”, 1979.

[J. B. MacQueen, 1967] J. B. MacQueen, "Some Methods for classification and Analysis of Multivariate Observations, Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability", 1967.

[J. Jeon, 2003] J. Jeon, V Lavrenko, and R Manmatha., “Automatic image annotation and retrieval using cross-media relevance models,” in SIGIR ’03, 2003, pp. 119–126.

[J. Li, 2000] J. Li, R. M. Gray, Image Segmentation and Compression Using Hidden Markov Models, Kluwer Academic Publishers, 2000.

[J. Shi, 2000] J. Shi and J. Malik, “Normalized Cuts and Image Segmentation,” IEEE Trans. Pattern Analysis and Machine Intelligence, 22(8):888–905, 2000.

[James Z. Wang, 2003] James Z. Wang and Jia Li, “Automatic Linguistic Indexing of Pictures by a Statistical Modeling Approach”, 2003.

[Jaume Vergés] Jaume Vergés-Llahí, State-of-the-Art Survey on Color Segmentation Methods.

[Jianping Fan, 2004] Jianping Fan, Yuli Gao, Hangzai Luo and Guangyou Xu, “Automatic Image Annotation by Using Concept-Sensitive Salient Objects for Image Content Representation“, 2004.

[Jiayu Tang, 2006] Jiayu Tang, Jonathon S. Hare and Paul H. Lewis, “Image Auto-Annotation Using A Statistical Model With Salient Regions”, 2006.

[Jing Huang, 1997] Jing Huang, S Ravi Kumar, Mandar Mitra, Wei-Jing Zhu and Ramin Zabih, “Image Indexing Using Color Correlograms” 1997.

[Jiwoon Jeon, 2004] Jiwoon Jeon and R. Manmatha, “Using Maximum Entropy for Automatic Image Annotation”, 2004.

[Kobus Barnard, 2001] Kobus Barnard, Pinar Duygulu, and David Forsyth, “Clustering Art”, 2001.

[Le Thi, 2004] Le Thi Lan and Alain Boucher, “An Interactive Image Retrieval System: From Symbolic to Semantic,” 2004.

[Linda Shapiro, 2003] Linda Shapiro and George Stockman, “Computer Vision”, 2003.

[M. Kass, 1987] M. Kass, A. Witkin and D. Terzopoulos. “Snakes: Active Contour Models”. Proc. Int. Conf. on Computer Vision ICCV87. London, England. 1987.

[Masashi Inoue, 2004] Masashi Inoue, “On the need for annotation based image retrieval”, 2004.
[Meng Zhu, 2007] Meng Zhu, “Semantic-associative visual content labelling and retrieval: A multimodal approach”, 2007.

[Michela Lecca, 2007] Michela Lecca, “An object recognition system for automatic image annotation and browsing of object catalogs”, 2007.

[P. Duygulu, 2002] P. Duygulu, K. Barnard, N. de Freitas, and D. Forsyth, “Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary.” In Seventh European Conference on Computer Vision, pages 97-112, 2002.

[Ranade S, 1980] Ranade S., “Use of trees for edge enhancement”, 1980.

[Ranade S, 1981] Ranade S., Shneier M., “Using *quadtrees* to smooth images”, 1981.

[Rong Jin, 2004] Rong Jin, Joyce Y. Chai and Luo Si , “Effective Automatic Image Annotation Via A Coherent Language Model and Active Learning”, 2004.

[S. L. Feng, 2004] S. L. Feng, R. Manmatha and V. Lavrenko, “Multiple Bernoulli Relevance Models for Image and Video Annotation“, 2004.

[Serge Belongie, 1998] Serge Belongie, Chad Carson, Hayit Greenspan, and Jitendra Malik, “Color- and Texture-Based Image Segmentation Using EM and Its Application to Content-Based Image Retrieval,” IEEE Proc. Int. Conf. Comp. Vision 1998.

[Song Chun Zhu, 1996] Song Chun Zhu and Alan Yuille, "Region Competition: Unifying Snakes, Region Growing and Bayes/MDL for Multi-band Image Segmentation," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 18, No. 9, 1996.

[Sophie Liu Xiao Fan, 2004] Sophie Liu Xiao Fan, Zhihong Man and Rashi Samur, "Edge Based Region Growing – A New Image Segmentation Method," 2004.

[Swain, 1991] M. J. Swain, and D. H. Ballard, "Color indexing," International Journal of Computer Vision, Vol. 7, No. 1, pp.11-32, 1991.

[T. Pavlidis, 1972] T. Pavlidis, "Segmentation of Pictures and Maps through Functional Approximation", 1972.

[Thomas Leung, 1998] Thomas Leung and Jitendra Malik, "Contour continuity in region based image segmentation", 1998.

[Tony F. Chan, 2001] Tony F. Chan and Luminita A. Vese, "Active Contours Without Edges", 2001.

[Tucker L.W., 1984] Tucker L.W., "Computer vision *quadtree* refinement", 1984.

[W.Y. Ma, 1997] W.Y. Ma and B.S. Manjunath, "Edge Flow: A Framework of Boundary Detection and Image Segmentation," 1997.

[Wu A. Y., 1982] Wu A. Y., Hong T. H., Rosenfeld A., "Threshold selection using *quadtrees*", 1982.

[Yuli Gao, 2006] Yuli Gao, Jianping Fan, Xiangyang Xue and Ramesh Jain, "Automatic Image Annotation by Incorporating Feature Hierarchy and Boosting to Scale up SVM Classifiers", 2006.

[Zhen Wu, 1993] Zhen Wu and Richard Leahy, "An Optimal Graph Theoretic Approach to Data Clustering: Theory and Its Application to Image Segmentation", 1993.

[Url1] Weisstein, Eric W. "K-Means Clustering Algorithm." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/K-MeansClusteringAlgorithm.html

[Url2] http://www.yale.edu/ceo/Projects/swap/landcover/Unsupervised_classification.htm

[Url3] http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

[Url4] http://en.wikipedia.org/wiki/Wikipedia:Picture_of_the_day/March_2007

[Url5] http://www.uio.no/studier/emner/matnat/ifi/INF3300/h06/undervisningsmateriale/segmentation_2006.pdf

[Url6] http://cs.gmu.edu/~kosecka/cs682/cs682-segmentation.pdf

[Url7] http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html

[Url8] http://www.emsps.com/photocd/corelcds.htm

Anexo A - *Quadtrees*

Introducción

Finkel y Bentley[Finkel R.A., 1974] utilizaron el término *quadtree* en 1974 para referirse a una partición del espacio en cuadrantes rectangulares. Consistía en la adaptación del árbol de búsqueda binaria a dos dimensiones (que podía ser fácilmente extendido a un número arbitrario de dimensiones).

Los orígenes del principio de descomposición recursiva en el cual están basados los *quadtrees* es difícil de conocer. Sin embargo, podemos rastrear sus primeras aplicaciones en las áreas de geometría. Lo más seguro es que el primer uso que se dio a los *quadtrees* fue como una forma fácil y óptima, de agregar ceros a matrices dispersas. Hoare [Hoare C.A.R., 1972] atribuye a Dijkstra la descomposición de una matriz en bloques cuadrados. Warnock [Warnock J.L, 1969] implementó un algoritmo de eliminación de superficies ocultas utilizando descomposición recursiva en una escena para buscar áreas rectangulares pequeñas y sencillas de desplegar. El robot del proyecto SRI [Nilsson N.J., 1969] utilizaba una descomposición del espacio en tres niveles para representar un mapa del robot. Tucker [Tucker L.W., 1984] utilizó refinamiento con *quadtrees* como una estrategia de control para un sistema de visión experto.

Procesamiento de Regiones con quadtrees

El *quadtree de regiones* (en adelante sólo *quadtree*) se caracteriza por ser una colección de bloques maximales que particionan una región dada. La representación mas simple de un *quadtree* consiste en una lista de bloques de tamaño *m*m* que corresponden a cada región en la máxima resolución de la descomposición.

Un *quadtree* requiere que sus bloques sean disjuntos, que sean de tamaños preestablecidos (eg. potencias de 2) y de posiciones determinadas. La motivación para su desarrollo fue obtener un método sistemático para encontrar y representar las partes homogéneas de una imagen.

Para poder transformar la información de la imagen en un *quadtree* necesitamos un criterio para decidir si una región es homogénea (uniforme). Uno de estos criterios puede ser que la desviación estándar de la intensidad de color este por debajo de un umbral dado. Utilizando este criterio, la imagen es subdividida en cuadrantes, subcuadrantes, etc. hasta que todos los bloques del *quadtree* correspondan a regiones homogéneas. Este proceso nos lleva a una descomposición regular. Si asociamos a cada nodo del *quadtree* la media de intensidad de color del bloque al que pertenece, el *quadtree* resultante sería una completa aproximación de la imagen. Si solicitamos que el umbral de homogeneidad sea 0, entonces tendremos que una región nunca será homogénea a menos que sea toda del mismo color. Este caso es de particular interés puesto que permite la total reconstrucción de la imagen a partir del *quadtree*.

Los bloques de un *quadtree* no necesariamente corresponden a regiones maximales de la imagen. Normalmente, existen bloques que pueden unirse. Para segmentar una imagen en bloques maximales, es necesario que permitamos la fusión de bloques adyacentes siempre y cuando la región resultante siga siendo homogénea. Esto puede realizarse a través de algoritmos de *split and merge*. Sin embargo, después de estas uniones, la estructura resultante dejará de ser un *quadtree* y se convertirá en un grafo de adyacencia. De esta manera, los *quadtrees* pueden ser utilizados como el paso inicial en el proceso de segmentación de imágenes. La Figura 6.1 muestra un ejemplo de segmentación basada en *quadtrees*.

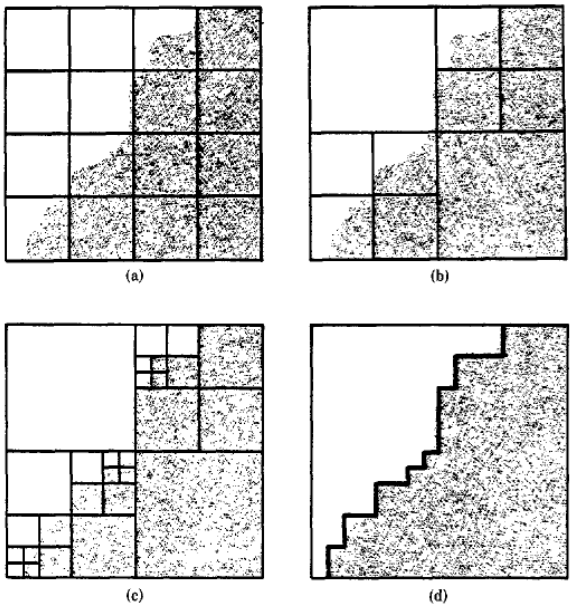


Figura 6.1: Ejemplo de segmentación de una imagen utilizando como base quadtrees. a) b) y c) Ilustran el proceso de la descomposición recursiva. d) Ilustra como luciría el resultado después de aplicar un algoritmo de split and merge a c). La estructura resultante en d) no es un quadtree, sin embargo, fue obtenida de uno.

Una alternativa a la estructura clásica de los *quadtrees* consiste en utilizar un método de descomposición irregular, por ejemplo, rectángulos de tamaño arbitrario en lugar de cuadrados. Esta alternativa presenta como ventaja principal la disminución del espacio requerido para almacenar el *quadtree* en memoria; mientras que la principal desventaja consiste en que la determinación de los puntos donde se hará la partición requiere de un proceso de búsqueda que agrega tiempo de ejecución.

Características de los quadtrees

Cualquier representación para la descomposición de imágenes debe poseer las siguientes características:

(1) La partición debe consistir de un patrón infinitamente repetitivo de tal manera que pueda ser utilizado para imágenes de cualquier dimensión.
(2) La partición debe ser infinitamente descomponible en un patrón cada vez más fino.

Existen muchas formas de particionar recursivamente una imagen. Según el patrón que utilicemos podemos cumplir una o ambas características. La Figura 6.2 nos muestra un ejemplo de diferentes patrones de descomposición recursiva.

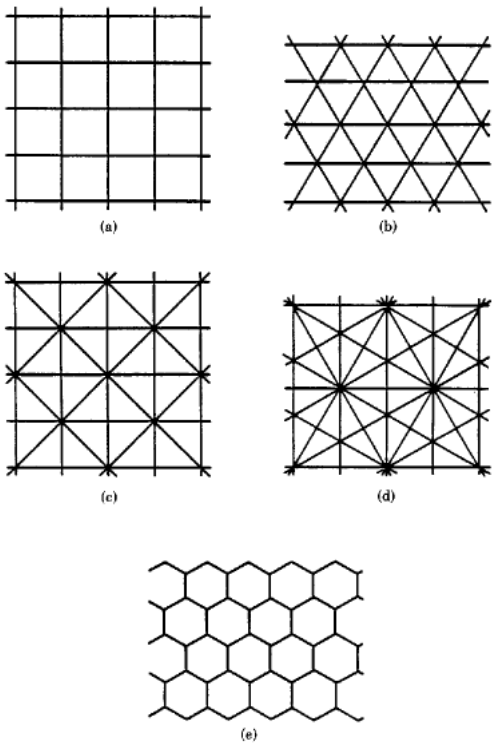


Figura 6.2: Diferentes patrones para dividir recursivamente una imagen. (a) cuadrado, (b) triángulo equilátero, (c) triángulo isóceles, (d) triángulo 30-60, (e) hexágono.

También existe la descomposición parcial, la cual se basa en componentes que siempre pueden ser unidos en unidades más grandes pero no siempre pueden ser divididos en unidades más pequeñas. Tal es el caso de los *septrees*. Los *septrees* son un estructura de descomposición recursiva que tiene como célula base un hexágono. Debido a que un hexágono no puede dividirse en más hexágonos, la base de la descomposición es un conjunto de 7 hexágonos que forman una estructura base capaz de ser embonada y agrupada con otras figuras iguales a ella hasta lograr una figura de la misma forma pero más grande, formada nuevamente por 7 figuras básicas. Este patrón puede reproducirse

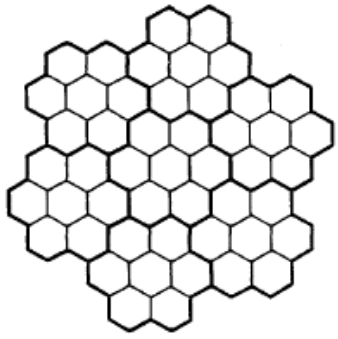


Figura 6.3: Septree. Patrón capaz de agruparse infinitamente pero no de descomponerse infinitamente.

de manera infinita hacia regiones cada vez más grandes pero no así para regiones cada vez más chicas, Figura 6.3.

Localización de Vecinos

Cada nodo del *quadtree* corresponde a un bloque en la imagen original. En un *quadtree* clásico, cada bloque tiene cuatro lados y cuatro esquinas. Llamamos a cada uno de los lados del bloque N, E, S y W y a las cuatro esquinas NW, NE, SW y SE.

Dados dos nodos P y Q, cuyos respectivos bloques no se sobreponen, y una dirección D, decimos que P y Q son *adyacentes* si existen dos píxeles p y q, contenidos en P y Q respectivamente, tales que q es adyacente al lado D de p o la esquina D de p es adyacente a la esquina opuesta de q.

Dado un nodo P y una dirección D, es común que exista más de un nodo Q adyacente. Debido a esto, es necesario dar información más precisa para localizar un bloque vecino, como si el vecino Q es adyacente a todo el lado D de P o sólo a una parte.

Para nombrar a los vecinos de P, utilizamos las siguientes expresiones. G significa mayor o igual, C significa esquina, S significa lado y N significa vecino:

1) **GSN**(P,D) = Q. El nodo Q corresponde al bloque más pequeño que es adyacente al nodo P en la dirección D y tiene un tamaño mayor o igual al del bloque correspondiente a P.
2) **CSN**(P, D, C) = Q. El nodo Q corresponde al bloque más pequeño que es adyacente al lado D de la esquina C del nodo P.
3) **GCN**(P, C) = Q. El nodo Q corresponde al bloque más pequeño que es contrario a la esquina C del nodo P y es de tamaño mayor o igual al bloque correspondiente a P.
4) **CCN**(P, C) = Q. El nodo Q corresponde al bloque más pequeño que es contrario a la esquina C del nodo P.

Para aclarar mejor estos conceptos, nos referiremos a la Figura 4.12 donde, F, G, I e H son los respectivos hijos NW, NE, SE y SW de C. Además, GSN(J, S) = L, CSN(J, E, SE) = 39, GCN(H, NE) = K y CCN(H, SW) = 38. En un *quadtree*, un bloque P no puede ser adyacente a dos bloques de tamaño mayor ya sea en lados opuestos o en esquinas opuestas.

Un bloque que no es adyacente al borde de la imagen tiene un mínimo de 5 vecinos y un máximo de 8. En caso de tener 8 vecinos todos ellos, excepto uno de la esquina, son del mismo tamaño.

Necesitamos ser capaces de localizar vecinos sin importar la localización o el tamaño del bloque. Además, no debemos crear ningún puntero a los nodos adyacentes, cada bloque sólo debe poseer 4 punteros a sus nodos hijos y 1 puntero a su nodo padre. El uso de más punteros haría que nuestro árbol dejara de ser un *quadtree*.

Localizar vecinos horizontales o verticales es muy sencillo. La idea básica consisten en recorrer ascendentemente el *quadtree* hasta encontrar un ancestro común con el vecino buscado, después descender en busca del nodo vecino. Debemos tomar en cuenta que los vecinos horizontales o verticales de un bloque no necesariamente son del mismo tamaño.

Existen algoritmos que agregan punteros entre los nodos vecinos al construir el *quadtree*. Con estos punteros logran que la búsqueda de vecinos sea mucho más directa. Todos los bloques vecinos del mismo tamaño, tienen un puntero entre ellos para indicar que son vecinos. Esto no es así cuando los vecinos no tienen el mismo tamaño, en este caso es necesario subir niveles en el *quadtree* hasta encontrar un nivel en el que el nodo visitado cuente con punteros a sus vecinos. Una vez hallado este nodo, podemos asegurar que sus vecinos son también vecinos de sus nodos hijo. En este esquema, los punteros que indican vecinos horizontales o verticales a lo largo de los niveles del árbol son llamados *cuerdas.*

Otra forma de representar la estructura de *quadtree* con punteros es la *red.* En ella, no tenemos que subir niveles para encontrar los vecinos de los nodos pues cada nodo cuenta con punteros específicos a sus nodos vecinos.

La ventaja de las cuerdas y las redes es que la cantidad de nodos que son visitados para encontrar vecinos es reducida. Sin embargo, la desventaja es el aumento en los requisitos de espacio en memoria. En promedio, para encontrar un vecino para un nodo dado en un *quadtree* convencional, utilizando la técnica del ancestro común, mínimo 4 nodos deben ser visitados mientras que en una estructura de cuerdas, un mínimo de 2 nodos debe ser visitado.

Representaciones basadas en punteros

La forma más natural de implementar un *quadtree* es utilizando una estructura de árbol donde cada nodo tiene cuatro punteros, uno para cada uno de sus hijos. Para facilitar algunas operaciones, puede agregarse un puntero adicional al nodo padre, de esta forma podemos recorrer libremente el árbol en ambas direcciones (arriba y abajo). Este esquema es ampliamente utilizado por muchos algoritmos de procesamiento de imágenes. Una representación alternativa es el denominado *bintree* en el cual el espacio es siempre dividido en dos partes alternando la división sobre los ejes X y Y. La ventaja de esta representación es que cada nodo posee solamente 2 punteros a sus hijos en lugar de 4. Además, su utilización generalmente da un menor número de nodos hoja. Esta estructura es particularmente atractiva cuando se trata de trabajar con datos de muchas dimensiones ya que se desperdicia menor espacio en los punteros que apuntan al valor *nulo*. La Figura 6.4 muestra el *bin-tree* equivalente al *quadtree* de la Figura 4.12.

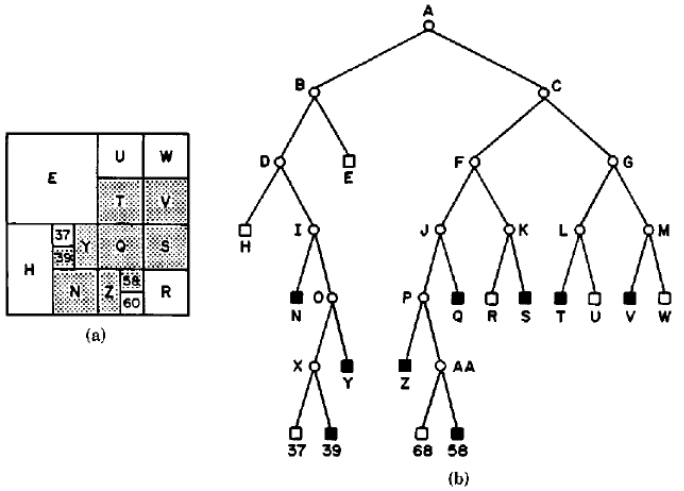


Figura 6.4: Bintree correspondiente a la Figura 3.1

El mayor problema de la representación con árboles de los *quadtrees* radica en la cantidad de memoria utilizada. No solamente necesitamos espacio para los bloques del *quadtree*, sino también para indicar los nodos intermedios en la búsqueda de bloques homogéneos y éstos nodos a su vez tiene sus propios punteros. Esta desventaja se vuelve mayor cuanto más grandes son las imágenes con las que tratamos puesto que pude ser que ocupen más espacio del que tenemos disponible en memoria. Debido a esto, se han ideado múltiples alternativas para representar un *quadtree* sin la utilización de punteros.

Representaciones sin punteros

Cuando una imagen es representada sólo por sus nodos hoja (bloques terminales), cada hoja puede ser codificada utilizando un número en base 4 llamado *código de localización*. Dichos códigos indican como localizar el bloque correspondiente desde el nodo raíz del *quadtree*. También existe una variante en base 5. Para eficientar aún más el uso de espacio, no hay necesidad de almacenar al código de localización de cada nodo hoja. Basta con que almacenemos solo los nodos NEGROS. Los códigos para los nodos BLANCOS pueden ser obtenidos a partir los primeros sin tener que construir físicamente el *quadtree*; todo esto, en una imagen blanco y negro. En este enfoque es posible acceder a los vecinos de cada nodo realizando operaciones sobre el código de localización.

Jones and Iyengar[1984] introdujeron el concepto de "bosque de *quadtrees*" que consiste en la descomposición de un *quadtree* en una colección de subquadtrees donde cada subquadtree corresponde a un bloque maximal. Un nodo interno del *quadtree* se dice de tipo GB si al menos dos de sus hijos son NEGROS, en cualquier otro caso se dice que el nodo es de tipo GW. Los nodos NEGROS y los nodos GB son considerados bloques maximales. Un bosque de *quadtrees* es el conjunto de bloques maximales que no esta contenido dentro de ningún otro bloque maximal y separan la región NEGRA de la imagen. Cada elemento del bosque es identificado por un **código de localización** en base

4. Esta representación resulta en ahorro de espacio debido a que los grandes espacios BLANCOS son ignorados.

Otra representación sin punteros consiste en realizar un recorrido en preorden del *quadtree* e ir creando una **cadena de caracteres** según el tipo de bloque encontrado durante el recorrido. Un "(" corresponde a un bloque GRIS, un "B" corresponde a un bloque NEGRO y un "W" corresponde a un bloque BLANCO. Esta representación es conocida como expresión DF. Esta representación es obra de Kawaguchi y Endo [Kawaguchi E., 1980]. Por ejemplo, la Figura 4.12 da como resultado la siguiente expresión-DF:

```
(W(WWBB(W(WBBBWB(BB(BBWW
```

Los hijos de cada nodo son recorridos en el siguiente orden: NW, NE, SW, SE. La imagen original puede ser reconstruida a partir de la expresión-DF.

Existe un enfoque muy similar llamado **codificación autoadaptativa de bloques**. La diferencia radica en que el alfabeto utilizado consiste solamente de "0" y "1", donde el "0" corresponde a los bloques BLANCOS y el "1" significa que es necesaria una subdivisión en ese punto.

Se ha demostrado que ambos métodos utilizan aproximadamente la misma cantidad de espacio.

Kawaguchi mostró que es posible realizar operaciones básicas de procesamiento de imágenes con una imagen representada por su expresión-DF, tales como calcular el centroide, rotación, escalamiento y traslación.

Es posible realizar búsqueda de bloques vecinos cuando utilizamos estructuras basadas en recorrido del *quadtree*, sin embargo los métodos para esto son generalmente más elaborados y requieren más tiempo que con las estructuras basadas en punteros. Es por esto que de ahora en adelante nos referiremos sólo a estructuras basadas en punteros.

Transformación

La representación en *quadtree* de imágenes binarias se propone debido a que su naturaleza jerárquica optimiza el rendimiento de un amplio número de operaciones. Sin embargo, la mayoría de las imágenes son generalmente representadas con otras estructuras como: arreglos binarios, *rasters*, códigos de cadenas y polígonos. Algunas de estas representaciones son particularmente útiles para ciertas tareas, por ejemplo: para gráficar en la memoria de video, es mucho más útil un *raster*. Debido a esto, siempre es necesario contar con algoritmos que nos permitan pasar de una estructura a otra con facilidad. En nuestro caso particular, existen muchos métodos para transformar un *quadtree* a otras estructuras.

La representación más común de una imagen es el arreglo de dos dimensiones. Existen muchas formas de construir un *quadtree* a partir de un arreglo. El enfoque más simple convierte todo el arreglo en un *quadtree*, un nodo por cada píxel. Una vez construido, el *quadtree* resultante, sus nodos hoja son examinados en busca de 4 nodos del mismo color para poder eliminarlos y dejar solo a su padre, de esta manera se va reduciendo la resolución del *quadtree* y el número de nodos. El proceso se repite hasta que ya no se pueda eliminar ningún nodo. Este enfoque es muy simple pero sumamente costoso en tiempo y en espacio dado que se crean mucho nodos innecesarios. De hecho, es incongruente que se ocupe demasiada memoria cuando se utilizan algoritmos basados en *quadtrees*.

Un método mucho más optimo consiste visitar primero los píxeles de la imagen y crear nodos hoja sólo hasta que se este seguro de que hemos encontrado una región maximal. Una forma de hacer esto es ir recorriendo las regiones de la imagen que sabemos corresponden a un quad, si todos los píxeles son del mismo tamaño, hemos encontrado una región maximal y podemos generar un nodo, en caso contrario, al primer píxel distinto, debemos avanzar un nivel en la resolución del *quadtree* y verificar todos los píxeles correspondientes a los quads de esta resolución. Este proceso se repite hasta encontrar todas las regiones maximales de todas la resoluciones.

También existe el caso contrario, cuando deseamos convertir un *quadtree* a un arreglo. Numerosos algoritmos existen para este caso. Básicamente, consisten en realizar un recorrido por el *quadtree* hasta llegar a sus nodos hojas para rellenar con píxeles del color del nodo a la región correspondiente en el arreglo. Este recorrido puede ser desde las hojas del árbol (*bottom-up*) o desde la raíz del árbol (*top-down*). Normalmente los algoritmos *bottom-up* visitan menos veces los nodos del *quadtree*.

Otra transformación común es pasar una estructura de *quadtree* a una cadena de códigos DF lo cual normalmente puede ser logrado con un recorrido en profundidad del *quadtree*.

Operaciones

Una de las grandes ventajas de los *quadtrees* es que muchas operaciones básicas pueden implementarse como cortes en el árbol. Los *quadtrees* son especialmente útiles para realizar operaciones binarias tales como la unión y la intersección de mapas de bits.

Para realizar la **unión** de S y T recorremos ambos *quadtrees* paralelamente, comparamos los nodos correspondientes en ambos *quadtrees* y construimos el *quadtree* resultante, U. Si, en la comparación de S y T, cualquiera de los dos nodos es NEGRO el nodo correspondiente en U es NEGRO. Si uno de los nodos es BLANCO, en S, el nodo resultante, en U, se pone al mismo valor del nodo correspondiente en T. Si ambos nodos son GRIS, el nodo en U es GRIS. El algoritmo se aplica recursivamente a todos los hijos de S y T. Existe un caso de particular atención: cuando dos nodos son GRIS, debemos

revisar si al combinar los nodos hijo de S y de T, obtenemos cuatro nodos NEGROS que den lugar a un solo nodo, entonces este nodo deberá ser color NEGRO. Podemos observar un ejemplo de unión de dos *quadtrees* en la Figura 6.6.
La **intersección** de dos *quadtrees* es igual de simple solo que los roles de los nodos BLANCO y NEGRO son invertidos: cuando dos nodos son GRIS, revisamos si existen cuadro nodos hijo BLANCOS, en S y T y realizamos la unión correspondiente para formar un solo nodo BLANCO. La Figura 6.6 ilustra un ejemplo de la intersección de dos *quadtrees*.

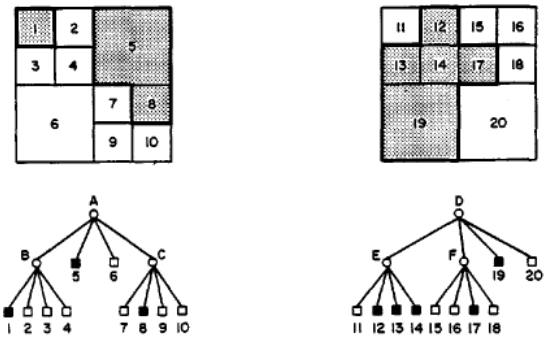


Figura 6.5: Dos imágenes y sus respectivos quadtrees.

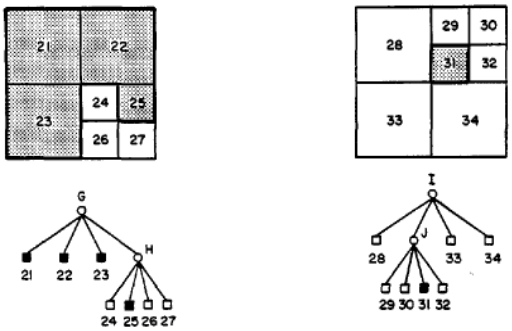


Figura 6.6: Unión (izquierda) e Intersección (derecha) de los quadtrees de la Figura 6.5.

Escalar el *quadtree* por una potencia de 2 es trivial (eg. nodos hoja formados por 8 píxeles en lugar de 4), sólo es necesario reducir un nivel su resolución. La **rotación** en múltiplos de 90 grados es igualmente simple. Consiste en la rotación recursiva de los hijos en cada nivel del *quadtree*. Los hijos NW, NE, SW y SE pasan a ser los hijos SW, NW, SE y NE respectivamente. La Figura 6.7 muestra un ejemplo de la rotación de *quadtrees*.

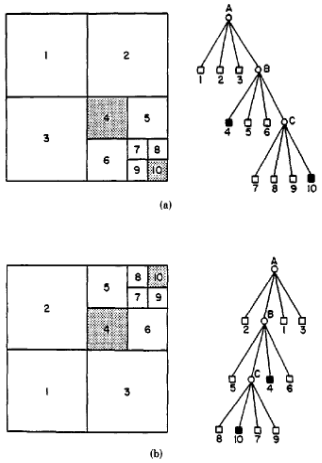


Figura 6.7: a) Imagen y quadtree original. b) Imagen y quadtree después de rotar la imagen 90 grados.

Otra operación útil en aplicaciones gráficas es conocida como *windowing* la cual es el proceso de extraer una región rectangular de una imagen representada por un *quadtree* y construir un nuevo *quadtree* para dicha región.

La mayoría de estas operaciones tienen un tiempo de ejecución proporcional al número de nodos en el *quadtree*.

Los *quadtrees* también han sido utilizados para operaciones de procesamiento de imágenes en escala de grises como la segmentación[Ranade S., 1980], mejoramiento de contornos[Ranade S., 1908], suavizado de imágenes[Ranade S., 1981] y selección de umbrales[Wu A. Y., 1982].

La habilidad de los *quadtrees* para realizar operaciones de conjuntos es una de las principales razones de su gran popularidad y preferencia sobre otras alternativas de representación como las cadenas de códigos.
Extracción de características utilizando *quadtrees*
A continuación mencionaremos algunas de las operaciones más cómunes con imágenes y la forma en que éstas pueden realizarse utilizando *quadtrees*. La mayoría de estas operaciones tienen un tiempo de ejecución proporcional al número de nodos en el *quadtree*.

Área y Momentos

Calcular el área y momentos de imágenes representadas por *quadtrees* es muy simple. Para calcular el área del *quadtree* realizamos un recorrido en postorden sumando el tamaño de todos los bloques NEGROS. Los momentos de la imagen se obtienen de sumar los momentos de los bloques NEGROS. El conocimiento del área y de los primeros momentos permite el cálculo de las coordenadas del centroide y éste a su vez permite el cálculo de los momentos centrales.

Componentes Conectados

El etiquetado de componentes conectados es una de las operaciones básicas en procesamiento de imágenes. Existe un método tradicional que consiste en recorrer la imagen fila por fila de izquierda a derecha asignando la misma etiqueta a píxeles NEGROS adyacentes en las direcciones derecha e inferior. Durante este proceso, surgen pares de equivalencias entre etiquetas por lo que es necesario un proceso posterior para designar una única etiqueta para cada conjunto de equivalencias y realizar la actualización correspondiente en los píxeles de la imagen.

También podemos realizar el etiquetado de componentes conectados utilizando *quadtrees*. El primer paso consiste en un recorrido en postorden del *quadtree* en el cual, para cada nodo NEGRO que encontremos -y etiquetemos-, buscaremos sus vecinos (utilizando las técnicas de localización de vecinos antes mencionadas) y les asignaremos la misma etiqueta. Cuando los nodos visitados ya tengan una etiqueta asignada, almacenaremos la equivalencia entre la etiqueta actual y la ya encontrada. Un segundo paso, combina todos los pares de equivalencias encontrados. Finalmente, recorremos nuevamente el *quadtree* y actualizamos las etiquetas en cada nodo para reflejar las equivalencias calculadas en el paso 2.

Podemos observar que este proceso es análogo al método tradicional con la ventaja de que el tiempo de ejecución depende únicamente del número de bloques en la imagen y no de su tamaño, en el método tradicional el tiempo de ejecución depende del número de píxeles de la imagen. De esta forma podemos observar que las estructuras de *quadtree* no solamente ahorran espacio sino también tiempo.

Perímetro

Para encontrar el perímetro de un polígono relleno representado por un *quadtree*, realizamos un recorrido en postorden del *quadtree* y para cada nodo NEGRO que encontremos, revisamos sus cuatro lados adyacentes en búsqueda de nodos BLANCOS. Para cada nodo BLANCO encontrado aumentamos la longitud del lado de este bloque al perímetro. Esta alternativa realiza esfuerzo adicional cuando se trata revisar las adyacencias entre nodos NEGROS pues revisa, dos veces, cada lado de ellos. Por esto, existe un método alternativo que explora únicamente los vecinos sureste y este. Para nodos NEGROS busca vecinos sureste y este BLANCOS y para nodos BLANCOS busca vecinos sureste y este NEGROS acumulando la longitud del perímetro de la misma forma que el algoritmo anterior.

Conteo de Componentes

Una forma de saber el número de objetos en una imagen es utilizando el número de Euler, G que es igual a V – E + F. V, E y F corresponden al número de vértices, contornos y caras, respectivamente. En una imagen binaria, representada por un arreglo de dos dimensiones, V es el número de píxeles negros, E es el número de pares de píxeles

negros horizontal o verticalmente adyacentes y F es el número de bloques de 2x2 píxeles negros.

Dyer[1980] redefinió los valores de V, E y F en el contexto de un *quadtree*. V es el número de bloques NEGROS, E es el número de pares NEGROS adyacentes horizontal o verticalmente y F es el número de bloques de 2x2 píxeles que están contenidos en tres o cuatro bloques del *quadtree* (uniones de bloques).

Desventajas

La principal motivación para el desarrollo de *quadtrees* es el deseo de reducir la cantidad de espacio requerido para almacenar datos usando bloques homogéneos de información. Esto, además, trae como consecuencia una disminución en el tiempo de ejecución de múltiples operaciones. Sin embargo, el *quadtree* no es siempre la representación ideal. El peor de los casos para un *quadtree*, en términos de espacio, ocurre cuando la región corresponde a un patrón de tablero de ajedrez. La Figura 6.8 muestra con claridad esta desventaja.

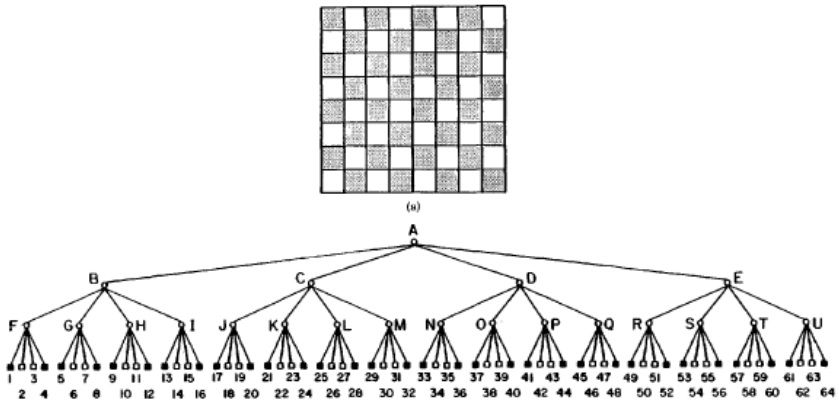


Figura 6.8: El peor de los escenarios para un quadtree

Otra desventaja importante de los *quadtrees* es que son extremadamente sensibles a la orientación y traslación de la imagen que representan. Rotar en un solo grado la imagen o trasladarla un solo píxel puede dar como resultado un *quadtree* completamente distinto al *quadtree* correspondiente a la imagen original. Esta desventaja ocasiona que el espacio ocupado por un *quadtree* representando a una misma imagen no sea siempre el mismo. La Figura 6.9 ilustra este problema.

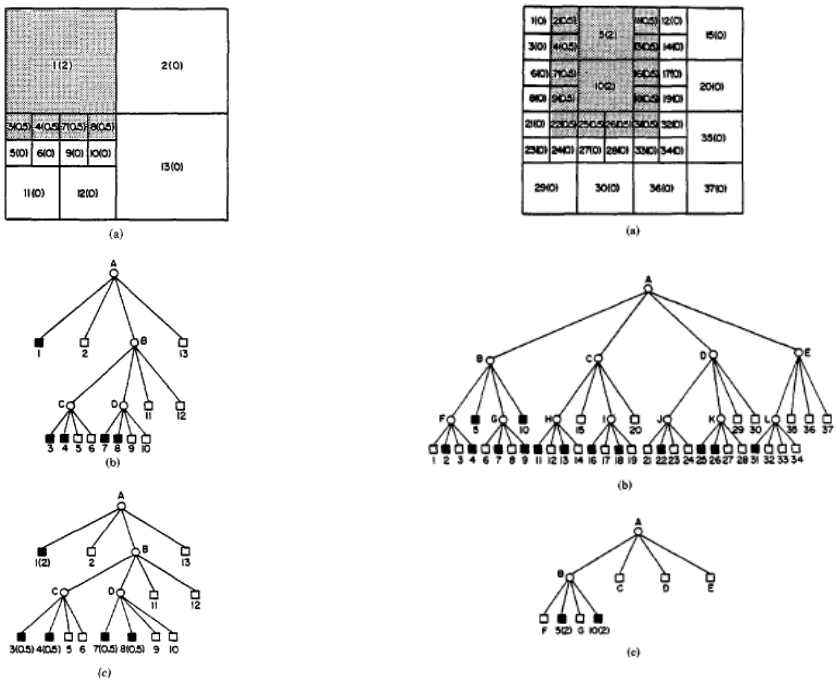


Figura 6.9: Ejemplo de la sensibilidad extrema de los quadtrees al desplazamiento de un solo píxel.

Existe una variante del *quadtree* que no tiene estos problemas llamada QMAT (*quadtree medial axis transform)*. La explicación de dicha estructura supera los alcances de este capítulo, para mayor información sobre esta estructura referirse a [Hanan Samet, 1984].

Aplicaciones

En [Hunter G. M., 1979] trabajaron con la representación de polígonos simples (sin agujeros) utilizando *quadtrees*. El *quadtree* resultante tiene tres tipos de nodos: interior, exterior y borde para indicar la parte del polígono que representa. Los nodos interiores y exteriores corresponden al área dentro y fuera del polígono. Éstos nodos pueden unirse con otros nodos adyacentes del mismo tipo para dar lugar a nodos que representen regiones de mayor tamaño. Los nodos de tipo borde no son unidos. Uno de los algoritmos que proponen consiste en lo siguiente: comenzar en el nodo raíz (todo el polígono) y dividirlo en cuatro regiones (cuadrantes) iguales. Cada nodo cuyos límites se intersecten con algún lado del polígono es dividido nuevamente. Este proceso continúa recursivamente hasta que se tienen nodos representando a todo el polígono. Este algoritmo no distingue entre los nodos exteriores e interiores por lo cual, después de construido el *quadtree*, se aplica un algoritmo de coloreado.

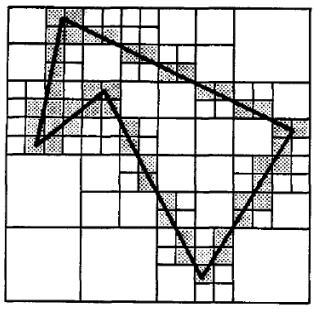


Figura 6.10: Ejemplo de la descomposición en quadtree de un polígono

[Hunter G. M., 1978] resolvió el problema de determinar si dos polígonos se intersectan utilizando *quadtrees*. Su solución consiste en construir el *quadtree* de cada uno de los polígonos y después intersectar ambos *quadtrees*. Si el *quadtree* resultante es nulo, entonces los polígonos no se intercectan. Sin embargo, esta solución esta sujeta a errores debido a las limitaciones propias del proceso de digitalización.

El *quadtree* también puede ser utilizado como un medio para aproximación de imágenes. Podemos obtener aproximaciones bruscas si simplemente ignoramos todos los nodos debajo de cierta resolución. Hay dos variantes de este método: aproximación interna y aproximación externa. La aproximación interna convierte todos los nodos GRIS en BLANCOS mientras que la aproximación externa convierte todos los nodos GRIS en NEGROS. Una aproximación más fina consiste en hacer NEGROS o BLANCOS todos los nodos GRIS en base al color de la mayoría de los píxeles que conforman el bloque que queremos descartar.

Los métodos de aproximación basados en *quadtrees* también han sido utilizados para la transmisión de imágenes en la cual se busca progresividad y compresión. Por progresividad entendemos la capacidad de que el dispositivo receptor pueda construir progresivamente una mejor aproximación de la imagen que recibe. Al final de la transmisión la imagen es construida perfectamente, la meta es recibir un bosquejo al principio y los detalles después, poco a poco, hasta formar completamente la imagen. Otra característica de este tipo de transmisión progresiva de datos es llamada *any-time* con lo cual se busca describir su capacidad para poder mostrar un resultado, aunque sea parcial, en cualquier instante de tiempo. Con esto la cantidad de tiempo disponible para llevar a cabo estas operaciones deja de ser un factor fundamental para su éxito.

La técnica más simple para esto es hacer un recorrido en anchura del *quadtree* e ir transmitiendo cada uno de los nodos, uno por uno, e ir avanzando de nivel hasta recorrer todo el *quadtree*. La desventaja de este método es que transmite mucha información redundante. Por ejemplo, cuando transmite un nodo GRIS y luego, los ese nodo también tiene hijos GRIS. En este caso no sólo no hay compresión durante la transmisión sino que además transmitimos más información de la necesaria. Existen muchas propuestas para

mejorar este método. Una de ellas es transmitir un bloque del *quadtree* sólo si su color es distinto al color de su bloque padre.

También existen algunos métodos de compresión que transmiten la imagen más rápido comprimiendo el *quadtree* antes de enviarlo. Una forma de hacer esto es que si un nodo GRIS tiene al menos tres hijos NEGROS, entonces lo convertimos en un nodo NEGRO y eliminamos a sus cuatro hijos. Este tipo de compresión es conocida compresión con pérdida de información ya que la imagen transmitida no es la misma que la original, aunque sí es una buena aproximación.

Anexo B – Manual de uso del segmentador

Clases ejecutables que segmentan imagenes utilizando nuestro algoritmo.

1. QuadTreeSegmentation.class

Parámetros de entrada desde la línea de comandos:

-i input: nombre del archivo de entrada [nombre_archivo.jpg]

-o output: nombre del archivo de salida
default: -i + _segments*

-q quadsize: tamaño mínimo del quad
default: 3

-h homogeneity th: umbral de homogeneidad entre regiones para ser consideradas "iguales" [0 - 1]
default: .10

-s sizeobject th: umbral de tamaño mínimo de los objetos a ser encontrados dentro de la imagen [0 - 1]
default: 0.025

-d debug: muestra información detallada del proceso de segmentación [false | true]
default: false

-gr gradient_image: mandar a un archivo el gradiente de la imagen [false | true]
default: false

-co constraint_image: mandar a un archivo la imagen de restricciones que guía la segmentación [false | true]
default: false

-qi quantized_image: mandar a un archivo la imagen con el espacio de colores discretizado [false | true]
default: false

-os oversegmentation: mandar a un archivo los resultados originales del algoritmo (sin ninguna mejora, sin unir las regiones pequeñas a las grandes) [true | false]
default: false

-ar annotated_result: mandar a un archivo los segmentos con los parámetros utilizados escritos dentro de la imagen [false | true]
default: false

-fs final_segmentation: mandar a un archivo los segmentos encontrados en la imagen [false | true]
default: true

-ov overlap_segmentation: mandar a un archivo la imagen original con los segmentos encontrados sobrepuestos [false | true]
default: false

-tr text_result: mandar a un archivo ASCII los resultados de la segmentación [true | false]
default: true

-sa segmentation_algorithm:
grid: Segmenta utilizando el segmentador en cuadros
En este caso es obligatorio especificar:
-w Número de columnas
-h Número de filas
default: Segmenta utilizando nuestra propuesta de segmentador

2. BatchSegmentator.class

-id input_directory: directorio de las imagenes de entrada
-od output_directory: direcotio de las imagenes de salida

Pueden también pasarse todos los parámetros utilizados en la clase QuadTreeSegmentation, excepto -i y -o

Ejemplos de uso:

> java QuadTreeSegmentation -i imagen.jpg
> java QuadTreeSegmentation -i directorio1/directorio2/imagen.jpg -o
> directorio3/resultado.jpg -ov true -q 2 -co true -h .05 -ar true
> java BatchSegmentator -id directorio1 -od directorio2
> java BatchSegmentator -id directorio1 -od directorio2 -ov true -q 2

Nota 1: Utilizar ultima versión de java: 1.6.01
Nota 2: Para leer archivos ASCII desde matlab:
http://web.cecs.pdx.edu/~gerry/MATLAB/plotting/loadingPlotData.html

Anexo C – Detalle de resultados del Conjunto A

Detalle de las imágenes del Conjunto A ilustrado en la Figura 5.1 y el tiempo que tomo segmentar cada imagen.

Imagen 1.- Ancho: 277 píxeles Alto: 320 píxeles. Segmentada en: 2620 milisegundos.
Imagen 2.- Ancho: 277 píxeles Alto: 320 píxeles. Segmentada en: 297 milisegundos.
Imagen 3.- Ancho: 320 píxeles Alto: 277 píxeles. Segmentada en: 203 milisegundos.
Imagen 4.- Ancho: 320 píxeles Alto: 277 píxeles. Segmentada en: 390 milisegundos.
Imagen 5 .- Ancho: 320 píxeles Alto: 216 píxeles. Segmentada en: 280 milisegundos.
Imagen 6.- Ancho: 320 píxeles Alto: 277 píxeles. Segmentada en: 500 milisegundos.
Imagen 7.- Ancho: 277 píxeles Alto: 320 píxeles. Segmentada en: 483 milisegundos.
Imagen 8.- Ancho: 320 píxeles Alto: 277 píxeles. Segmentada en: 265 milisegundos.
Imagen 9.- Ancho: 277 píxeles Alto: 320 píxeles. Segmentada en: 406 milisegundos.
Imagen 10.- Ancho: 320 píxeles Alto: 277 píxeles. Segmentada en: 921 milisegundos.
Imagen 11.- Ancho: 320 píxeles Alto: 213 píxeles. Segmentada en: 452 milisegundos.
Imagen 12.- Ancho: 320 píxeles Alto: 225 píxeles. Segmentada en: 421 milisegundos.
Imagen 13.- Ancho: 320 píxeles Alto: 240 píxeles. Segmentada en: 687 milisegundos.
Imagen 14.- Ancho: 800 píxeles Alto: 443 píxeles. Segmentada en: 2262 milisegundos.
Imagen 15.- Ancho: 800 píxeles Alto: 534 píxeles. Segmentada en: 4690 milisegundos.
Imagen 16.- Ancho: 303 píxeles Alto: 243 píxeles. Segmentada en: 468 milisegundos.
Imagen 17.- Ancho: 200 píxeles Alto: 300 píxeles. Segmentada en: 386 milisegundos.
Imagen 18.- Ancho: 116 píxeles Alto: 261 píxeles. Segmentada en: 140 milisegundos.

Se segmentaron 18 imagenes en 15.871 segundos.

Anexo D – Listados de ejecución del algoritmo

Listado 6.1: Reemplazos de etiquetas realizados durante la etapa de recorrido en quadtree.

La etiqueta (región) 7 fue sustituida por 1
La etiqueta (región) 12 fue sustituida por 5
La etiqueta (región) 14 fue sustituida por 4
La etiqueta (región) 13 fue sustituida por 5
La etiqueta (región) 13 fue sustituida por 5
La etiqueta (región) 9 fue sustituida por 2
La etiqueta (región) 16 fue sustituida por 2
La etiqueta (región) 16 fue sustituida por 2
La etiqueta (región) 17 fue sustituida por 9
La etiqueta (región) 9 fue sustituida por 2
La etiqueta (región) 18 fue sustituida por 1
La etiqueta (región) 18 fue sustituida por 1
La etiqueta (región) 10 fue sustituida por 1
La etiqueta (región) 10 fue sustituida por 1
La etiqueta (región) 10 fue sustituida por 1
La etiqueta (región) 11 fue sustituida por 3
La etiqueta (región) 23 fue sustituida por 12
La etiqueta (región) 23 fue sustituida por 12
La etiqueta (región) 12 fue sustituida por 5
La etiqueta (región) 24 fue sustituida por 1
La etiqueta (región) 26 fue sustituida por 1
La etiqueta (región) 26 fue sustituida por 1
La etiqueta (región) 27 fue sustituida por 1
La etiqueta (región) 27 fue sustituida por 1
La etiqueta (región) 28 fue sustituida por 6
La etiqueta (región) 30 fue sustituida por 5
La etiqueta (región) 31 fue sustituida por 6
La etiqueta (región) 31 fue sustituida por 6
La etiqueta (región) 14 fue sustituida por 4
La etiqueta (región) 36 fue sustituida por 2
La etiqueta (región) 35 fue sustituida por 1
La etiqueta (región) 35 fue sustituida por 1
La etiqueta (región) 37 fue sustituida por 2
La etiqueta (región) 9 fue sustituida por 2
La etiqueta (región) 36 fue sustituida por 2
La etiqueta (región) 38 fue sustituida por 2
La etiqueta (región) 38 fue sustituida por 2
La etiqueta (región) 39 fue sustituida por 9
La etiqueta (región) 17 fue sustituida por 9
La etiqueta (región) 17 fue sustituida por 9
La etiqueta (región) 41 fue sustituida por 19
La etiqueta (región) 41 fue sustituida por 19
La etiqueta (región) 19 fue sustituida por 3
La etiqueta (región) 9 fue sustituida por 2
La etiqueta (región) 3 fue sustituida por 2
La etiqueta (región) 45 fue sustituida por 20
La etiqueta (región) 47 fue sustituida por 4
La etiqueta (región) 50 fue sustituida por 21
La etiqueta (región) 21 fue sustituida por 4
La etiqueta (región) 10 fue sustituida por 1
La etiqueta (región) 53 fue sustituida por 3
La etiqueta (región) 53 fue sustituida por 3
La etiqueta (región) 11 fue sustituida por 3
La etiqueta (región) 11 fue sustituida por 3
La etiqueta (región) 54 fue sustituida por 3
La etiqueta (región) 53 fue sustituida por 3
La etiqueta (región) 55 fue sustituida por 3
La etiqueta (región) 11 fue sustituida por 3
La etiqueta (región) 55 fue sustituida por 11
La etiqueta (región) 55 fue sustituida por 11
La etiqueta (región) 56 fue sustituida por 5
La etiqueta (región) 57 fue sustituida por 25
La etiqueta (región) 57 fue sustituida por 25
La etiqueta (región) 25 fue sustituida por 1
La etiqueta (región) 61 fue sustituida por 1
La etiqueta (región) 62 fue sustituida por 28
La etiqueta (región) 62 fue sustituida por 28
La etiqueta (región) 6 fue sustituida por 5
La etiqueta (región) 63 fue sustituida por 1
La etiqueta (región) 63 fue sustituida por 1
La etiqueta (región) 28 fue sustituida por 6
La etiqueta (región) 29 fue sustituida por 3
La etiqueta (región) 65 fue sustituida por 3
La etiqueta (región) 14 fue sustituida por 4
La etiqueta (región) 67 fue sustituida por 14
La etiqueta (región) 67 fue sustituida por 14
La etiqueta (región) 69 fue sustituida por 32
La etiqueta (región) 71 fue sustituida por 31
La etiqueta (región) 72 fue sustituida por 5
La etiqueta (región) 71 fue sustituida por 31
La etiqueta (región) 70 fue sustituida por 31
La etiqueta (región) 31 fue sustituida por 6
La etiqueta (región) 31 fue sustituida por 6
La etiqueta (región) 31 fue sustituida por 6
La etiqueta (región) 31 fue sustituida por 6
La etiqueta (región) 75 fue sustituida por 73
La etiqueta (región) 75 fue sustituida por 73
La etiqueta (región) 8 fue sustituida por 1
La etiqueta (región) 8 fue sustituida por 1
La etiqueta (región) 80 fue sustituida por 2
La etiqueta (región) 255 fue sustituida por 82
La etiqueta (región) 37 fue sustituida por 2
La etiqueta (región) 255 fue sustituida por 37
La etiqueta (región) 86 fue sustituida por 2
La etiqueta (región) 9 fue sustituida por 2
La etiqueta (región) 90 fue sustituida por 39
La etiqueta (región) 255 fue sustituida por 9
La etiqueta (región) 255 fue sustituida por 95
La etiqueta (región) 17 fue sustituida por 9
La etiqueta (región) 17 fue sustituida por 9
La etiqueta (región) 102 fue sustituida por 42
La etiqueta (región) 103 fue sustituida por 19
La etiqueta (región) 104 fue sustituida por 43
La etiqueta (región) 103 fue sustituida por 19
La etiqueta (región) 19 fue sustituida por 3
La etiqueta (región) 19 fue sustituida por 3
La etiqueta (región) 42 fue sustituida por 19
La etiqueta (región) 105 fue sustituida por 19
La etiqueta (región) 105 fue sustituida por 19
La etiqueta (región) 106 fue sustituida por 45
La etiqueta (región) 45 fue sustituida por 20
La etiqueta (región) 111 fue sustituida por 2
La etiqueta (región) 255 fue sustituida por 4
La etiqueta (región) 115 fue sustituida por 2
La etiqueta (región) 50 fue sustituida por 21
La etiqueta (región) 50 fue sustituida por 21
La etiqueta (región) 50 fue sustituida por 21
La etiqueta (región) 21 fue sustituida por 4
La etiqueta (región) 65 fue sustituida por 3
La etiqueta (región) 10 fue sustituida por 1
La etiqueta (región) 123 fue sustituida por 19
La etiqueta (región) 255 fue sustituida por 125
La etiqueta (región) 129 fue sustituida por 22
La etiqueta (región) 129 fue sustituida por 22
La etiqueta (región) 10 fue sustituida por 1
La etiqueta (región) 11 fue sustituida por 3
La etiqueta (región) 11 fue sustituida por 3
La etiqueta (región) 11 fue sustituida por 3
La etiqueta (región) 136 fue sustituida por 54
La etiqueta (región) 136 fue sustituida por 54
La etiqueta (región) 54 fue sustituida por 3
La etiqueta (región) 54 fue sustituida por 3
La etiqueta (región) 130 fue sustituida por 3
La etiqueta (región) 138 fue sustituida por 3
La etiqueta (región) 139 fue sustituida por 12
La etiqueta (región) 140 fue sustituida por 11
La etiqueta (región) 140 fue sustituida por 11
La etiqueta (región) 140 fue sustituida por 11
La etiqueta (región) 12 fue sustituida por 11
La etiqueta (región) 12 fue sustituida por 11
La etiqueta (región) 141 fue sustituida por 12
La etiqueta (región) 12 fue sustituida por 5
La etiqueta (región) 143 fue sustituida por 142
La etiqueta (región) 143 fue sustituida por 3
La etiqueta (región) 142 fue sustituida por 3
La etiqueta (región) 144 fue sustituida por 5
La etiqueta (región) 145 fue sustituida por 25
La etiqueta (región) 146 fue sustituida por 1
La etiqueta (región) 146 fue sustituida por 1
La etiqueta (región) 147 fue sustituida por 25
La etiqueta (región) 148 fue sustituida por 25
La etiqueta (región) 152 fue sustituida por 25
La etiqueta (región) 158 fue sustituida por 1
La etiqueta (región) 159 fue sustituida por 1
La etiqueta (región) 159 fue sustituida por 1
La etiqueta (región) 58 fue sustituida por 1
La etiqueta (región) 58 fue sustituida por 1
La etiqueta (región) 12 fue sustituida por 11
La etiqueta (región) 163 fue sustituida por 62
La etiqueta (región) 163 fue sustituida por 62
La etiqueta (región) 12 fue sustituida por 5
La etiqueta (región) 255 fue sustituida por 5
La etiqueta (región) 255 fue sustituida por 5
La etiqueta (región) 255 fue sustituida por 166
La etiqueta (región) 6 fue sustituida por 5
La etiqueta (región) 28 fue sustituida por 5
La etiqueta (región) 28 fue sustituida por 6
La etiqueta (región) 155 fue sustituida por 6
La etiqueta (región) 174 fue sustituida por 29
La etiqueta (región) 175 fue sustituida por 30
La etiqueta (región) 30 fue sustituida por 5
La etiqueta (región) 176 fue sustituida por 5
La etiqueta (región) 176 fue sustituida por 5
La etiqueta (región) 179 fue sustituida por 67
La etiqueta (región) 67 fue sustituida por 14
La etiqueta (región) 182 fue sustituida por 65
La etiqueta (región) 51 fue sustituida por 4
La etiqueta (región) 66 fue sustituida por 3
La etiqueta (región) 255 fue sustituida por 4
La etiqueta (región) 185 fue sustituida por 14
La etiqueta (región) 14 fue sustituida por 4
La etiqueta (región) 14 fue sustituida por 4
La etiqueta (región) 179 fue sustituida por 67
La etiqueta (región) 67 fue sustituida por 14
La etiqueta (región) 69 fue sustituida por 32
La etiqueta (región) 188 fue sustituida por 70
La etiqueta (región) 188 fue sustituida por 70
La etiqueta (región) 70 fue sustituida por 31
La etiqueta (región) 70 fue sustituida por 31
La etiqueta (región) 255 fue sustituida por 70
La etiqueta (región) 70 fue sustituida por 31
La etiqueta (región) 193 fue sustituida por 73
La etiqueta (región) 195 fue sustituida por 4
La etiqueta (región) 195 fue sustituida por 4
La etiqueta (región) 198 fue sustituida por 74
La etiqueta (región) 198 fue sustituida por 74
La etiqueta (región) 255 fue sustituida por 33
La etiqueta (región) 34 fue sustituida por 33
La etiqueta (región) 34 fue sustituida por 33
La etiqueta (región) 34 fue sustituida por 33
La etiqueta (región) 200 fue sustituida por 190
La etiqueta (región) 255 fue sustituida por 75
La etiqueta (región) 202 fue sustituida por 34
La etiqueta (región) 193 fue sustituida por 73
La etiqueta (región) 75 fue sustituida por 73
La etiqueta (región) 75 fue sustituida por 73
La etiqueta (región) 77 fue sustituida por 15
La etiqueta (región) 77 fue sustituida por 15
La etiqueta (región) 78 fue sustituida por 6
La etiqueta (región) 78 fue sustituida por 6
La etiqueta (región) 78 fue sustituida por 6
La etiqueta (región) 208 fue sustituida por 79
La etiqueta (región) 209 fue sustituida por 78
La etiqueta (región) 209 fue sustituida por 79

Listado 6.2: Relación de todas las etiquetas que fueron reemplazadas por una misma etiqueta.

```
El segmento 1 se unió con los segmentos: [ 7 18 10 24 26 27 35 25 61 63 8 146 158 159 58 ]
El segmento 2 se unió con los segmentos: [ 9 16 36 37 38 3 80 86 111 115 ]
El segmento 3 se unió con los segmentos: [ 11 19 53 54 55 29 65 130 138 143 142 66 ]
El segmento 4 se unió con los segmentos: [ 14 47 21 51 195 ]
El segmento 5 se unió con los segmentos: [ 12 13 30 56 6 72 144 28 176 ]
El segmento 6 se unió con los segmentos: [ 28 31 155 78 ]
El segmento 9 se unió con los segmentos: [ 17 39 ]
El segmento 11 se unió con los segmentos: [ 55 140 12 ]
El segmento 12 se unió con los segmentos: [ 23 139 141 ]
El segmento 14 se unió con los segmentos: [ 67 185 ]
El segmento 15 se unió con los segmentos: [ 77 ]
El segmento 19 se unió con los segmentos: [ 41 103 42 105 123 ]
El segmento 20 se unió con los segmentos: [ 45 ]
El segmento 21 se unió con los segmentos: [ 50 ]
El segmento 22 se unió con los segmentos: [ 129 ]
El segmento 25 se unió con los segmentos: [ 57 145 147 148 152 ]
El segmento 28 se unió con los segmentos: [ 62 ]
El segmento 29 se unió con los segmentos: [ 174 ]
El segmento 30 se unió con los segmentos: [ 175 ]
El segmento 31 se unió con los segmentos: [ 71 70 ]
El segmento 32 se unió con los segmentos: [ 69 ]
El segmento 33 se unió con los segmentos: [ 34 ]
El segmento 34 se unió con los segmentos: [ 202 ]
El segmento 39 se unió con los segmentos: [ 90 ]
El segmento 42 se unió con los segmentos: [ 102 ]
El segmento 43 se unió con los segmentos: [ 104 ]
El segmento 45 se unió con los segmentos: [ 106 ]
El segmento 54 se unió con los segmentos: [ 136 ]
El segmento 62 se unió con los segmentos: [ 163 ]
El segmento 65 se unió con los segmentos: [ 182 ]
El segmento 67 se unió con los segmentos: [ 179 ]
El segmento 70 se unió con los segmentos: [ 188 ]
El segmento 73 se unió con los segmentos: [ 75 193 ]
El segmento 74 se unió con los segmentos: [ 198 ]
El segmento 78 se unió con los segmentos: [ 209 ]
El segmento 79 se unió con los segmentos: [ 208 209 ]
El segmento 142 se unió con los segmentos: [ 143 ]
El segmento 190 se unió con los segmentos: [ 200 ]
```

Listado 6.3: Resultado de revisar que segmentos son completamente similares obtenida a partir del Listado 6.2

```
El segmento 3 es similar a los segmentos: [ 11 19 29 ]
El segmento 5 es similar a los segmentos: [ 12 ]
El segmento 6 es similar a los segmentos: [ 28 31 ]
El segmento 20 es similar a los segmentos: [ 45 ]
El segmento 28 es similar a los segmentos: [ 62 ]
El segmento 73 es similar a los segmentos: [ 75 ]
```

Listado 6.4: Etiqueta, área y extremos de cada segmento de la imagen de la Figura 4.22.

```
Objeto 1:  área = 8,   e1(304,466), e2( 305,466), e3( 305,469), e4( 304,469]  Objeto 24:  área = 14,  e1(264,279), e2( 265,279), e3( 265,285), e4( 264,285]
Objeto 2:  área = 12,  e1(301,459), e2( 303,459), e3( 303,462), e4( 301,462]  Objeto 25:  área = 21,  e1(266,264), e2( 268,264), e3( 268,270), e4( 266,270]
Objeto 3:  área = 20,  e1(289,459), e2( 293,459), e3( 293,462), e4( 289,462]  Objeto 26:  área = 8,   e1(269,241), e2( 270,241), e3( 270,244), e4( 269,244]
Objeto 4:  área = 32,  e1(271,474), e2( 278,474), e3( 278,477), e4( 271,477]  Objeto 27:  área = 14,  e1(234,339), e2( 235,339), e3( 235,345), e4( 234,345]
Objeto 5:  área = 20,  e1(264,474), e2( 268,474), e3( 268,477), e4( 264,477]  Objeto 28:  área = 57,  e1(231,316), e2( 233,316), e3( 233,334), e4( 231,334]
Objeto 6:  área = 32,  e1(246,474), e2( 253,474), e3( 253,477), e4( 246,477]  Objeto 29:  área = 26,  e1(239,309), e2( 243,309), e3( 240,315), e4( 239,315]
Objeto 7:  área = 15,  e1(261,448), e2( 265,448), e3( 265,450), e4( 261,450]  Objeto 30:  área = 33,  e1(236,305), e2( 238,305), e3( 238,315), e4( 236,315]
Objeto 8:  área = 20,  e1(261,444), e2( 265,444), e3( 265,447), e4( 261,447]  Objeto 31:  área = 20,  e1(239,301), e2( 243,301), e3( 243,304), e4( 239,304]
Objeto 9:  área = 15,  e1(256,448), e2( 260,448), e3( 260,450), e4( 256,450]  Objeto 32:  área = 12,  e1(186,316), e2( 188,316), e3( 188,319), e4( 186,319]
Objeto 10: área = 16,  e1(244,440), e2( 245,440), e3( 245,447), e4( 244,447]  Objeto 33:  área = 12,  e1(181,309), e2( 183,309), e3( 183,312), e4( 181,312]
Objeto 11: área = 20,  e1(251,406), e2( 255,406), e3( 255,409), e4( 251,409]  Objeto 34:  área = 20,  e1(219,264), e2( 223,264), e3( 223,267), e4( 219,267]
Objeto 12: área = 14,  e1(254,399), e2( 255,399), e3( 255,405), e4( 254,405]  Objeto 35:  área = 16,  e1(99,470),  e2( 100,470), e3( 100,477), e4( 99,477]
Objeto 13: área = 21,  e1(251,399), e2( 253,399), e3( 253,405), e4( 251,405]  Objeto 36:  área = 20,  e1(94,466),  e2( 98,466),  e3( 98,469),  e4( 94,469]
Objeto 14: área = 8,   e1(249,399), e2( 250,399), e3( 250,402), e4( 249,402]  Objeto 37:  área = 20,  e1(89,474),  e2( 93,474),  e3( 93,477),  e4( 89,477]
Objeto 15: área = 48,  e1(229,474), e2( 240,474), e3( 240,477), e4( 229,477]  Objeto 38:  área = 16,  e1(114,421), e2( 115,421), e3( 115,428), e4( 114,428]
Objeto 16: área = 20,  e1(229,459), e2( 233,459), e3( 233,462), e4( 229,462]  Objeto 39:  área = 28,  e1(81,444),  e2( 85,440),  e3( 85,447),  e4( 81,447]
Objeto 17: área = 48,  e1(216,474), e2( 225,474), e3( 225,477), e4( 216,477]  Objeto 40:  área = 59,  e1(84,433),  e2( 90,440),  e3( 90,443),  e4( 86,443]
Objeto 18: área = 27,  e1(239,448), e2( 243,444), e3( 243,450), e4( 239,450]  Objeto 41:  área = 24,  e1(81,417),  e2( 83,417),  e3( 83,424),  e4( 81,424]
Objeto 19: área = 14,  e1(229,448), e2( 230,448), e3( 230,454), e4( 229,454]  Objeto 42:  área = 61,  e1(81,395),  e2( 85,406),  e3( 85,413),  e4( 84,413]
Objeto 20: área = 22,  e1(224,376), e2( 225,376), e3( 225,386), e4( 224,386]  Objeto 43:  área = 30,  e1(84,391),  e2( 85,391),  e3( 85,405),  e4( 84,405]
Objeto 21: área = 20,  e1(251,301), e2( 255,301), e3( 255,304), e4( 251,304]  Objeto 44:  área = 20,  e1(89,376),  e2( 93,376),  e3( 93,379),  e4( 89,379]
Objeto 22: área = 16,  e1(249,301), e2( 250,301), e3( 250,308), e4( 249,308]  Objeto 45:  área = 20,  e1(56,474),  e2( 60,474),  e3( 60,477),  e4( 56,477]
Objeto 23: área = 8,   e1(264,294), e2( 265,294), e3( 265,297), e4( 264,297]  Objeto 46:  área = 32,  e1(41,474),  e2( 48,474),  e3( 48,477),  e4( 41,477]
```

Objeto 47: área = 35, e1(76,429), e2(80,429), e3(80,435), e4(76,435]
Objeto 48: área = 8, e1(69,425), e2(70,425), e3(70,428), e4(69,428]
Objeto 49: área = 40, e1(44,425), e2(53,425), e3(53,428), e4(44,428]
Objeto 50: área = 28, e1(9,474), e2(15,474), e3(15,477), e4(9,477]
Objeto 51: área = 303, e1(0,478), e2(100,478), e3(100,480), e4(0,480]
Objeto 52: área = 36, e1(3,466), e2(8,466), e3(8,473), e4(6,473]
Objeto 53: área = 36, e1(0,466), e2(5,470), e3(5,473), e4(0,473]
Objeto 54: área = 20, e1(64,414), e2(68,414), e3(68,417), e4(64,417]
Objeto 55: área = 14, e1(79,399), e2(80,399), e3(80,405), e4(79,405]
Objeto 56: área = 16, e1(79,391), e2(80,391), e3(80,398), e4(79,398]
Objeto 57: área = 20, e1(59,414), e2(63,414), e3(63,417), e4(59,417]
Objeto 58: área = 8, e1(34,406), e2(35,406), e3(35,409), e4(34,409]
Objeto 59: área = 8, e1(29,410), e2(30,410), e3(30,413), e4(29,413]
Objeto 60: área = 34, e1(31,395), e2(35,395), e3(35,405), e4(34,405]
Objeto 61: área = 12, e1(151,350), e2(153,350), e3(153,353), e4(151,353]
Objeto 62: área = 20, e1(149,335), e2(153,331), e3(150,338), e4(149,338]
Objeto 63: área = 12, e1(131,316), e2(133,316), e3(133,319), e4(131,319]
Objeto 64: área = 12, e1(86,346), e2(88,346), e3(88,349), e4(86,349]
Objeto 65: área = 12, e1(91,335), e2(93,335), e3(93,338), e4(91,338]
Objeto 66: área = 8, e1(84,271), e2(85,271), e3(85,274), e4(84,274]
Objeto 67: área = 24, e1(91,268), e2(98,268), e3(98,270), e4(91,270]
Objeto 68: área = 8, e1(99,260), e2(100,260), e3(100,263), e4(99,263]
Objeto 69: área = 20, e1(91,260), e2(95,260), e3(95,263), e4(91,263]
Objeto 70: área = 16, e1(79,320), e2(80,320), e3(80,327), e4(79,327]
Objeto 71: área = 24, e1(76,320), e2(78,320), e3(78,327), e4(76,327]
Objeto 72: área = 16, e1(59,301), e2(60,301), e3(60,308), e4(59,308]
Objeto 73: área = 20, e1(54,301), e2(58,301), e3(58,304), e4(54,304]
Objeto 74: área = 8, e1(79,275), e2(80,275), e3(80,278), e4(79,278]
Objeto 75: área = 6, e1(49,279), e2(50,279), e3(50,281), e4(49,281]
Objeto 76: área = 28, e1(46,271), e2(50,271), e3(50,278), e4(49,278]
Objeto 77: área = 28, e1(44,271), e2(48,275), e3(48,278), e4(44,278]
Objeto 78: área = 21, e1(46,264), e2(48,264), e3(48,270), e4(46,270]
Objeto 79: área = 36, e1(41,267), e2(43,267), e3(43,278), e4(41,278]
Objeto 80: área = 16, e1(39,245), e2(40,245), e3(40,252), e4(39,252]
Objeto 81: área = 53, e1(39,241), e2(43,241), e3(43,255), e4(41,255]
Objeto 82: área = 45, e1(36,241), e2(38,241), e3(38,255), e4(36,255]
Objeto 83: área = 26, e1(264,234), e2(268,234), e3(265,240), e4(264,240]
Objeto 84: área = 20, e1(266,230), e2(270,230), e3(270,233), e4(266,233]
Objeto 85: área = 78, e1(266,189), e2(268,189), e3(268,214), e4(266,214]
Objeto 86: área = 45, e1(266,174), e2(268,174), e3(268,188), e4(266,188]
Objeto 87: área = 16, e1(269,140), e2(270,140), e3(270,147), e4(269,147]
Objeto 88: área = 22, e1(274,84), e2(275,84), e3(275,94), e4(274,94]
Objeto 89: área = 9, e1(271,73), e2(273,73), e3(273,75), e4(271,75]
Objeto 90: área = 111, e1(276,43), e2(278,43), e3(278,79), e4(276,79]
Objeto 91: área = 32, e1(241,20), e2(248,20), e3(248,23), e4(241,23]
Objeto 92: área = 32, e1(231,16), e2(238,16), e3(238,19), e4(231,19]
Objeto 93: área = 8, e1(229,20), e2(230,20), e3(230,23), e4(229,23]
Objeto 94: área = 20, e1(219,12), e2(223,12), e3(223,15), e4(219,15]
Objeto 95: área = 40, e1(201,8), e2(210,8), e3(210,11), e4(201,11]
Objeto 96: área = 20, e1(186,4), e2(190,4), e3(190,7), e4(186,7]
Objeto 97: área = 20, e1(179,4), e2(183,4), e3(183,7), e4(179,7]
Objeto 98: área = 120, e1(174,0), e2(203,0), e3(203,3), e4(174,3]
Objeto 99: área = 8, e1(44,234), e2(45,234), e3(45,237), e4(44,237]
Objeto 100: área = 24, e1(41,200), e2(43,200), e3(43,207), e4(41,207]
Objeto 101: área = 8, e1(49,189), e2(50,189), e3(50,192), e4(49,192]
Objeto 102: área = 14, e1(44,189), e2(45,189), e3(45,195), e4(44,195]
Objeto 103: área = 33, e1(41,185), e2(43,185), e3(43,195), e4(41,195]
Objeto 104: área = 21, e1(79,178), e2(85,178), e3(85,180), e4(79,180]
Objeto 105: área = 15, e1(69,178), e2(73,178), e3(73,180), e4(69,180]
Objeto 106: área = 14, e1(44,174), e2(45,174), e3(45,180), e4(44,180]
Objeto 107: área = 45, e1(46,159), e2(48,159), e3(48,173), e4(46,173]
Objeto 108: área = 16, e1(49,147), e2(50,147), e3(50,154), e4(49,154]
Objeto 109: área = 33, e1(51,129), e2(53,129), e3(53,139), e4(51,139]
Objeto 110: área = 40, e1(106,4), e2(115,4), e3(115,7), e4(106,7]
Objeto 111: área = 32, e1(91,8), e2(98,8), e3(98,11), e4(91,11]
Objeto 112: área = 20, e1(81,12), e2(85,12), e3(85,15), e4(81,15]
Objeto 113: área = 8, e1(59,114), e2(60,114), e3(60,117), e4(59,117]
Objeto 114: área = 22, e1(54,114), e2(55,114), e3(55,124), e4(54,124]
Objeto 115: área = 45, e1(56,99), e2(58,99), e3(58,113), e4(56,113]
Objeto 116: área = 16, e1(59,91), e2(60,91), e3(60,98), e4(59,98]
Objeto 117: área = 16, e1(64,80), e2(65,80), e3(65,87), e4(64,87]
Objeto 118: área = 57, e1(61,65), e2(63,65), e3(63,83), e4(61,83]
Objeto 119: área = 14, e1(59,84), e2(60,84), e3(60,90), e4(59,90]
Objeto 120: área = 16, e1(64,50), e2(65,50), e3(65,57), e4(64,57]
Objeto 121: área = 32, e1(69,42), e2(70,42), e3(70,57), e4(69,57]
Objeto 122: área = 21, e1(66,39), e2(68,39), e3(68,45), e4(66,45]
Objeto 123: área = 8, e1(74,24), e2(75,24), e3(75,27), e4(74,27]
Objeto 124: área = 193, e1(304,459), e2(320,459), e3(315,473), e4(309,473]
Objeto 125: área = 136, e1(286,466), e2(305,470), e3(300,477), e4(294,477]
Objeto 126: área = 176, e1(299,451), e2(320,451), e3(320,458), e4(299,458]
Objeto 127: área = 75, e1(281,459), e2(285,459), e3(285,473), e4(281,473]
Objeto 128: área = 139, e1(241,414), e2(255,410), e3(250,424), e4(246,424]
Objeto 129: área = 970, e1(229,466), e2(280,459), e3(280,473), e4(229,473]
Objeto 130: área = 80, e1(201,406), e2(210,406), e3(210,413), e4(201,413]
Objeto 131: área = 123, e1(224,387), e2(233,387), e3(230,398), e4(229,398]
Objeto 132: área = 52, e1(231,361), e2(235,361), e3(233,372), e4(231,372]
Objeto 133: área = 75, e1(226,361), e2(230,361), e3(230,375), e4(226,375]
Objeto 134: área = 71, e1(239,316), e2(245,313), e3(245,323), e4(239,323]
Objeto 135: área = 479, e1(241,290), e2(263,275), e3(255,300), e4(244,300]
Objeto 136: área = 89, e1(261,245), e2(270,245), e3(268,259), e4(266,259]
Objeto 137: área = 92, e1(89,451), e2(95,451), e3(95,465), e4(91,465]
Objeto 138: área = 40, e1(61,466), e2(65,466), e3(65,473), e4(61,473]
Objeto 139: área = 75, e1(71,406), e2(75,406), e3(75,420), e4(71,420]
Objeto 140: área = 124, e1(44,418), e2(55,406), e3(55,420), e4(44,420]
Objeto 141: área = 72, e1(71,361), e2(80,365), e3(80,368), e4(71,368]
Objeto 142: área = 35, e1(151,354), e2(155,354), e3(155,360), e4(151,360]
Objeto 143: área = 204, e1(86,268), e2(100,286), e3(100,293), e4(91,293]
Objeto 144: área = 47, e1(99,264), e2(105,268), e3(105,270), e4(101,270]
Objeto 145: área = 224, e1(49,256), e2(60,256), e3(60,270), e4(49,270]
Objeto 146: área = 118, e1(271,234), e2(275,230), e3(275,255), e4(271,255]
Objeto 147: área = 10, e1(274,211), e2(275,211), e3(275,215), e4(274,215]
Objeto 148: área = 75, e1(261,219), e2(265,219), e3(265,233), e4(261,233]
Objeto 149: área = 121, e1(266,151), e2(270,148), e3(270,173), e4(266,173]
Objeto 150: área = 40, e1(276,106), e2(280,106), e3(280,113), e4(276,113]
Objeto 151: área = 149, e1(271,99), e2(275,95), e3(273,135), e4(271,135]
Objeto 152: área = 75, e1(51,234), e2(55,234), e3(55,248), e4(51,248]
Objeto 153: área = 84, e1(41,208), e2(45,211), e3(45,225), e4(41,225]
Objeto 154: área = 139, e1(46,196), e2(50,193), e3(48,222), e4(46,222]
Objeto 155: área = 75, e1(36,226), e2(40,226), e3(40,240), e4(36,240]
Objeto 156: área = 46, e1(56,121), e2(60,118), e3(60,128), e4(56,128]
Objeto 157: área = 83, e1(64,72), e2(70,65), e3(65,79), e4(64,79]
Objeto 158: área = 459, e1(254,436), e2(280,421), e3(270,447), e4(259,447]
Objeto 159: área = 397, e1(259,417), e2(280,406), e3(260,432), e4(259,432]
Objeto 160: área = 523, e1(234,369), e2(250,361), e3(248,398), e4(244,398]
Objeto 161: área = 174, e1(189,316), e2(200,316), e3(200,330), e4(191,330]
Objeto 162: área = 922, e1(31,410), e2(75,391), e3(35,428), e4(31,428]
Objeto 163: área = 150, e1(41,361), e2(50,361), e3(50,375), e4(41,375]
Objeto 164: área = 392, e1(49,282), e2(65,290), e3(65,300), e4(51,300]
Objeto 165: área = 439, e1(266,219), e2(280,181), e3(268,225), e4(266,225]
Objeto 166: área = 1135, e1(204,0), e2(270,0), e3(270,30), e4(266,30]
Objeto 167: área = 297, e1(49,163), e2(60,159), e3(53,184), e4(51,184]
Objeto 168: área = 1448, e1(269,448), e2(320,421), e3(298,458), e4(294,458]
Objeto 169: área = 1786, e1(234,354), e2(280,286), e3(260,360), e4(234,360]
Objeto 170: área = 1361, e1(21,286), e2(53,309), e3(50,330), e4(21,330]
Objeto 171: área = 1703, e1(54,144), e2(80,106), e3(63,180), e4(59,180]
Objeto 172: área = 1843, e1(21,0), e2(120,0), e3(60,30), e4(21,30]
Objeto 173: área = 651, e1(0,0), e2(20,0), e3(20,30), e4(0,30]
Objeto 174: área = 9428, e1(86,384), e2(320,478), e3(319,480), e4(101,480]
Objeto 175: área = 10899, e1(91,339), e2(233,335), e3(118,439), e4(116,439]
Objeto 176: área = 20406, e1(249,395), e2(320,0), e3(320,420), e4(281,420]
Objeto 177: área = 27521, e1(44,241), e2(265,249), e3(93,364), e4(89,364]
Objeto 178: área = 32670, e1(64,88), e2(275,46), e3(263,210), e4(241,210]
Objeto 179: área = 23340, e1(0,31), e2(93,466), e3(93,473), e4(9,473]

Listado 6.5: Objetos que serán eliminados debido a que su área es menor al 1.25% de la imagen.

Objeto 1: área = 8
Objeto 2: área = 12
Objeto 3: área = 20
Objeto 4: área = 32
Objeto 5: área = 20
Objeto 6: área = 32
Objeto 7: área = 15
Objeto 8: área = 20
Objeto 9: área = 15
Objeto 10: área = 16
Objeto 11: área = 20
Objeto 12: área = 14
Objeto 13: área = 21
Objeto 14: área = 8
Objeto 15: área = 48
Objeto 16: área = 20
Objeto 17: área = 48
Objeto 18: área = 27
Objeto 19: área = 14
Objeto 20: área = 22
Objeto 21: área = 20
Objeto 22: área = 16
Objeto 23: área = 8
Objeto 24: área = 14
Objeto 25: área = 21
Objeto 26: área = 8
Objeto 27: área = 14
Objeto 28: área = 57
Objeto 29: área = 26
Objeto 30: área = 33
Objeto 31: área = 20
Objeto 32: área = 12
Objeto 33: área = 12
Objeto 34: área = 20
Objeto 35: área = 16
Objeto 36: área = 20
Objeto 37: área = 20
Objeto 38: área = 16
Objeto 39: área = 28
Objeto 40: área = 59
Objeto 41: área = 24
Objeto 42: área = 61
Objeto 43: área = 30
Objeto 44: área = 20
Objeto 45: área = 20
Objeto 46: área = 32
Objeto 47: área = 35
Objeto 48: área = 8

Objeto 49: área = 40
Objeto 50: área = 28
Objeto 51: área = 303
Objeto 52: área = 36
Objeto 53: área = 36
Objeto 54: área = 20
Objeto 55: área = 14
Objeto 56: área = 16
Objeto 57: área = 20
Objeto 58: área = 8
Objeto 59: área = 8
Objeto 60: área = 34
Objeto 61: área = 12
Objeto 62: área = 20
Objeto 63: área = 12
Objeto 64: área = 12
Objeto 65: área = 12
Objeto 66: área = 8
Objeto 67: área = 24
Objeto 68: área = 8
Objeto 69: área = 20
Objeto 70: área = 16
Objeto 71: área = 24
Objeto 72: área = 16
Objeto 73: área = 20
Objeto 74: área = 8
Objeto 75: área = 6
Objeto 76: área = 28
Objeto 77: área = 28
Objeto 78: área = 21
Objeto 79: área = 36
Objeto 80: área = 16
Objeto 81: área = 53
Objeto 82: área = 45
Objeto 83: área = 26
Objeto 84: área = 20
Objeto 85: área = 78
Objeto 86: área = 45
Objeto 87: área = 16
Objeto 88: área = 22
Objeto 89: área = 9
Objeto 90: área = 111
Objeto 91: área = 32
Objeto 92: área = 32
Objeto 93: área = 8
Objeto 94: área = 20
Objeto 95: área = 40
Objeto 96: área = 20
Objeto 97: área = 20
Objeto 98: área = 120
Objeto 99: área = 8
Objeto 100: área = 24
Objeto 101: área = 8
Objeto 102: área = 14
Objeto 103: área = 33
Objeto 104: área = 21
Objeto 105: área = 15
Objeto 106: área = 14
Objeto 107: área = 45
Objeto 108: área = 16
Objeto 109: área = 33
Objeto 110: área = 40
Objeto 111: área = 32
Objeto 112: área = 20
Objeto 113: área = 8
Objeto 114: área = 22
Objeto 115: área = 45
Objeto 116: área = 16
Objeto 117: área = 16
Objeto 118: área = 57
Objeto 119: área = 14
Objeto 120: área = 16
Objeto 121: área = 32
Objeto 122: área = 21
Objeto 123: área = 8
Objeto 124: área = 193
Objeto 125: área = 136
Objeto 126: área = 176
Objeto 127: área = 75
Objeto 128: área = 139
Objeto 129: área = 970
Objeto 130: área = 80
Objeto 131: área = 123
Objeto 132: área = 52
Objeto 133: área = 75
Objeto 134: área = 71
Objeto 135: área = 479
Objeto 136: área = 89
Objeto 137: área = 92
Objeto 138: área = 40
Objeto 139: área = 75
Objeto 140: área = 124
Objeto 141: área = 72
Objeto 142: área = 35
Objeto 143: área = 204
Objeto 144: área = 47
Objeto 145: área = 224
Objeto 146: área = 118
Objeto 147: área = 10
Objeto 148: área = 75
Objeto 149: área = 121
Objeto 150: área = 40
Objeto 151: área = 149
Objeto 152: área = 75
Objeto 153: área = 84
Objeto 154: área = 139
Objeto 155: área = 75
Objeto 156: área = 46
Objeto 157: área = 83
Objeto 158: área = 459
Objeto 159: área = 397
Objeto 160: área = 523
Objeto 161: área = 174
Objeto 162: área = 922
Objeto 163: área = 150
Objeto 164: área = 392
Objeto 165: área = 439
Objeto 166: área = 1135
Objeto 167: área = 297
Objeto 168: área = 1448
Objeto 169: área = 1786
Objeto 170: área = 1361
Objeto 171: área = 1703
Objeto 172: área = 1843
Objeto 173: área = 651

Listado 6.6: Izquierda.- Lista de reemplazos con ciclos. Derecha.- Lista con ciclos corregidos (rojo).

0: -1, 1: 124, 2: 124, 3: 168, 4: 174, 5: 129, 6: 129, 7: 9, 8: 7, 9: 7, 10: 18, 11: 13, 12: 13, 13: 176, 14: 13, 15: 129, 16: 0, 17: 174, 18: 10, 19: 174, 20: 131, 21: 22, 22: 21, 23: 169, 24: 176, 25: 177, 26: 136, 27: 175, 28: 177, 29: 134, 30: 177, 31: 30, 32: 161, 33: 177, 34: 177, 35: 36, 36: 137, 37: 179, 38: 175, 39: 40, 40: 39, 41: 0, 42: 56, 43: 42, 44: 174, 45: 138, 46: 179, 47: 179, 48: 179, 49: 179, 50: 179, 51: 174, 52: 53, 53: 52, 54: 57, 55: 42, 56: 42, 57: 54, 58: 162, 59: 179, 60: 179, 61: 142, 62: 177, 63: 175, 64: 177, 65: 177, 66: 143, 67: 177, 68: 144, 69: 177, 70: 177, 71: 179, 72: 164, 73: 164, 74: 177, 75: 77, 76: 164, 77: 75, 78: 76, 79: 77, 80: 82, 81: 177, 82: 155, 83: 177, 84: 148, 85: 165, 86: 149, 87: 149, 88: 151, 89: 178, 90: 176, 91: 178, 92: 166, 93: 178, 94: 178, 95: 166, 96: 98, 97: 98, 98: 166, 99: 177, 100: 153, 101: 154, 102: 154, 103: 179, 104: 177, 105: 177, 106: 107, 107: 106, 108: 179, 109: 179, 110: 172, 111: 172, 112: 172, 113: 156, 114: 179, 115: 114, 116: 115, 117: 157, 118: 119, 119: 179, 120: 179, 121: 122, 122: 179, 123: 178, 124: 2, 125: 124, 126: 124, 127: 129, 128: 174, 129: 15, 130: 175, 131: 174, 132: 160, 133: 175, 134: 29, 135: 177, 136: 26, 137: 36, 138: 45, 139: 162, 140: 162, 141: 179, 142: 61, 143: 177, 144: 68, 145: 164, 146: 176, 147: 176, 148: 177, 149: 86, 150: 176, 151: 88, 152: 177, 153: 100, 154: 101, 155: 153, 156: 113, 157: 117, 158: 9, 159: 158, 160: 132, 161: 32, 162: 179, 163: 179, 164: 75, 165: 176, 166: 98, 167: 171, 168: 3, 169: 132, 170: 179, 171: 167, 172: 179, 173: 179, 174: -1, 175: -1, 176: -1, 177: -1, 178: -1, 179: -1.

0: -1, 1: 124, 2: 124, 3: 168, 4: 174, 5: 129, 6: 129, 7: 9, 8: 9, 9: 9, 10: 18, 11: 176, 12: 176, 13: 176, 14: 176, 15: 129, 16: 0, 17: 174, 18: 18, 19: 174, 20: 174, 21: 22, 22: 22, 23: 160, 24: 176, 25: 177, 26: 136, 27: 175, 28: 177, 29: 134, 30: 177, 31: 177, 32: 161, 33: 177, 34: 177, 35: 137, 36: 137, 37: 179, 38: 175, 39: 40, 40: 40, 41: 0, 42: 56, 43: 56, 44: 174, 45: 138, 46: 179, 47: 179, 48: 179, 49: 179, 50: 179, 51: 174, 52: 53, 53: 53, 54: 57, 55: 56, 56: 56, 57: 57, 58: 179, 59: 179, 60: 179, 61: 142, 62: 177, 63: 175, 64: 177, 65: 177, 66: 177, 67: 177, 68: 144, 69: 177, 70: 177, 71: 179, 72: 77, 73: 77, 74: 177, 75: 77, 76: 77, 77: 77, 78: 77, 79: 77, 80: 153, 81: 177, 82: 153, 83: 177, 84: 177, 85: 176, 86: 149, 87: 149, 88: 151, 89: 178, 90: 176, 91: 178, 92: 166, 93: 178, 94: 178, 95: 166, 96: 166, 97: 166, 98: 166, 99: 177, 100: 153, 101: 154, 102: 154, 103: 179, 104: 177, 105: 177, 106: 107, 107: 107, 108: 179, 109: 179, 110: 179, 111: 179, 112: 179, 113: 156, 114: 179, 115: 179, 116: 179, 117: 157, 118: 179, 119: 179, 120: 179, 121: 179, 122: 179, 123: 178, 124: 124, 125: 124, 126: 124, 127: 129, 128: 174, 129: 129, 130: 175, 131: 174, 132: 160, 133: 175, 134: 134, 135: 177, 136: 136, 137: 137, 138: 138, 139: 179, 140: 179, 141: 179, 142: 142, 143: 177, 144: 144, 145: 77, 146: 176, 147: 176, 148: 177, 149: 149, 150: 176, 151: 151, 152: 177, 153: 153, 154: 154, 155: 153, 156: 156, 157: 157, 158: 9, 159: 9, 160: 160, 161: 161, 162: 179, 163: 179, 164: 77, 165: 176, 166: 166, 167: 171, 168: 168, 169: 160, 170: 179, 171: 171, 172: 179, 173: 179, 174: -1, 175: -1, 176: -1, 177: -1, 178: -1, 179: -1.