UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

Facultad de Ingeniería Eléctrica

División de Estudios de Posgrado

DETECCIÓN DE REGIONES CLONADAS EN IMÁGENES DIGITALES

TESIS

Que para obtener el grado de

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

Sistemas Computacionales

Presenta

Salvador Daniel Pelayo Gómez

Félix Calderón Solorio

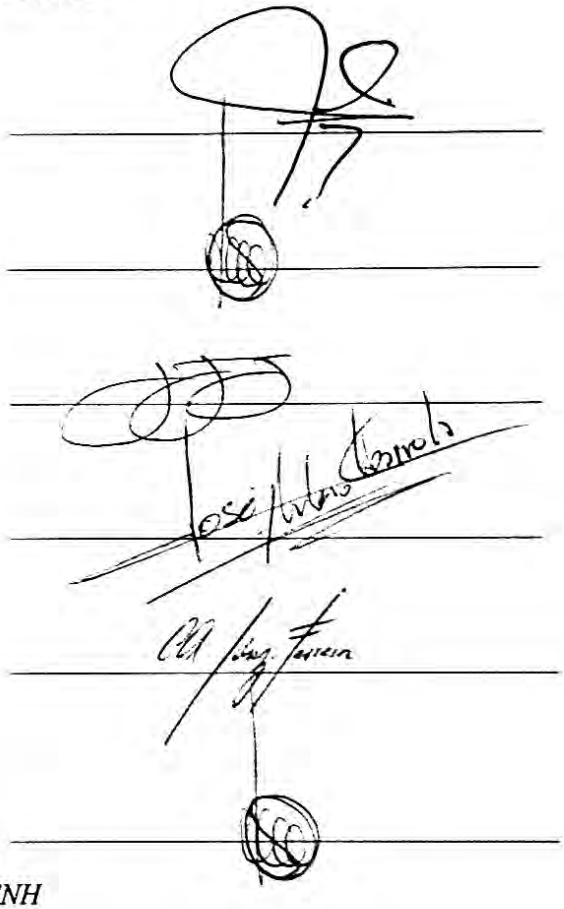
Doctor en Ciencias con Orientacin en Ciencias de la Computación

Director de Tesis

Morelia, Michoacán. Octubre 2017

DETECCIÓN DE REGIONES CLONADAS EN IMÁGENES DIGITALES

Los Miembros del Jurado de Examen de Grado aprueban la **Tesis de Maestría en Ciencias en Ingeniería Eléctrica** de ***Salvador Peñayo Gómez***



Dr. Jaime Cerda Jacobo
Presidente del Jurado

Dr. Félix Calderón Solorio
Director de Tesis

Dr. Juan José Flores Romero
Vocal

Dr. José Antonio Camarena Ibarrola
Vocal

Dr. Carlos Alberto Júnez Ferreira
Revisor Externo (FIC. UMSNH)

Dr. Félix Calderón Solorio
Jefe de la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica. UMSNH
(Por reconocimiento de firmas).

UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO
Septiembre 2017

Contenido

Lista de Símbolos

$\mathbb{C}$	Números complejos.
$\mathbb{R}$	Números reales.
e	Número de Euler o constante de Napier.
N	Total de muestras para en una señal discreta.
M	Total de muestras para una señal discreta en una segunda dimensión.
$\phi(n)$	Función exponencial compleja.
$\phi_k(n)$	Familia de exponenciales complejas.
$x(n)$	Señal discreta en el dominio del tiempo.
$X(k)$	Señal discreta en el dominio de la frecuencia.
$\overset{\mathcal{F}}{\Leftrightarrow}$	Aplicación de la Transformada de Fourier.
$*$	Operador para Convolución.
*	Conjugado para números complejos.
I	Imagen digital.
N_r	Número de renglones de una imagen digital.
N_c	Número de columnas de una imagen digital.
$I(r, c)$	Píxel de una imagen, dado por las coordenadas r, c.
R	Canal de color Rojo de una imagen digital.
G	Canal de color Verde de una imagen digital.
B	Canal de color Azul de una imagen digital.
$R(r, c)$	Píxel del canal Rojo, dado por las coordenadas r, c.
$G(r, c)$	Píxel del canal Verde, dado por las coordenadas r, c.
$B(r, c)$	Píxel del canal Azul, dado por las coordenadas r, c.
$I(k)$	k-ésimo píxel de I.
C_k	Vector de caracteristicas para el k-ésimo píxel de una imagen digital.
v_k	Vector con coeficientes de frecuencia, para una vecindad del píxel k-ésimo.
u_k	Vector con las coordenadas r y c del píxel k-ésimo.
g_k	Etiqueta que identifica al grupo que pertenece el píxel k-ésimo.
$w_{R,k}$	Ventana para el canal Rojo, construida sobre la vecindad del píxel k.
$w_{G,k}$	Ventana para el canal Verde, construida sobre la vecindad del píxel k.

$w_{B,k}$	Ventana para el canal Azul, construida sobre la vecindad del píxel k.
$W_{R,k}$	Ventana para el canal Rojo, construida sobre la vecindad del píxel k en el dominio de la frecuencia.
$W_{G,k}$	Ventana para el canal Verde, construida sobre la vecindad del píxel k en el dominio de la frecuencia.
$W_{B,k}$	Ventana para el canal Azul, construida sobre la vecindad del píxel k en el dominio de la frecuencia.
$N_v(k)$	Conjuto de etiquetas para la vecindad geometrica del píxel k.
D_E	Distancia Euclidiana entre dos vectores.
D_{cos}	Distancia Coseno entre dos vectores.
ϵ	Umbral de error o diferencia permitida.
τ	Radio para delimitar una vecindad geometrica.
I_s	Imagen digital generada por nuestro algoritmo $DeReC$.

Lista de Figuras

Lista de Tablas

Dedicatoria

Este trabajo se lo dedico primeramente a Dios por darme la oportunidad de estar aquí y la capacidad para poder estudiar.

A mi esposa Marisol y a mi hijito Alan Daniel, gracias por acompañarme en la vida. Gracias por todo el apoyo, por todas las veces que sólo por ustedes me levanté. Los amo con todo mi corazón. Este logro no sería posible sin ustedes, ya que son mi razón de esforzarme y mis ganas de ser mejor cada día.

También se lo dedico a mis papás y hermanos, gracias a todos sus consejos, sacrificio y su ayuda hoy puedo estar aquí, porque ustedes me formaron y educaron. En especial a mi hermano Juan de Dios, por ser estos dos años mi compañero y haberme convencido de estudiar la maestría. Espero ser un orgullo para ustedes que tanto me quieren y tanto los quiero.

Por último va dedicado a todas las personas que se han dado el tiempo de enseñarme, explicarme y ayudarme a aprender. A mi asesor Dr. Félix Calderón Solorio, a mi amigo Adán Garnica y a mi profesor Dr. Juan José Flores Romero, así como a todos los profesores que me dieron clases, gracias por su dedicación y por la gran labor que hacen de educar.

Salvador Daniel Pelayo Gómez

Resumen

En esta tesis se presenta una técnica para detección de regiones copiadas y/o clonadas en imágenes digitales. Con este propósito presentamos el algoritmo $DeReC$, el cual está basado en la Transformada Coseno Discreta (DCT, de sus siglas en inglés). $DeReC$ permite detectar regiones clonadas en imágenes con compresión sin pérdida (PNG) y con compresión con pérdida (en específico JPEG, dado que este es el formato de compresión de imágenes más utilizado en la actualidad). El algoritmo consiste en generar un conjunto de vectores de características para todos los píxeles de la imagen, con la ayuda de la DCT, posteriormente por medio de búsquedas aproximadas, utilizando $kd-trees$ y $Vp-trees$ encontramos al vecino más cercano para cada vector del conjunto; finalmente se validan esos vecinos cercanos para determinar si los píxeles que representan pertenecen a una región clonada o no. Como resultado el algoritmo entrega un mapa binario de la imagen, donde muestra en blanco las regiones clonadas y en negro las regiones no clonadas. Se realizan múltiples experimentos, como son ajustes de parámetros, medición de tiempos de ejecución y validación de resultados obtenidos en imágenes PNG y JPEG. Estos experimentos nos permiten medir la efectividad del algoritmo desarrollado. Adicionalmente se hacen comparaciones contra resultados obtenidos por otros algoritmos encontrados en la literatura. Los resultados obtenidos por $DeReC$ son satisfactorios, ya que logra detectar las regiones clonadas en los formatos para los que se realizaron pruebas (PNG y JPEG), al medir la efectividad de la detección encontramos que está a la par de otras propuestas.

Palabras clave: Regiones Clonadas en Imágenes, Imágenes Alteradas, Análisis Forense de Imágenes, $DeReC$, DCT, $Vp-trees$, $kd-trees$.

Abstract

This thesis presents a technique to detect copied and/or cloned regions in digital images. We present the algorithm $DeReC$, based on the Discrete Cosine Transform (DCT). $DeReC$ allows us to detect cloned regions in images with lossless compression (PNG) and lossy compression (specifically JPEG, since this is the image compression format most commonly used today). The algorithm consists in generating a set of feature vectors for each pixel in the image using the DCT, after that, using approximated searches with $kd-trees$ and $vp-trees$, we find the nearest neighbor for each vector in the set; finally these nearest neighbors are validated to determine if the pixels represent a cloned region. The result of the algorithm is a binary map where the cloned regions are shown in white. We present results of multiple experiments where parameters were setted, execution times were measured and the results obtained for PNG and JPEG images were validated. These experiments allow us to measure the performance of $DeReC$. Additionally, we present a comparison between our results and results obtained with algorithms found in the literature. The results obtained by $DeReC$ are satisfactory, since it is able to detect cloned regions in the formats for which tests were performed (PNG and JPEG), when measuring the performance of the detection we found that it is on par with other proposals.

Capítulo 1

Introducción

En este capítulo vamos a definir lo que es el análisis forense de imágenes digitales, así como la importancia que tienen las imágenes digitales en la actualidad. Nos enfocaremos en especifico en las alteraciones basadas en regiones clonadas, planteando de manera formal dicho problema, para esto haremos una revisión del estado del arte. Nos fijaremos objetivos claros que nos permitan resolver el problema de la detección de regiones clonadas. Y por último describiremos brevemente los capítulos posteriores.

1.1. Planteamiento del Problema

Las imágenes digitales o fotografías digitales son ampliamente utilizadas en la actualidad, con el fin de documentar nuestro modo de vida. Como lo mencionan [Gloe, Kirchner, Winkler and Rainer, 2007], debido a las nuevas tecnologías existentes la edición o manipulación de imágenes digitales es muy sencilla y una práctica muy común, ya que existen diferentes herramientas para edición de imágenes digitales como: GIMP, Photoshop, Corel Draw, Paint, entre varias más. Generalmente no existe alguna repercusión en esta práctica, por ejemplo retocar fotografías, remover objetos de la escena, cambiar el fondo de una fotografía, etc., no implica repercusiones siempre y cuando el objetivo de editar una imagen sea con propósitos estéticos. Pero no

siempre esto es así, ya que existen casos donde la edición de las imágenes sí puede tener consecuencias, por ejemplo cuando una imagen alterada se utiliza como evidencia, cuando se pretende alterar la realidad, pseudo-periodismo, plagio, intriga, lo cual puede derivar en la toma de decisiones erróneas. Estos pueden ser algunos casos en los cuales una imagen falsa puede generar consecuencias como bien lo mencionan [Gloe et al., 2007] y [Ng, Chang and Sun, 2004].

Es necesario estar conscientes de la importancia que juegan las imágenes digitales en la actualidad y la facilidad que existe para poder alterar una imagen digital, por esto, en el presente trabajo abordaremos el tema del análisis forense de imágenes, en especifico hablaremos sobre la detección de regiones clonadas. Nuestra objetivo es poder generar una herramienta que nos permita detectar imágenes que han sido alteradas, a las cuales se les ha agregado regiones clonadas, ya sea para tapar un elemento de la escena o agregar alguno existente en otro lugar.

Según [Gloe et al., 2007] el análisis forense de imágenes digitales o fotografías digitales es el estudio que se realiza sobre una imagen con el fin de encontrar evidencias de que la imagen ha sido o no alterada. Creemos que es necesario realizar herramientas que de manera automática nos permitan detectar imágenes digitales alteradas, por dos razones principales: primeramente por el papel tan importante que tienen en la actualidad y debido a que en algunos casos resulta muy difícil para un observador no experimentado poder detectar dichas alteraciones.

Dada una imagen digital y suponiendo que existe alguna región clonada, debemos determinar donde se encuentra dicha clonación. Como ejemplo veamos las imágenes en las Figuras 1.1 y 1.2 que corresponden al lanzamiento de unos misiles balísticos, sin embargo una de ellas es la imagen original y la otra la imagen alterada. Estas imágenes pueden circular en medios de información con las implicaciones legales a que den lugar, pero cual de las dos imágenes es la original y cual la alterada si a simple vista no se nota el cambio. Con este ejemplo queremos mostrar la importancia

de desarrollar algoritmos de detección de regiones clonadas, que nos permitan saber dada una sola imagen si esta fue manipulada. Para este ejemplo, la imagen de la Figura 1.1 es la imagen alterada y la Figura 1.2 es la imagen original. Para la imagen manipulada podemos ver que se copió el misil (lado derecho) y se pegó del lado izquierdo, por eso se ven dos misiles, otra alteración es con respecto al vehículo que está del lado derecho el cual se cubrió copiando tierra y poniéndola encima de dicho vehículo, por último se agregaron algunos soldados en la parte inferior (Fig. 1.1).

Figura 1.1: Imagen alterada con regiones clonadas.

Figura 1.2: Imagen original sin alteraciones.

1.2. Antecedentes

Existen diferentes tipos de análisis forense que se pueden realizar en una imagen digital, aunque este trabajo se va a enfocar en especifico a detección de áreas clonadas. A continuación describiremos algunos de los más comunes.

1.2.1. Análisis de meta-datos.

Las imágenes digitales actuales utilizan diferentes tipos de formatos, por ejemplo JPEG, PNG, GIF, etc. Algunos de estos formatos soportan incluir más información en el archivo, aparte de la información referente a la imagen digital; dicha información adicional se conoce como meta-datos. Los meta-datos pueden ser de diferentes tipos, como por ejemplo datos referentes a la ubicación geográfica donde fue tomada la foto, información técnica de la cámara que se utilizó para tomar la foto, fecha y hora de la toma, condiciones de iluminación, nombre del archivo; estos son algunos de los meta-datos más comunes. El análisis de meta-datos consiste en validar que dicha información realmente coincida con la imagen digital, existen diferentes formas de hacer esto, [Orozco, González, Villalba and Castro, 2012] realizan un trabajo en relación a este tipo de detección, que se enfoca a encontrar anomalías en fotografiás digitales que fueron capturadas con dispositivos móviles y que están almacenadas en formato JPEG/Exif. Esta técnica de análisis forense es muy poco efectiva, dado que no todos los formatos de imagen soportan los mismos meta-datos y aunado a eso también es posible realizar el borrado de los meta-datos sin alterar la imagen. Si los meta-datos son alterados o eliminados esta técnica forense no puede ser utilizada.

1.2.2. Detección de fotomontajes.

Un fotomontaje es una técnica que se ha utilizado desde antes que existieran las imágenes digitales, la cual consiste en crear una imagen como mezcla de otras, las cuales pueden ser recortadas y unidas, el resultado es una única imagen. Esta técnica

es considerada incluso como un arte, ya que realizar un fotomontaje creíble implica mucho trabajo. Gracias a las imágenes digitales y las actuales herramientas de edición, es muy común encontrar fotomontajes y estos son cada vez más fáciles de realizar. [Ng et al., 2004] en su artículo abordan este tipo de alteraciones y presentan una solución utilizando un proceso estadístico.

1.2.3. Detección de áreas clonadas

La clonación de áreas en una imagen es una técnica de manipulación que consiste en copiar áreas de una imagen sobre sí misma. En una imagen la luz, los colores, las sombras y otros factores son consistentes, debido a esta consistencia al realizar copias de una imagen en si misma las regiones clonadas son difíciles de detectar. El presente trabajo trata sobre este tema en particular.

Hay diversos autores que han abordado el tema de detección de regiones clonadas. Uno de los trabajos más citados es el de [Fridrich, Soukal and Lukáš, 2003]; en el cual hablan sobre la importancia y necesidad de poder detectar falsificaciones en las imágenes digitales. En ese mismo trabajo proponen una solución para la detección de áreas clonadas, para lo cual crean bloques traslapados, utilizando la DCT transforman dichos bloques al dominio de la frecuencia, después los cuantifican y por último buscan cuales de los bloques resultantes son similares y los marcan como clonados.

[Langille and Gong, 2006] presentan una solución también basada en bloques y por medio de técnicas de correspondencia buscan bloques similares, adicionalmente mejoran la eficiencia del algoritmo al utilizar arboles kd (kd-trees) para el agrupamiento de vecinos cercanos.

[Mahdian Babak, 2007], por su parte, presentan una solución basada en momentos invariantes a emborronado (blur moment invariants) para detectar regiones clonadas aún cuando se haya realizado algún tipo de emborronado. Estos emborronados normalmente son agregados para hacer menos visibles las alteraciones, también hacen uso de arboles kd para mejorar la eficiencia.

Por su parte [Li, Wu, Tu and Sun, 2007] presentan una solución utilizando la Transformada Wavelet Discreta (DWT) y descomposición en valores singulares (SVD), también hacen uso de bloques los cuales son transformados para así generar sus vectores de características.

[Huang, Guo and Zhang, 2008] hacen uso de la extracción robusta de características de una imagen con ayuda de SIFT (Scale-Invariant Feature Transform), una vez que tienen el conjunto de puntos importantes de la imagen los comparan para detectar duplicados, su algoritmo detecta puntos característicos duplicados y no áreas en sí, pero a partir de los puntos es más simple para un observador encontrar las áreas clonadas y también es posible a partir de dichos puntos generar un algoritmo que calcule las áreas clonadas.

Por su parte [Bayram, Sencar and Memon, 2009] en su artículo proponen un método robusto que les permite detectar regiones clonadas a pesar de la compresión, el escalamiento y rotaciones, que se pueden aplicar a las regiones clonadas. Para ello crean bloques traslapados a los cuales transforman utilizando la Transformada Fourier-Mellin (FMT), luego cuantizan los valores obtenidos y generan un vector de 45 características. Una vez que cuentan con los vectores buscan bloques duplicados utilizan dos técnicas: ordenamiento lexicográfico y conteo de filtros Bloom. En sus conclusiones comentan tener mucho mejor desempeño utilizando los filtros Bloom.

[Shivakumar and Baboo, 2011] hacen uso de SURF (Speeded-Up Robust Features) para obtener descriptores de puntos de interés en la imagen. Una vez que tiene un conjunto de puntos interesantes y sus descriptores por medio de kd-trees agrupan descriptores similares y los comparan. Sí los descriptores son similares marcan los puntos como duplicados. Por último realizan un procedimiento de verificación con la intención de quitar puntos mal clasificados. Lo que obtienen como resultado es un conjunto de puntos que corresponden con las áreas clonadas.

En el trabajo presentado por [Hashmi, Anand and Keskar, 2014] hacen uso de la Transformada Wavelet, la cual transforma al dominio de la frecuencia, para obtener

a partir de la imagen original 4 partes $\{LL, LH, HL, HH\}$. Dado que la parte con mayor cantidad de información es LL sobre esta aplican el algoritmo SIFT con el cual obtienen un conjunto de descriptores, sobre dicho conjunto se realizan búsquedas de proximidad para encontrar descriptores duplicados y marcar los puntos como clonados. La ventaja que mencionan sobre otros métodos que también hacen uso de SIFT es que al trabajar solo con las bajas frecuencias eliminan gran cantidad de información lo que ayuda al desempeño considerablemente.

El trabajo más reciente que encontramos al respecto es el presentado por [Malviya and Ladhake, 2016], el cual se basa en que al alterar una imagen se introduce inconsistencia en el color de los píxeles. Para realizar la detección de regiones clonadas en la imagen primeramente aplican un filtro a dicha imagen con el objetivo de eliminar posible ruido. Después crean bloques de $N \times M$ y para cada bloque realizan una transformación afín conocida como 8Z, ahora utilizando un Correlograma Automatico de Color (ACC, Auto Color Correlogram) hacen la extracción de características, con lo que obtienen un vector de características para cada bloque, por último buscan vectores similares para encontrar las regiones clonadas.

Existen algunos otros métodos para detectar regiones clonadas como por ejemplo los basados en mezcla de clasificadores. Los trabajos anteriormente mencionados los podemos considerar como clasificadores (dicotomizadores), ya que determinan para cada píxel si este es o no clonado. En el trabajo propuesto por [Ferreira et al., 2016] platean una propuesta basada en una mezcla de clasificadores haciendo uso de BKS (Behavior Knowledge Space). Su trabajo nos será de gran ayuda, ya que podemos utilizar los resultados que presentan para varios de los clasificadores ya mencionados anteriormente y compararlos contra los nuestros.

1.3. Objetivos de la Tesis

1.3.1. Objetivo general

El principal objetivo del presente trabajo es poder desarrollar una herramienta automática que permita detectar regiones clonadas en una imagen con compresión JPEG, que preferentemente no presente transformaciones geométricas como: rotación, escalamiento y cizallamiento.

1.3.2. Objetivos particulares

- Generar un algoritmo de detección de regiones clonadas con tiempo de ejecución competitivo.
- Buscar que el algoritmo diseñado necesite muy pocos parámetros para que sea fácil de utilizar incluso por usuarios no expertos en el área del análisis forense de imágenes digitales.
- Realizar comparaciones contra las principales propuestas que encontremos en la literatura, al menos contra las 4 mejores. Estas comparaciones deberán ser planteadas de forma objetiva utilizando una medida de error.
- El algoritmo diseñado debe competir en cuanto calidad de respuesta, según las medidas de error y tiempo de ejecución, frente a otros algoritmos.
- Hacer pruebas y validaciones que nos permitan comprobar el cumplimiento de nuestros objetivos.

1.4. Descripción de Capítulos

En el capítulo 2 abordaremos el marco teórico, enfocándonos en las herramientas que vamos a utilizar para poder desarrollar el algoritmo de detección.

En el capítulo 3 hacemos el desarrollo de nuestro algoritmo, explicamos cada una de las etapas de las que consta así como su justificación. En la parte final del capítulo presentamos el pseudocódigo del algoritmo implementado.

En el capítulo 4 realizamos pruebas y comparaciones entre los resultados que obtenemos con nuestro algoritmo y los resultados que se han obtenido con otros algoritmos, para calificar el desempeño de nuestra propuesta de forma objetiva se hace uso de imágenes que tienen regiones clonadas conocidas a priori, a los resultados obtenidos les aplicamos un análisis estadístico llamado *F-Measure*, de esta forma podemos cuantificar y comparar los resultados.

Por último en el capítulo 5 presentamos nuestras conclusiones generales en base a los resultados obtenidos y las pruebas realizadas. También se presenta una lista de posibles trabajos futuros.

Capítulo 2

Marco Teórico

En este capítulo abordaremos el marco teórico. Primeramente vamos a explicar lo que es la Transformada de Fourier Discreta (DFT) y la Transformada Coseno Discreta (DCT) ya que dichas transformadas son básicas cuando se trata del procesamiento digital de señales. Después hablaremos sobre lo que son las imágenes digitales y algunos de los diferentes formatos que existen. Nos enfocaremos en el funcionamiento del método de compresión que se utiliza en las imágenes JPEG, ya que poder detectar las regiones clonadas en este formato es más complejo. Por último haremos una breve explicación del problema de búsqueda de vecinos cercanos y abordaremos algunas de las técnicas que se utilizan para resolver este problema de forma eficiente, principalmente nos enfocaremos en el funcionamiento de los árboles kd (k-dimensional Tree) y Vp (Vantage-point Tree).

2.1. Transformada Discreta de Fourier

En esta tesis se hace uso de la Transformada Discreta Coseno, la cual es un caso especial de la Transformada Discreta de Fourier. No vamos a entrar en gran detalle sobre la Transformada Discreta de Fourier, solo abordaremos la forma más general que se utiliza en el procesamiento digital de señales. La información incluida en este

capítulo es un breve resumen del tema, si se desea obtener mayor información pueden consultar los trabajos realizados por [Lim, 1990] y [Mitra, 2011].

2.1.1. Definición

La Transformada Discreta de Fourier, llamada DFT por sus siglas en inglés (Discrete Fourier Transform), es una función lineal e invertible que pertenece al conjunto de los números complejos ($\mathbb{C}$). La DFT nos permite transformar funciones del domino del tiempo al dominio de la frecuencia. A diferencia de la Transformada de Fourier Continua, la versión discreta requiere una función que sea una secuencia discreta y de duración finita, esta transformada evalúa solo los componentes de frecuencia que sean suficientes para poder reconstruir el segmento finito que se analiza. Si la señal a transformar se extiende de forma infinita y periódica solo basta con analizar un único periodo de la señal, el cual cumple con ser finito.

2.1.2. Representación de Señales Periódicas

La exponencial compleja $\phi(n) = e^{j(2\pi/N)n}$ es una señal periódica con periodo N, a partir de esta señal podemos construir toda una familia de exponenciales complejas ahora con periodo N/k dadas por (2.1).

$$\phi_k(n) = e^{jk(2\pi/N)n} \tag{2.1}$$

recordemos que las exponenciales complejas también pueden ser escritas como una suma de un coseno y un seno (ecuación de Euler), tal como se muestra en (2.2).

$$\phi_k(n) = cos\left(\frac{2\pi k}{N}n\right) + j\ sen\left(\frac{2\pi k}{N}n\right) \tag{2.2}$$

donde k es un número entero.

Una señal discreta periódica $x(n)$ de longitud N puede ser representada como una

combinación lineal de exponenciales complejas como se muestra en (2.3).

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\phi_k(n)$$

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{jk(2\pi/N)n} \tag{2.3}$$

en donde $X(k)$ es la Transformada de Fourier de $x(n)$. La ecuación (2.3) es conocida como la Transformada Discreta Inversa de Fourier.

Existen diferentes maneras de calcular $X(k)$, una de ellas es a partir de (2.3) despejar $X(k)$. Es posible reescribir (2.3) en forma de un sistema de ecuaciones y despejar $X(k)$ resolviendo el sistema, pero no es práctico, ya que el sistema a resolver es de tamaño $N \times N$ y puede ser un sistema muy grande. En lugar de hacer esto en la siguiente sección se explica como se calcula la DFT.

2.1.3. Cálculo de la DFT

Una forma práctica en la que podemos realizar el cálculo de $X(k)$ a partir de (2.3) es multiplicando ambos lados de la ecuación por $e^{-jr(2\pi/N)n}$ (donde r es un número entero) y sumar para los N términos de la serie como sigue:

$$\begin{aligned}
\sum_{n=0}^{N-1} x(n)e^{-jr(2\pi/N)n} &= \frac{1}{N}\sum_{n=0}^{N-1}\sum_{k=0}^{N-1} X(k)e^{jk(2\pi/N)n}e^{-jr(2\pi/N)n} \\
\sum_{n=0}^{N-1} x(n)e^{-jr(2\pi/N)n} &= \frac{1}{N}\sum_{k=0}^{N-1} X(k)\sum_{n=0}^{N-1} e^{j(k-r)(2\pi/N)n}
\end{aligned}$$

Podemos notar que el término $\sum_{n=0}^{N-1} e^{j(k-r)(2\pi/N)n}$ es de la forma $\sum_{n=0}^{N-1} a^n$, donde $a = e^{j(k-r)(2\pi/N)}$. Podemos resolver dicho sumatorio tomando en cuenta dos casos, primero cuando $k = r$:

$$\sum_{n=0}^{N-1} e^{j(0)(2\pi/N)n} = \sum_{n=0}^{N-1} 1 = N$$

El siguiente caso es cuando $k \neq r$, en este caso podemos hacer uso de la sucesión dada en (2.4)

$$\sum_{n=0}^{N-1} a^n = \frac{1-a^N}{1-a} \tag{2.4}$$

y obtenemos

$$\begin{aligned}
\sum_{n=0}^{N-1} e^{j(k-rk)(2\pi/N)n} &= \frac{1-e^{j(k-r)(2\pi/N)N}}{1-e^{j(k-r)(2\pi/N)}} \\
&= \frac{1-e^{j(k-r)(2\pi)}}{1-e^{j(k-r)(2\pi/N)}} \\
&= \frac{1-1}{1-e^{j(k-r)(2\pi/N)}} = 0
\end{aligned}$$

Por lo tanto si tenemos valores de r iguales a k la suma valdrá N y cero en todos los demás casos, de tal manera que ya podamos despejar $X(k)$, la forma general en la que se representa la Transformada Discreta de Fourier esta dada por (2.5).

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-jk(2\pi/N)n} \tag{2.5}$$

Ahora ya tenemos forma de calcular la Transformada Discreta de Fourier con (2.5) y la Transformada Discreta Inversa de Fourier con (2.3).

2.1.4. Propiedades de la DFT

En está sección vamos a algunas de las propiedades de la Transformada Discreta de Fourier que son relevantes para nuestro trabajo. En la Tabla 2.1 se muestran a forma de resumen. Estas propiedades son mencionadas ya que algunas se preservan en la Transformada Discreta Coseno. No se realizá la demostración de dichas propiedades, para entenderlas más a fondo pueden consultar [Lim, 1990].

Tabla 2.1: Principales propiedades de la DFT.

Propiedad	**Descripción**	**Ecuación**
Periodicidad	Una señal cualquiera se puede considerar periódica, con periodo N, si cumple que $X(k) = X(k+N)$, para todo k.	$X(k) = X(k+N)$
Linealidad	Un sistema es lineal si satisface el principio de superposición, que engloba las propiedades de escalado y aditividad.	$DFT[ax_1(n) + bx_2(n)] = aX_1(k) + bX_2(k)$
Relación de Parseval	Relaciona como se encuentra la energía de una señal con respecto de su transformada.	$N\sum_{n=0}^{N-1} \lvert x(n)\rvert^2 = \sum_{k=0}^{N-1} \lvert X(k)\rvert^2$

Propiedad de Simetría, aproximación a la DCT

Para cualquier señal $x(n)$ la parte real la podemos calcular como:

$$\mathcal{R}[x(n)] = \frac{1}{2}[x(n) + x^*(n)]$$

y la parte imaginaria como:

$$\mathcal{I}[x(n)] = \frac{1}{2}[x(n) - x^*(n)]$$

Si $x(n)$ es una señal puramente real se cumple:

$$\frac{1}{2}[x(n) - x^*(n)] = 0$$

lo que es equivalente a (2.6).

$$x(n) = x^*(n) \tag{2.6}$$

Al aplicar la transformada de Fourier a ambos lados de (2.6) obtenemos:

$$X(k) = X^*(-k)$$

en el caso de una señal real y par sabemos que $x(n) = x(-n)$ y que $x(n) = x^*(n)$, de esta manera en el dominio de Fourier y según la definición de parte Real e Imaginaria tendremos:

$$\begin{aligned} X(k) &= X(-k) = \mathcal{R}[X(-k)] + \mathcal{I}[X(-k)] \\ X(k) &= X^*(-k) = \mathcal{R}[X(-k)] - \mathcal{I}[X(-k)] \end{aligned}$$

como vemos la única posibilidad para que esto se cumpla es que $\mathcal{I}[X(-k)] = 0$, es decir que la parte imaginaria sea cero. En base a lo anterior podemos reescribir la DFT para una señal real y par como se ve en (2.7), aquí tenemos nuestra primera aproximación a lo que es la Transformada Discreta Coseno, la cual será explicada a detalle en la sección 2.2.

$$X(k) = \sum_{n=0}^{N-1} x(n) cos\left(\frac{2\pi}{N} nk\right) \tag{2.7}$$

2.1.5. Transformada Discreta de Fourier en 2D

La DFT en 2D puede ser definida sobre una cuadrícula de tamaño $N \times M$, en lugar de un vector. Para poder obtener una ecuación que nos permita calcular la transformada y su inversa podemos generalizar la definición que ya tenemos y aplicarla primero por renglones y luego por columnas sobre nuestra señal 2D o primero por columnas y luego por renglones, ya que es indistinto. La ecuación (2.8) nos muestra como se calcula la transformada en 2D y la ecuación (2.9) como se calcula la inversa.

$$X(k,l) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n,m) e^{-j(\frac{2\pi k}{N} n + \frac{2\pi l}{M} m)} \tag{2.8}$$

$$x(n,m) = \frac{1}{NM} \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} X(k,l) e^{j(\frac{2\pi k}{N} n + \frac{2\pi l}{M} m)} \tag{2.9}$$

2.2. Transformada Discreta Coseno

Según [Khayam, 2003] en los últimos años la Transformada Discreta Coseno se ha convertido en la trasformación de facto en la gran mayoría de los sistemas visuales. Ya que esta ha sido utilizada para el desarrollo de los actuales estándares de codificación, como por ejemplo MPEG (Moving Picture Experts Group) y JTV (JRiver Recorded TV).

2.2.1. Definición

La Transformada Discreta Coseno, comúnmente solo llamada DCT de sus siglas en inglés (Discrete Cosine Transform), es una función lineal e invertible del dominio real $\mathbb{R}^N$ al dominio real $\mathbb{R}^N$. La DCT expresa una secuencia finita de N puntos en términos de una suma de funciones coseno a diferentes frecuencias, [Mitra, 2011]. En particular la DCT es una Transformada de Fourier Discreta de una señal real que se forza a ser par. Supongamos una señal real con N muestras $x^{'} = [x^{'}(0), x^{'}(1), x^{'}(2), ..., x^{'}(N-1)]$, primeramente construiremos una señal $x(n)$ con $2N$ puntos tal como vemos en (2.10).

$$x(n) = \begin{cases} x^{'}(n) & 0 \leq n \leq N-1 \\ x^{'}(-n-1) & -N \leq n \leq -1 \end{cases} \tag{2.10}$$

Podemos notar que $x(n)$ tiene un periodo $2N$ y es par con respecto al punto $n = -1/2$. Si a nuestra nueva señal le aplicamos la DFT lo que obtendríamos seria una nueva señal transformada la cual también es real, el resultado obtenido sería equivalente al que se genera cuando calculamos la DCT. Dado que no es práctico estar transformando la señal original para hacerla par la forma de calcular la DCT se

realiza de tal manera que la señal se comporte como par. Para ver más información respecto a la DCT podemos consultar [Mitra, 2011].

Por las razones mencionadas anteriormente de que la DCT es un caso especial de la DFT hay varias maneras de calcularla y existen diferentes definiciones matemáticas. La forma matemática más común que se utiliza en el procesamiento de señales digitales es la DCT de tipo II, dada por (2.11).

$$X(k) = \sum_{n=0}^{N-1} x(n)\ cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \tag{2.11}$$

donde $x(n)$ es la señal de entrada, $X(k)$ es la señal transformada y N es la longitud de la señal.

Es común que en implementaciones hechas en software se realice algún escalamiento en los componentes de la señal (Trasformada Coseno Discreta Ortogonal), con el fin de calcular de manera eficiente la transformada y la transformada inversa.

2.2.2. Aplicaciones de la DCT

La DCT es ampliamente utilizada en el procesamiento de señales digitales e imágenes, ya que tiene la propiedad de compactar la energía, dado que la mayor parte de la información tiende a concentrarse en unos pocos componentes de baja frecuencia. Algunas de las aplicaciones que se han realizado utilizando alguna forma de la DCT son: compresión de audio (MP3, AAC, WMA, Vorbis) y compresión de imágenes (JPEG).

2.2.3. Propiedades de la DCT

Varias de las propiedades que observamos en la DFT se conservan para la DCT y algunas otras son diferentes. En esta sección mencionaremos las que consideramos más importantes para nuestro trabajo, para ver más propiedades y mayor información sobre la DCT podemos consultar [Mitra, 2011] y [Khayam, 2003].

Conservación de la Energía (Relación de Parseval)

En procesamiento digital de señales la cantidad de energía que existe en una señal esta dada por (2.12)

$$\sum_{n=0}^{N-1} x(n)^2 \tag{2.12}$$

Existe una relación entre la cantidad de energía de una señal y la cantidad de energía de la señal transformada por la DCT. Está relación está dada por (2.13), lo que significa que la DCT conserva la energía.

$$\sum_{n=0}^{N-1} x(n)^2 = \frac{1}{N} X(0)^2 + \frac{2}{N} \sum_{n=1}^{N-1} X(n)^2 \tag{2.13}$$

Compactación de Energía

Si consecutivos valores de $x(n)$ están positivamente correlacionados, la DCT concentra la energía en pocas muestras de $X(k)$ y las descorrelaciona. Esta propiedad no se presenta de forma tan notoria en la DFT. En la imagen de la Figura 2.1 vemos una gráfica que nos muestra como compacta la energía la DCT y la DFT, entre mayor cantidad de muestras se toman mayor es la cantidad de energía que se abarca, cuando se toman todas las muestras tenemos el 100 % de la energía. Nótese como la gráfica crece mucho más rápido para la DCT, lo que significa que en pocas muestras hay mayor cantidad de energía.

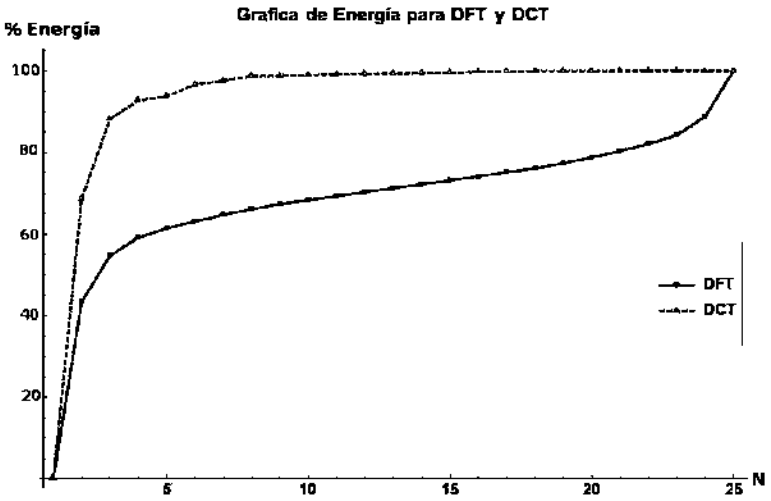


Figura 2.1: Comportamiento de la compactación de energía según la cantidad de muestras que son tomadas tanto para la DFT como para la DCT, ejemplo con una señal de longitud 24.

DCT de una constante

Dada una señal $x(n) = C$, la cual es constante para todos los valores de n, podemos calcular su transformada $DCT[x(n)]$ como se ve en (2.14).

$$X(k) = \begin{cases} CN & k = 0 \\ 0 & k > 0 \end{cases} \tag{2.14}$$

DCT de una señal invertida

Supongamos una señal $x(n) = [x(0), x(1), x(2), ..., x(N-1)]$ y su señal invertida $x'(n) = [x(N-1), x(N-2), x(N-3), ..., x(0)]$. Si calculamos $X(k) = DCT[x(n)]$ y $X'(k) = DCT[x'(n)]$ se puede observar en (2.15 y 2.16) que los resultados que se obtienen para las señales transformadas tienen la misma magnitud pero diferentes signos para los $X(i)$ cuando i es impar.

$$X(k) = [X(0), X(1), X(2), X(3), ...] \tag{2.15}$$

$$X'(k) = [X(0), -X(1), X(2), -X(3), ...] \tag{2.16}$$

Dada esta propiedad si calculamos el valor absoluto a cada punto de la señal tenemos que $|X(k)| = |X'(k)|$.

Estas propiedades y el hecho de que la compresión JPEG utiliza la DCT es lo que nos motiva a utilizarla en el presente trabajo.

2.2.4. Transformada Discreta Coseno en 2D

La DCT en 2D puede ser definida sobre una cuadrícula de tamaño $N \times M$, en lugar de un vector. Para poder obtener una ecuación que nos permita calcular la transformada y su inversa podemos generalizar la definición que ya tenemos y aplicarla primero por renglones y luego por columnas sobre nuestra señal 2D o primero por columnas y luego por renglones, ya que es indistinto. Para este trabajo la DCT en 2D va ser de gran importancia ya que es la transformación que vamos a utilizar para generar vectores de características robustos. En la ecuación 2.17 vemos como se puede calcular la DCT para una matriz.

$$X(k,l) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n,m)\ cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \tag{2.17}$$

Complejidad Computacional

La complejidad computacional asociada a la DCT es similar a la DFT, si hacemos un análisis de complejidad llegaremos a que calcular la DCT en una dimensión es del orden $O(N^2)$. Aunque existe un algoritmo rápido para la DCT similar al de la DFT, el cual tiene un orden $O(Nlog(N))$. En el caso de la DCT en 2D como sabemos puede ser calculada como la DCT en una dimensión para cada renglón y luego para cada columna, si utilizamos el algoritmo rápido ($O(Nlog(N))$) la complejidad puede ser calculada para los renglones $N \times Nlog(N)$ y para las columnas como $M \times Mlog(M)$ y la complejidad total sería la suma de ambas. Si tenemos una matriz cuadrada donde

$N = M$ la complejidad puede calcularse como $2N^2log(N)$ por lo que simplemente se considera del orden $O(N^2log(N))$.

2.3. Imagen Digital

[Ordoñez, 2005] menciona que existen dos modos principales que se utilizan para representar una imagen digital: Imagen como Mapa de Bits e Imagen Vectorial, cada representación está diseñada con diferentes propósitos. Las imágenes como mapa de bits, también conocidas como bitmaps, son cuadrículas que almacenan puntos de color en su interior, estos puntos son conocidos como píxeles, a pesar de que la imagen es un conjunto de puntos da la sensación de ser una imagen de tono continuo. Mientras tanto las imágenes vectoriales, o gráficos orientados a objetos, lo que almacenan son reglas para dibujado de objetos geométricos, las reglas generalmente contienen información referente a la figura geométrica, la ubicación, el color de relleno, el color del trazo, etc. El modo de representación que se utiliza en fotografiás digitales es mapa de bits, en nuestro trabajo nos vamos a enfocar en este tipo de representación, la cual definiremos de manera formal.

2.3.1. Definición

[Chris Solomon, 2011] define una imagen digital como una representación discreta de datos que posee tanto información espacial (disposición) como de intensidad.

Disposición Espacial

La disposición espacial (layout) es un arreglo bi-dimensional (2D) discreto, que forma una cuadrícula de tamaño $N_r \times N_c$, donde N_r es el total de renglones y N_c es el total de columnas. Un elemento perteneciente a dicha disposición espacial puede ser representado como $I(r, c)$ donde r y c nos proporcionan una coordenada en 2D ($r \in \{0, 1, 2, 3, ..., N_r - 1\}$ y $c \in \{0, 1, 2, 3, ..., N_c - 1\}$). r nos da la posición por

renglones, mientras que c nos da la posición por columnas. Cada elemento de esta forma $(I(r, c))$ es llamado píxel, en la Figura 2.2 se aprecia visualmente como esta compuesta la cuadricula de tamaño $N_r \times N_c$ y marcado lo que es un píxel.

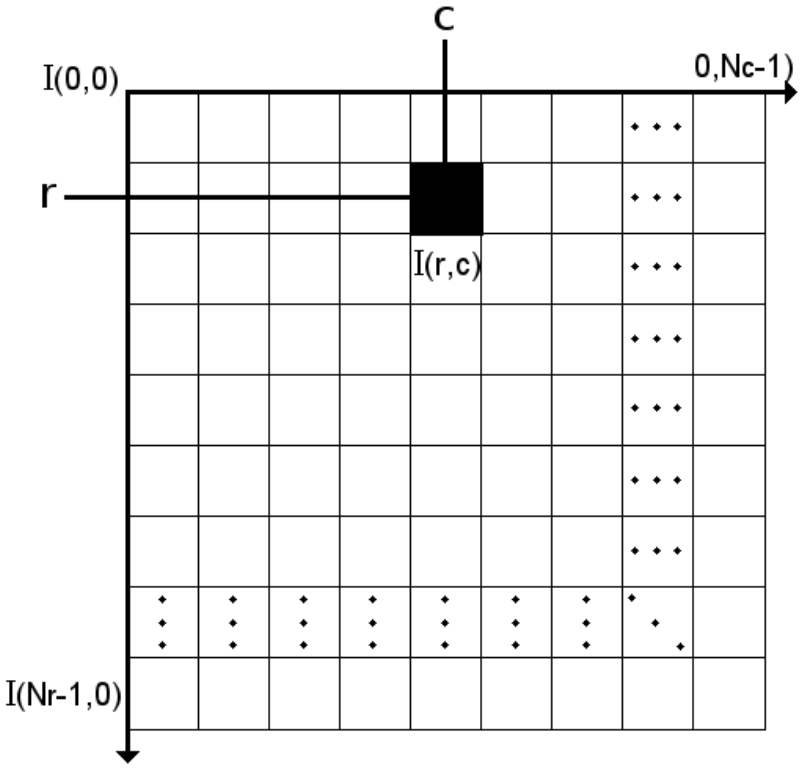


Figura 2.2: Representación de la disposición espacial de una imagen digital. $I(r, c)$ representa al píxel en la coordenada (r, c).

Intensidad

La intensidad son los valores numéricos que almacena un píxel, se almacena un único valor por cada canal de color de la imagen, ya que una imagen puede contener uno o más canales de color, dando como resultado que un píxel sea un vector. Dichos valores definen como va lucir el color de un píxel en particular. En el caso más simple cuando tenemos una imagen con un solo canal, el píxel contiene un único

valor numérico que se interpreta en un mapa de escala de gris, donde el valor mínimo posible representa al color Negro y el valor máximo el color Blanco. En el caso de imágenes a color es normal hacer uso de tres canales, el modelo más común es conocido como RGB donde cada canal representa un color especifico (Rojo, Verde y Azul respectivamente), por lo que los píxeles deben almacenar tres valores, siendo vectores de la forma:

$$I(r, c) = [R(r, c), G(r, c), B(r, c)]$$

No entraremos en mayor detalle sobre lo que son las imágenes digitales, para más información recomendamos consultar [Jain, 1989] y [Chris Solomon, 2011].

2.3.2. Formatos en Imágenes Digitales

En la actualidad hay gran cantidad de formatos en los cuales se puede almacenar una imagen digital. En la Figura 2.3 se muestra una clasificación que podemos realizar para los diferentes tipos de formato, según su funcionalidad. Note que hay formatos flexibles que aparecen en múltiples categorías. Brevemente describiremos cada uno de los formatos mencionados en la Figura 2.3.

- TIFF: Sus siglas vienen de "Tag Image File Format", de todos los formatos mencionados este es el más flexible, ya que permite almacenar una imagen sin compresión, con compresión con perdida y con compresión sin perdida. A pesar de tener esta propiedad de flexibilidad es muy común que se utilice solamente para almacenar imágenes sin compresión y sin perdida, dando como resultado archivos que necesitan gran cantidad de almacenamiento, pero con gran calidad. Las imágenes almacenadas en este formato tiene una profundidad de color de 32 bits, lo que significa que cada píxel puede tomar 2^{32} valores diferentes.

- BMP: Su nombre es "Bit Map"de ahí la abreviación (BMP). Este es un formato de almacenamiento sin perdida, puede utilizar compresión o no usarla, siendo flexible. Es propiedad de Microsoft, por lo que existen implementaciones nativas

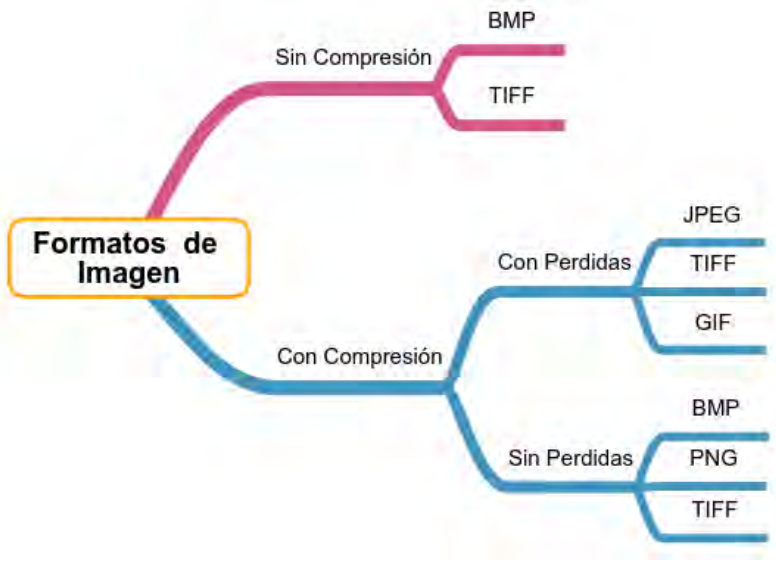


Figura 2.3: Clasificación de diferentes formatos para imágenes digitales.

solo en el sistema operativo Windows, fuera de dicho sistema es muy poco utilizado. Este formato tiene una profundidad de color variable de $1, 4, 8$ y 24 bits, la máxima cantidad de colores permitidos para cada píxel es 2^{24}. Cuando se utiliza la máxima profundidad de color los archivos resultantes tienen un tamaño superior que otros formatos con la misma profundidad de color.

- GIF: Su nombre corresponde a las siglas de "Graphics Interchange Format", la principal funcionalidad de este formato es la capacidad de poder almacenar varias ventanas (frames) las cuales se muestran de manera secuencial generando animaciones. Este formato no es apropiado para imágenes de alta calidad, ya que solo utiliza una paleta de 256 colores diferentes, es decir que solo tiene una profundidad de color de 8 bits.

- PNG: Sus siglas significan "Portable Networks Graphics", que es el nombre del grupo que lo desarrolla. Es uno de los formatos más utilizados en la actualidad, fue diseñado con la intensión de ser un formato utilizado en redes (de ahí su nombre), principalmente para Internet. Cuenta con una profundidad de color de

24 bits y un algoritmo de compresión sin pérdidas muy potente, generando archivos que requieren poco espacio de almacenamiento y por ende generan menor cantidad de transmisión de datos sobre la red. Dicho formato puede soportar un canal adicional de color que se utiliza para almacenar la transparencia.

- JPEG: Este formato toma su nombre de "Joint Photographic Experts Group", el cual es el grupo que lo desarrolló. Es el formato más común para almacenar fotografiás digitales e imágenes de tono continuo. Utiliza un método de compresión con pérdidas, el cual degrada la calidad de la imagen en favor de archivos cuyo tamaño sea menor. La gran mayoría de cámaras digitales actuales lo usan por defecto para almacenar las fotografiás tomadas. Al igual que otros formatos tiene una profundidad de color de 24 bits que son $16,777,216$ colores diferentes.

En el presente trabajo nos enfocaremos solamente en los formatos JPEG y PNG, dado que son los más utilizados para almacenar imágenes con compresión con pérdidas y sin pérdidas respectivamente.

2.4. Reto al Utilizar JPEG

Ya que como parte del objetivo general se desea detectar regiones clonadas en imágenes JPEG, es importante hablar un poco acerca de la complejidad que existe en esto. El proceso de compresión que utiliza JPEG realiza una transformación sobre los valores de una imagen, pero dicha transformación es dependiente de la posición geométrica, esto sucede ya que JPEG divide la imagen en bloques de 8×8 y cada bloque lo procesa de manera independiente. Ahora podemos empezar a notar que una región clonada al aparecer en otra zona de la imagen difícilmente va a estar alineada con los bloques de la misma manera como se encuentra la región original, provocando que al realizarse el proceso de compresión las regiones clonadas sean procesadas de forma diferente y por lo tanto tengan valores diferentes.

Supongamos una imagen I sin compresión JPEG, tomamos el valor de dos píxeles de la imagen que sean iguales pero estén en posiciones distintas, a los que llamaremos $I(i,j)$ y $I(k,l)$. Dado que ambos píxeles fueron seleccionados por tener valores iguales, se cumple que:

$$I(i,j) - I(k,l) = 0$$

A dicha imagen le aplicamos el proceso de compresión JPEG, obteniendo una nueva imagen $I^{'}$.

$$I^{'} = JPEG[I]$$

Esto nos genera un nuevo par de píxeles $I^{'}(i,j)$ y $I^{'}(k,l)$, y al calcular su diferencia tenemos que:

$$I^{'}(i,j) - I^{'}(k,l) \neq 0$$

Es decir que el proceso de compresión alteró los valores de manera diferente, provocando que ya no sean iguales los píxeles. Por lo tanto nuestra propuesta debe ser robusta a estos cambios, para lograrlo entraremos en más detalle sobre el funcionamiento del algoritmo de compresión JPEG.

2.5. Algoritmo de Compresión JPEG

JPEG es el nombre de un comité de expertos "Grupo Conjunto de Expertos en Fotografía"(Joint Photographic Experts Group) que creó un estándar de compresión y codificación de archivos para imágenes digitales de tono continuo.

2.5.1. Definición

JPEG es un método de codificación comúnmente utilizado en la compresión con pérdida, para imágenes digitales, principalmente utilizado en imágenes fotográficas. Las imágenes procesadas con este método son almacenadas en un archivo con formato JPEG/Exif o JPEG/JFIF (estos diferentes formatos generalmente solo son llamados

JPEG). Es importante resaltar que no el lo mismo el método de codificación JPEG que el formato JPEG, pero ambos términos están estrechamente relacionados. Toda la información referente al método de compresión que utiliza JPEG puede ser encontrada en el estándar [ISO/IEC 10918-1, 1994].

La idea principal de la codificación JPEG es eliminar información no crucial en una imagen digital, esto es posible debido a que nuestros ojos no captan los cambios abruptos de color o iluminación en áreas pequeñas, por lo que si esta información es eliminada de una imagen digital nuestros ojos percibirán relativamente igual la imagen original que la imagen que no cuenta con esta información, pero en cambio el espacio necesario para almacenar la imagen será menor gracias a la información que ha sido eliminada.

2.5.2. Etapas para generar la compresión

En este apartado vamos a abordar las principales etapas del método de codificación JPEG, el entender como se realiza la codificación nos puede permitir encontrar un método para detectar regiones clonadas en imágenes con compresión JPEG.

El algoritmo de codificación JPEG en términos generales consta de las siguientes partes: Transformación del espacio de color, submuestreo de canales, separación en bloques, transformación de los bloques utilizando la DCT, cuantización digital y codificación entrópica. Las cuales están descritas a detalle en [Wallace, 1992]. Vamos a explicarlas brevemente para entender de forma general como trabaja JPEG.

Transformación del espacio de color

Las imágenes digitales comúnmente son almacenadas o capturadas en un espacio de color RGB. Lo primero que se hace es transformar el espacio de color RGB al espacio de color YCbCr. Este espacio de color consta de tres canales donde: Y es la componente de luminancia o brillo, que es la imagen en escala de gris.

Cb y Cr son la crominancia, Cb (diferencia de azul) relativiza la imagen entre el azul y rojo; mientras Cr (diferencia de rojo) relativiza la imagen entre verde y rojo.
Este espacio de color es más adecuado para hacer la detección de cambios de iluminación y color, ya que cada canal es lo que almacena, Y son los cambios de iluminación y Cb y Cr son los cambios en color.

Submuestreo de canales

Se puede realizar un proceso de submuestreo en los canales Cb y Cr, recordemos que estos canales contienen la información sobre los cambios de color. Este proceso de submuestreo es posible ya que nuestros ojos son más sensibles a los cambios de brillo que a los cambios de color, por lo que es posible considerar menos información sobre el color. Este paso no es obligatorio y puede no ser realizado.

Separación en bloques

Cada canal de la imagen se subdivide en bloques de 8×8 píxeles, dichos bloques no se traslapan y se manejan de forma independiente. El primer bloque de cada canal se encuentra en la parte superior izquierda.

Transformación de los bloques utilizando la DCT

Una vez que se cuenta con los bloques individuales se procede a realizar la transformación, para esto es necesario que los valores estén centrados sobre 0, las imágenes de 8 bits se encuentran en un rango de valores $[0, 255]$ por lo que es necesario restar 128 a cada valor para quedar en un rango $[-128, 127]$. Ahora los bloques son transformados al dominio de la frecuencia por medio de la DCT, dando como resultado bloques de 8×8 pero cuya interpretación esta relacionada con coeficientes de frecuencias.

Cuantización Digital

Hasta este punto toda la información referente a la imagen solo ha sido transformada de diferentes maneras pero aún no se había realizado ninguna pérdida de información. En este paso cada bloque es dividido punto a punto por una matriz de cuantización y redondeado al valor entero más cercano. Existen dos matrices de cuantización una para el brillo (canal Y) y otra para el color (canales Cb y Cr), estás matrices están diseñadas con la intensión de eliminar información que no altere de forma sustancial lo que se percibe en la imagen al observarla. En la ecuación (2.18) se muestra la matriz típica de cuantización para el canal del brillo o luminancia (Y), mientras que en (2.19) se muestra la matriz de cuantización que se utiliza para el color (canales Cb y Cr). Para ver información detallada respecto a la matrices de cuantización se puede consultar el trabajo de [Watson, 1993].

$$Q_l = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \tag{2.18}$$

$$Q_c = \begin{pmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{pmatrix} \tag{2.19}$$

Supongamos un bloque G de dimensión 8×8 y una matriz de cuantización Q (ya sea Q_l 2.18 o Q_c 2.19) también de las mismas dimensiones, nuestro nuevo bloque ya cuantizado, que llamaremos B será calculado como se muestra en la ecuación 2.20.

$$B(i,j) = round\left(\frac{G(i,j)}{Q(i,j)}\right) \tag{2.20}$$

donde (i,j) representan el elemento en el renglón i y la columna j.

Nótese como este proceso de redondeo es el responsable de que se genere pérdida de información.

Codificación Entrópica

La codificación entrópica es una forma especial de compresión sin pérdida, para realizar la codificación primeramente se ordenan los elementos del bloque cuantizado (B) en un vector, el orden en que se toman los elementos del bloque es en forma de Zigzag (ver Figura 2.4). Esta forma especifica de ordenar los elementos de la matriz en zigzag es con la intensión de poner al inicio del vector los coeficientes de las frecuencias más bajas y al final los coeficientes de las frecuencias más altas, ya que estos últimos son de menos interés y normalmente valen cero debido al proceso de cuantización. Una vez que se tiene el vector de longitud 64 con los componentes ordenados se aplica el algoritmo de codificación Huffman, el cual genera la compresión del vector

sin pérdida de información. Para ver más acerca de esta codificación proponemos revisar el artículo original [Huffman, 1952].

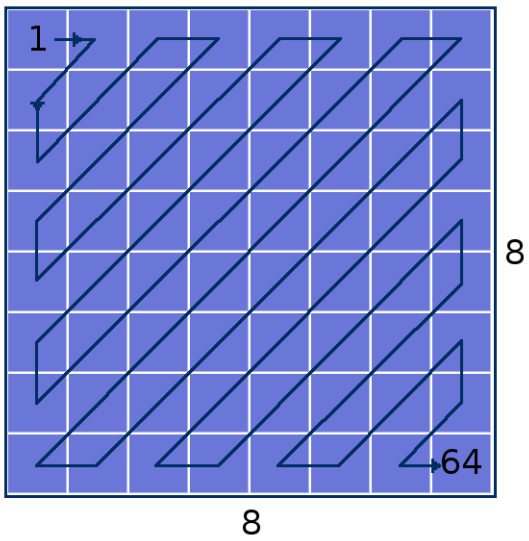


Figura 2.4: Ordenamiento de los componentes de un bloque de 8×8 en forma de zigzag para la codificación entrópica utilizado por JPEG.

Todo esté proceso es completamente reversible con lo cual es posible obtener nuevamente la imagen digital a partir de los vectores comprimidos. La forma como se almacena la información ya comprimida no es relevante para nosotros, así que solo mencionaremos que existen dos tipos principales de contenedores (archivos) que se utilizan para almacenar imágenes comprimidas usando JPEG, el más común es JPEG/Exif el cual utilizan la mayoría de cámaras digitales. El otro contenedor que se utiliza es JPEG/JFIF. Es importante hacer énfasis en que no es lo mismo el contenedor (archivo) que almacena la imagen que el método de compresión, normalmente siempre se hace referencia a JPEG de forma indistinta.

2.5.3. Ejemplo para un bloque

Supongamos que contamos con un bloque G de 8×8 resultado de dividir el canal del brillo (luminancia) de una imagen, dicho bloque contiene los valores que se muestran a continuación.

$$G = \begin{pmatrix} 42 & 106 & 122 & 132 & 80 & 246 & 174 & 68 \\ 126 & 98 & 196 & 46 & 90 & 90 & 166 & 86 \\ 124 & 138 & 136 & 246 & 222 & 268 & 240 & 196 \\ 126 & 116 & 142 & 244 & 208 & 212 & 140 & 138 \\ 94 & 66 & 134 & 174 & 90 & 78 & 118 & 138 \\ 70 & 194 & 134 & 102 & 160 & 28 & 46 & 184 \\ 112 & 148 & 64 & 146 & 182 & 146 & 116 & 218 \\ 176 & 132 & 156 & 90 & 174 & 194 & 220 & 202 \end{pmatrix}$$

Podemos ver esos mismos valores como la imagen en escala de gris que representan en la Figura 2.5.

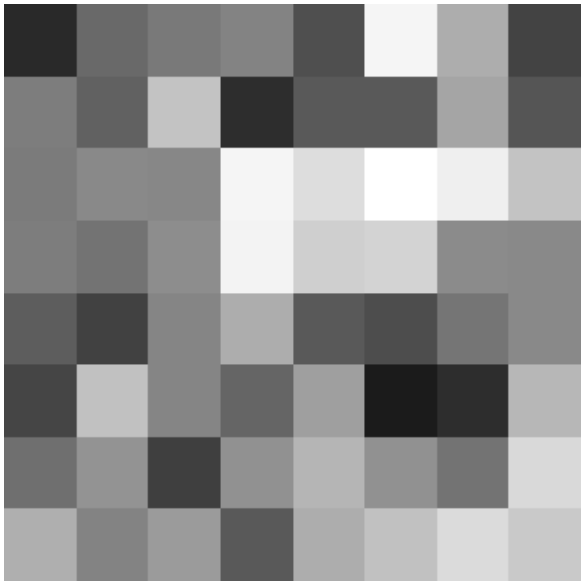


Figura 2.5: Bloque de 8×8 píxeles, del canal del brillo de una imagen.

Ahora al bloque G lo cambiamos de rango restando 128 a cada uno de los valores,

con lo que nos queda de la siguiente forma:

$$G = \begin{pmatrix} -86 & -22 & -6 & 4 & -48 & 118 & 46 & -60 \\ -2 & -30 & 68 & -82 & -38 & -38 & 38 & -42 \\ -4 & 10 & 8 & 118 & 94 & 140 & 112 & 68 \\ -2 & -12 & 14 & 116 & 80 & 84 & 12 & 10 \\ -34 & -62 & 6 & 46 & -38 & -50 & -10 & 10 \\ -58 & 66 & 6 & -26 & 32 & -100 & -82 & 56 \\ -16 & 20 & -64 & 18 & 54 & 18 & -12 & 90 \\ 48 & 4 & 28 & -38 & 46 & 66 & 92 & 74 \end{pmatrix}$$

Una vez que se encuentra en el rango correcto calculamos su DCT, quedando como se ve a continuación:

$$G = \begin{pmatrix} 107.0 & -80.2 & -39.2 & -8.37 & -6.89 & -4.83 & 1.10 & -8.67 \\ -14.3 & -3.88 & -40.5 & 19.6 & -56.6 & 49.5 & 13.4 & -11.4 \\ -10.8 & -16.9 & 36.5 & 33.1 & -53.0 & -2.94 & 11.2 & 15.7 \\ -143. & 31.9 & 7.33 & -43.7 & -3.82 & -27.6 & -7.91 & -7.76 \\ 1.24 & -10.4 & -20.6 & 19.9 & -17.6 & 39.9 & 19.8 & -15.9 \\ 63.7 & -29.8 & -43.0 & 29.6 & 19.0 & -21.0 & -10.5 & -55.6 \\ 57.5 & -16.2 & -13.1 & 1.23 & -8.76 & -17.3 & -10.0 & -14.6 \\ 21.7 & -37.8 & -5.06 & 10.0 & 5.91 & 26.6 & -3.02 & -27.9 \end{pmatrix}$$

El siguiente paso a realizar es la cuantización por que hacemos uso de la ecuación (2.20). Como el bloque pertenece al canal de brillo usamos la matriz de cuantización dada por Q_l (2.18), con lo que obtenemos el valor del nuevo bloque ya cuantizado

(B):

$$B = \begin{pmatrix} 7 & -7 & -4 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -3 & 1 & -2 & 1 & 0 & 0 \\ -1 & -1 & 2 & 1 & -1 & 0 & 0 & 0 \\ -10 & 2 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 3 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

El último paso que debemos de realizar es el de la codificación entrópica, para esto aplicamos a B el esquema de zigzag que se explicó anteriormente, con lo que obtenemos un vector dado por la secuencia: [7, -7, -1, -1, 0, -4, 0, -3, -1, -10, 0, 2, 2, 1, 0, 0, -2, 1, 0, 0, 3, 1, -1, -1, -1, -1, 1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]. El algoritmo JPEG aplica una codificación Huffman para comprimir la información, pero dado que no entraremos en detalle del funcionamiento de la codificación Huffman, imaginemos un proceso de compresión muy simple que es no almacenar los ceros que se encuentran al final del vector, quedando la secuencia como: [7, -7, -1, -1, 0, -4, 0, -3, -1, -10, 0, 2, 2, 1, 0, 0, -2, 1, 0, 0, 3, 1, -1, -1, -1, -1, 1, 0, 0, 0, 0, 0, 0, -1].

Podemos notar como podemos pasar de almacenar 64 valores a solo necesitar almacenar 34 ya que los últimos 30 eran ceros y podíamos evitar almacenarlos, lo que nos da una tasa de compresión del 46.8 % para nuestro bloque de ejemplo. Con este ejemplo se puede mostrar la gran cantidad de información que es posible eliminar, sin degradar notablemente el contenido de la imagen digital, esto debido a que las matrices de cuantización están diseñadas para que la información que se elimine sea poco perceptible. A pesar de no haber utilizado la codificación Huffman para este ejemplo lo que se quiere hacer notar es como al tener tantos ceros es posible comprimir en gran medida la información.

2.6. Búsqueda de Vecinos Cercanos

El problema de búsqueda de vecinos cercanos es muy común en el área de clasificación y reconocimiento de patrones, por lo que ha sido altamente investigado. Por ejemplo [Cover and Hart, 1967] en su trabajo presentan una forma de clasificar patrones con base en vecinos cercanos, utilizando una regla de vecindad basada en una probabilidad de error. En la literatura que trata sobre el tema es posible encontrar información sobre vecinos cercanos y los diferentes usos que se les dan. El libro "Pattern Classification"de [Duda, Hart and Stork, 2012] trata temas al respecto, donde hablan sobre estimación de vecinos cercanos, la regla de vecinos cercanos y métricas y clasificación utilizando vecinos cercanos.

De manera general con base en la literatura consultada podemos resumir el problema al siguiente enunciado: "Dado un conjunto de puntos en un espacio métrico y un punto de interés de dicho conjunto, encontrar todos los puntos que están cercanos (según una métrica o medida de distancia) a ese punto de interés". A continuación abordaremos este problema más a fondo.

2.6.1. Definición

Existen diferentes formas del problema, como por ejemplo encontrar el vecino más cercano, encontrar los k vecinos más cercanos o encontrar los vecinos que se encuentran a una distancia menor a ϵ. En general, la forma de encontrar la solución es similar en todos ellos. Por esto vamos a explicar el problema de encontrar a los k vecinos más cercanos, ya que es el problema más genérico.

Dado un conjunto de puntos $\{p_1, p_2, p_3, ..., p_N\}$ con $p_i \in \mathbb{R}^M$, donde M es la dimensión de cada punto, y una función de distancia $d(p_i, p_j)$, la cual nos dice que tan cerca se encuentra el punto p_i del punto p_j (por ejemplo la distancia euclidiana). Encontrar los k vecinos más cercanos de p_i.

Para esto se puede hace uso de diferentes técnicas, [Andrés, 1996] en su tesis

doctoral menciona las siguientes:

- Búsqueda exhaustiva.
- Búsqueda de vecinos en espacios métricos.

Explicaremos brevemente en que consiste cada una de estas técnicas.

2.6.2. Búsqueda Exhaustiva

Este es el nombre que se le da al método más simple de buscar vecinos cercanos, el cual consiste en realizar todas las comparaciones posibles, es decir para encontrar a los vecinos más cercanos de p_i se debe calcular la distancia contra todos los p_j, $d(p_i, p_j) \forall j \in \{1, 2, 3, ..., N\}$ y $j \neq i$. Una vez que contamos con todas las distancias calculadas podemos ordenar los puntos p_j de menor a mayor distancia y quedarnos con los k menores. Como se puede notar para calcular los vecinos cercanos de un único punto es necesario realizar $N - 1$ operaciones de distancia, en caso de querer calcular los vecinos cercanos de nuestros N puntos debemos de realizar el proceso anterior N veces, lo que nos lleva a que en total hacemos $N \times N - 1$ operaciones, esto da lugar a un algoritmo de complejidad $O(N^2)$.

La ventaja de esta solución es que es simple de implementar y el resultado obtenido es exacto, la desventaja es que en la mayoría de los casos no es práctica ya que requiere mucho tiempo de cómputo cuando el conjunto de puntos es muy grande.

2.6.3. Búsqueda en Espacios Métricos

La mayoría de los métodos propuestos que se encuentran en la literatura pertenecen a este grupo, debido a que en los problemas de clasificación y reconocimiento de patrones es posible representar nuestros datos en forma de vectores (comúnmente como vectores de características) a los cuales es posible asignar una métrica de distancia la cual nos genera un espacio métrico, por ejemplo la distancia euclidiana

entre dos vectores nos genera un espacio métrico. Estos métodos utilizan estructuras de datos, generalmente árboles, las cuales son diseñadas para segmentar el espacio de tal manera que a la hora de realizar búsquedas de vecinos cercanos solo sea necesario realizarlas en un segmento y no en todo el espacio. Las soluciones que realizan un particionado del espacio de búsqueda son conocidas como búsquedas aproximadas de vecinos cercanos ([Yianilos, 1993]).

Las ventajas de estos métodos es que resuelven el problema de la complejidad computacional asociada a las búsquedas exhaustivas. Como desventaja tenemos que el crear dichas estructuras también requiere tiempo de computo y dependiendo de como se realice el particionado del espacio de búsqueda puede ser imposible encontrar a los k verdaderos vecinos cercanos de un punto dado.

[Yianilos, 1993] en su artículo presenta un compendio de las estructuras de datos y algoritmos que pueden ser utilizados para realizar búsquedas de vecinos cercanos en espacios métricos. Las estructuras que nosotros vamos a utilizar para realizar la búsqueda de vecinos cercanos van a ser k-dimensional tree y Vantage-point tree.

2.6.4. k-d Tree

Un árbol k-d (k-dimensional tree) es una estructura de datos que pertenece a los árboles binarios, la cual organiza puntos en un espacio euclidiano de k dimensiones, generando una partición del espacio. Un árbol k-d crea particiones utilizando solo hiperplanos perpendiculares a los ejes del sistema de coordenadas. Primeramente realiza una partición utilizando el primer eje, la siguiente partición la realiza con sobre el siguiente eje y así sucesivamente. Cuando se llega al último eje, si aún es necesario seguir particionando, se vuelve a particionar utilizando el primer eje. Por ejemplo para un espacio métrico en dos dimensiones con ejes de coordenadas x y y, la primera partición puede hacerse en base a x, la siguiente en base a y, la tercera nuevamente en base a x y así de manera sucesiva. En general es indistinto sobre cual eje se empieza a particionar primero. Para seleccionar el punto sobre el que se

realizará la partición se deben ordenar los puntos en base al eje sobre el cual se va particionar, una vez ordenados se selecciona el punto que es la media, en caso de haber dos puntos en la media se toma uno de forma indistinta. El realizar la construcción del árbol utilizando este método no lleva a generar un árbol balanceado.

En la Figura 2.6 vemos como se ve el particionado de un espacio 2D utilizando un árbol k-d, nótese que la primera partición (la mayor) fue hecha sobre el eje y.

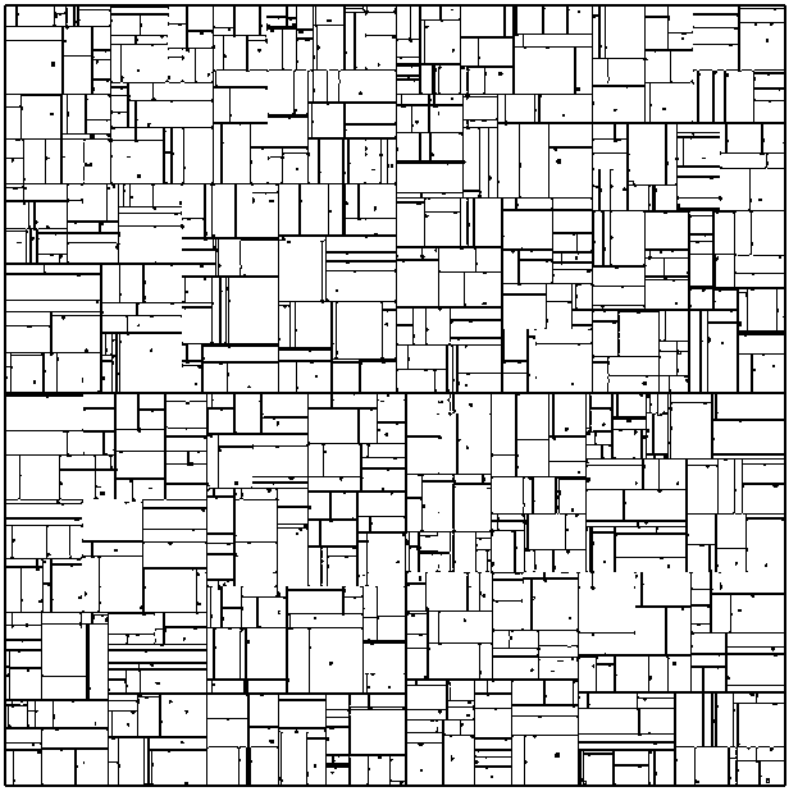


Figura 2.6: Particionado realizado utilizando un árbol kd. Figura tomada de [Yianilos, 1993].

Complejidad

Algunas de las operaciones que pueden ser implementadas para un árbol kd son las siguientes:

- Construcción del árbol para N elementos. Complejidad $O(Nlog(N))$.

- Insertar un elemento sobre un árbol ya construido. Complejidad $O(log(N))$.

- Eliminar un elemento del árbol. Complejidad $O(log(N))$.

- Buscar un vecino cercano. Complejidad $O(log(N))$.

- Buscar n vecinos cercanos. Complejidad $O(nlog(N))$.

2.6.5. Vp Tree

Un árbol Vantage-point (Vp Tree) es una estructura de datos basada en un árbol binario de búsqueda, de forma similar al árbol kd, que nos permite segmentar o particionar un espacio métrico, esto lo hace eligiendo un punto de dicho espacio (Vantage-point) generando una circunferencia de radio τ donde ese punto es el centro, por lo que los demás puntos quedan en dos partes: los que están dentro de la circunferencia y los que están fuera. Para cada una de las partes este proceso se repite de forma recursiva, generando particiones cada vez más pequeñas (con un τ menor). En el árbol resultante se cumple que los vecinos en el árbol son muy probablemente vecinos en el espacio métrico.

En la Figura 2.7 vemos como se ve el particionado de un espacio 2D utilizando un árbol Vp, los semicírculos más grandes son los generados por las primeras particiones, los más pequeños por las últimas.

Algunas implementaciones realizadas de árboles Vp calculan el punto de ventaja (Vantage-point) de manera aleatoria, algunos trabajos presentan diferentes formas de calcular dicho punto. Para más información sobre el funcionamiento, desempeño y uso de estos árboles pueden consultar el trabajo de [Brin, 1995].

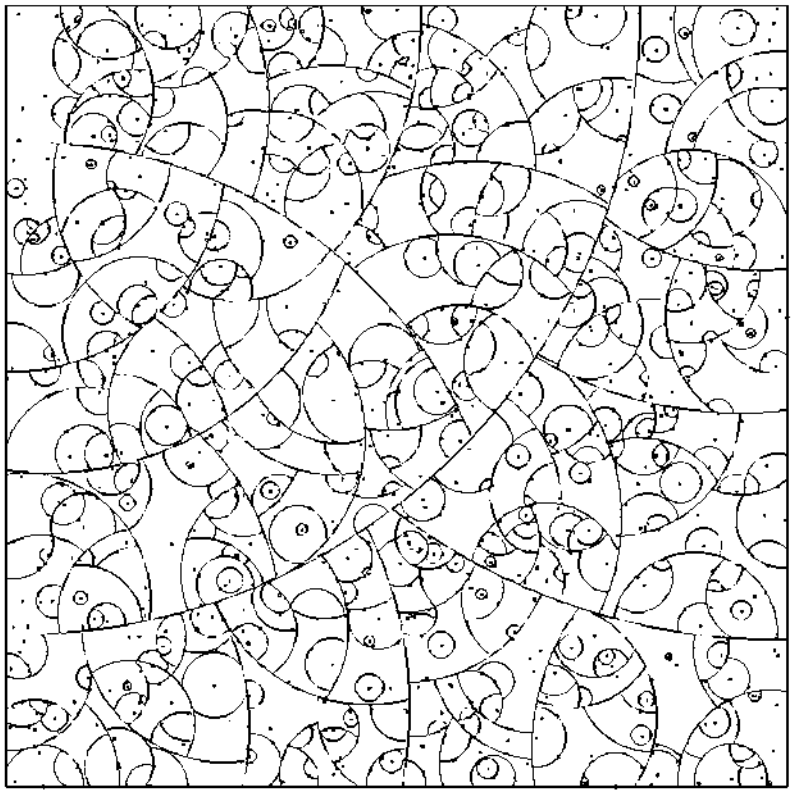


Figura 2.7: Particionado realizado utilizando un árbol Vp. Figura tomada de [Yianilos, 1993].

Complejidad

De forma similar que en los árboles kd las operaciones que pueden ser implementadas para un árbol Vp son las siguientes:

- Construcción del árbol para N elementos. Complejidad $O(Nlog(N))$.
- Insertar un elemento sobre un árbol ya construido. Complejidad $O(log(N))$.
- Eliminar un elemento del árbol. Complejidad $O(log(N))$.
- Buscar un vecino cercano. Complejidad $O(log(N))$.
- Buscar n vecinos cercanos. Complejidad $O(nlog(N))$.

Podemos ver que las operaciones en ambas estructuras de datos son las mismas y su complejidad computacional también lo es, lo que cambia es la forma en la que se realiza el particionado del espacio cuando se construye el árbol.

2.6.6. Ejemplo para k-d tree y Vp tree

Vamos a mostrar un ejemplo simple de como se realiza el particionado del espacio para los diferentes tipos de árbol que ya explicamos. Para esto vamos a definir un conjunto de puntos en un espacio bidimensional, en la Tabla 2.2 se muestran los 10 puntos que vamos a utilizar en este ejemplo.

Tabla 2.2: Puntos para ejemplo del particionado con k-d tree y Vp tree.

Punto	**Coordenada x**	**Coordenada y**
P1	8	4
P2	1	4
P3	3	8
P4	2	8
P5	7	1
P6	6	2
P7	1	10
P8	5	0
P9	3	10
P10	6	9

En la Figura 2.8 se muestran dibujados los números de cada punto de la Tabla 2.2 sobre un espacio bidimensional. De forma gráfica es posible notar fácilmente quienes son los vecinos cercanos de cada punto.

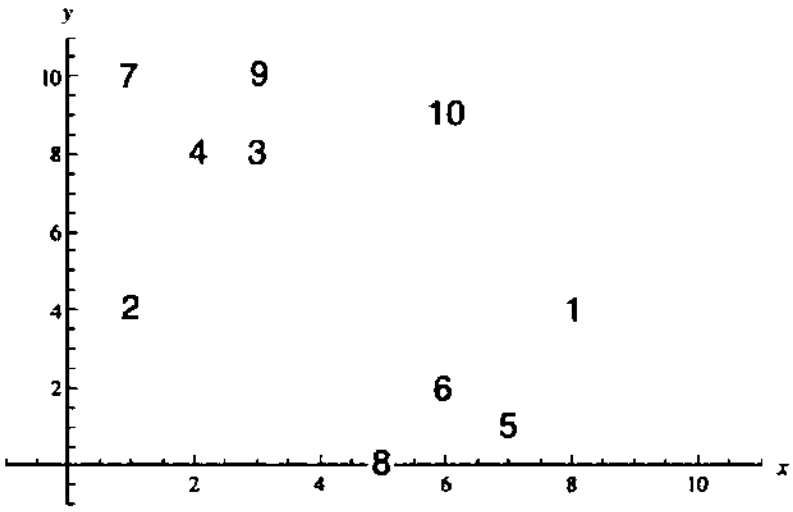


Figura 2.8: Dibujo de los puntos de la Tabla 2.2 en un espacio bidimensional.

k-d tree

Lo primero que se debe de hacer para generar el árbol k-d es ordenar los puntos por una de sus coordenadas, nosotros iniciaremos por la coordenada en x. Una vez ordenados los puntos tomamos el que está en medio, el cual se vuelve el nodo principal del árbol, para nuestro ejemplo nos quedamos con el punto P8 y se genera la primera partición. Ahora tenemos dos conjuntos de puntos, los que están a la izquierda de P8 y los que están a la derecha, para cada conjunto repetimos el procedimiento de ordenarlos, pero ahora sobre la coordenada y. Una vez ordenados ambos conjuntos tomamos el punto que queda en medio de cada uno y dichos puntos serán los nodos hijos de P8. Este procedimiento se sigue repitiendo de forma recursiva para cada nueva partición, cambiando de coordenada sobre la que se ordena hasta que ya no haya puntos por meter en el árbol.

Siguiendo el procedimiento anterior obtenemos una segmentación del espacio tal como se muestra en la Figura 2.9 y el árbol al que se llega se muestra en la Figura 2.10.

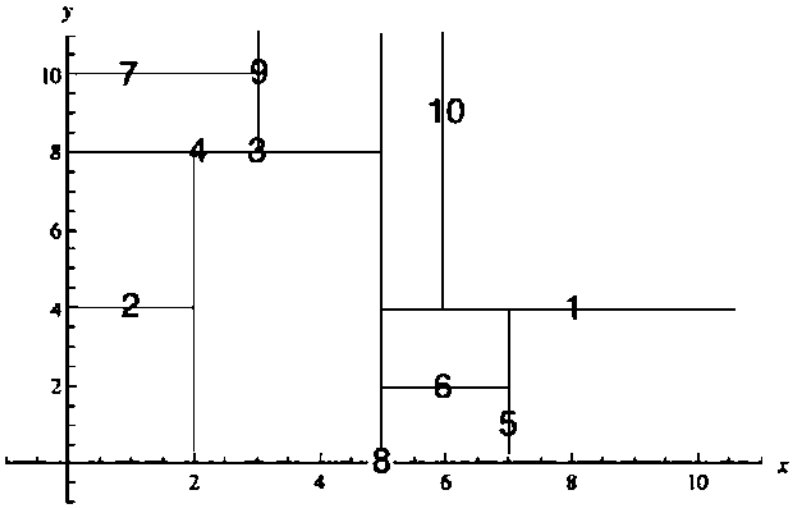


Figura 2.9: Espacio particionado siguiendo la construcción de un k-d tree.

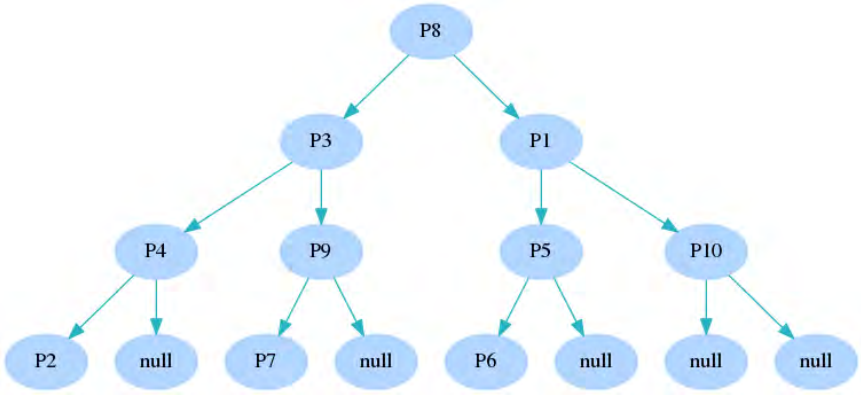


Figura 2.10: k-d tree generado para los puntos de la Tabla 2.2.

Vp tree

Para generar el Vp tree lo primero que se debe realizar es tomar un "Punto de Ventaja"(Vantage Point), en este caso lo haremos de manera aleatoria. Una vez que ya se tiene el punto de ventaja se debe trazar una circunferencia, la cual particiona el espacio en dos conjuntos, los puntos dentro de la circunferenia y los puntos fuera de ella. El punto de ventaja se toma como nodo principal para el árbol. Ahora para cada conjunto se selecciona nuevamente un punto de ventaja, los cuales se vuelven

hijos del nodo principal, y se trazan circunferencias para particionar nuevamente el espacio, solo que estás tienen un radio menor. Se repite el procedimiento de manera recursiva para cada subconjunto que se particiona y metiendo el punto de ventaja dentro del árbol.

Siguiendo el procedimiento anterior obtenemos una segmentación del espacio tal como se muestra en la Figura 2.11 y el árbol al que se llega se muestra en la Figura 2.12.

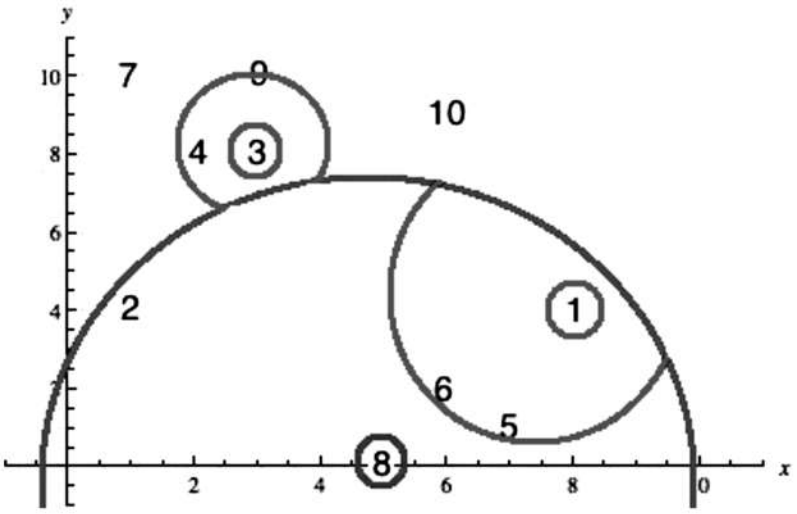


Figura 2.11: Espacio particionado siguiendo la construcción de un Vp tree.

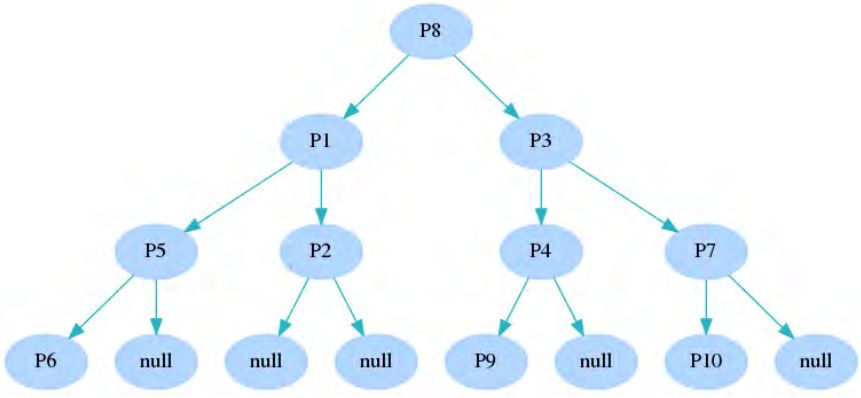


Figura 2.12: Vp tree generado para los puntos de la Tabla 2.2.

Noté como tanto el árbol k-d como el Vp son similares pero no iguales, esto ocurrió así ya que elegimos como primer punto de ventaja en el Vp el mismo que es el nodo principal en el árbol k-d.

2.7. Conclusiones

Este capítulo lo enfocamos a analizar las herramientas que nos serán de ayuda para resolver el problema planteado. Primeramente analizamos la DFT y la DCT por la importancia que tienen en el procesamiento digital de señales. Prestando gran atención a las propiedades de dichas transformadas, las cuales son claves para resolver el problema.

Después nos enfocamos en lo que son las imágenes digitales y los diferentes formatos que existen, pero de manera especial nos centramos en JPEG por ser el formato más utilizado para almacenar imágenes digitales de tono continuo.

Por último tratamos el problema de vecinos cercanos y explicamos dos estructuras de datos que vamos a utilizar para resolver de forma eficiente este problema, las cuales son los k-d trees y los Vp trees. Aún no podemos concluir cual de las dos estructuras no va funcionar mejor, aunque en la literatura se reportan mejores resultados en general para los Vp trees aplicados a diferentes problemas en lo que se necesita búsqueda de vecinos cercanos. En el capítulo de resultados haremos pruebas que nos permitan determinar cual de las dos estructuras es mejor para resolver nuestro problema.

Capítulo 3

Detección de Regiones Clonadas

En el presente capítulo se describirán cada una de las partes de las que consta el algoritmo de detección de regiones clonadas, que llamamos $DeReC$. De forma general, el algoritmo propuesto consta de las siguientes etapas:

- Generación de vectores de características.
- Ordenamientos de vectores de características.
- Búsqueda de píxeles duplicados y agrupamiento.
- Generación de imagen binaria.
- Relleno de huecos.

Cada una de estas etapas será descrita en las secciones siguientes.

3.1. Definición de la Notación

Para poder abordar estas etapas primeramente vamos a definir la notación que utilizaremos para una imagen digital. Definimos una imagen de entrada con posibles regiones clonadas I de tamaño $N = N_r \times N_c$, donde N_r es el número de renglones y

N_c es el número de columnas y dicha imagen está compuesta por tres canales: Rojo, Verde y Azul (R, G y B). Un píxel de la imagen I es un vector de la forma:

$$I(r,c) = [R(r,c), G(r,c), B(r,c)]^T$$

donde r y c representan una coordenada en la forma (renglón, columna).

Si numeramos los píxeles de la imagen usando un indice $I(k)$ (un píxel de I en la $k-$ésima posición), empezando por el píxel superior izquierdo y seguimos la numeración avanzando a la derecha por columnas. Continuamos así para todos los renglones de la imagen, con lo que obtenemos $k \in \{0, 1, 2, ..., N-1\}$. Proponemos la siguiente notación:

$$\begin{aligned} I(r,c) &\equiv I(k) \\ I(k) &= [R(k), G(k), B(k)]^T \end{aligned}$$

donde la relación que existe entre r, c y k está dada por:

$$\begin{aligned} k &= N_c\, r + c \\ r &= \lfloor \tfrac{k}{N_c} \rfloor \\ c &= k\,\%N_c \end{aligned}$$

donde $\%$ es el residuo de la división entera.

3.2. Generación de Vectores de Características

Para poder realizar la detección de las regiones clonadas comenzaremos por definir un vector de características C_k dado por (3.1), correspondiente al k-ésimo píxel de la imagen, como una composición de dos vectores v_k, u_k y un escalar g_k. El vector v_k almacena información de frecuencia sobre un vecindario de la imagen con origen en la k-ésima posición. El vector $u_k = [r, c]$ contiene la información de las coordenadas correspondientes al k-ésimo píxel. Finalmente g_k es la etiqueta que identificará

al grupo al que pertenece el vector de características, inicialmente $g_k = 0 \ \forall \ k \in \{0, 1, 2, \cdots, N-1\}$. Conforme se realiza la detección y clasificación de las regiones clonadas g_k puede ir tomando valores enteros positivos. Como se realiza la agrupación será explicado en la sección 3.4.

$$C_k = [v_k, u_k, g_k] \tag{3.1}$$

Para determinar el vector v_k, es necesario hacer una descomposición de la imagen en ventanas traslapadas de tamaño 8×8 píxeles. Para esto en cada coordenada k de la imagen se construirán tres ventanas $w_{R,k}$, $w_{G,k}$ y $w_{B,k}$ (3.2) con la información de color de los píxeles vecinos sobre la imagen origen I.

$$\begin{gathered} w_{R,k} = \{R(r+n, c+m)\} \\ w_{G,k} = \{G(r+n, c+m)\} \\ w_{B,k} = \{B(r+n, c+m)\} \\ \forall \ n \in [0, 1, \cdots, 7], m \in [0, 1, \cdots, 7] \end{gathered} \tag{3.2}$$

Dadas las propiedades de compactación de la energía y de una señal invertida de la DCT(vistas en la sección 2.2), proponemos que el vector v_k (3.3) sea formado por los valores absolutos de los cuatro componentes de frecuencia más baja de la DCT (en dos dimensiones) para cada una de las ventanas $W_{R,k}$ $W_{G,k}$ y $W_{B,k}$ dadas por (3.4).

$$v_k = \begin{bmatrix} W_{R,k}(0,0), W_{R,k}(0,1), W_{R,k}(1,0), W_{R,k}(1,1), \\ W_{G,k}(0,0), W_{G,k}(0,1), W_{G,k}(1,0), W_{G,k}(1,1), \\ W_{B,k}(0,0), W_{B,k}(0,1), W_{B,k}(1,0), W_{B,k}(1,1) \end{bmatrix} \tag{3.3}$$

$$\begin{aligned} W_{R,k} &= |DCT(w_{R,k})| \\ W_{G,k} &= |DCT(w_{G,k})| \\ W_{B,k} &= |DCT(w_{B,k})| \end{aligned} \tag{3.4}$$

Donde v_k es un vector de dimensión 12, nótese como pasamos de tener tres ventanas de tamaño 64 a solo 12 valores, los cuales en la mayoría de los casos contienen

la información más relevante de cada ventana. En la Figura 3.1 se puede apreciar como son construidas las ventanas (3.2) a partir de la imagen I para un píxel k, cada ventana pertenece a un canal de color y todas inician y terminan en las mismas coordenadas r y c.

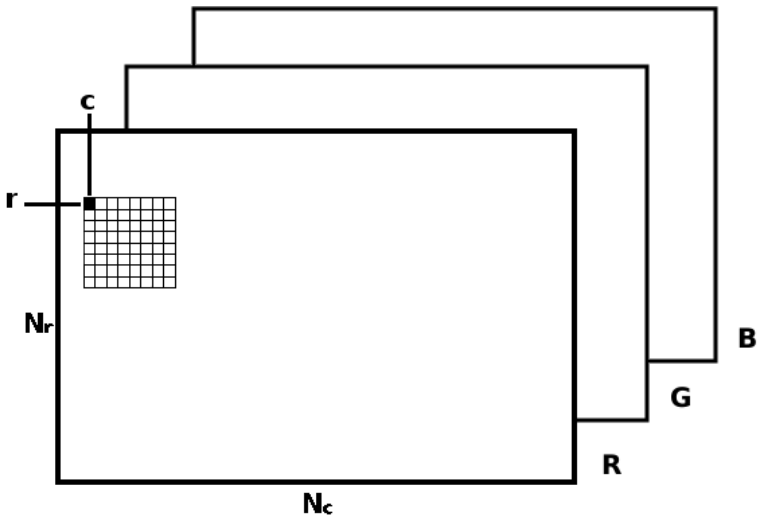


Figura 3.1: Representación de como se crean las ventanas (3.2) para un píxel k.

3.3. Ordenamiento de Vectores de Características

Para determinar regiones similares o clonadas, es necesario para cada uno de los vectores característicos v_k encontrar sus vecinos más cercanos de manera eficiente, los cuales son candidatos a ser píxeles duplicados. Un proceso de comparación de todos los $v_i \; \forall \; i \in \{0, 1, 2, \cdots, N-1\}$ contra todos los $v_j \; \forall \; j \in \{0, 1, 2, \cdots, N-1\}$ implica un algoritmo de complejidad $O(N^2)$, por lo que utilizaremos algoritmos de búsqueda de vecinos cercanos del orden $O(NlogN)$, basados en arboles binarios.

Los algoritmos que proponemos para realizar la búsqueda de vecinos cercanos están basados en dos estructuras de datos: Vantage-point Trees (Vp-Trees) y k-dimensional Trees (kd-Trees) (mencionadas en la sección 2.6). Estos algoritmos nos

permiten segmentar un espacio de búsqueda de tal manera que los vectores cercanos, según una medida de distancia, se agrupen cerca. De esta manera, al realizar búsquedas de vecinos cercanos no es necesario comparar contra todos los vectores. Ambos algoritmos pertenecen a métodos de búsqueda aproximada, esto es que los vecinos que se consideran cercanos no necesariamente son los más cercanos, ya que la partición del espacio puede provocar que vecinos realmente cercanos se alejen. A pesar de esto, los resultados que se obtienen con estos algoritmos son muy buenos y sobre todo eficientes.

Los Algoritmos 1 y 2, son dos algoritmos recursivos que realizan un ordenamiento del conjunto C (C es el conjunto de todos los vectores de características C_k). Ambos algoritmos necesitan un parámetro NSS que controla el tamaño de cada partición en el espacio de búsqueda. Entre menor sea el valor de NSS se crearán más particiones y estas serán más pequeñas. Una vez que está ordenado C encontrar los vecinos cercanos de v_k, que llamaremos $N_v(k)$, es simple, solo basta con buscar dentro de la partición en que se encuentra v_k y como la partición es de tamaño NSS podemos calcular $N_v(k) = \{v_{k+1}, v_{k+2}, \cdots, v_{k+NSS}\}$.

En el caso del Algoritmo 1, C se ordena por los valores de la p-ésima componente del vector v_k haciendo $d_k = v_k[p]$, tal como se particiona cuando se construye un árbol k-d. Mientras que para el Algoritmo 2, se utiliza una medida de distancia Coseno (3.5) entre un vector pivote v_p calculado aleatoriamente con cada uno de los elementos v_k.

$$D_{cos}(x, y) = 1 - \frac{\sum_{i=0}^{M-1} x_i * y_i}{\|x\|\|y\|} \tag{3.5}$$

donde M representa la dimensionalidad del los vectores.

El Algoritmo 1 recibe como parámetros el conjunto de vectores característicos a ordenar (C), la cantidad de vectores característicos que hay en C (N), el tamaño del que se desea cada partición (NSS) y la coordenada por la que se debe de empezar a particionar (h). Lo primero que realiza el algoritmo es determinar sobre que coordenada sigue el particionado (p), después generar un vector d el cual contienen

los valores de la coordenada p de todos los vectores. Una vez que tiene d lo utiliza para ordenar C utilizando *quickSort*. Ya que C fue ordenado se crean dos vectores C_1 y C_2 los cuales contienen la mitad de C cada uno. Se aplica de forma recursiva el ordenamiento ahora para C_1 y C_2. El algoritmo regresa la unión de C_1 y C_2 que es equivalente a construir un k-d tree sobre un vector para los elementos de C.

Algoritmo 1 $kd(C, N, NSS, h)$

1: **si** $N \geq NSS$ **entonces**
2: $p = h\,\%N$
3: $d = [\,]$
4: **para** $k \in \{0, 1, 2, \cdots, N-1\}$ **hacer**
5: $d[k] = v_k[p]$
6: **fin para**
7: $C \leftarrow quickSort(C, d)$
8: $C_1 \leftarrow C[0; N/2 - 1]$
9: $C_2 \leftarrow C[N/2; N - 1]$
10: $C_1 \leftarrow kd(C_1, N/2, NSS, h + 1)$
11: $C_2 \leftarrow kd(C_2, N/2, NSS, h + 1)$
12: **fin si**
13: **devolver** $C_1 \cup C_2$

El Algoritmo 2 recibe como parámetros el conjunto de vectores característicos a ordenar (C), la cantidad de vectores característicos que hay en C (N) y el tamaño del que se desea cada partición (NSS). Lo primero que realiza el algoritmo es determinar sobre que punto se hará el particionado, para esto genera un número aleatorio p entre 0 y $N-1$. Después generar un vector d el cual contienen los valores de distancia entre v_p y todos los demás vectores. Una vez que tiene d lo utiliza para ordenar C utilizando *quickSort*. Ya que C fue ordenado se crean dos vectores C_1 y C_2 los cuales contienen la mitad de C cada uno. Se aplica de forma recursiva el ordenamiento ahora para C_1

y C_2. El algoritmo regresa la unión de C_1 y C_2 que es equivalente a construir un Vp tree sobre un vector para los elementos de C.

Algoritmo 2 $vp(C, N, NSS)$

si $N \geq NSS$ **entonces**
  $p = rand(0, N-1)$
  $d = [\ ]$
  para $k \in \{0, 1, 2, \cdots, N-1\}$ **hacer**
    $d[k] = D_{cos}(v_k, v_p)$
  fin para
  $C \leftarrow quickSort(C, d)$
  $C_1 \leftarrow C[0; N/2-1]$
  $C_2 \leftarrow C[N/2; N-1]$
  $C_1 \leftarrow vp(C_1, N/2, NSS)$
  $C_2 \leftarrow vp(C_2, N/2, NSS)$
fin si
devolver $C_1 \cup C_2$

3.4. Búsqueda de Píxeles Duplicados y Agrupamiento

Vamos a considerar que dos píxeles pueden estar duplicados si sus vectores de características tienen una distancia entre ellos menor o igual a ϵ, donde ϵ es un parámetro que nos permite ser más estrictos o menos estrictos en cuanto a la decisión. La regla de decisión se muestra en (3.6).

$$posibleDuplicado(v_k, v_l) = \begin{cases} si & D_{cos}(v_k, v_l) \leq \epsilon \\ no & D_{cos}(v_k, v_l) > \epsilon \end{cases} \tag{3.6}$$

Realizar la búsqueda del vecino más cercano del vector v_k implica buscar sobre $N_v(k)$, esto es realizar NSS comparaciones tal y como se muestra en (3.7).

$$NN(v_k) = v_m \mid D_{cos}(v_m, v_k) < D_{cos}(v_n, v_k), \\ \forall\ v_m\ \wedge\ \forall\ v_n \in N_v(k) \tag{3.7}$$

Una vez que encontramos al vecino más cercano de v_k que llamaremos v_l verificamos si cumplen la regla de decisión (3.6). En caso de cumplirla buscaremos un grupo al cual pertenecen dichos vectores y marcamos g_k y g_l al grupo encontrado. Para realizar la búsqueda del grupo es necesario tener el conjunto de vecinos cercanos geométricos (3.8) alrededor de la posición del píxel a una distancia menor o igual a τ, considerando solamente las coordenadas geométricas u_k.

$$N_u(k) = \{g_j \mid D_E(u_k, u_j) < \tau\ \ \forall\ j \in \{0, 1, 2, \cdots, N-1\} \text{ y } g_j \neq 0\} \tag{3.8}$$

Donde D_E es la distancia Euclidiana entre dos vectores y se calcula como se muestra en (3.9).

$$D_E(x, y) = \sqrt{\sum_{i=0}^{M-1} (x_i - y_i)^2} \tag{3.9}$$

Como tenemos un par de vectores v_k y v_l realizamos el cálculo de $N_u(k)$ y $N_u(l)$ con (3.8) los cuales serán un conjunto de etiquetas numéricas con valores de grupos (enteros positivos), siempre y cuando dicho valor sea diferente de 0 (la etiqueta de grupo 0 se utiliza para decir que aún no ha sido clasificado el píxel). Debido a la regla de decisión (3.6) nótese que hay píxeles para los cuales siempre su etiqueta será 0. Para decidir a que grupo pertenecen los vectores k y l se realiza la intersección de los

conjuntos $N_u(k)$ y $N_u(l)$. Se consideran dos casos: uno cuando la intersección es vacía y otro cuando no lo es, como se muestra en (3.10).

$$g_k = g_l = \begin{cases} \text{Si } N_u(k) \cap N_u(l) = \varnothing & \text{Se crea un grupo nuevo} \\ \text{Si no} & \text{mín}(N_u(k) \cap N_u(l)) \end{cases} \tag{3.10}$$

donde g_k y g_l siempre son valores enteros positivos. La creación de un grupo nuevo se hace llevando un contador entero positivo, el cual se incrementa en 1 y el valor resultante se toma como el nuevo grupo.

Para explicar de mejor manera la creación y asignación de grupos supongamos dos vectores v_k y v_l que ya se sabe que son vecinos cercanos. El primer caso que tenemos es cuando todavía no ha sido etiquetado ningún vector, es decir g_k en todos los vectores es 0. Con ayuda de la Figura 3.2 podemos ver como v_k y v_l no tienen vecinos geométricos etiquetados, es decir, $N_u(k) = \varnothing$ y $N_u(l) = \varnothing$, por lo que al calcular la intersección de ambos conjuntos el resultado es $\varnothing$ y en base a (3.10) debemos crear un grupo nuevo. Como para este primer caso partimos del supuesto que aún no existen grupos hacemos que $g_k = 1$ y $g_l = 1$, ahora la siguiente vez que sea necesario crear un grupo podremos utilizar el valor 2 y así de manera sucesiva.

Si ya existen grupos, véase la Figura 3.3 y 3.4, se consideran dos casos en los que $N_u(k)$ y $N_u(l)$ ya no son conjuntos vacíos. Por lo tanto al realizar la intersección si obtenemos un valor y no es necesario crear un grupo nuevo. En el caso de la Figura 3.3 la intersección nos regresa un único valor $N_u(k) \cap N_u(l) = \{2\}$ por lo que $g_k = 2$ y $g_l = 2$. Mientras tanto en la Figura 3.4 vemos como la intersección nos regresa un conjunto con más de un valor $N_u(k) \cap N_u(l) = \{1, 2\}$, por lo que debemos tomar el valor mínimo de la intersección y quedando $g_k = 1$ y $g_l = 1$.

Todo este procedimiento se realiza con la intención de poder encontrar áreas clonadas y no solo píxeles de forma aislada, el objetivo de quedarnos con el mínimo de la intersección es generar la menor cantidad de grupos posible. La creación de grupos es automática así como se plantea y para cada imagen puede resultar una canti-

dad diferente de grupos totales. Esta es una de las grades fortalezas del algoritmo implementado.

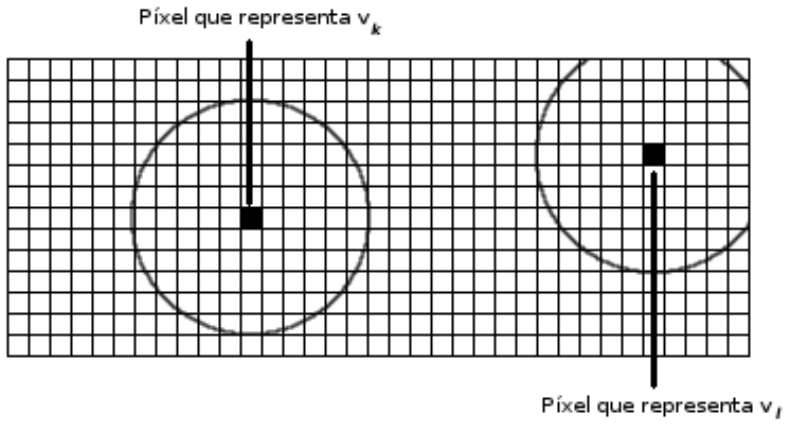


Figura 3.2: $N_u(k) = \varnothing, N_u(l) = \varnothing, N_u(k) \cap N_u(l) = \varnothing$, por lo que se debe crear un nuevo grupo y ambos vectores marcarlos a ese grupo.

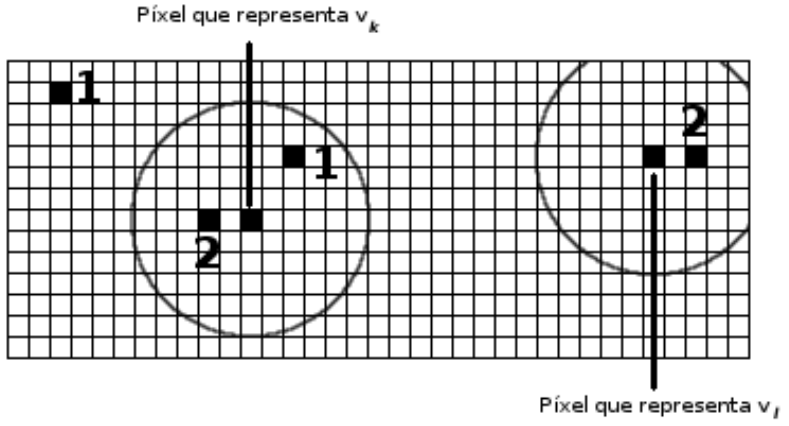


Figura 3.3: $N_u(k) = \{1, 2\}, N_u(l) = \{2\}, N_u(k) \cap N_u(l) = \{2\}$, por lo que ambos se marcan pertenecientes al grupo 2.

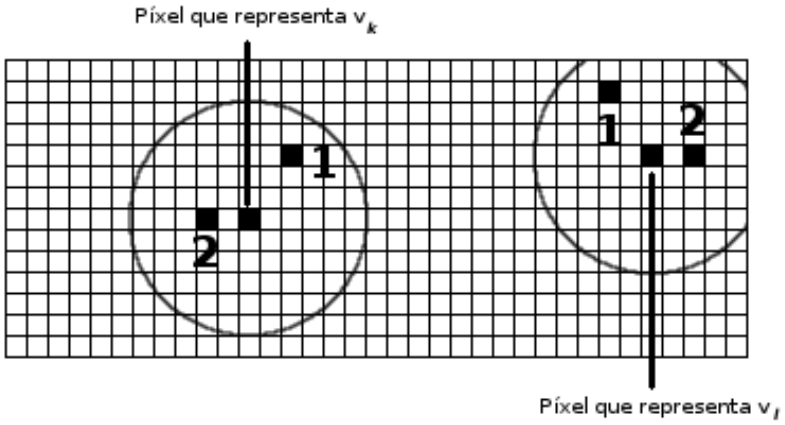


Figura 3.4: $N_u(k) = \{1, 2\}, N_u(l) = \{1, 2\}, N_u(k) \cap N_u(l) = \{1, 2\}$, por lo que ambos se marcan pertenecientes al grupo 1.

3.5. Generación de Imagen Binaria

Nuestro algoritmo regresa como resultado del procesamiento una imagen binaria (solo regiones blancas y negras) que llamaremos I_s, la cual tiene el mismo tamaño de I. Al iniciar el procedimiento I_s será una imagen en negro total. Una vez que ya se tienen todos los vectores marcados como pertenecientes a un grupo, para agregar las regiones clonadas en la imagen binaria resultante I_s se hace un conteo de cuantos vectores C_k pertenecen al mismo grupo. Para todos los grupos que tengan al menos 128 elementos y una marca de grupo diferente de 0 tomaremos sus coordenadas u_k, en cuya posición marcaremos de color blanco la imagen I_s. Con lo cual obtenemos una imagen donde en blanco se muestran regiones clonadas y en negro regiones normales.

Decidimos tomar el valor de al menos 128 elementos por grupo ya que cada elemento representa un píxel y nuestras ventanas son precisamente de 64 píxeles, dado que siempre se marcan en pares los píxeles clonados, cuando detectamos una ventana clonada tendremos precisamente 128 píxeles. Áreas menores a ese tamaño que están clonadas en la imagen no son relevantes ya que son muy pequeñas y difíciles de

detectar por el tamaño de ventana utilizado.

3.6. Relleno de Huecos

Ahora que ya se cuenta con una imagen binaria que representa las regiones clonadas, I_s. Es necesario hacer una operación morfológica para rellenar los posibles huecos que quedan en la imagen. Esto se debe a que las búsquedas de vecinos cercanos se hicieron de manera aproximada y por lo tanto no siempre se encuentran todos los píxeles duplicados. Por otra parte, cuando la imagen a analizar tienen compresión JPEG también es difícil encontrar exactamente todos los píxeles duplicados o clonados.

La operación morfológica común para resolver esto es una erosión ($I_s \ominus E$) seguida de una dilatación ($I_s \oplus E$), que se conoce como apertura ($I_s \circ E = (I_s \ominus E) \oplus E$), donde E se conoce como elemento estructurante, que sirve para controlar la forma en que se realiza la dilatación. En nuestro algoritmo utilizamos como elemento estructurante el mostrado en (3.11). No vamos a adentrarnos en el tema de las operaciones morfológicas sobre imágenes, para cualquier consulta revisar el libro de [Serra, 1983].

$$E = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{3.11}$$

Realizar esta operación de apertura implica tiempo de cómputo adicional, pero nos permite quitar puntos negros en zonas blancas y puntos blancos en zonas negras,

obteniendo una mayor precisión en la detección.

3.7. Algoritmo $DeReC$

En base a todas las etapas que han sido explicadas presentamos un resumen de lo que hace el algoritmo $DeReC$. Como es posible calcular los conjuntos de vecinos $N_v(k)$ y $N_u(k)$ al rededor de un posición k de la imagen origen I, podemos hacer un Algoritmo de Detección de Regiones Clonadas ($DeReC$) basado en la intersección de los vecindarios.

Primero vamos a definir los algoritmos parciales para algunas de la etapas que necesita $DeReC$. En el Algoritmo 3 se encuentran las etapas de búsqueda de vecinos cercanos y agrupamiento. En el Algoritmo 4 se describe la etapa de generación de la imagen binaria.

Algoritmo 3 Etapas de búsqueda de vecinos cercanos y agrupamiento.

para $k = 0$ hasta $N - 1$ **hacer**
  $v_l = NN(v_k)$ con (3.7)
  si $D_{cos}(v_k, vl) < \epsilon$ **entonces**
    Calcula $N_u(k)$ y $N_u(l)$ con (3.8)
    si $N_u(k) \cap N_u(l) = \varnothing$ **entonces**
      $T_g \leftarrow T_g + 1$
      $g_k = T_g$, $g_l = T_g$
    si no
      $j \leftarrow \text{ínf}(N_u(k) \cap N_u(l))$
      $g_k = j, g_l = j$
    fin si
  fin si
fin para

Algoritmo 4 Etapa de generación de imagen binaria

1: **para** $m = 1$ hasta T_g **hacer**
2: $S[m] = \sum_{j=0}^{N-1} \delta(m - g_j)$
3: **fin para**
4: **para** $k = 0$ hasta $N - 1$ **hacer**
5: **si** $g_k \neq 0$ y $S[g_k] > 128$ **entonces**
6: $I_s(u_k) = 1$
7: **si no**
8: $I_s(u_k) = 0$
9: **fin si**
10: **fin para**

El Algoritmo 5 ($DeReC$), recibe como parámetros una imagen de entrada con posibles regiones clonadas I, un umbral ϵ, el tamaño de las particiones NSS y algoritmo de ordenamiento a utilizar alg.

El Algoritmo $DeReC$, para cada posición k calculará el vecino mas cercano v_l a v_k en el conjunto $N_v(k)$ y tomando en cuenta los conjuntos $N_u(k)$ y $N_u(l)$ realiza la intersección para marcar g_k y g_l con un valor de grupo. Si la intersección es vacía $N_u(k) \cap N_u(l) = \varnothing$ se marcaran los grupos $g_k = g_l = T_g$ con un nuevo valor T_g. De lo contrario se marcan con el valor mínimo de la intersección, para garantizar que se creen el menor número de grupos. Finalmente para generar la imagen binaria que muestra las posibles regiones clonadas I_s, marcaremos la imagen I_s en la posición u_k con 1 si $g_k \neq 0$ y al menos existen 128 elementos en el grupo, y 0 en caso contrario. A nuestra imagen resultante I_s se le aplica una apertura para eliminar huecos.

Para el Algoritmo $DeRec$ la función δ se define como $\delta(x) = 1$ si $x = 0$ y 0 en caso contrario y la función mín(A) es el valor mínimo del conjunto A.

Algoritmo 5 Algoritmo $DeReC(I, \epsilon, NSS, alg)$

$T_g = 0$
$N_r = Renglones(I)$
$N_c = Columnas(I)$
$N = N_r \times N_c$
$C = [\ ]$
para $k = 0$ hasta $N - 1$ **hacer**
    Calcular v_k con (3.4)
    $u_k = [\lfloor \frac{k}{N_c} \rfloor, k\,\%N_c)]$
    $g_k = 0$
    $C[k] = [v_k, u_k, g_k]$
fin para
si alg==”vp” **entonces**
    $C = vp(C, N, NSS)$ Algoritmo 2
si no, **si** alg==”kd” **entonces**
    $C = kd(C, N, NSS, 0)$ Algoritmo 1
fin si
Ejecutar etapas del Algoritmo 3
Ejecutar etapas del Algoritmo 4
Fijar el valor de E según (3.11)
$I_s = I_s \circ E$
devolver I_s

3.8. Conclusiones

Hemos propuesto un algoritmo, al que llamamos $DeReC$, buscando principalmente que sea capaz de detectar regiones clonadas en imágenes con compresión JPEG y que el tiempo de ejecución necesario sea aceptable. Para poder detectar las regiones

clonadas en imágenes con compresión JPEG hicimos uso de la DCT, mientras que para lograr un tiempo de ejecución aceptable nos enfocamos en tres puntos clave: primeramente generar vectores de características de tan solo dimensión 12, realizar la búsqueda del vecino más cercano de forma eficiente con ayuda de algoritmos de ordenamiento (*vpSorting* y *kdSorting*) y por último detectar falsos positivos con forme se iban clasificando los píxeles, para evitar un post-procesamiento complejo. Como resultado tenemos un algoritmo simple y eficiente.

Capítulo 4

Resultados

4.1. Introducción

Para analizar el desempeño de *DeReC* vamos a realizar diferentes pruebas, para esto utilizaremos una base de datos que consta de 12 imágenes en formato PNG (compresión sin perdida) y 12 imágenes en formato JPEG (compresión con perdida, Calidad = 95 %), las imágenes formato JPEG fueron creadas a partir de las imágenes PNG que contienen las regiones clonadas. En la Tabla 4.1 se muestran las características de las imágenes utilizadas.

Haremos pruebas de efectividad de detección, esto será posible gracias a que en nuestra base de datos contamos con las máscaras binarias que se utilizaron para generar la imagen alterada; idealmente el Algoritmo *DeReC* debería dar como respuesta dichas máscaras binarias. Para medir la calidad de la respuesta utilizaremos una métrica de error conocida como *F-Measure*, la cual se explicará detalladamente más adelante.

También haremos pruebas de eficiencia midiendo el tiempo de cómputo que le toma al algoritmo llegar al resultado. Todas las pruebas fueron realizadas en un equipo de cómputo que cuenta con un procesador Intel® Core™ i5-5200U @ 2.20 GHz. y 12GB de memoria RAM.

Tabla 4.1: Propiedades de las imágenes con las que consta nuestra base de datos.

Imagen ID	N_r	N_c	N
1	390	694	270660
2	390	694	270660
3	390	694	270660
4	233	416	96928
5	233	416	96928
6	233	416	96928
7	324	478	154872
8	292	520	151840
9	292	520	151840
10	584	1040	607360
11	584	1040	607360
12	584	1040	607360

4.1.1. Métricas de Error

Para medir el desempeño del algoritmo en cuanto a efectividad de detección, haremos un análisis estadístico conocido como F_1score o *F-measure*, para el cual necesitamos el conteo de Verdaderos Positivos (TP), Verdaderos Negativos (TN), Falsos Positivos (FP) y Falsos Negativos (FN). Con estos valores calculamos: Tasa de Verdaderos Positivos (TPR), esta tasa se calcula como se ve en (4.1), Tasa de Verdaderos Negativos (TNR) como se muestra en (4.2), Tasa de Falsos Positivos (FPR) según (4.3) y Tasa de Falsos Negativos(FNR) como vemos en (4.4). También mediremos la exactitud (*Accuracy*) y la precisión (*Precision*) utilizando (4.5) y (4.6) respectivamente. Finalmente calculamos la medida *F-measure* (4.7), la cuál nos da información importante y resumida respecto a la precisión y sensibilidad del sistema clasificador. Dicha medida idealmente debe de ser igual a 1 lo que significaría que

el sistema logra clasificar sin errores. Para ver más información sobre el uso de esta métrica de error recomendamos revisar el trabajo de [Powers, 2011].

$$TPR = \frac{TP}{Total\ de\ Positivos\ Reales} \tag{4.1}$$

$$TNR = \frac{TN}{Total\ de\ Negativos\ Reales} \tag{4.2}$$

$$FPR = \frac{FP}{Total\ de\ Negativos\ Reales} \tag{4.3}$$

$$FNR = \frac{FN}{Total\ de\ Positivos\ Reales} \tag{4.4}$$

$$ACC = \frac{TPR + TNR}{2} \tag{4.5}$$

$$Precision = \frac{TPR}{TPR + FNR} \tag{4.6}$$

$$F\text{-}Measure = 2 \times \frac{Precision \times TPR}{Precision + TPR} \tag{4.7}$$

El *Total de Positivos Reales* es la cantidad de píxeles en la mascara original (Ground truth) que tienen un valor de 1 (color blanco) y el *Total de Negativos Reales* son los píxeles que tienen un valor de 0 (color negro).

4.2. Pruebas Generales del Algoritmo DeReC

En esta sección vamos a realizar pruebas generales del algoritmo implementado, para encontrar los valores adecuados de los parámetros de $DeReC$, nos vamos enfocar en definir lo siguiente:

- **Algoritmo de Ordenamiento**. Recordemos que hemos propuesto dos algoritmos para realizar el ordenamiento de nuestros vectores de características: kd y vp (en la sección 3.3). Ambos nos permiten segmentar el espacio de búsqueda y realizar búsqueda de vecinos cercanos de forma eficiente.

- **NSS**. Este parámetro es el encargado de controlar la cantidad de veces que es particionado el espacio de búsqueda para los algoritmos kd y vp.

- ϵ. Este parámetro $DeReC$ controla el umbral que determina si dos vectores de características los podemos considerar similares. Valores muy cercanos a cero hacen que nuestro algoritmo sea más estricto y valores cercanos a uno hacen que nuestro algoritmo sea completamente permisivo, dicho rango ($[0, 1]$) se debe al uso de la Distancia Coseno para medir distancia entre dos vectores. En ambos extremos los resultados obtenidos no sirven para detectar las regiones clonadas, ya que en un caso (valores casi de cero) la imagen de respuesta podría ser completamente negra (no detecta regiones clonadas) y en el otro caso (valores cercanos o iguales a uno) la imagen de respuesta podría ser color blanco (detecta que todas las regiones son similares entre si).

Los parámetros NSS y ϵ así como el algoritmo de ordenamiento determinan de forma directa el comportamiento de $DeReC$. Tanto en eficiencia de cómputo como en la eficacia de la detección, por lo que es muy importante poderlos definir de forma óptima. Midiendo el tiempo de cómputo y mediante las métricas de error, que nos permiten medir la calidad de la respuesta, podremos saber que valores son más adecuados para cada parámetro.

4.2.1. Comparación entre kd y vp

Para las imágenes con compresión sin pérdida (PNG), de nuestra base de datos, vamos a realizar pruebas de calidad entre los resultados obtenidos utilizando kd y vp. Estas pruebas nos dirán cual de los dos algoritmos segmenta mejor el espacio de búsqueda para nuestros vectores de características. Obtendremos el valor de $F\text{-}Measure$ para cada imagen utilizando cada uno de los algoritmos de ordenamiento, mientras todos los demás parámetros se mantienen constantes, $\epsilon = 1 \times 10^{-6}$ y $NSS = 30$.

En la tabla 4.2 vemos los resultados obtenidos para la medida de error $F\text{-}Measure$ en cada tipo de ordenamiento así como el tiempo en segundos necesario para llegar a la solución. Si observamos la tabla podemos notar que en la mayoría de los casos obtiene mejores resultados el algoritmo vp. Ahora calculamos la media de $F\text{-}Measure$ para cada algoritmo y la desviación estándar obteniendo para kd: $\mu_F = 0.8925$, $\sigma_F = 0.06225$ y para vp: $\mu_F = 0.9225$, $\sigma_F = 0.03671$. Nótese como la media es más alta para vp, lo que nos dice que obtiene mejores resultados y la desviación estándar es menor lo que nos dice que los resultados varían poco al rededor de la media. Podemos concluir que los resultados que obtenemos con vp son de mayor calidad, por lo que desde ahora para todas las demás pruebas es el algoritmo que vamos a utilizar por defecto.

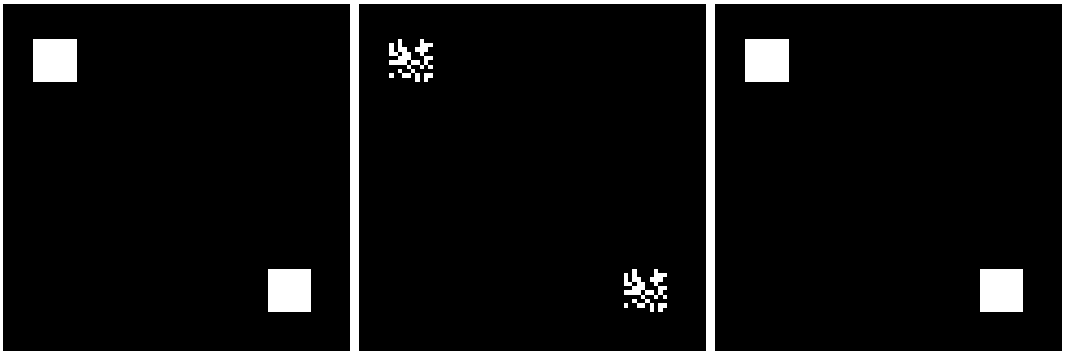
En cuanto al tiempo necesario de ejecución de forma general podemos decir que ambos son similares, lo cual era de esperarse ya que ambos algoritmos de ordenamiento se basan en el particionado del espacio de búsqueda. Las estructuras de datos en las que se basan son árboles binarios y la complejidad de cada uno es del mismo orden ($O(NlogN)$).

Tabla 4.2: Comparación de resultados obtenidos entre *kd* y *vp*.

	kd		**vp**	
Imagen	*F-Measure*	**Tiempo**	*F-Measure*	**Tiempo**
1	0.9	10.993	0.91	10.670
2	0.93	13.069	0.96	12.371
3	0.92	11.739	0.94	10.591
4	0.87	4.736	0.85	3.465
5	0.91	4.062	0.92	3.863
6	0.72	3.957	0.95	4.081
7	0.94	6.407	0.94	6.144
8	0.95	6.223	0.95	5.981
9	0.84	6.62	0.85	5.561
10	0.9	28.865	0.94	23.669
11	0.93	27.175	0.94	27.808
12	0.9	25.894	0.92	24.142

Otra observación importante es el hecho de que los resultados obtenidos para *F-Measure* utilizando *kd* no distan tanto de los obtenidos con *vp* gracias al relleno de huecos que se implementó. Ya que dicho relleno nos permite eliminar falsos negativos que aparecen en áreas cercanas a verdaderos positivos, por lo tanto al haber menos cantidad de falsos negativos el valor calculado para *F-Measure* es mayor. En pruebas sin relleno de huecos el algoritmo de ordenamiento *vp* tenía resultados mejores, esto se atribuye a que el particionado del espacio de búsqueda para nuestro problema en particular se realiza mejor con *vp*, ya que con *kd* no siempre encontramos los verdaderos vecinos cercanos, provocando que haya píxeles sin detectar y generando huecos en la imagen binaria resultante. En la Figura 4.1 se aprecia el resultado obtenido sin relleno de huecos. Véase como detecta mejor *vp*, aunque con *kd* si es posible encontrar

la zona clonada, hay gran cantidad de falsos negativos.



(a) Máscara Real (b) Máscara kdSorting (c) Máscara vpSorting

Figura 4.1: La imagen (a) es la máscara real, de forma ideal es la que debe ser encontrada. La imagen (b) muestra la máscara que se encontro utilizando *kd* sin relleno de huecos, nótese que hay varios huecos aunque el contorno general si fue encontrado. La imagen (c) muestra el resultado obtenido utilizando *vp*, a simple vista luce igual que la máscara real aunque le faltan algunos píxeles, pero son tan pocos que no se notan.

4.2.2. Determinación de Valores Adecuados para NSS

Como ya hemos definido que *vp* nos permite obtener mejores resultados ahora vamos a determinar el parámetro NSS para *vp* solamente. Determinar este parámetro de manera adecuada nos puede beneficiar en dos aspectos: Menor tiempo de cómputo y mejor calidad de respuesta. Lo que vamos a hacer para determinar esté parámetro será realizar el cálculo de las métricas de error para diferentes valores de NSS, utilizando la medida $F\text{-}Measure$. También realizaremos una comparativa del tiempo de respuesta para diferentes valores de NSS. Con los resultados que obtengamos de ambas pruebas podremos determinar valores óptimos de este parámetro, de tal manera que se maximice la calidad de la respuesta y minimice el tiempo.

Experimento Propuesto

Para poder determinar los valores adecuados de NSS vamos a realizar el análisis de las imágenes de nuestra base de datos que tienen tamaños similares, variando NSS en el rango $[2, 100]$. Para cada valor de NSS procesaremos las imágenes, mediremos el tiempo necesario para procesar cada una, evaluaremos los resultados con las métricas de error y los promediaremos. Los datos generados nos ayudan a identificar el comportamiento general que presenta el algoritmo para diferentes valores de NSS.

La tabla 4.3 muestra los resultados que se obtienen del experimento. Lo primero que podemos observar es como incrementa el tiempo de cómputo conforme incrementa el valor de NSS. Para los resultados de las métricas notamos que iban incrementando conforme mayor era NSS, pero llega un punto donde prácticamente no varían. El mejor resultado, en cuanto a calidad, se obtiene para $NSS = 10$ ya que en ese valor la mayoría de las métricas fueron las más altas.

Tabla 4.3: Tiempo de cómputo y métricas de error promediados, obtenidos para diferentes valores de NSS.

NSS	**Tiempo**	**TPR**	**TNR**	**ACC**	**Precision**	**F-Measuere**
2	1.753	0.86	1	0.93	0.99	0.92
3	1.888	0.87	1	0.93	0.99	0.93
5	2.298	0.87	0.99	0.93	0.99	0.93
7	2.355	0.87	0.99	0.93	0.99	0.93
10	2.590	0.88	0.99	0.94	0.99	0.93
15	3.163	0.87	0.99	0.93	0.99	0.92
20	3.569	0.88	0.99	0.93	0.98	0.93
50	6.149	0.88	0.99	0.93	0.98	0.93
100	10.419	0.88	0.99	0.93	0.98	0.93

En la Figura 4.2 se observa la curva ROC correspondiente a la Tabla 4.3, varios

puntos salen juntos por lo que pareciera haber menos resultados que los que hay en la tabla. Para construir la curva se hace uso de TPR y $FPR = 1 - TNR$, donde en el eje X se utiliza FPR mientras que en Y se usa TPR. La curva nos permite encontrar el valor adecuado para el parámetro NSS. Lo que se debe de hacer es tomar el valor de NSS que genera el punto en la curva que se encuentra en la parte más superior y más a la izquierda. En esté caso tenemos que los valores de $10, 20, 50$ y 100 son los que nos maximizan la calidad de la respuesta.

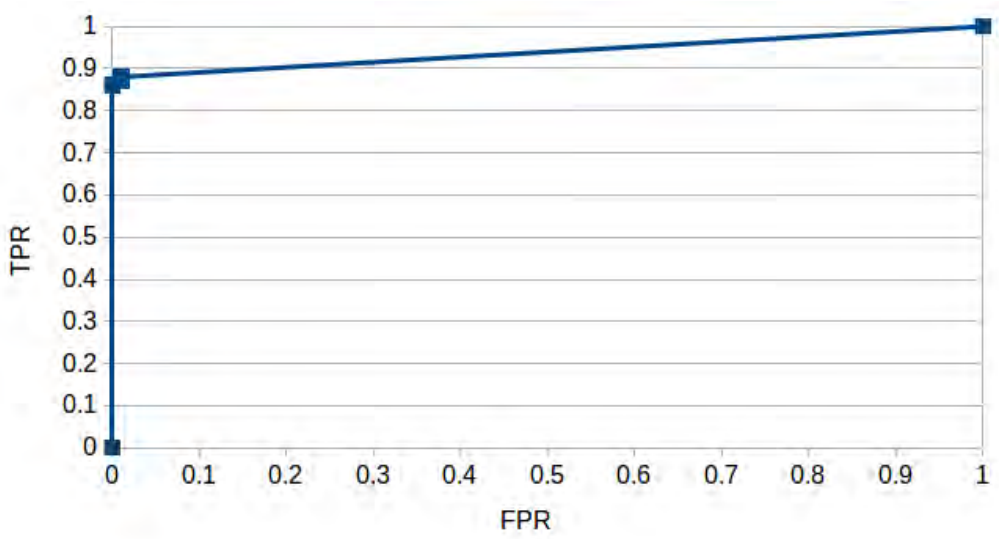


Figura 4.2: Curva ROC para el parámetro NSS.

De este experimento podemos deducir que valores pequeños de NSS ayudan a que el algoritmo sea más eficiente en tiempo de cómputo. Si analizamos como se realiza la búsqueda del vecino más cercano de un vector veremos que una vez que el conjunto de vectores fue ordenado con vp, encontrar el vecino más cercano implica realizar una búsqueda secuencial sobre los NSS vecinos, si el valor de NSS es más grande, esta búsqueda secuencial es más lenta, ya que compara contra más vectores. Analicemos un caso extremo en el cual tenemos N vectores y $NSS = N+1$, dado que la condición de ejecución de $vpSorting$ es $N \geq NSS$, $vpSorting$ no se ejecuta, y ahora buscar el vecino más cercano de cada vector implica buscar sobre los NSS vecinos,

por lo que tenemos un algoritmo de complejidad $O(N \times NSS)$ que es equivalente que $O(N^2 + N)$, que en términos de complejidad solamente se escribe $O(N^2)$, es decir, en el caso extremo nuestra búsqueda se vuelve exhaustiva.

Como dijimos al inicio del experimento deseamos encontrar un valor de NSS que nos ayude a maximizar la calidad de la respuesta y minimizar el tiempo de cómputo necesario, según esta premisa proponemos $NSS = 3$ como el valor óptimo para las imágenes con ID 4, 5 y 6. Ya que en este valor el tiempo es uno de los menores y la calidad es muy aceptable, casi idéntica a la obtenida con $NSS = 10$ que era uno de los valores óptimos obtenidos por la curva ROC.

Ahora vamos a realizar pruebas iguales a la anterior pero con imágenes en otro rango de tamaño, para determinar si el valor óptimo de NSS es dependiente del tamaño de la imagen. Para esto usando las imágenes en nuestra base de datos repetimos el experimento. En la Tabla 4.4 se muestran los valores NSS que consideramos óptimos para los diferentes tamaños de imágenes.

Tabla 4.4: Valores óptimos de NSS según rangos de tamaños de imagen.

Tamaño de Imagen	**NSS óptimo**
1 a 99,999	3
100,000 a 199,999	5
200,000 a 499,999	6
500,000 a 1,000,000	7

Para imágenes dentro del rango que se muestran en la Tabla 4.4 proponemos que se utilice el valor NSS de la tabla, ya que esto permitirá resultados eficientes y de calidad. Para imágenes más grandes fuera de estos rangos proponemos utilizar valores un poco mayores de NSS, también recomendamos no utilizar valores mayores a $NSS = 50$ ya que puede afectar demasiado en el tiempo de cómputo cuando se trabaje con imágenes de alta resolución. En general la calidad de la detección no se

ve afectada para valores mayores de NSS pero sí el tiempo de cómputo.

4.2.3. Determinación de Valores Adecuados para ϵ

Al realizar pruebas que nos permitieran encontrar valores adecuados de ϵ lo primero que notamos es que dichos valores varían dependiendo de si estamos analizando imágenes en formado PNG o en formato JPEG. También dichos valores pueden ser diferentes dependiendo de la imagen que se desee analizar, pero en general varían más por el formato que por las características de la imagen. Debido a esto vamos a recomendar un rango de posibles valores para este parámetro según el formato de la imagen.

Determinación de ϵ para Imágenes PNG

Procesamos las 12 imágenes en formato PNG con $DeReC$ para diversos valores de ϵ, calculamos la medida $F\text{-}Measure$ y promediaremos los resultados. La intensión será observar como varía el valor medio de $F\text{-}Measure$ conforme cambia el valor de ϵ.

En la Tabla 4.5 se observan los resultados obtenidos para diferentes valores de ϵ, estos valores fueron tomados en base a la experiencia de varios experimentos y pruebas que se le realizaron al algoritmo.

Tabla 4.5: Valores medios de F-Measure para diferentes valores de ϵ utilizando las imágenes PNG.

Valor de ϵ	**FPR**	**TPR**	**F-Measure Media**
1×10^{-40}	0.00	0.785	0.87
1×10^{-35}	0.00	0.790	0.88
1×10^{-30}	0.00	0.785	0.87
1×10^{-25}	0.00	0.790	0.88
1×10^{-20}	0.00	0.790	0.88
1×10^{-15}	0.00	0.795	0.88
1×10^{-10}	0.00	0.795	0.88
1×10^{-7}	0.01	0.800	0.88
1×10^{-6}	0.11	0.805	0.83
1×10^{-5}	0.25	0.820	0.80
1×10^{-4}	0.32	0.885	0.81

En la Figura 4.3 se observa la curva ROC correspondiente a la Tabla 4.5, podemos apreciar que el punto que se encuentra más a izquierda y más arriba es el correspondiente con el valor de $\epsilon = 1 \times 10^{-7}$, por lo que es el que maximiza la calidad de la repuesta.

Note como el rango posibles de valores es muy amplio, debido principalmente a dos razones. Las regiones clonadas en imágenes sin pérdida (como lo es PNG) suelen ser prácticamente iguales, es decir que se pueden utilizar valores de ϵ pequeños. La otra razón se debe a nuestra propuesta de agrupamiento, ya que para valores más grandes de ϵ deberían aparecer zonas que realmente no son clonadas (Falsos Positivos) pero esas regiones son desechadas por el agrupamiento realizado y como resultado la detección es correcta a pesar de que se utilicen valores grandes de ϵ.

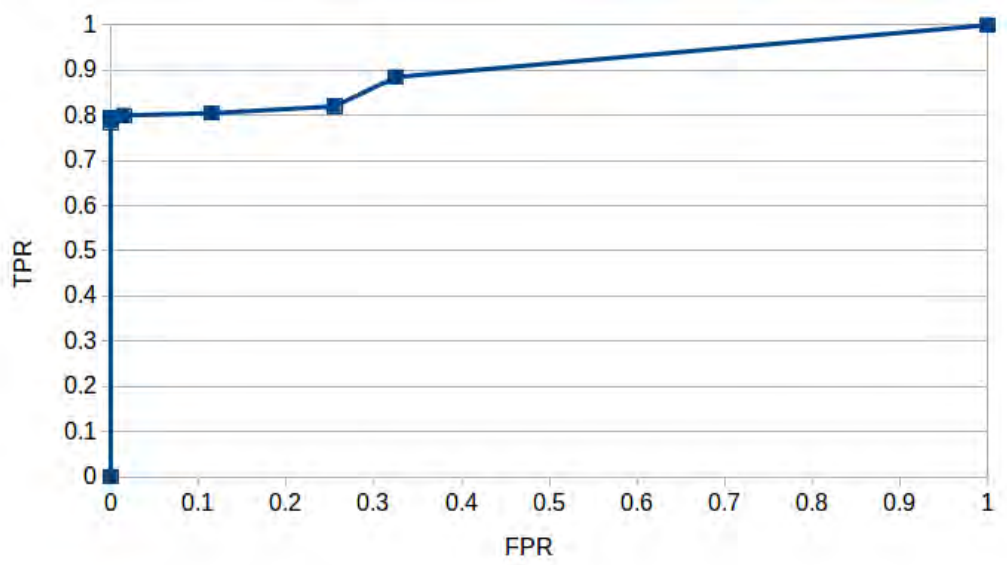


Figura 4.3: Curva ROC para el parámetro ϵ en imagenes PNG.

En base a nuestra experiencia recomendamos utilizar un valor de 1×10^{-7}, véase que el rendimiento es bueno para valores cercanos de 1×10^{-7}. Recomendamos tomar este valor ya que si la imagen tuviera algún tipo de emborronado en la región clonada o algún escalamiento y/o rotación utilizar valores más pequeños de ϵ podría provocar no detectar de forma adecuada las regiones clonadas. Con todas las pruebas realizadas concluimos que el rango de valores que puede tomar ϵ para imágenes PNG está entre 1×10^{-40} y 1×10^{-4}, es muy robusto el algoritmo para casi cualquier valor de este rango.

Determinación de ϵ para Imágenes JPEG

Ahora utilizando nuestras 12 imágenes en formato JPG realizamos el mismo procedimiento que se utilizó con las imágenes PNG, para diferentes valores de ϵ calcularemos el valor medio de *F-Measure*.

En la Tabla 4.6 se observan las medias obtenidas para cada valor de ϵ, nótese como para valores menores o iguales a 1×10^{-6} no es posible detectar regiones clonadas

en este formato, esto es debido al proceso de compresión que hace JPEG. El valor con la media más alta lo encontramos en 3×10^{-4} por lo que proponemos dicho valor para ser el utilizado por defecto. Si vemos la curva ROC en la Figura 4.4 el punto más a la izquierda y arriba que está en el gráfico precisamente corresponde con el valor de $\epsilon = 3 \times 10^{-4}$. A pesar de que para cada imagen el valor de ϵ óptimo es diferente, podemos iniciar con este valor y ajustarlo de forma iterativa si lo que se desea es gran precisión, si lo único que se desea es encontrar a groso modo la(s) zona(s) clonada(s) ese valor nos podrá servir de buena manera. De la experiencia obtenida haciendo pruebas en el algoritmo creemos que un rango adecuado de valores de ϵ para imágenes JPEG puede ir desde 1×10^{-5} hasta 1×10^{-3}.

Tabla 4.6: Valores medios de F-Measure obtenidos para diferentes valores de ϵ utilizando imágenes JPEG.

Valor de ϵ	**FPR**	**TPR**	**F-Measure Media**
1×10^{-6}	0.00	0.00	0.00
1×10^{-5}	0.015	0.475	0.62
7×10^{-5}	0.04	0.515	0.66
9×10^{-5}	0.025	0.525	0.64
1×10^{-4}	0.045	0.685	0.77
2×10^{-4}	0.05	0.85	0.81
3×10^{-4}	0.08	0.875	0.83
4×10^{-4}	0.13	0.71	0.77
1×10^{-3}	0.12	0.96	0.74

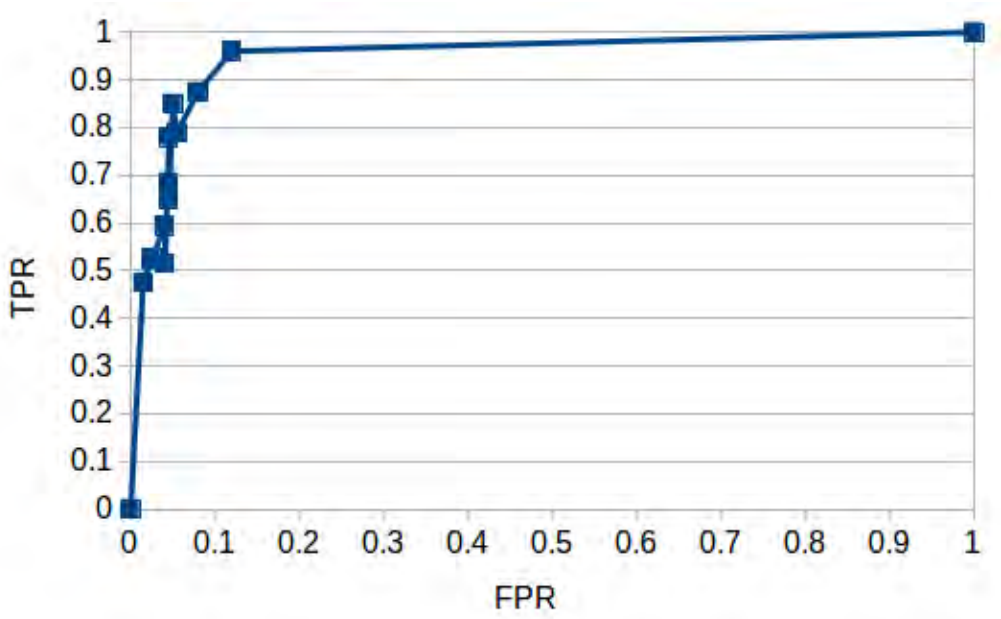


Figura 4.4: Curva ROC para el parámetro ϵ en imagenes JPG.

Recomendaciones

Si se desea ajustar el parámetro ϵ de manera fina (que puede ser diferente para cada imagen) recomendamos el siguiente procedimiento:

- Utilizar el valor propuesto según el formato de la imagen.
- Si la cantidad de regiones detectadas parecen ser muchas disminuir el valor de ϵ en 1×10^{-7} para PNG y en 5×10^{-5} para JPEG.
- Si la cantidad de regiones detectadas parecen ser muy pocas aumentar el valor de ϵ en 1×10^{-7} para PNG y en 5×10^{-5} para JPEG.
- Repetir de manera iterativa el proceso anterior, pudiendo realizar decrementos o incrementos de ϵ cada vez más pequeños, para refinar el resultado. Si entre diferentes valores de ϵ la imagen resultante parece no cambiar de forma significativa es que la solución entregada es la detección correcta del algoritmo.

Aunque en general utilizar el valor propuesto para cada tipo de imagen nos genera resultados donde las áreas clonadas se pueden apreciar, el realizar un ajuste de ϵ solo es con la intensión de obtener una imagen donde las regiones clonadas se ajusten de forma más precisa.

4.3. Pruebas de Agrupamiento

Una de las características que hacen que nuestro algoritmo sea robusto es la implementación de grupos automáticos, gracias a esta idea es posible utilizar valores de ϵ "grandes"que nos generarían Falsos Positivos pero que el agrupamiento elimina. La idea es simple, fácil de implementar, eficiente y ayuda en gran medida a mejorar los resultados. Para demostrarlo en esta sección vamos a realizar y presentar resultados obtenidos con el agrupamiento y sin él.

Para esto vamos a utilizar la imagen con identificador 11 en su versión JPG. Escogimos esta imagen porque tiene múltiples regiones clonadas y los grupos deberían detectar cada región como perteneciente a un grupo en específico. La Figura 4.5 muestra la imagen a utilizar, la cual consta de dos regiones clonadas, la primera es un bloque de pasto que se utilizó para tapar un registro metálico y la segunda región es una tapadera de cemento que se duplico.

Figura 4.5: Imagen 11, con los contornos de las regiones clonadas.

Empezaremos con una prueba para ver como luce un resultado si se utilizan los

grupos y si no se utilizan, en la Tabla 4.7 vemos los valores obtenidos para nuestras métricas de error, hay que notar como decrementan estos valores de manera drástica al no hacer uso de los grupos. De todos los valores de las métricas el que más llama la atención es TNR, el cual mide la Tasa de Verdaderos Negativos, ese valor tan pequeño lo que nos dice es que no encontramos los verdaderos negativos. Al utilizar como métrica principal *F-Measure* es importante no dejarnos llevar solamente por el valor obtenido por ésta, a pesar de ser la que intenta reunir la mayor cantidad de información para representarla en un solo valor, siempre es importante estar pendientes de todos los demás resultados y de esta manera poderlos interpretar correctamente.

Tabla 4.7: Comparación de resultados obtenidos para la imagen ID 11, utilizando nuestra propuesta de agrupamiento y sin utilizarla.

Tipo	**TPR**	**TNR**	**ACC**	**Precision**	**F-Measure**	**Tiempo**
Con Grupos	0.89	0.98	0.93	0.98	0.93	28.39 seg.
Sin Grupos	0.97	0.22	0.6	0.55	0.71	24.42 seg.

En ambos casos (cuando se utilizaron grupos y cuando no) el valor de ϵ fue el mismo 1×10^{-4}. Utilizamos un valor grande para que fuera más notorio el efecto que tiene el agrupamiento. El no utilizar grupos la única ventaja que tiene es un menor tiempo de ejecución, pero este es mínimo.

La Figura 4.6 muestra ambas máscaras obtenidas, (a) haciendo uso de los grupos y (b) sin utilizarlos. Al no hacer uso de los grupos y por el hecho de haber utilizado un valor grande de ϵ la respuesta obtenida esta llena de Falsos Positivos. Tal como se había mencionado el haber obtenido un valor tan pequeño de TNR significaba no haber encontrado los Verdaderos Negativos (tan solo se encontraron el 22 %), esto en la imagen resultante luce como regiones blancas que debieron ser negras.

(a) Máscara obtenida con grupos (b) Máscara obtenida sin grupos

Figura 4.6: Comparación entre las máscaras obtenidas si se utiliza o no el agrupamiento propuesto.

Otra de las ventajas de utilizar el agrupamiento es que cada región clonada con su contraparte pueden ser marcadas con mismo grupo. En la Figura 4.7 se muestra la máscara obtenida con agrupamiento en donde cada grupo fue marcado con un color diferente y se nota claramente como las regiones originales y sus contrapartes clonadas son de los mismos colores.

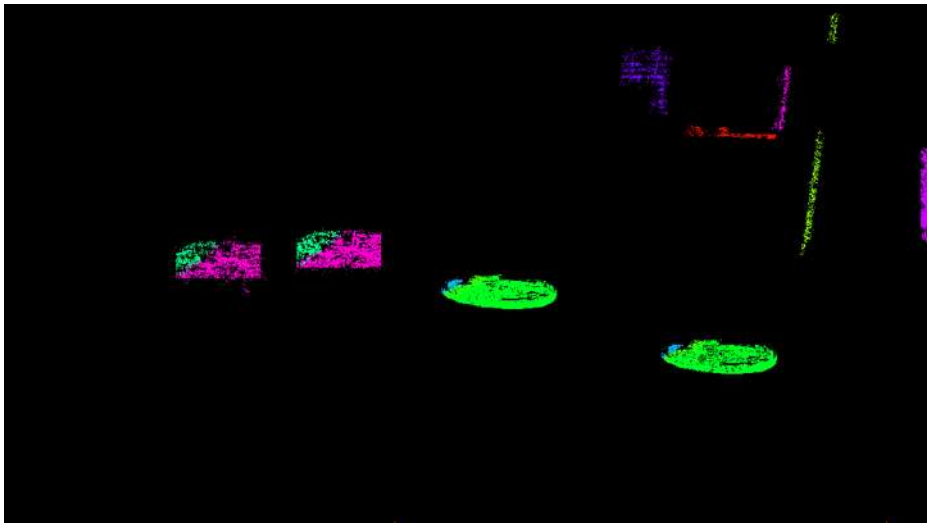


Figura 4.7: Máscara obtenida con grupos, para un $\epsilon = 1 \times 10^{-4}$, en donde cada grupo fue marcado con un color diferente.

4.4. Resultados para Imágenes PNG

En esta sección vamos a presentar las pruebas realizadas a las imágenes en nuestra base de datos con compresión sin perdida (PNG). A dichas imágenes les aplicamos el algoritmo *DeReC*, utilizando los valores propuestos para los parámetros según la sección anterior. En la Tabla 4.8 vemos los resultados obtenidos según las métricas para las 12 imágenes de la base de datos, solo se muestran los valores de TPR y TNR ya que $FPR = 1 - TNR$ y $FNR = 1 - TPR$. Los resultados están ordenados de mayor a menor utilizando el valor de $F\text{-}Measure$, se puede observar que ninguno de los valores es inferior a 0.8 es decir que los resultados son de buenos a excelentes.

Tabla 4.8: Métricas de error de los resultados obtenidos por el algoritmo *DeReC* para las imágenes PNG.

Imagen ID	TPR	TNR	ACC	Precision	F-Measure
2	0.92	1	0.96	1	0.96
6	0.91	0.99	0.95	0.99	0.95
8	0.91	1	0.95	1	0.95
3	0.89	1	0.94	1	0.94
7	0.89	1	0.94	1	0.94
10	0.88	1	0.94	1	0.94
11	0.89	1	0.94	1	0.94
5	0.86	1	0.93	1	0.92
12	0.85	1	0.93	1	0.92
1	0.84	1	0.92	1	0.91
4	0.74	1	0.87	1	0.85
9	0.73	1	0.87	1	0.85

En la Figura 4.8 se observan los resultados obtenidos por *DeReC* para la base de datos de imágenes en formato PNG. El orden presentado es al igual que el mostrado

en la tabla 4.8. Para cada imagen de la base de datos tenemos tres imágenes en la figura, la primera es la máscara con la que se realizo la clonación, la segunda es la máscara que encuentra *DeReC* y la tercera es la imagen manipulada en donde se marcan los contornos de las regiones que detecto *DeReC*.

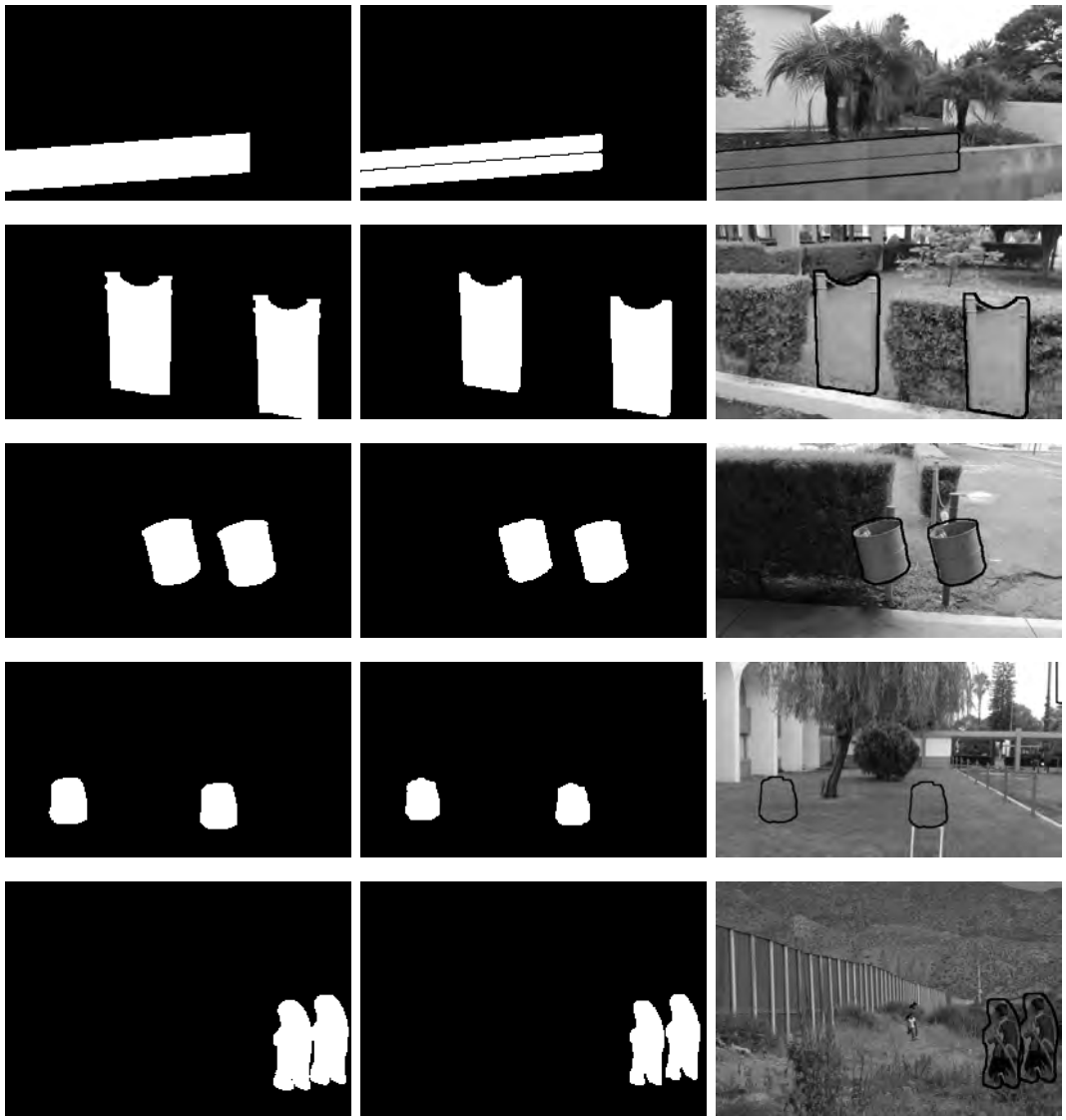


Figura 4.8: Resultados para la images en formato PNG, de nuestra base de datos.

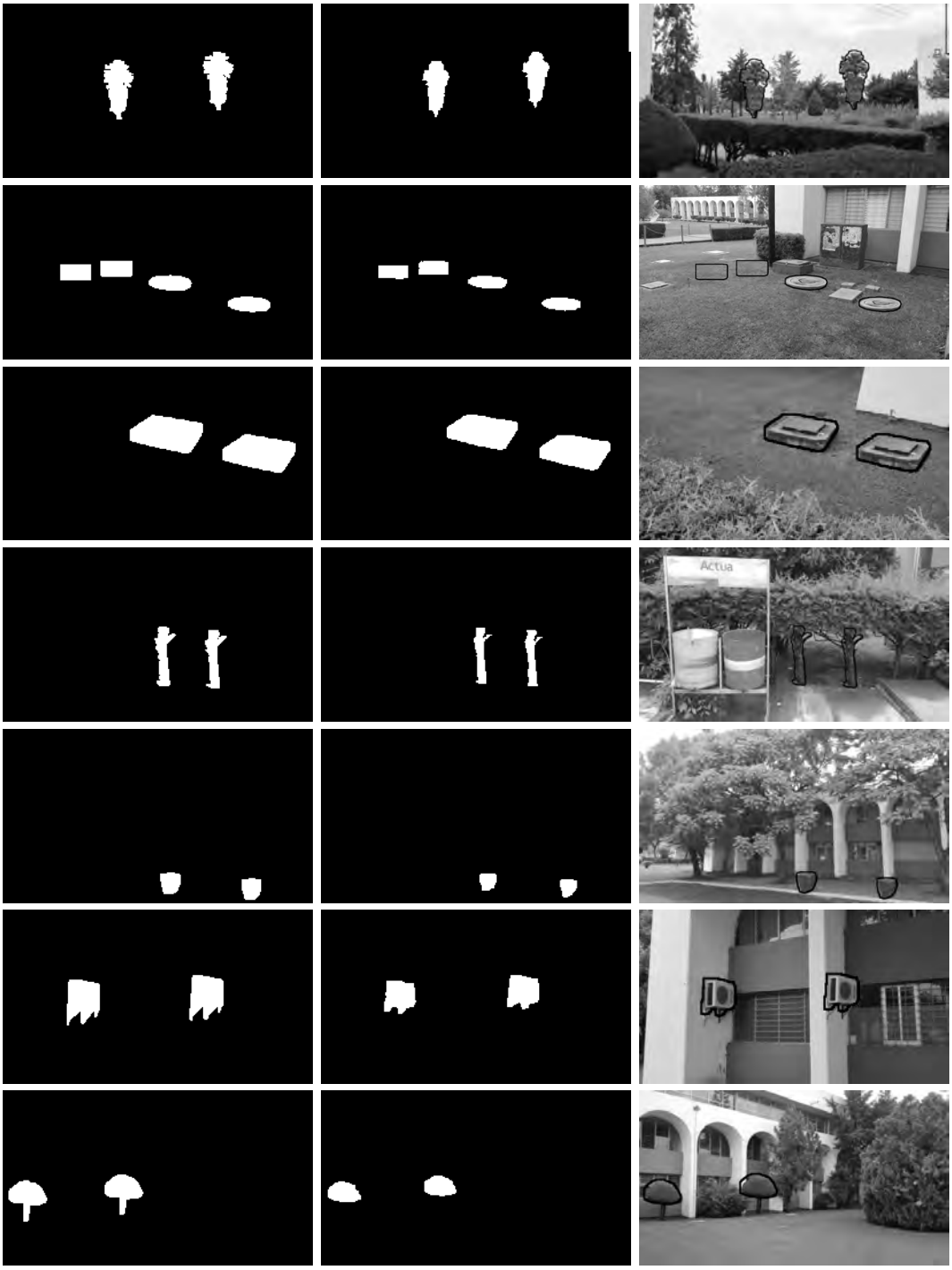


Figura 4.9: Continuación de resultados para la images en formato PNG, de nuestra base de datos.

4.5. Resultados para Imágenes JPEG

Para cada una de las imágenes JPEG en nuestra base de datos realizamos la búsqueda de regiones clonadas utilizando nuestro algoritmo, para las máscaras binarias obtenidas como resultado hacemos el calculo de las métricas, la Tabla 4.9 nos muestra los resultados obtenidos, en la tabla de las cuatro tasas TPR, TNR, FPR y FNR solo se muestran dos TPR y TNR ya que $FPR = 1-TNR$ y $FNR = 1-TPR$, de esta manera no saturamos la tabla y se hace más comprensible. Cada renglón de la tabla es para una imagen, el número es el identificador de la imagen y los resultados están ordenados por el valor de $F\text{-}Measure$ de mayor a menor.

Tabla 4.9: Métricas de error de los resultados obtenidos por el algoritmo $DeReC$ para las imágenes JPEG.

Imagen ID	**TPR**	**TNR**	**ACC**	**Precision**	**F-Measure**
6	0.91	0.98	0.95	0.98	0.95
2	0.92	0.97	0.95	0.97	0.94
8	0.86	0.98	0.92	0.98	0.92
10	0.86	0.99	0.93	0.99	0.92
5	0.84	0.99	0.91	0.99	0.91
7	0.84	1	0.92	1	0.91
3	0.82	0.99	0.91	0.99	0.9
12	0.81	1	0.9	0.99	0.89
1	0.76	1	0.88	1	0.86
4	0.76	0.97	0.86	0.96	0.85
9	0.7	1	0.85	1	0.82
11	0.7	0.99	0.84	0.98	0.82

En la Figura 4.10 se observan los resultados obtenidos por $DeReC$ para la base de datos de imágenes en formato JPEG. El orden presentado es al igual que el mostrado

en la tabla 4.9. Para cada imagen de la base de datos tenemos tres imágenes en la figura, la primera es la máscara con la que se realizo la clonación, la segunda es la máscara que encuentra *DeReC* y la tercera es la imagen manipulada en donde se marcan los contornos de las regiones que detecto *DeReC*. A pesar de que los resultados obtenidos evidentemente son peores que los obtenidos para el formato PNG (que era de esperarse), en general podemos decir que son buenos, ya que ayudan a detectar de buena forma las regiones clonadas.

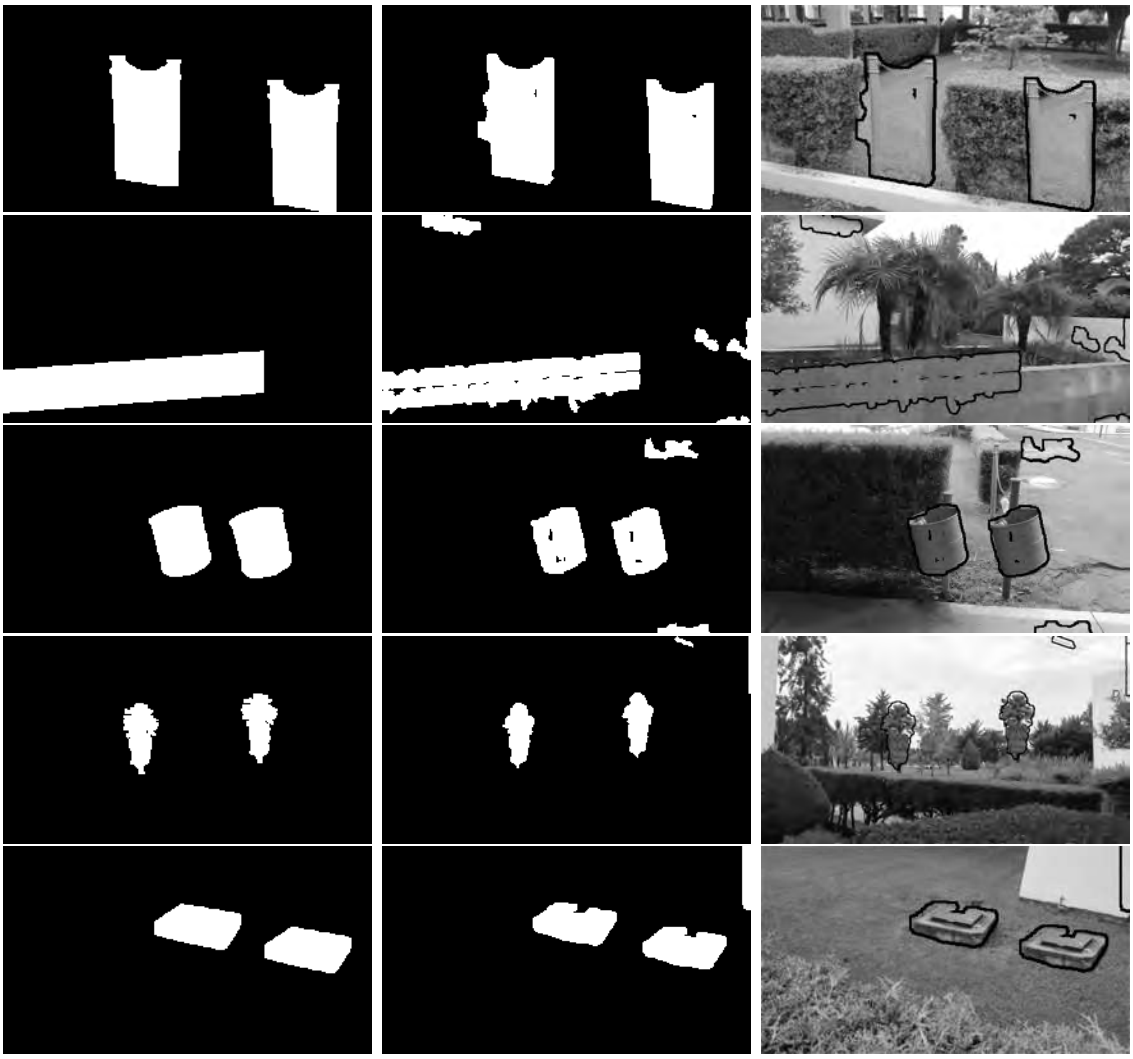


Figura 4.10: Resultados para la images en formato PNG, de nuestra base de datos.

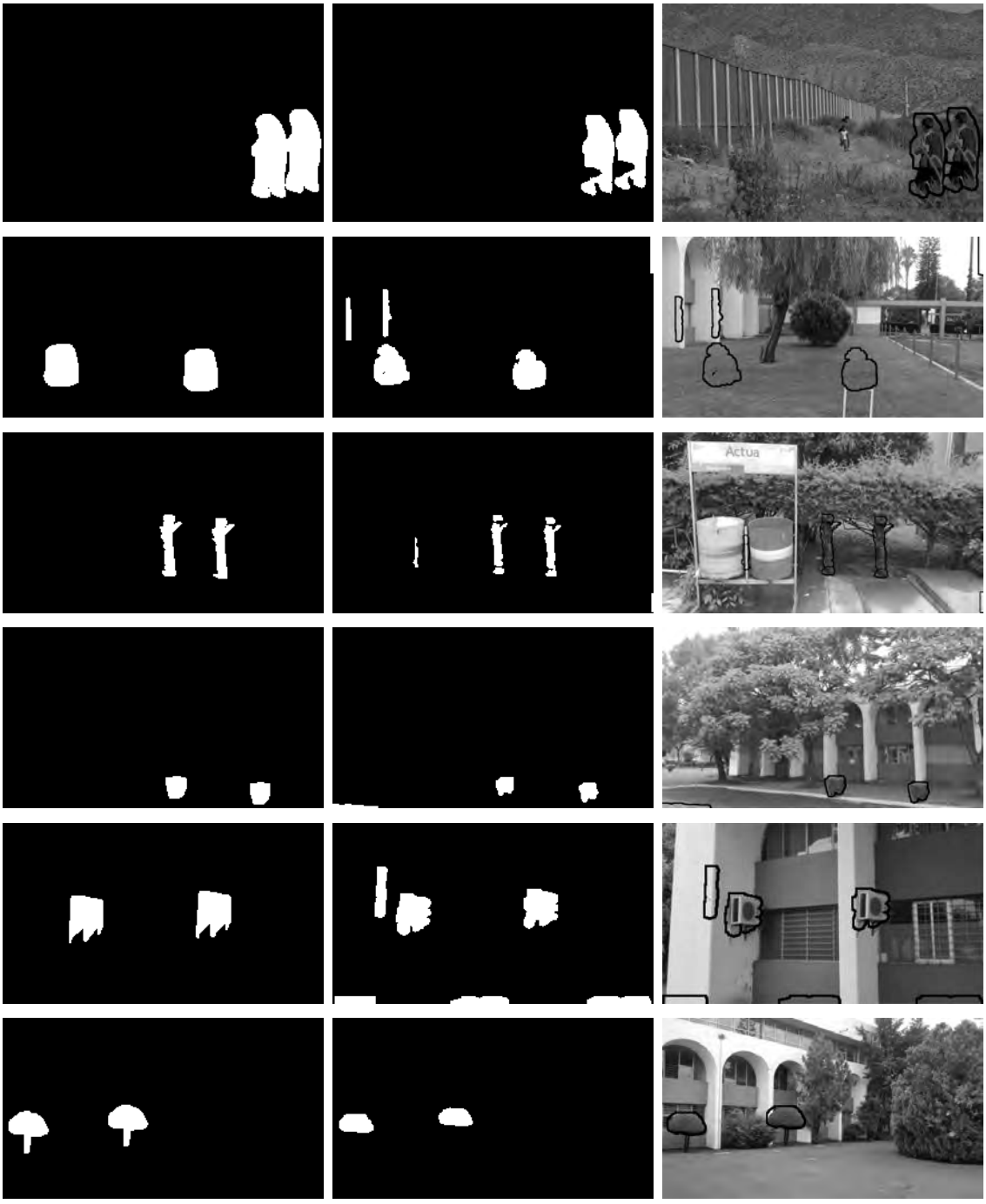


Figura 4.11: Continuación de resultados para las imágenes en formato JPG, de nuestra base de datos.

Figura 4.12: Continuación de resultados para las imágenes en formato JPG, de nuestra base de datos.

De todos los valores en la Tabla 4.9, la columna referente a *F-Measure* es la que más nos interesa. Podemos observar que en ninguno de los casos se obtuvo un valor por debajo de 0.8, en general concluimos que el algoritmo presentado obtiene buenos resultados en la detección de regiones clonadas.

4.5.1. Imágenes con Escalamiento y Rotación

Otra prueba que consideramos interesante (que no la incluimos en los resultados para el formato PNG, ya que la prioridad de nuestra propuesta era detectar regiones en imágenes con formato JPEG), es intentar detectar regiones clonadas que cuentan con escalamiento y/o rotación. *DeReC* no fue planeado para que soportara estas características, pero dadas las propiedades de la DCT creemos que puede ser capaz de lograr detectar este tipo de transformaciones geométricas, aunque sea en menor medida.

Para comprobar esta teoría aplicamos el algoritmo *DeReC* a una imagen que cuenta con una región clonada, la cual a sido escalada y rotada, dicha imagen se muestra en la Figura 4.13.

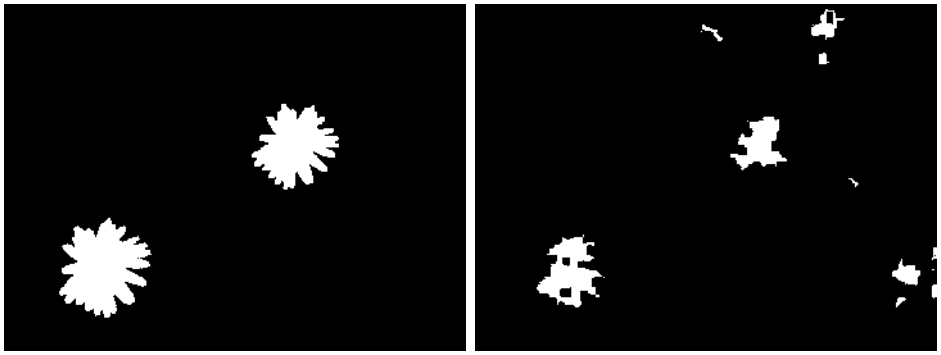
Figura 4.13: Imagen con una región clonada, la cual fue rotada y escalada.

Una vez aplicado nuestro algoritmo por medio de las métricas de error evaluamos el resultado obtenido, la Tabla 4.10 muestra los valores calculados para las métricas de error.

Tabla 4.10: Métricas de error de los resultados obtenidos por el algoritmo *DeReC* para la imagen de la Figura 4.13.

Imagen ID	**TPR**	**TNR**	**ACC**	**Precision**	**F-Measure**
Margaritas	0.37	0.99	0.68	0.97	0.53

Para analizar el resultado de forma visual tenemos la máscara que se utilizó para la clonación (Ground Truth) y el resultado obtenido por el algoritmo en la Figura 4.14. En la Figura 4.15 vemos los contornos de las áreas detectadas marcados en color negro sobre la imagen alterada. Dichos contornos no se ajustan por completo a las margaritas que están clonadas, pero son suficientes para darnos una idea de donde se encuentra la alteración. Por lo tanto nuestro algoritmo aunque no fue diseñado para detectar regiones clonadas con estás alteraciones, si es capaz de hacerlo, obteniendo resultados suficientes para encontrar la alteración.



(a) Máscara Real (b) Máscara Obtenida

Figura 4.14: Máscaras real y obtenida para una imagen alterada con una región clonada que cuenta con escalamiento y rotación.

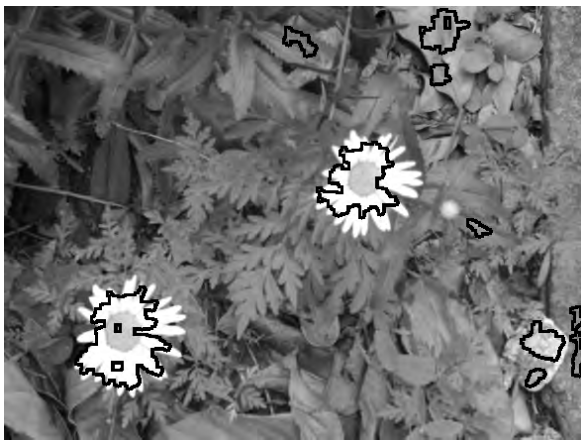


Figura 4.15: Contornos detectados para una imagen con una región clonada, la cual fue rotada y escalada.

4.5.2. Comparación con Otros Trabajos

[Ferreira et al., 2016] presentan un trabajo en donde hacen la comparación de diferentes algoritmos de detección de regiones clonadas, en su trabajo proponen un método utilizando mezcla de clasificadores, por esta razón no nos comparamos directamente contra su trabajo ya que como tal no proponen como hacer la detección de

regiones clonadas, sino como en base a varios clasificadores obtener un mejor resultado. Lo que nos llama la atención de su trabajo es que aplican estas mismas métricas para medir el desempeño de algunos clasificadores propuestos por otros autores. Presentan una base de datos que consta de 108 imágenes en las cuales existen regiones clonadas, algunas de ellas con escalamientos, otras con rotaciones e incluso algunas con ambas transformaciones. Sobre su base de datos hacen pruebas a varios clasificadores, por lo que utilizando su base probaremos a $DeReC$ y lo compararemos contra los resultados de las propuestas de otros autores.

En los resultados obtenidos para los clasificadores de otros autores en imágenes con compresión JPEG el mejor resultado es el que corresponde al trabajo realizado por [Shivakumar and Baboo, 2011], obteniendo un valor medio de $F\text{-}Measure = 0.765$. En la Tabla 4.11 se muestran los resultados obtenidos por nosotros y los resultados obtenidos por [Ferreira et al., 2016] para múltiples algoritmos, utilizando la misma base de datos.

Tabla 4.11: Comparación de resultados, utilizando la media de las métricas de error, para imágenes con regiones clonadas escaladas y/o rotadas.

Algoritmo	**TPR**	**TNR**	**ACC**	**Precision**	**F-Measure**
$DeReC$	0.582	0.911	0.746	0.861	0.682
Shivakumar and Baboo (SURF)	0.622	0.760	0.809	0.996	0.765
Shivakumar and Baboo (SIFT)	0.539	0.850	0.768	0.997	0.698
Amerini et al. (Hierarch-SIFT)	0.482	0.790	0.740	0.995	0.648
Silva et al. (Multiscale Voting)	0.301	0.310	0.647	0.972	0.459

Como puede verse los resultados que obtenemos con $DeReC$ son muy cercanos a los resultados que obtienen otros algoritmos o incluso mejores. En las Figura 4.16 se muestran algunas de las imágenes procesadas para esta prueba, donde los contornos negros indican las zonas clonadas detectadas.

Figura 4.16: Algunos resultados obtenidos para la base de datos de [Ferreira et al., 2016].

4.5.3. Pruebas para Diferentes Niveles de Calidad en JPEG

El algoritmo de compresión JPEG permite un parámetro para controlar la calidad que va desde 0 hasta 100, donde 0 significa la mayor compresión posible y por lo tanto la menor calidad de imagen y 100 es la mayor calidad posible y una menor compresión que a su vez implica menor perdida de información.

Debido a que diferentes valores de calidad generan diferentes imágenes con diferente información creemos que es conveniente saber hasta que punto nuestro algoritmo aún es capaz de detectar las regiones clonadas. La prueba que hicimos fue comprimir una imagen con diferentes niveles de calidad y aplicar nuestro algoritmo, a la respuesta obtenida le calculamos las métricas de error y vemos como se comporta el error en base al nivel de compresión, los resultados obtenidos se muestran en la Tabla 4.12.

Tabla 4.12: Respuesta obtenida para diferentes niveles de calidad utilizados en una imagen JPEG.

Calidad	**TPR**	**TNR**	**ACC**	**Precision**	**F-Measure**
95	0.9	0.99	0.95	0.99	0.94
90	0.9	0.99	0.94	0.99	0.94
85	0.85	0.99	0.92	0.99	0.91
80	0.89	0.99	0.94	0.99	0.94
75	0.87	0.95	0.91	0.94	0.91
70	0.81	0.99	0.9	0.98	0.89
65	0.77	0.96	0.86	0.95	0.85
60	0.68	0.98	0.83	0.97	0.8
50	0.6	0.96	0.78	0.94	0.73
40	0.6	0.98	0.79	0.97	0.74
30	0.53	1	0.77	1	0.7
20	0.56	0.98	0.77	0.96	0.71
10	0.61	0.94	0.77	0.91	0.73

Como era de esperarse al disminuir la calidad de la imagen detectar las regiones clonadas es más difícil por lo que la eficacia del algoritmo disminuye tal como se muestra en la Tabla 4.12.

[Ferreira et al., 2016] en su trabajo solo realizan pruebas con imágenes JPEG que tienen una calidad mayor a 70 ya que para valores menores los resultados que obtienen son muy malos. En nuestro caso a pesar de que se degrada la calidad de la detección conforme baja la calidad de la imagen, es paulatina la degradación y podemos detectar las regiones clonadas aún a niveles de calidad muy bajos.

4.6. Pruebas de Rendimiento

En esta sección vamos a analizar el tiempo de respuesta que le toma a nuestro algoritmo encontrar una solución, en la Tabla 4.13 se muestra el tiempo necesario de computo para procesar cada imagen de nuestra base de datos en formato JPEG, también están los tamaños de las imágenes procesadas, ya que el tiempo de computo es directamente proporcional al tamaño de la imagen.

Tabla 4.13: Tiempo necesario para procesar cada imagen JPEG de nuestra base de datos.

Imagen ID	**Tiempo en segundos**
1	5.390
2	8.176
3	5.787
4	1.781
5	1.756
6	1.776
7	4.513
8	3.044
9	3.390
10	12.356
11	14.551
12	14.174

El tiempo total que le lleva al algoritmo detectar las regiones clonadas en una imagen no solo depende del tamaño de la imagen, también afecta que tantos píxeles se detectan como clonados ya que para dichos píxeles es necesario buscar un grupo, por esta razón imágenes con tamaños similares pueden obtener tiempos diferentes, incluso imágenes más pequeñas pueden tardar más tiempo en ser procesadas que otras

más grandes.

En la literatura no encontramos información contra la que nos pudiéramos comparar respecto al tiempo de ejecución de forma directa, primeramente porque hay artículos en el área que no mencionan el tiempo de computo y segundo porque los artículos que si lo mencionan utilizan equipos muy diferentes al que nosotros usamos y no es posible hacer la comparación, ya que los equipos más lentos es obvio que obtendrán peores tiempos. A pesar de esto consideramos que el tiempo obtenido es razonable y menor que incluso el obtenido por otros autores en equipos de computo superiores al nuestro.

[Ferreira et al., 2016] mencionan en su trabajo tiempos de computo incluso de miles de segundos, utilizando una computadora con mayor capacidad que la nuestra (Intel® Core ™ i7-5820K CPU @3.30GHz con 62GB de RAM), pero no dicen de forma clara como obtienen esos tiempo ni para que tamaños de imagen, por lo que no hay punto de comparación. Una consideración importante a mencionar es que el computo se realizó utilizando solamente un núcleo del procesador.

Capítulo 5

Conclusiones

5.1. Conclusiones Generales

Gracias a todas las pruebas y validaciones realizadas al algoritmo propuesto podemos concluir que tenemos un algoritmo altamente robusto; al compararlo con la literatura en el área encontramos que la calidad de nuestros resultados es superior a las propuestas de otros autores. Aunque no fue posible comparar directamente nuestros tiempos de procesamiento contra otros trabajos (debido a que no todos los autores presentan tiempos de cómputo y los que presentan estos datos utilizan equipos diferentes al de nosotros). Sabemos que nuestra propuesta es muy eficiente, ya que incluso en imágenes de gran tamaño el análisis toma solo algunos segundos, otras propuestas toman varios cientos de segundo, y creemos que aún el posible realizar mejoras en el desempeño.

Los puntos más fuertes de nuestra propuesta y los cuales no se encontraron en propuestas similares son principalmente el uso del algoritmo *vpSorting* para el ordenamiento de los vectores de características, el cual es eficiente en tiempo de cómputo y a la hora de hacer búsquedas de vecinos cercanos sobre los vectores ordenados los resultados son excepcionales. En la mayoría de casos se encuentran los verdaderos vecinos cercanos, esto significa que el ordenamiento se realiza de manera adecuada, y

la búsqueda de vecinos cercanos se vuelve simple y eficiente. El otro punto fuerte es la propuesta de agrupamiento, la cual nos permite eliminar Falsos Positivos de manera eficiente y al mismo tiempo que se encuentran los píxeles duplicados, permitiendo tanto clasificar las regiones clonadas, como depurar el resultado y todo esto de forma automática ya que la búsqueda de grupos fue ajustada de tal manera en la que no es necesario definir parámetros por el usuario, tampoco se necesita información a priori sobre como quedarán los grupos, ni cuantos deben ser. El agrupamiento propuesto por nosotros hace algo similar a lo que haría un proceso convencional de clustering, pero de forma más eficiente y simple.

En general nuestro algoritmo obtuvo buenos resultados tanto en imágenes con compresión sin pérdida (PNG) como en imágenes con compresión con pérdida (JPEG) y como era de esperarse los resultados fueron mejores para las imágenes del primer grupo (PNG). Aunque solo realizamos pruebas para estos dos formatos de imagen estamos completamente seguros que el algoritmo es capaz de obtener resultados en casi cualquier tipo de imagen, para los formatos sin pérdida obtendrá resultados de forma similar que con PNG y para los formatos con pérdida los resultados serán similares que a JPEG, ya que la mayoría de formatos de compresión con pérdida suelen basarse en quitar información poco perceptible a nuestros ojos, tal como lo hace JPEG.

A pesar de que el algoritmo presentado es robusto aún es posible realizar mejoras en él, ya que solo es capaz de detectar regiones clonadas con poco escalamiento, pequeñas rotaciones y cambios de tono de color mínimos. Si volvemos más robustos nuestros vectores de características en esos aspectos tendremos una herramienta muy potente.

5.2. Trabajos Futuros

Aún hay muchas cosas que se pueden realizar en este trabajo con la intención de tener una herramienta más potente y automática que permita detectar las regiones clonadas. A continuación se mencionan algunas áreas de oportunidad del algoritmo.

1. Implementar vectores de características que sean robustos a escalamientos, cambios de color y rotaciones. De esta manera nuestro algoritmo podría detectar regiones clonadas que tengan este tipo de transformaciones de manera más eficiente. Y creemos que a pesar de no habernos planteado este objetivo estamos muy cerca de lograrlo.

2. Realizar una detección bajo un esquema Grueso-Fino, primero detectar las regiones clonadas como lo hace actualmente el algoritmo y por medio de un proceso más fino encontrar los bordes de forma adecuada de las regiones clonadas.

3. Trabajar con solo algunas de las regiones de la imagen. Si podemos determinar cuales regiones de la imagen son interesantes y en las que puedan existir regiones clonadas, podríamos generar vectores de características solo para dichas regiones y no para toda la imagen. Lo que simplificaría en gran medida el tiempo de cómputo necesario para procesar una imagen, esto sería de gran utilidad pensando en los tamaños de imagen actuales y que cada día son mayores.

4. Hacer pruebas utilizando diferentes técnicas de búsquedas de vecinos cercanos. Actualmente solo hicimos dos propuestas, pero existen algoritmos más nuevos que permiten hacer búsquedas aproximadas de vecinos cercanos en millones de datos de forma eficiente.

5. Realizar pruebas para diferentes formatos de imágenes con compresión con pérdida, aquí solo se abordo JPEG por ser el más común, pero existen muchos otros formatos de imagen los cuales también realizan una compresión con per-

dida, por ejemplo JPEG 2000. No nos enfocaríamos en los formatos sin pérdida ya que deberán funcionan de manera similar que PNG.

6. Utilizar diferentes tipos de transformaciones que nos permitan trabajar en el dominio de la frecuencia, como puede ser utilizar la Transformada Wavelet Discreta.

7. Definir los valores de los parámetros de forma automática según las características de cada imagen, de tal manera que la herramienta final solo necesite una imagen de entrada para poder generar la imagen de salida.

8. Proponer una base de datos con cientos de imágenes, las cuales contengan características diferentes y así poder validar que sea robusto el algoritmo.

9. Realizar pruebas en imágenes con ruido.

10. Un trabajo que ayudaría en gran medida a popularizar nuestro algoritmo sería implementarlo como módulo para algún editor importante, por ejemplo GIMP, que es un editor de imágenes OpenSource al cual se le puede extender la funcionalidad por medio de módulos.

Referencias

[Andrés 1996] Andrés, María Luisa M.: *Algoritmos de búsqueda de vecinos más próximos en espacios métricos*, Universidad Politécnica de Valencia, Tesis Doctoral, 1996

[Bayram et al. 2009] Bayram, Sevinc ; Sencar, Husrev T. ; Memon, Nasir: An efficient and robust method for detecting copy-move forgery. In: *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on* IEEE (Veranst.), 2009, S. 1053–1056

[Brin 1995] Brin, S.: Near Neighbor Search in Large Metric Spaces. In: *21th International Conference on Very Large Data Bases (VLDB 1995)*, 1995

[Chris Solomon 2011] Chris Solomon, Toby B.: *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. Wiley-Black Well, 2011. – ISBN 978-0470844731

[Cover and Hart 1967] Cover, Thomas ; Hart, Peter: Nearest neighbor pattern classification. In: *IEEE transactions on information theory* 13 (1967), Nr. 1, S. 21–27

[Duda et al. 2012] Duda, Richard O. ; Hart, Peter E. ; Stork, David G.: *Pattern classification*. John Wiley & Sons, 2012

[Ferreira et al. 2016] Ferreira, Anselmo ; Felipussi, Siovani C. ; Alfaro, Car-

los ; Fonseca, Pablo ; Vargas-Muñoz, John E. ; Santos, Jefersson A. dos ; Rocha, Anderson: Behavior Knowledge Space-Based Fusion for Copy–Move Forgery Detection. In: *IEEE Transactions on Image Processing* 25 (2016), Nr. 10, S. 4729–4742

[Fridrich et al. 2003] Fridrich, A J. ; Soukal, B D. ; Lukáš, A J.: Detection of copy-move forgery in digital images. In: *Proceedings of Digital Forensic Research Workshop* Citeseer (Veranst.), 2003

[Gloe et al. 2007] Gloe, Thomas ; Kirchner, Matthias ; Winkler, Antje ; Rainer, Böhme: Can we trust digital image forensics? In: *Proceedings of the 15th ACM international conference on Multimedia* ACM (Veranst.), 2007, S. 78–86

[Hashmi et al. 2014] Hashmi, Mohammad F. ; Anand, Vijay ; Keskar, Avinas G.: Copy-move image forgery detection using an efficient and robust method combining un-decimated wavelet transform and scale invariant feature transform. In: *AASRI Procedia* 9 (2014), S. 84–91

[Huang et al. 2008] Huang, Hailing ; Guo, Weiqiang ; Zhang, Yu: Detection of copy-move forgery in digital images using SIFT algorithm. In: *Computational Intelligence and Industrial Application, 2008. PACIIA'08. Pacific-Asia Workshop on* Bd. 2 IEEE (Veranst.), 2008, S. 272–276

[Huffman 1952] Huffman, David A.: A method for the construction of minimum-redundancy codes. In: *Proceedings of the IRE* 40 (1952), Nr. 9, S. 1098–1101

[ISO/IEC 10918-1 1994] Digital compression and coding of continuous-tone still images / International Organization for Standardization. 1994. – Standard ISO/IEC 10918-1

[Jain 1989] Jain, Anil K.: *Fundamentals of Digital Image Processing*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1989. – ISBN 978-0133361650

[Khayam 2003] Khayam, Syed A.: The discrete cosine transform (DCT): theory and application. In: *Michigan State University* 114 (2003)

[Langille and Gong 2006] Langille, Aaron ; Gong, Minglun: An efficient match-based duplication detection algorithm. In: *Computer and Robot Vision, 2006. The 3rd Canadian Conference on* IEEE (Veranst.), 2006, S. 64–64

[Li et al. 2007] Li, Guohui ; Wu, Qiong ; Tu, Dan ; Sun, Shaojie: A sorted neighborhood approach for detecting duplicated regions in image forgeries based on DWT and SVD. In: *Multimedia and Expo, 2007 IEEE International Conference on* IEEE (Veranst.), 2007, S. 1750–1753

[Lim 1990] Lim, Jae S.: *Two-dimensional Signal and Image Processing*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1990. – ISBN 0-13-935322-4

[Mahdian Babak 2007] Mahdian Babak, Saic S.: Detection of copy-move forgery using a method based on blur moment invariants. In: *Forensic science international* 171 (2007), Nr. 2, S. 180–189

[Malviya and Ladhake 2016] Malviya, Ashwini V. ; Ladhake, Siddharth A.: Pixel Based Image Forensic Technique for Copy-move Forgery Detection Using Auto Color Correlogram. In: *Procedia Computer Science* 79 (2016), S. 383–390

[Mitra 2011] Mitra, Sanjit K.: *Digital Signal Processing: A Computer Based Approach*. Mcgraw Hill Higher Education, 2011. – ISBN 0071289461

[Ng et al. 2004] Ng, Tian-Tsong ; Chang, Shih-Fu ; Sun, Qibin: Blind detection of photomontage using higher order statistics. In: *Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on* Bd. 5 IEEE (Veranst.), 2004, S. V–V

[Ordoñez 2005] Ordoñez, Cristian: Formatos de imagen digital. In: *Revista Digital Universitaria [en línea]. 10 de mayo 2005, Vol. 6, No. 5. Disponible*

en: http://www.revista.unam.mx/vol.6/num5/art50/int50.htm. ISSN: 1607-6079. (2005)

[Orozco et al. 2012] Orozco, Ana Lucila S. ; González, David Manuel A. ; Villalba, Luis Javier G. ; Castro, Julio César H.: Anomalías en el Seguimiento de Exif en el Análisis Forense de Metadatos de Imágenes de Moviles. In: *Actas del XII Reunión Espanola sobre Criptología y Seguridad de la Información, Donostia-San Sebastián, España* (2012)

[Powers 2011] Powers, David M.: Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. In: *Journal of Machine Learning Technologies* 2 (2011), Nr. 1, S. 37–63. – ISSN 2229-3981

[Serra 1983] Serra, Jean: *Image Analysis and Mathematical Morphology*. Academic Pr, 1983. – ISBN 0126372403

[Shivakumar and Baboo 2011] Shivakumar, BL ; Baboo, Lt Dr S S.: Detection of region duplication forgery in digital images using SURF. In: *IJCSI International Journal of Computer Science Issues* 8 (2011), Nr. 4

[Wallace 1992] Wallace, Gregory K.: The JPEG still picture compression standard. In: *IEEE transactions on consumer electronics* 38 (1992), Nr. 1, S. xviii–xxxiv

[Watson 1993] Watson, Andrew B.: DCT quantization matrices visually optimized for individual images. In: *proc. SPIE*, 1993

[Yianilos 1993] Yianilos, Peter N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: *SODA* Bd. 93, 1993, S. 311–321