Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Detecção de Cópias de Imagens baseada em similaridade de conteúdo

Victor Quezado Rodrigues Silva

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Flávio de Barros Vidal

Brasília
2016

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Detecção de Cópias de Imagens baseada em similaridade de conteúdo

Victor Quezado Rodrigues Silva

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Flávio de Barros Vidal (Orientador)
CIC/UnB

PCF Jorge de Albuquerque Lambert
SEPAEL/INC/DPF

Prof. Dr. Marcus Vinicius Lamar
CIC/UnB

Prof. Dr. Ricardo Pezzuol Jacobi
Coordenador do Curso de Engenharia da Computação

Brasília, 9 de dezembro de 2016

Dedicatória

Dedico esse projeto aos meus pais, que me deram suporte e me permitiram chegar onde cheguei, aos amigos que fizeram da minha estadia na universidade algo extremamente agradável e construtivo, aos professores, que se ofereceram a prestar seus conhecimentos para mim e a colegas, ao meu orientador por me apresentar este tema que muito me fascinou e por me orientar nesse trabalho de grande escala e, por fim, mas não menos importante, à ciência da computação que me apresentou esse incrível universo, cheio de lógica e desafios.

"*Everyday life is like programming, I guess. If you love something you can put beauty into it.* " - Donald Knuth.

Agradecimentos

Gostaria de agradecer ao Prof. Dr. Flávio de Barros Vidal que me apresentou esse tema e guiou-me com presteza durante essa pesquisa. Também agradeço ao Kelvin, amigo, que me ajudou nas etapas iniciais de desenvolvimento e, graças às discussões que tivemos sobre o assunto, ajudou-me a superar as dificuldades encontradas pelo caminho.

Resumo

A produção de cópias de imagens digitais cresceu muito rapidamente nas últimas décadas. Algoritmos de detecção de cópias são desenvolvidos para lidar com esse problema, seja com a finalidade de otimização de armazenamento, detecção de quebra de direitos autorais ou detecção de pornografia infantil. Para tratar dessa questão, este projeto propõe uma nova técnica usando uma variação de um algoritmo de detecção de cópias baseado em conteúdo, testando novos parâmetros e introduzindo um novo método que permite a comparação entre duas imagens sem a necessidade de usar o processo de *maximum a posteriori*. Para a descoberta desses novos parâmetros e avaliação do desempenho do algoritmo proposto, são usados três banco de dados. Os resultados destes testes são apresentados e, em maioria, satisfatórios para a solução do problema proposto.

Palavras-chave: opencv, visão computacional, CBCD, CBICD, DCT, partição em blocos, análise forense

Abstract

The quantity of digital image copies has grown a lot in the last decades. Image copy detection algorithms are developed to solve this problem, either for storage optimization, copyrights infringement or child pornography tracking. Then, this paper propose a novel technique that is a variation of a content-based image copy detection algorithm, tests new parameters and introduces a new comparison method that let the comparison between two images not be bound by the process of *maximum a posteriori*. This project uses three image databases to find out these new parameters and the performance of the proposed algorithm. The results of these tests are presented and are mostly satisfactory for the proposed problem.

Keywords: opencv, computer vision, CBCD, CBICD, DCT, block partition, forensic analysis

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CBCD *Content-Based Copy Detection.*

CBIR *Content-Based Image Retrieval.*

CUDA *Compute Unified Device Architecture.*

DCT *Discrete Cosine Transform.*

DCT-2D Transformada Discreta do Cosseno Bidimensional.

GPU *Graphics Processing Unit.*

ISS *Image Secret Sharing.*

LLE *Local Linear Embedding.*

MAP *Maximum a Posteriori.*

OpenCL *Open Computing Language.*

OpenCV *Open Source Computer Vision.*

RAISE *RAw ImageS datasEt.*

SVD *Single Value Decomposition.*

UCID *Uncompressed Colour Image Database.*

Capítulo 1

Introdução

Nas últimas décadas, o mundo digital tem evoluído muito rapidamente e, com ele, as tecnologias que envolvem mídias digitais. A produção, tanto por criação como alteração, de mídia digital está muito mais acessível. Somado a isto, há também a evolução da Internet e das tecnologias de informação, que por sua vez facilitam a disperção dessas mídias ao redor do globo. Apesar do grande potencial dessas transformações, tais facilidades também ajudam na prática de tarefas ilícitas.

Neste contexto, cópias de imagens se tornaram uma constante, seja de forma inocente ou não. Cópia pode ser definida não somente como uma representação idêntica (bit a bit) da original, mas também podem possuir alterações geométricas (rotação, redimensionamento, translação, etc.), de qualidade ou de iluminação, ou, ainda, uma representação parcial da imagem. Então, encontrar essas cópias de forma precisa, capaz de lidar com certo grau de alteração, tornou-se uma necessidade. Existem várias finalidades para encontrar tais duplicatas, dentre elas:

Detecção de Pirataria: Geralmente, o uso mais comum. Podemos citar a situação em que um indíviduo, pessoa física ou jurídica, possui como propriedade intelectual um acervo de imagens, mas este fora propagado sem a devida autorização, ocasionando uma quebra de direitos autorais. Para tratar tal situação, é necessário verificar bancos de dados, por vezes gigantescos e, portanto, dispendiosa (ainda mais se for atribuída a um ser humano), e decidir se uma imagem é ou não uma cópia.

Otimização de Espaço: Alguns sites possuem armazenamento de imagens e é comum que certas imagens alcancem grandes proporções de compartilhamento, ou seja, a mesma imagem será compartilhada ou mesmo enviada para o site por uma grande quantidade de pessoas. Assim, é vantajoso possuir um algoritmo capaz de classificar a nova imagem como cópia, para ao invés de adicionar mais outra ao banco de

dados, simplesmente cria-se uma nova referência da imagem já existente para aquele usuário. Neste caso, a imagem tem de ser bem próxima, se não igual, à original.

Detecção de Pornografia Infantil: Da mesma forma que a detecção de pirataria, busca-se uma imagem ou um conjunto delas contra um banco de dados à procura de imagens contendo pornografia infantil. No caso, uma entidade autorizada possui um acervo de imagens conhecidas e busca em servidores de *download* essas imagens e suas respectivas cópias e procura os usuários que as estão baixando. E quando encontrado um suspeito, será gerado um novo banco de dados usando os possíveis dados suspeitos contidos em seu computador e será realizada uma nova busca visando provas incriminatórias [4].

Existem várias formas de realizar esta detecção, dentre elas, as principais são: *Watermarking* e *Content-Based Copy Detection* (CBCD). *Watermarking* produz uma espécie de selo de autenticação, seja explícita ou implicitamente, no entanto tal método não só altera a imagem original, como costuma ter alta complexidade, proporcional ao grau de resistência a alterações.

Neste projeto, abordaremos a segunda técnica, CBCD. Esta se baseia na extração de características na imagem sem a necessidade de informações pré-determinadas, como no *watermarking*. As técnicas de extração destas características variam muito de solução pra solução, mas seu diferencial é extrair informações interessantes a partir de um conhecimento zero da imagem original e, através dessas informações, conseguir discernir uma imagem da outra e apontar se são similares o suficientes para serem cópias. Pode ser usada em conjunto com o *Watermarking*, mas não há interesse nesta possibilidade, uma vez que o foco está no potencial da detecção baseada em conteúdo.

Outro ponto a ser observado é que deve ser evitada a confusão entre CBCD, *Content-Based Image Retrieval* (CBIR) e detecção de forjas. Enquanto o CBCD busca a detecção de imagens que são apenas transformações de uma outra imagem, o CBIR busca algo mais amplo, como a detecção de similaridade entre duas imagens. Por exemplo, dada uma cena, mas retiradas de ângulos diferentes e estas, por sua vez, similares, mas visivelmente diferentes uma da outra, o CBCD não seria capaz de detectar, mas o CBIR seria. Já na detecção de forjas, busca-se descobrir se houve adição e/ou remoção de elementos na imagem e, se possível, ainda tenta descobrir qual das duas imagens é a original e qual é a cópia.

Como o propósito deste projeto está em apenas descobrir se duas imagens quaisquer são similares o suficiente para uma ser chamada de cópia e não se são próximas ou se foram forjadas, o foco é apresentar uma solução de CBCD.

Nos capítulos seguintes serão dados mais detalhes sobre a implementação do método proposto. No Capítulo 2, outros trabalhos que tentam solucionar o problema de detecção

de cópias baseada em conteúdo são apresentados. No Capítulo 3, é feita uma análise da estrutura do algoritmo usado como base e do algoritmo proposto, apresentando as similaridades e diferenças entre ambos. No Capítulo 4, serão relacionados os materias que foram utilizados ao longo do projeto, bem como será discutido o processo de ajuste, que levará à versão final do algoritmo, e teste, em que será posto à prova o método. No Capítulo 5, os valores obtidos nos ajustes e testes serão apresentados. Por fim, no Capítulo 6 será feita uma análise dos resultados obtidos e serão apresentadas possíveis melhorias e trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

Neste capítulo, é descrito o algoritmo-base, como também são mostradas soluções alternativas. Primeiramente, é feito um histórico de soluções antes do desenvolvimento da técnica de [1]. Posteriormente, a solução apresentada em [1] é descrita em detalhes. Depois, é visto técnicas que também usam da DCT, algumas complementares à solução de [1]. Na sequência, são mostrados métodos que oferecem soluções que não usam a DCT como base.

2.1 Trabalhos Anteriores

Métodos CBIR já existiam anteriormente ao citado em [1]. Por exemplo, um dos métodos CBIR populares da época era o método de interseção de histograma [5]. Entretanto, justamente por serem técnicas CBIR e pelo fato de histogramas de cores não preservarem informações espaciais [6], tornaram tais métodos fracos no quesito detecção de cópias.

Um dos métodos criados para lidar com o problema da localização dos pixels foi a divisão em blocos. Usando um conjunto de cores que descreviam todas as cores da imagem, Hsu [7] dividia a imagem em subimagens e extraía os histogramas de cada subimagem. Então, para comparação de duas imagens, comparavam-se seus histogramas de cores e a similaridade entre as subimagens. Porém, tal técnica demanda alto custo computacional e, portanto, demora muito para processar conjuntos de imagens muito grandes.

Chang [8] propôs o uso de *Wavelets* e conseguiu resultados significativos, onde 8 cópias de 10 em um espaço de 30000 imagens foram detectadas. Mas seus testes não foram tão profundos, uma vez que experimentarou apenas para alterações como *sharpening*, suavização e *despeckling*. No entanto, não há dados sobre a eficiência do algoritmo para alterações mais agressivas.

Outro fator que ocorreu na comparação entre imagens foi o aparecimento de um método de comparação baseado em correlação:

$$C = \frac{\sum_{i=1}^{N} \|I_{1,i} - I_{2,i}\|}{N} \tag{2.1}$$

Entretanto, tal método possui diversas limitações. Se as imagens não possuírem o mesmo número de pixels devido a um redimensinoamento de uma delas, por exemplo, tal fórmula apresentará erros. Ainda, não apresenta robustez, visto que um único pixel com valor que não segue a norma do restante da imagem pode distorcer o resultado. Além disso, é fraco contra variações não lineares na intensidade entre os pixels correspondentes. Para tratar desses problemas, Bhat então sugeriu o uso de medidas ordinais para correspondência de imagens estéreo [9][10]. Tal técnica consiste em particionar uma imagem em $M \times N$ blocos de mesmo tamanho. Então para cada bloco, realiza-se a média de todas as suas intensidades, criando-se uma nova imagem de tamanho $M \times N$. Em seguida, a matriz é transformada em uma *rank matrix*. Esta consiste de uma matriz onde seus elementos indicam o índice do elemento da matriz original caso esta fosse ordenada de forma crescente. A Figura 2.1 ilustra bem essa situação, onde à esquerda está a subimagem e à direita está a *rank matrix*.

10	40	20
50	70	90
30	80	60

$$\begin{bmatrix} 1 & 4 & 2 \\ 5 & 7 & 9 \\ 3 & 8 & 6 \end{bmatrix}$$

Figura 2.1: Exemplo de subimagem e *rank matrix* [1].

2.2 Trabalho Base

No entanto, no caso de ataques irregulares como espelhamento vertical ou horizontal, a proposta de Bhat não reage bem. Para lidar com o problema, Kim [1] utiliza medidas ordinais em conjunto com a DCT-2D, onde, ao invés de realizar a comparação entre matrizes ranqueadas baseadas na imagem original, Kim [1] decidiu usar a comparação dos coeficientes AC da subimagem.

Isto é, a imagem é transformada para escala de cinza e então será dividida em 64 blocos (8×8) de tamanho igual e não-sobrepostos. A média das intensidades de cada bloco é calculada e armazenada em uma nova imagem de tamanho 8×8. Depois, executa a Transformada Discreta do Cosseno na imagem resultante e aplica-se a medida ordinal

dos coeficientes AC, ou seja, gera-se uma *rank matrix* 1×63 a partir dos coeficientes AC. Dessa matriz, extrai-se uma matriz de tamanho 1×35. Esta será a assinatura de uma imagem. Para compará-las, basta usar a norma L1. Para normalizar a distância calculada pela norma L1, usa-se o *Maximum a Posteriori*.

2.3 Técnicas de Detecção de Cópias Baseadas em Conteúdo que utilizam Transformadas Discretas do Cosseno

2.3.1 Transformada Discreta do Cosseno

A *Discrete Cosine Transform* (DCT) é uma ferramenta bastante utilizada em processamento de sinais. Similar à Transformada de Fourier, a DCT consegue expressar uma sequência infinita de pontos em termos de uma soma de cosenos, mas opera somente no domínio dos números reais. Devido a este projeto ser relacionado a imagens e estas poderem ser consideradas funções discretas bidimensionais, será usado um subtipo: Transformada Discreta do Cosseno Bidimensional (DCT-2D). Uma das vantagens da DCT é ser bastante resistente a transformações na imagem. Devido a essas características, transformações como borramento, compressão, etc. tendem a afetar detalhes, isto é, componentes de alta frequência da imagem, e estas mudanças podem ser facilmente observadas e quantizadas e, portanto, é natural gerar métricas de similaridades a partir delas. Inclusive, segundo Skrepth *et al* [11], algoritmo baseados em DCT extraem características melhores que outras técnicas no domínio espacial e *Wavelet*.

A Equação 2.2 mostra a fórmula da DCT de duas dimensões em uma matriz de dimensões $N_1 \times N_2$, onde X_{k_1,k_2} representa o pixel da imagem resultante na posição (k_1, k_2) e x_{n_1,n_2} representa o pixel da imagem original na posição (n_1, n_2).

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cos\left[\frac{\pi}{N_1}(n_1 + \frac{1}{2})k_1\right] \cos\left[\frac{\pi}{N_2}(n_2 + \frac{1}{2})k_2\right] \tag{2.2}$$

Sendo X a imagem original transformada, pode-se chamar cada elemento de X de coeficiente. O elemento no canto superior esquerdo, ou seja, o elemento na posição (0,0) de X é referido como coeficiente DC e possui a maior quantidade de energia da imagem original. Os outros coeficientes são chamados de coeficientes AC. São eles que possuem informações detalhadas sobre a imagem e serão usados neste projeto.

2.3.2 Propostas

Um dos problemas que o método de Kim [1] possuía era a inabilidade de lidar com rotações. Este método consegue detectar cópias com rotações de até $\pm 1°$, porém mais do que isso era limitado às características da imagem, isto é, se a imagem com rotação era muito similar à original. Para lidar essa falha, em 2005, Wu *et al* [12] propuseram um método que ao invés de realizar uma divisão em blocos quadrados, a divisão em blocos seria através de elipses concêntricas. Entretanto, foi notado que a eficiência de tal método caía significativamente a partir dos 15°de rotação. Então em 2009, Zou *et al* [13] propuseram o uso de cículos concêntricos ao invés de elipses. Porém, existem dois tipos de rotação:

- Rotação que corta pedaços dela
- Rotação que preenche os espaços sobrando com a cor preta

Nesse contexto, o método em [13] solucionava bem o primeiro caso, mas não tão bem o segundo. Logo, Li *et al* [14], propuseram, em 2011, uma alternativa que, como incremento, pré-processa a imagem através da normalização da mesma, conseguindo reter mais informações.

Em 2012, Baaziz *et al* [15] apresentaram um método que combinava o método de Kim [1] com outro fator existente nos coeficientes AC da DCT: o sinal. Para isso, realiza-se a solução em [1] até a parte da extração da subimagem transformada. Desse ponto, ocorrem 2 caminhos em paralelo sobre esta subimagem: a extração de 35 coeficientes AC a partir da matriz 6×6 superior esquerda e a extração do sinal dos 35 coeficientes AC em zigue-zague a partir da posição (0,0). Em seguida, esses coeficientes são usados para a realização de uma medida de similaridade.

Zou *et al* [16] propuseram outro método com o intuito de melhorar a técnica de [1]. Seu método consistia em usar informações sobre as bordas das imagens. Entretanto, como alterações em imagens facilmente afetam as bordas, os autores propuseram usar um conjunto de cópias da imagem original, passando-as pelo algoritmo de [1], até a extração da *rank matrix* e usando *k-means cluster* para reduzir o número de comparações. Em seguida, retiram-se as informações de borda usando *Wavelet* de *Haar*. Usa-se, então, todas essas informações como auxílio na detecção de cópias.

Em 2014, Tang *et al* [17] propuseram um *hash* para imagens, sendo que um de seus usos era a detecção de cópias. Tal técnica normaliza a imagem, divide-a em blocos e transforma cada bloco para o domínio da frequência usando DCT. Em seguida, são extraídos os coeficientes DCT dominantes da primeira linha/coluna de cada bloco para a construção de matrizes de características. Depois, é realizada a compressão de matriz através do

cálculo e quantificação das distâncias das colunas. Com isso, produz-se um *hash* da imagem. Para o cálculo da distância entre os dois *hashes*, usa-se a norma L1.

Em 2016, Tang *et al* [18] apresentaram um método de *hash*, onde a imagem é normalizada e, a partir desta, ângulos de vetores de cores são extraídos. A partir deles, usa-se a DCT para extrair uma matriz de características estável. Então, *Local Linear Embedding* (LLE) é usado na matriz de características para a redução de dados, sendo que variações nos resultados da LLE são quantizado e encriptados de forma a contruir o *hash*. Novamente, usa-se a norma L1 para fins de comparação.

2.4 Trabalhos Correlatos

Em 2009, Kang *et al* [19] propuseram uma técnica baseada em *Single Value Decomposition* (SVD) e divisão em blocos, com o intuito de aumentar a robustez e a discriminabilidade de algoritmos de detecção de cópias baseados em conteúdo. Tal técnica consistia em converter a imagem RGB para o espaço de cor YUV, extrair a componente Y, dividí-la em $N \times N$ blocos, aplicar SVD para cada um desses blocos e o maior valor singular de cada bloco era extraído para uma matriz D_{LSV}, sendo então passado pela SVD novamente e transformado em um vetor de tamanho N, para, em seguida, ser normalizado e usado para comparação de similaridades.

Em 2011, Arnia *et al* [20] apresentaram um algoritmo que usa os coeficientes da transformada *Wavelet*. Este consiste em transformar a imagem para o domínio da frequência através da transformada *Wavelet* e realizar a comparação entre ambas as imagens através da equiparação dos sinais dos coeficientes *Wavelet* das imagens. Foram realizados experimentos usando um e dois níveis de decomposição e várias combinações de sub-bandas.

Em 2012, Zhou *et al* [21] propuseram um método que usa a Transformada de Fourier-Mellin. Para isso, converte-se a imagem em uma nova imagem em escala de cinza e de tamanho 256×256. Uma vez tendo-se a imagem pré-processada, os coeficientes são extraídos a partir da Transformada de Fourier-Mellin. Para a identificação de cópias, são usadas as proporções entre coeficientes adjacentes.

Em 2014, Zhou *et al* [22] apresentaram um procedimento que combina técnicas de detecção local e detecção global. A razão disso é que assinaturas globais são sensíveis à rotação, corte, entre outras transformações geométricas, já assinaturas locais não conseguem um bom índice de discriminação de cópias. Para lidar com esse problema, primeiramente, pontos de interesse são extraídos da imagem através do detector de Hessian-Affine. Depois, a imagem é dividida em círculos, distribuindo os pontos de interesse entre os círculos. Para combinar as vantagens do método global baseado em círculos com os pontos

de interesse, as características das distribuições desses pontos de interesse são usadas para se criar uma assinatura para aquela imagem.

Em 2015, Zou *et al* [23] propuseram uma técnica de *hashing* baseado em kernel de múltiplas características. O propósito desta é usar um kernel de múltiplas características para converter assinaturas reais em assinaturas binárias e, durante esse processo, mapear as características importantes da imagem em questão nessa assinatura binária.

Hsieh *et al* [24] apresentaram uma técnica de *hashing*, onde, primeiramente, pré-processa-se a imagem reduzindo-a para o tamanho 256×256, convertendo-a para escala de cinza e passando-a pelo algoritmo de *Image Secret Sharing* (ISS) [25]. Depois, extrai-se um vetor de características e a partir dele contrói-se um *hashing* de múltiplas tabelas.

Em 2016, Yan *et al* [26] decidem focar-se na performance de algoritmos de extração de características. Para isso, eles usam um *hashing* baseado em *Machine Learning* para que se consiga produzir uma rápida agregação de características.

Baber *et al* [27] apresentam uma solução para lidar com os principais desafios de descritores de características baseados em *patch*: ambiguidade e distinção insuficiente. Para isso, Baber usa textura e gradientes na vizinhança destas características observadas para influenciar seus respectivos valores.

Capítulo 3

Metodologia Proposta

Neste capítulo, será dada uma visão geral da solução proposta na metodologia, assim como uma comparação da solução escolhida com a apresentada em [1]. Uma vez que muitos parâmetros do algoritmo foram escolhidos através de testes, então o Capítulo 4 descreverá melhor o desenvolvimentos dos testes e o Capítulo 5 mostrará os resultados destes testes e quais valores foram escolhidos como parâmetros para compor o algoritmo.

3.1 Descrição do problema

Dados os conjuntos O, que representa o conjunto de imagens originais, T, que representa o conjunto de imagens a serem testadas e C, que representa o conjunto de imagens consideradas cópias, sendo que $C \subseteq T$, deseja-se um algoritmo `A` que itere sobre toda imagem $\mathbf{O_i} \in O, \forall i = 1 \ldots n$ e $\mathbf{T_j} \in T, \forall j = 1 \ldots m$, e encontre toda imagem $\mathbf{C_k} \in C, \forall k = 1 \ldots q$.

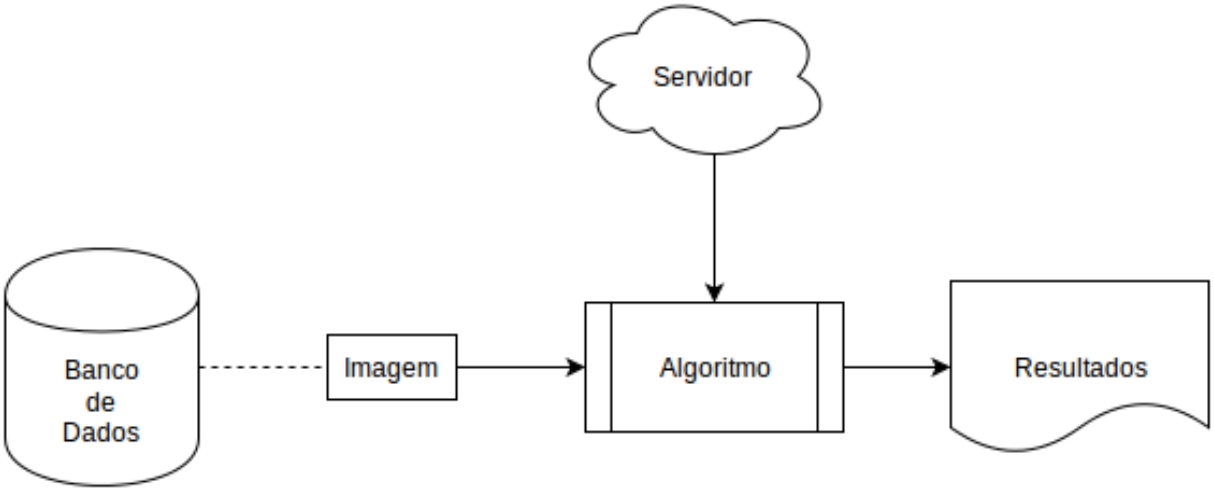


Figura 3.1: Estrutura de um sistema de detecção de cópias.

Conforme a Figura 3.1 ilustra, Banco de Dados é a representação do conjunto O, a imagem representando uma imagem $\mathbf{O_i}$, o servidor representando T e o Resultado contendo o conjunto C.

O algoritmo `A` em questão pode ser definido por `ALG`(O, T, C). Ainda, dados um algoritmo de pré-processamento PP(**I**,**I'**), um algoritmo de extração de características EC(**I**,**I'**), um algoritmo que mede similaridade MS(**I**,**I'**,s) e um conjunto H que contém todos os valores de similaridade possíveis que determinam se uma imagem é cópia da outra, pode-se definir `ALG` como:

```
   Data: O,T
   Result: C
 1 for i ← 1 to n do
 2 |   for j ← 1 to m do
 3 |   |   O'← PP(O_i);
 4 |   |   T'← PP(T_j);
 5 |   |   O"← EC(O');
 6 |   |   T"← EC(T');
 7 |   |   s← MS(O",T");
 8 |   |   if s ∈ H then
 9 |   |   |   insert T_j into C
10 |   |   end
11 |   end
12 end
```

Algoritmo 1: Definição do algoritmo `ALG`(O, T, C)

A Figura 3.2 representa uma versão genérica do que o Algoritmo da Figura 3.1 e `ALG` deve fazer. Por ser apenas uma versão genérica, consegue ilustrar bem o que é desejado de um algoritmo CBCD, mas não deve ser considerado como forma obrigatória.

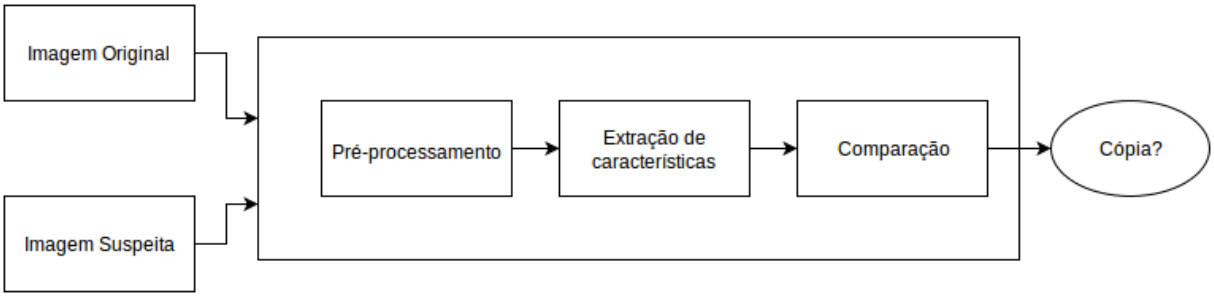


Figura 3.2: Visão do diagrama de detecção de cópia.

3.2 Proposta

Devido ao fato de que o algoritmo proposto neste projeto é baseado na metodologia de [1], portanto ainda será mantido o esqueleto da solução original. Então, dadas uma imagem original $\mathbf{O_1}$ e uma imagem de teste $\mathbf{T_1}$:

1. Ambas serão convertidas para imagens em escala de cinza($\mathbf{G_O}$, $\mathbf{G_T}$).
2. Será extraída de cada uma, subimagem de tamanho $N \times N$($\mathbf{S_O}$, $\mathbf{S_T}$) através da divisão de $\mathbf{G_O}$, $\mathbf{G_T}$ em blocos de tamanhos iguais e não sobrepostos.
3. Então, aplica-se a DCT em $\mathbf{S_O}$ e $\mathbf{S_T}$, resultando em uma nova imagem de tamanho $N \times N$($\mathbf{X_O}$ e $\mathbf{X_T}$).
4. Os $\mathbf{K_O}$ e $\mathbf{K_T}$ primeiros coeficientes AC de X_O e X_T serão, então, extraídos, sendo $\mathbf{K_O}$,$\mathbf{K_T} \leq N^2 - 1$
5. Usando-se os coeficientes $\mathbf{K_O}$ e $\mathbf{K_T}$, calcula-se a distância D entre essas duas imagens usando uma norma L1.

$$D = \sum_{a=0}^{N^2-1} \|\mathbf{K_{O_a}} - \mathbf{K_{T_a}}\| \tag{3.1}$$

6. D é então normalizada, gerando d, tornando-se um valor mais restrito e, portanto, facilitando a comparação entre estes.
7. Dado um *threshold* h, se $d < h$, então $\mathbf{O_1}$ e $\mathbf{T_1}$ serão próximas o suficientes para serem declaradas cópia uma da outra.

A Figura 3.3 demonstra de forma geral o esqueleto dos algoritmos. Porém, mesmo seguindo essas diretrizes, existem certos pontos onde ambas as propostas se assemelham e outros pontos em que se diferem.

Na parte de divisão em blocos da imagem, ambos os algoritmos permaneceram bastante semelhantes. Isto é, para a construção da subimagem, $\mathbf{G}$ será dividida em $N \times N$ blocos de tamanhos iguais. Sendo $S_{i,j,k}$ a posição (j,k) da subimagem S_i e $B_{i,j,k}$ o bloco na posição (j,k) na imagem G_i, então o valor de $S_{i,j,k}$ será a média aritmética dos elementos de $B_{i,j,k}$. A diferença é que [1] utiliza $\mathbf{N}$ igual a 8. Neste projeto, resolveu-se testar novos valores de $\mathbf{N}$, com o intuito de melhorar os resultados.

Quanto à parte de extração de componentes AC, o conceito geral permanece o mesmo. Mas existem possíveis pontos de análise para uma futura melhora: o valor de K, quais coeficientes extrair e se haverá algum processamento neles. [1] extrai uma matriz quadrada de ordem $M < N$ a partir do canto superior esquerdo da subimagem transformada

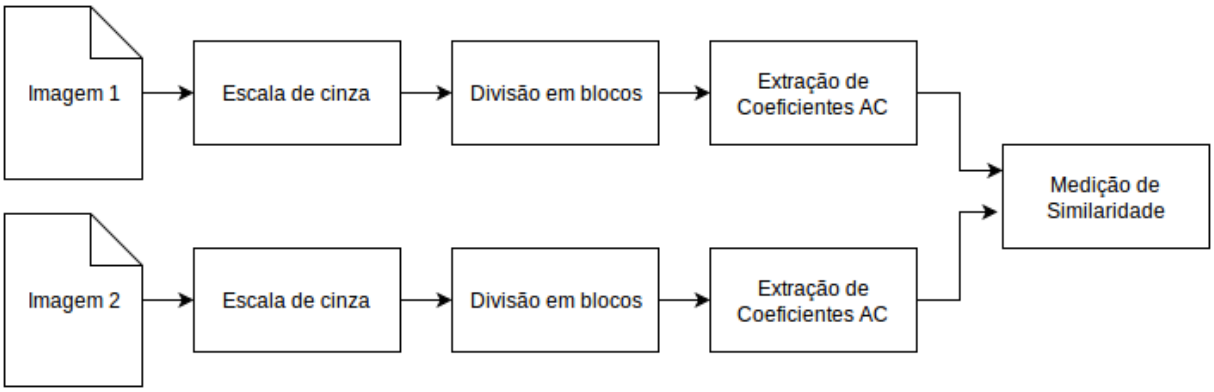


Figura 3.3: Fluxo do algoritmo no caso de comparação entre duas imagens.

e define $K = 35$, pois $M = 6$ apresentou os melhores resultados. Neste projeto apresentamos uma alternativa a ser testada, onde, simplesmente, iteramos linearmente sobre a matriz, para ver como se compara a solução de [1]. Quanto a parte de processamento dos coeficientes AC, foram testadas as possibilidades de usá-los de forma pura, com valores absolutos, ordenados ou ambos. O Capítulo 4 oferecerá mais detalhes.

A parte onde ocorre a medição da similaridade entre duas imagens é a que mais difere da solução-base. Primeiramente, [1] usa uma *rank matrix* para efeito de comparação, já este projeto oferece uma solução utilizando diretamente os valores dos coeficientes. Outro fator é que enquanto ambos usam a norma L1, uma vez que esta é mais robusta para *outliers* (dados que não obedecem o comportamento da maioria) do que a norma L2 [28], para medir as distâncias, o método de normalização é completamente diferente. [1] usa a técnica de *Maximum a Posteriori* (MAP), ou seja, dadas uma imagem original o e um conjunto $\mathbf{T}$ com imagens para testes, Δ o conjunto de todas as distâncias entre os vetores contendo os componentes AC de uma imagem $T_i \in \mathbf{T}, \forall i = 1 \ldots n$ e e os componentes AC de o, então a medida de similaridade $\mathbf{d}$ será a norma L1 entre esses vetores em o e e em T_i, representada por D, dividida pelo maior valor em Δ, isto é:

$$d = \frac{D}{max(\Delta)} \tag{3.2}$$

Entretanto, tal solução fica totalmente dependente do banco de dados ao qual foi aplicada, caso este possua uma pequena quantidade de imagens ou contenha nenhuma cópia, a chance de apresentar erros é muito grande. Para isso, esse projeto propõe uma nova forma de realizar essa normalização, que resultou de uma tentativa acidental de reproduzir o código de [1] e produzia bons resultados. A nova forma consiste em dividir D pela norma L1 entre o vetor Θ de componentes AC de o e o vetor invertido τ de componentes AC de T_i:

$$d = \frac{D}{\sum_{a=1}^{r} \|\Theta_a - \tau_a\|} \tag{3.3}$$

Quanto ao *threshold*, [1] propôs que $h = 0.37$ como o melhor valor de *threshold*. Mas, uma vez tendo novo método de medição de similaridade, então um novo h precisa ser estimado.

Capítulo 4

Desenvolvimento da Aplicação

Neste capítulo, defini-se quais materiais foram utilizados, como por exemplo, tecnologias e base de dados para os testes. Além disso, é apresentado o passo-a-passo dos testes que levam à definição dos parâmetros do algoritmo e do teste final. Os resultados destes testes são apresentados no Capítulo 5.

4.1 Materiais

O projeto foi escrito na linguagem C++ utilizando a biblioteca OpenCV [29]. Foram usados três banco de dados de imagens para a realização da calibração: UCID.v2 [2], RAISE [3], Copydays [30]. Exemplos de imagens de cada *dataset* se encontram no Anexo I.

4.1.1 OpenCV

Open Source Computer Vision (OpenCV) [29] é uma biblioteca que contém um enorme acervo de algoritmos de visão computacional. Por ser liberada sob uma licença BSD, é gratuita, *open-source* e pode até ser incluída em soluções proprietárias.

Escrita em C/C++ e com suporte à arquitetura *multicore*, visa eficiência computacional com foco em aplicações em tempo real. Além disso, possui interface com as linguagens C++, C, Python e Java, e com as plataformas Windows, Linux, Mac OS, iOS e Android.

Ainda, o OpenCV apresenta integração com ferramentas de terceiros para ajudar no desenvolvimento de aplicações de visão computacional. São exemplos dessas bibliotecas: Tesseract, para a realização de reconhecimento de caracteres ópticos, e *Open Computing Language* (OpenCL) [31] e *Compute Unified Device Architecture* (CUDA) [32] para aproveitamento de tecnologias *manycores*.

Devido a todas essas características, OpenCV e C++ foram escolhidas como ferramentas de desenvolvimento neste trabalho.

4.1.2 Bancos de Dados

UCID.v2

Uncompressed Colour Image Database (UCID) [2], pronunciado "*use it*", é um banco de dados de imagens cuja preocupação é atender todos os requisitos de um banco de dados voltado a CBIR. Atualmente, UCID está na versão dois e possui 1338 imagens não compactadas em formato TIF.

Deste banco de dados foram usadas todas as 1338 imagens para realização dos experimentos. O diferencial deste banco de imagens em relação aos outros é possuir a maior quantidade de imagens a serem comparadas e, por consequência, há uma maior chance do algoritmo ser confundido. Por suas imagens serem pequenas, com tamanho médio de 130*kilobytes*, possibilita os testes, já que as iterações são mais rápidas.

Outro ponto interessante é o foco em algoritmos de CBIR. As imagens apresentam similaridades umas com as outras, o que é um ponto positivo, pois pode-se testar com mais precisão o aparecimento de falsos positivos.

(a) Compressão (b) Original (c) Rotação

Figura 4.1: Exemplos de imagens usadas em UCID.v2 [2].

RAISE

RAw ImageS datasEt (RAISE) [3] é um banco de dados de 8156 imagens em alta resolução sem compressão que ilustram vários temas e cenários. Voltado à pesquisa forense, categoriza as imagens de acordo com o conteúdo. Uma imagem está contida em pelo menos uma das categorias: *Outdoor*, *Indoor*, Paisagem, Natureza, Pessoas, Objetos, Construções.

Deste banco de dados, foram importadas 1000 imagens, abrangendo todas as categorias. Destas, 100 foram escolhidas aleatoriamente. Seu diferencial em relação aos outros *datasets* é que este possui imagens grandes, com média de 20 MB por imagem e dimensões na ordem do milhar. Justamente, por causa dessas características, foram escolhidas apenas 100 imagens.

(a) Compressão (b) Original (c) Rotação

Figura 4.2: Exemplos de imagens usadas em RAISE [3].

Copydays

O banco de dados Copydays [30] é um banco de imagens composto por fotos tiradas durante as férias dos próprios autores do *dataset* e respectivas cópias. Para a produção destas cópias, cada imagem original sofreu ataques de compressão, recorte e "*strong*". Então, tem-se:

Originais: Conjunto de imagem originais.

Cortadas: Imagens que foram cortadas, apresentando apenas parcialmente seu conteúdo. Corte varia de 10% a 80% da área da imagem original.

Redimensionamento + Compressão JPEG: Imagens foram redimensionadas para 1/16 do tamanho original e posteriormente comprimidas usando os fatores de qualidade: 3, 5, 8, 10, 15, 20, 30, 50, 75.

Strong attacks: 229 imagens que foram atacadas de diversas formas, não se limitando a apenas um ataque. Alguns exemplos de ataques são: *print and scan*, borramento, *paint*.

Foram utilizadas todas as imagens do banco de dados para o projeto. O que o difere dos outros bancos é que Copydays possui seu próprio conjunto de imagens cópias. Os outros *datasets* tiveram cópias artificiais a eles adicionadas, conforme apresentado na seção 4.2.

(a) Original (b) Corte (c) Ataque forte

Figura 4.3: Exemplos de ataques em Copydays [30].

4.2 Preparação do banco de dados

Para usar de forma eficiente os *datasets*, decidiu-se usar uma abordagem 7:3 para ajustes e testes, respectivamente. Isto é, 70% dos bancos de dados seriam usados para ajustar o algoritmo proposto e os 30% restante seriam utilizados para realização do teste da solução. As imagens contidas nos conjuntos de 70% e 30% foram escohidas aleatoriamente.

Durante a fase de ajustes, os três bancos foram usados isolados um do outro, por exemplo, para descobrir o melhor *threshold* o algoritmo que realizaria essa iteração seria executado separadamente para cada um dos bancos. Porém, a melhor solução que acatasse satisfatoriamente os conjuntos de imagens simultaneamente fora a escolhida.

Para a construção de um banco de dados que simulasse bem as dificuldades em um possível servidor contendo cópias de imagens, foram geradas cópias para todas as imagens dos bancos Ucid.v2 e RAISE. Não foram geradas para Copydays, pois este já possuía cópias previamente, exemplos no Anexo II). Foram geradas 46 cópias (exemplos dessas cópias usando `ucid00001.tif` se encontram no Anexo II):

Compressão: As imagens foram convertidas para o formato JPEG e comprimidas com fatores de qualidade de compressão: 10, 20, 30, 40, 50, 60, 80, 90, 100.

Redimensionamento: As imagens foram redimensionadas nas escalas de 0.5, 1, 1.5 e 2 em cada direção, por exemplo, uma das cópias é o *resize* de 2x na horizontal e 0.5x na vertical.

Rotação: As imagens foram rotacionadas nos ângulos: 45°, 90°, 135°, 180°, 225°, 270°, 315°.

Espelhamento: As imagens foram espelhadas tanto no eixo vertical quanto no eixo horizontal.

Equalização: Os histogramas das imagens foram equalizados.

Escala de cinza: As imagens foram convertidas para escala de cinza.

Filtro por média: Usando filtros de tamanho 3, 5, 7, 9, a imagem foi borrada usando um filtro de média aritmética.

Filtro por mediana: O mesmo que acima, entretanto usando um filtro de mediana.

***Sharpening*:** Utilizando de um filtro Laplaciano de tamanho 3×3 com o valor do centro igual a 8 e o valor do restante das células do filtro iguais a -1, soma-se o resultado da imagem filtrada com a imagem original.

4.3 Divisão de blocos

Conforme mencionado anteriormente no Capítulo 3, um dos possíveis pontos de melhoria é justamente testar novos valores de N, para a geração de uma subimagem $N \times N$. Como foi dito em [1], os blocos devem possuir tamanhos iguais e não devem se sobrepor, mas não foi apresentado nenhuma solução para lidar com o problema que aparece ao processar uma imagem cujas dimensões não são múltiplas de N, portanto, apesar de ser uma falha, decidiu-se cortar partes das imagens que não cabiam na subimagem. Para testar novos tamanhos, foram usados valores de 4 a 32, inclusive, mas por causa de uma limitação no funcionamento da função `dct()` do OpenCV, foram testados apenas valores pares.

Uma vez que os outros parâmetros não haviam sido testados, resolveu-se usar valores padrões para eles. Nesta fase do ajuste, seriam usados todos os coeficientes AC sem qualquer alteração e um valor de *threshold* igual a 0.5.

4.4 Extração de componentes AC

Outro ponto de possível melhoria identificado foi a quantidade K de coeficientes AC. A quantidade de coeficientes AC é importante, pois se K for muito pequeno, o algoritmo

poderia produzir uma quantidade maior de falsos positivos. Por outro lado, caso K for muito grande, a quantidade de detalhes seria maior e, portanto, mais fácil dos ataques à imagem sucederem, aumentando a probabilidade de falsos negativos serem gerados.

Além disso, dado que $K \leq N^2 - 1$, surge, então, a questão de quais coeficientes serão coletados. Dois métodos se apresentam nessa questão: extrair uma matriz quadrada de ordem M a partir do campo superior direito da imagem transformada e a iteração linear desta imagem, ou seja, considerando a imagem como uma matriz, itera ela de forma sequêncial, lendo todas as colunas de todas as linhas.

Ainda, há a questão de que se deve ou não ser usado algum processamento sobre o vetor contendo os coeficientes extraídos. Os processamentos escolhidos foram:

Puro: Os valores não sofrem alteração alguma, isto é, o vetor é usado da mesma forma em que foi produzido.

Valores absolutos: Os valores são colocados em forma absoluta. Dessa forma, o que importa é a magnitude do coeficiente.

Ordenado: Os valores dos coeficientes AC são ordenados de forma decrescente, ou melhor, quanto maior o valor mais no ínicio do vetor eles deverão estar. O objetivo foi simular o que acontece no algoritmo de compressão de imagens JPEG, onde a matriz é organizada em um vetor usando um percorrimento em *zig-zag* da matriz de coeficientes e, às vezes, com pequenas alterações de rotas para extrair a maior quantidade possível de energia da imagem. A ideia é colocar os maiores coeficientes AC de cada imagem para serem comparados, para ver se a energia era similar.

Valores absolutos e ordenados: Os valores são colocados em forma absoluta e, posteriomente, são ordenados em ordem decrescente. O intuito é mesclar as intenções das alternativas supracitadas.

Uma vez que todos esses pontos de melhorias estão interligados, ou seja, possuem uma alta dependência um do outro, no momento dos ajustes, estes são feitos em conjunto, através de combinação dos valores já citados. Novamente, o valor do *threshold* não havia sido decidido, portanto usou-se $h = 0.5$ para julgar os resultados.

4.5 Definição de *threshold*

Visto que mudanças ocorreram em vários trechos do código, então o *threshold* também deverá ser alterado, já que esse fator é o que determina se deve considerar uma imagem cópia ou não. Para descobrir o novo valor, realizam-se testes com valores variando de 0.05 a 0.5, com passo de 0.01.

4.6 Teste

Finalmente, o algoritmo, com seus novos parâmetros já coletados, será posto à prova contra os 30% finais de cada banco de dados. Para efeito de comparação, o algoritmo proposto em [1] é implementado com todos os valores propostos e será executado sobre a mesma base. Não ocorreu novo treinamento para a definição de novo *threshold*, pois a acredita-se que a solução do artigo de Kim [1] seria uma solução para ser usada em qualquer outro banco.

Conforme será mostrado no Capítulo 5, será usada uma métrica para quantificar o resultado e também será mostrado quais tipos de cópias foram resgatadas em média em cada banco.

Capítulo 5

Resultados

Neste capítulo, é discutida a métrica escolhida para medir a performance dos valores testados. Depois, serão mostrados os resultados de cada teste de ajuste realizado, bem como a definição de quais parâmetros são usados para compor o algoritmo. Mostra-se, assim, o resultado final para os 30% restante dos *datasets*, conforme discutido na seção 4.2.

5.1 Métricas

Para calcular o rendimento da solução proposta, serão usadas algumas métricas comuns nessa área, especificamente a *F1-Score* apresentada em [33]. Mas antes de definir quais métricas serão usadas, é necessário um conhecimento prévio de nomenclatura:

Positivo (P): Conjunto contendo elementos que o algoritmo definiu que se encontravam dentro dos parâmetros para um dado propósito.

Negativo (N): Conjunto contendo elementos que o algoritmo definiu que não se encontravam dentro dos parâmetros para um dado propósito.

Verdadeiro positivo (TP): Esse conjunto representa todos os elementos que a solução definiu como positivos e sabe-se que estes realmente o são.

Falso positivo (FP): Esse conjunto representa todos os elementos que a solução definiu como positivos, porém sabe-se que estes não o são.

Verdadeiro negativo (TN): Esse conjunto representa todos os elementos que a solução definiu como negativos e sabe-se que estes realmente o são.

Falso negativo (FN): Esse conjunto representa todos os elementos que a solução definiu como negativos e sabe-se que estes não o são.

No caso foi decidido utilizar a *F1-Score*. Esta é baseada em duas outra métricas que também serão mostradas neste capítulo: *Precision* e *Recall*.

Precision pode ser definido como a proporção de elementos positivos que são relevantes à solução, ou seja,

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

Já *Recall* é a proporção de elementos de elementos relevantes extraídos e que foram bem sucedidos,

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

Agora que *Precision* e *Recall* foram definidas, pode-se definir *F1-Score*. Esta é uma métrica bastante conhecida, capaz de medir a acurácia de um teste. Seu cálculo pode ser dado pela média harmônica entre *Precision* e *Recall*, conforme mostra a equação 5.3:

$$F1 - Score = 2 \cdot \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{5.3}$$

5.2 Ajustes

Usando 70% das três bases de dados utilizadas, é realizado o ajuste dos parâmetros: quantidade de blocos, coefiecientes AC e *threshold*. Os gráficos contendo os *F1-Scores* serão amostrados a seguir, porém os gráficos contendo *Precision* e *Recall* estarão no Anexo III.

5.2.1 Divisão de blocos

A Figura 5.1 mostra os valores observados ao se usar matrizes quadradas de ordem N, sendo $N = 4 \ldots 32$ para a geração da subimagem. Percebe-se que utilizar $N < 8$ produz resultados ruins em relação aos outros valores. Outro ponto a ser observado é que para o banco de dados Copydays, o melhor resultado acontece em $N = 10$, em compensação, em UCID.v2, os resultados começam a se estabilizar a partir de $N = 14$. Por isso, para melhor atender a todos os conjuntos de imagens, decidiu que um bom valor seria usar $N = 12$ para a criação da subimagem, i.e., a imagem original será convertida em uma imagem de tamanho 12×12.

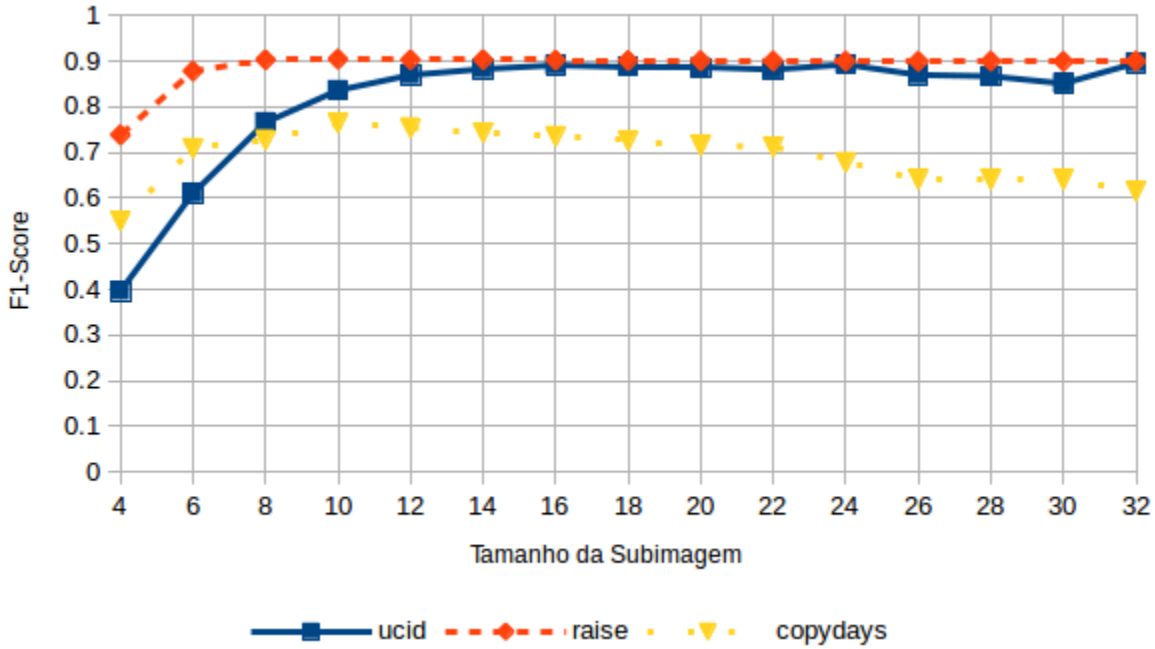


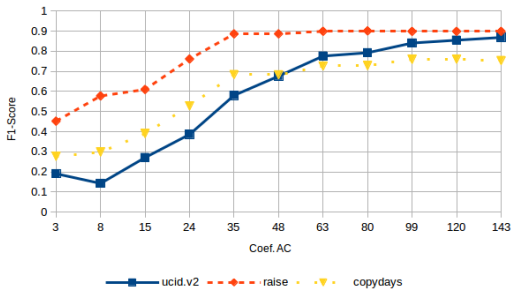
Figura 5.1: *F1-Scores* com a variação do tamanho da subimagem.

5.2.2 Extração de componentes AC

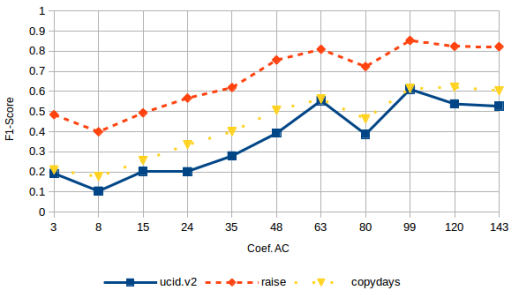
As Figuras 5.2 mostram os *F1-Scores* dos ajustes durante a extração dos coeficientes AC, onde Próprio representa percorrimento proposto, **Kim** o percorrimento da técnica descrita em [1], isto é, a extração da matriz quadrada superior esquerda de ordem N, **Puro** extração de coeficientes AC sem alterações, **Absoluto** extração das magnitudes dos coeficientes AC, **Ordenado** extração de coeficientes AC em ordem decrescente e **Ambos**, extração das magnitudes em ordem decrescente.

Pelos resultados apresentados nas Figuras 5.2, percebe-se que ordenar os coeficientes AC só piora o desempenho do algoritmo, o que revela que a posição dos coeficientes é uma informação essencial de uma imagem. Já os valores absolutos apresentam resultados razoáveis utilizando uma grande quantidade de coeficientes. Porém, esses valores caem bruscamente em UCID.v2. O melhor resultado para os três *datasets* foi o de usar 143 coeficientes AC de forma pura, isto é, sem qualquer alteração.

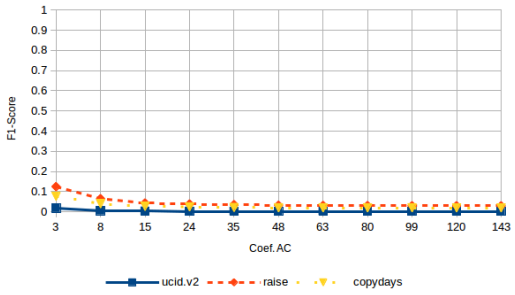
Uma vez que estão sendo usado todos os coeficientes, o método proposto por [1] e por este projeto para a forma de coleta dos coeficientes é idêntica. Portanto, foi usada a forma mais eficiente para coletá-los na devida ordem.



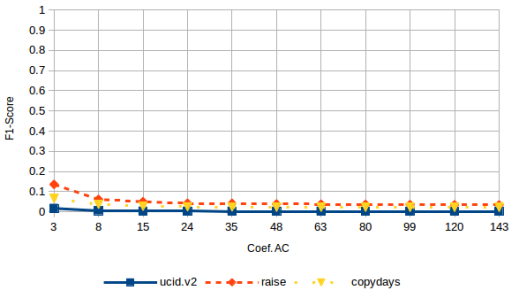
(a) Próprio/Puro



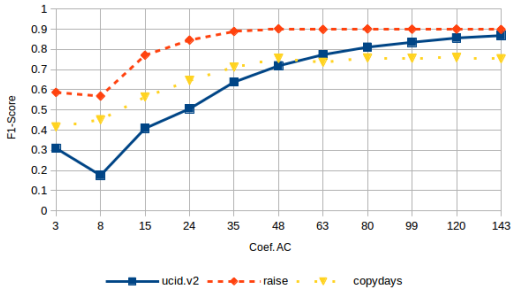
(b) Próprio/Absoluto



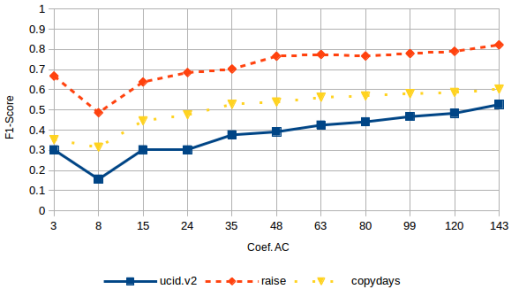
(c) Próprio/Ordenado



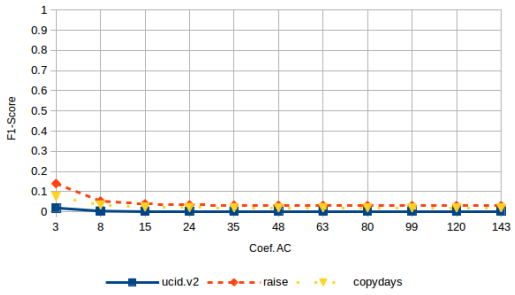
(d) Próprio/Ambos



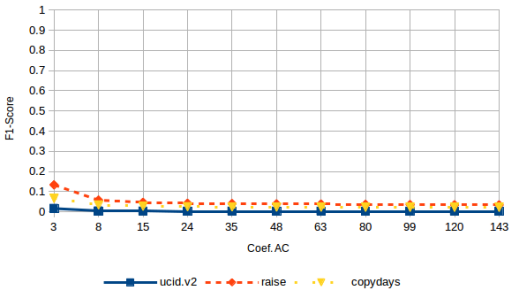
(e) Kim/Puro



(f) Kim/Absoluto



(g) Kim/Ordenado



(h) Kim/Ambos

Figura 5.2: *F1-Scores* dos ajustes durante a extração dos coeficientes AC.

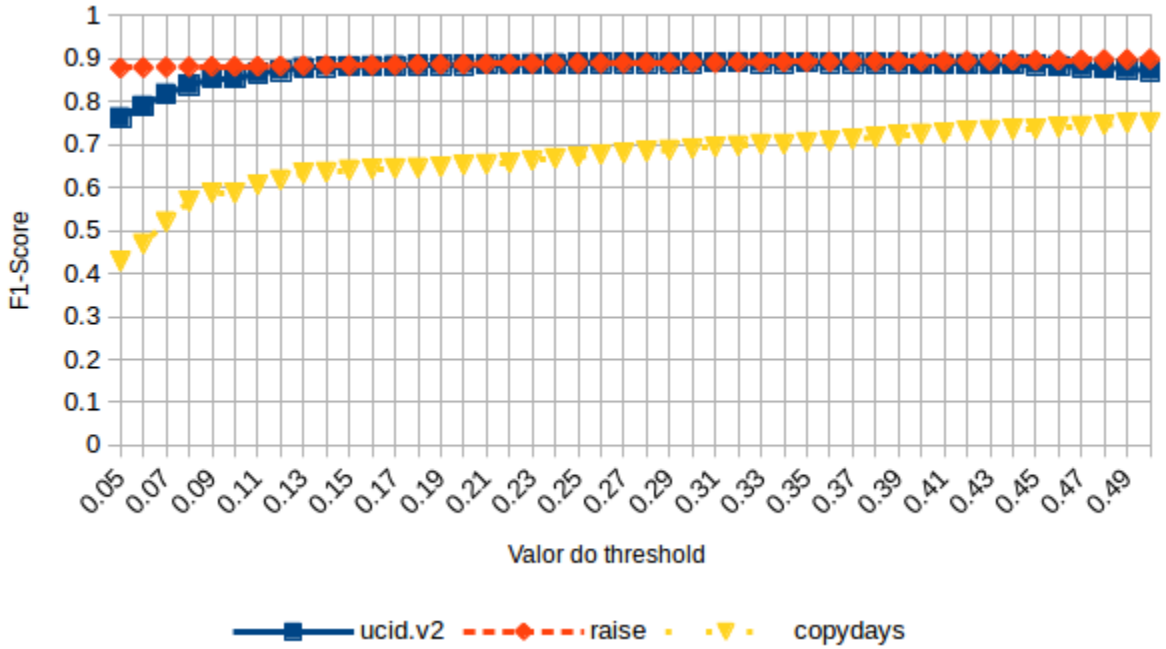


Figura 5.3: *F1-Scores* da variação do valor do *threshold*.

5.2.3 Determinação de *threshold*

A Figura 5.3 mostra os *F1-Scores* ao ajustar os valores de *threshold* da solução. Vê-se que em Copydays e RAISE, este de forma menos brusca, o aumento do *threshold* aumenta o *F1-Score*, entretanto, estes possuem muito menos imagens que UCID.v2 e, portanto, até mesmo um décimo de acréscimo no valor do *threshold* é capaz de adicionar um novo elemento aos positivos com alta probabilidade de ser um verdadeiro positivo. Por isso, decidiu-se usar o UCID.v2 como referência nesse caso. Este demonstra uma queda lenta, mas gradual a partir de $threshold > 0.36$, logo a partir desse ponto, incrementar o valor apenas afeta negativamente a solução. Utilizando os valores de *F1-Scores* com até 5 casas decimais obtidos durantes os ajustes, percebeu-se que um bom valor que atende bem a UCID.v2 e RAISE e, razoavelmente, a Copydays seria $threshold = 0.32$.

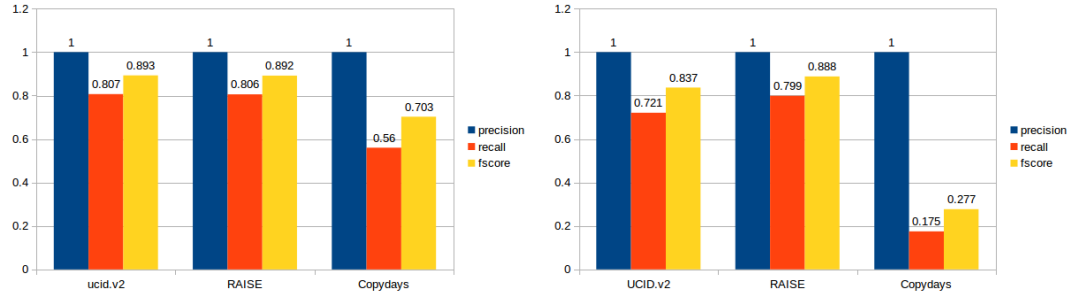
5.3 Testes

Aqui são apresentados os resultados do algoritmo proposto, já com os parâmetros determinados anteriormente utilizando os 30% restantes de todos bancos de imagens. A Figura 5.4 mostra que UCID.v2 e RAISE possuem resultados similares e bons, com *F1-score* quase em 0.9). Observa-se que de forma geral, a solução proposta, embora não por muito,

superou a solução de [1]. RAISE possui o resultado mais próximo, mas acredita-se que seja devido ao fato do banco ser pequeno.

Já Copydays possui um resultado insatisfatório, mas esperado, uma vez que em UCID.v2 e RAISE as cópias foram produzidas especialmente para este projeto, enquanto Copydays já possuía sua própria coleção, sem mencionar o fato de que alguns destes ataques são muito fortes e acreditava-se que a solução não possuiria robustez o suficiente para aguentá-los, conforme foi demonstrado. No entanto, de forma inesperada, os resultados da metodologia proposta foram significativamente superiores à de [1].

Um fato interessante de ser notado é a *Precision*, que apresenta o valor 1 para todos eles. Isso significa que o método pode até capturar poucos verdadeiros positivos, mas todos os positivos eram verdadeiros positivos, isto é, não houve erro na acusação de uma ser cópia.



(a) Solução Proposta (b) Solução de [1]

Figura 5.4: Resultado dos testes nos 30% tanto da solução proposta quanto da solução de Kim [1].

Em seguida, foi feita uma análise da proporção de imagens encontradas corretamente de um certo tipo de cópia em relação à quantidade total de cópias deste respectivo tipo. A Tabela 5.2 demonstra que os resultados de ambos UCID.v2 e RAISE possuem resultados similares, com apenas algumas poucas diferenças. Ela também indica que houve um descobrimento de 100% das cópias de compressão, redimensionamento, escala de cinza e suavizadas com filtros passa-baixa de média e de mediana. Em compensação, uma das fraquezas principais de [1], a incapacidade de detector cópias com rotação, não foi corrigida ou sequer suavizada, demonstrada pelo 0% de detecção desse tipo de cópia nos resultados.

A Tabela 5.2 também mostra os resultados de [1] em relação às bases UCID.v2 e RAISE. Percebe-se então que houve melhoria na detecção de compressão, equalização de histograma, *sharpening* e filtros de média aritmética e mediana. Entretanto, a técnica de [1] foi superior na detecção de rotação de 180°, espelhamento vertical e horizontal. Nos outros tipos de cópias, os valores são os mesmos.

		Solução Proposta		Kim	
		UCID.v2	RAISE	UCID.v2	RAISE
Original		1	1	1	1
Compressão	10	1	1	0.502488	0.666667
	20	1	1	0.830846	0.866667
	30	1	1	0.927861	0.966667
	40	1	1	0.947761	1
	50	1	1	0.975124	1
	60	1	1	0.977612	1
	70	1	1	0.977612	1
	80	1	1	0.982587	1
	90	1	1	1	1
	100	1	1	1	1
Redimensionamento ($largura \times comprimento$)	0.5×0.5	1	1	1	1
	0.5×1	1	1	1	1
	0.5×1.5	1	1	1	1
	0.5×2	1	1	1	1
	1×0.5	1	1	1	1
	1×1	1	1	1	1
	1×1.5	1	1	1	1
	1×2	1	1	1	1
	1.5×0.5	1	1	1	1
	1.5×1	1	1	1	1
	1.5×1.5	1	1	1	1
	1.5×2	1	1	1	1
	2×0.5	1	1	1	1
	2×1	1	1	1	1
	2×1.5	1	1	1	1
	2×2	1	1	1	1
Rotação	45°	0	0	0	0
	90°	0	0	0	0
	135°	0	0	0	0
	180°	0	0	0.0124378	0.633333
	225°	0	0	0	0
	270°	0	0	0	0
	315°	0	0	0	0
Espelhamento	Horizontal	0.0199005	0	1	1
	Vertical	0	0	1	1
Equalização de histograma		0.915423	0.866667	0	0
Escala de cinza		1	1	1	1
Sharpening		0.987562	1	0.0174129	0.366667

Tabela 5.1: Resultado do algoritmo proposto e Kim por tipo de cópia produzida para os *datasets* UCID.v2 e RAISE.

Filtro de Média	3×3	1	1	0.982587	1
	5×5	1	1	0.945274	1
	7×7	1	1	0.855721	1
	9×9	1	1	0.733831	1
Filtro de Mediana	3×3	1	1	0.634328	0.933333
	5×5	1	1	0.325871	0.866667
	7×7	1	1	0.176617	0.733333
	9×9	1	1	0.0995025	0.533333

Tabela 5.2: Continuação: Resultado do algoritmo proposto e Kim por tipo de cópia produzida para os *datasets* UCID.v2 e RAISE.

Já a Tabela 5.3 apresenta os resultados de Copydays. Alguns resultados eram esperados, como por exemplo, o baixo índice nas cópias que possuem cortes e nos *strong attacks*, pois há muita perda ou adição de informação na imagem, algo que dificilmente será recuperado. Entretanto, o resultado da compressão não era esperado. Conforme foi observado nos outros bancos, era esperado pelo menos para compressões com fator de qualidade acima de 10, tivessem *ratio* 1. As imagens foram investigadas para procurar o que as distinguem do restante, mas não apresentam nenhuma característica desse tipo.

Ainda ocorre a mostra dos resultados de [1] em relação ao banco de dados de Copydays. Percebe-se que tal técnica não lida bem com cortes e, estranhamente, com a compressão, pois era esperado um resultado muito próximo dos encontrados na solução proposta neste projeto. Era esperado um resultado fraco nos *strong attacks*, mas foi inferior à solução proposta nesse projeto.

Copydays			
		Solução Proposta	Kim
Original		1	1
Corte	10	0.729167	0
	15	0.520833	0
	20	0.1875	0
	30	0.0416667	0
	40	0	0
	50	0	0
	60	0	0
	70	0	0
	80	0	0
Compressão	3	0.895833	0
	5	0.916667	0.0833333
	8	0.9375	0.208333
	10	0.9375	0.1875
	15	0.9375	0.291667
	20	0.9375	0.354167
	30	0.9375	0.354167
	50	0.9375	0.479167
	75	0.9375	0.4375
Ataques fortes		0.16	0.04

Tabela 5.3: Resultado do algoritmo proposto e Kim por tipo de cópia no *datasets* Copydays.

Capítulo 6

Conclusão

Um dos objetivos principais do método sugerido, que é propor um método capaz de analisar se 2 imagens são cópias, sem utilizar o MAP, foi alcançado. Quanto à parte de desempenho, os resultados apresentados em UCID.v2 e RAISE foram muito bons, uma vez que se aproximaram de um *F1-Score* igual a 0.9 e de forma geral foram superiores ao algoritmo de [1]. Já em Copydays, houve uma disparidade muito grande nos resultados encontrados, favorecendo a solução desenvolvida neste projeto. No entanto, apesar desse resultado ser vantajoso para a proposta deste projeto, os resultados foram ruins, uma vez que *F1-Score* abaixo de 0.75 é um resultado que indica a impropriedade de uma solução para um dado problema, isto é, para Copydays, a solução desenvolvida não serve. Quanto à robustez, ambas metodologias apresentaram resultados excelentes nos testes, uma vez que o valor de *Precision* é igual a 1, apontando com muita veracidade que uma imagem é cópia.

Os valores encontrados na separação de tipos de imagens, sugere que houve avanços em algumas classes (compressão, equalização de histograma, *sharpening* e filtros de média aritmética e mediana) de cópias, mas em outras houve regressão(rotação de 180°, espelhamento vertical e horizontal). Ainda, em UCID.v2 e RAISE, a técnica proposta conseguiu resultado perfeito em detecção de compressão, redimensionamento, filtros de média, filtros de mediana), mas não detectou cópias de rotação ou de espelhamento.

Quanto à Copydays, foi uma surpresa não haver perfeição na detecção de compressão. Estima-se que ou há disparidade na implementação das compressões usadas em OpenCV e Copydays ou as imagens possuem características que as tornam suscetíveis à perda de informação na compressão. Quanto aos *strong attacks*, os resultados foram insatisfatórios, mas foram os esperados, uma vez que mesmo para a visão humana é difícil observar similaridades.

6.1 Trabalhos futuros

Existem vários pontos de melhorias. O primeiro tange a questão dos resultados. Antes de tudo, os testes precisam ser melhorados, isto é, com bancos de dados maiores, com maior diversidade de originais e cópias. Além disso, cópias com rotações ainda são uma adversidade muito grande ao algoritmo proposto e para o algoritmo de [1]. Uma opção seria usar métodos mais modernos que tentam avançar nesse quesito, conforme mencionado no Capítulo 2. Outra opção seria usar técnicas diferentes, ou seja, utilizar outras características das imagens ou, até mesmo, utilizar técnicas de inteligência artificial para treinar o algoritmo para vários tipos de ataques. Por exemplo, poder-se-ia usar um algoritmo híbrido, utilizando DCT e outro método de extração de características e utilizar uma rede neural para atribuir pesos a certos coeficientes AC.

Outro ponto de melhora é a questão de desempenho. Não foi o foco principal e foi pouco abordada neste projeto, mas ainda é um fator fundamental em um algoritmo de CBCD. Há, por exemplo, a possibilidade de preparar o algoritmo para suportar o uso de *threads* e/ou *Graphics Processing Unit* (GPU) para cálculos em paralelo.

Este é um campo vasto, que caminha a passos largos, bastante estimulante e com muito espaço para contribuições. Percebe-se que ainda existe muito a ser feito nesta área e espera-se que esse projeto tenha contribuído pelo menos um pouco no avanço desta.

Referências

[1] Kim, Changick: *Content-based image copy detection*. Em *Signal Processing: Image Communication 18*, páginas 169–184, Palo Alto, CA 94304, USA, 2003. ix, 4, 5, 7, 10, 12, 13, 14, 19, 21, 24, 27, 29, 31, 32

[2] Schaefer, Gerald e Michal Stich: *UCID - an uncompressed colour image database*. Em *Storage and Retrieval Methods and Applications for Multimedia*, volume 5307, páginas 472–480, 2004. ix, 15, 16

[3] Dang-Nguyen, Duc Tien, Cecilia Pasquini, Valentina Conotter e Giulia Boato: *RAISE - a raw images dataset for digital image forensics*. Em *Storage and Retrieval Methods and Applications for Multimedia*, páginas 219–224, 2004. ix, 15, 16, 17

[4] *Tecnologia ajuda a polícia a combater a pornografia infantil na internet*. http://g1.globo.com/fantastico/noticia/2016/09/tecnologia-ajuda-policia-combater-pornografia-infantil-na-internet.html. Accessado: 2016-11-8. 2

[5] Swain, Michael J. e Dana H. Ballard: *Color indexing*. International Journal of Computer Vision, 7(1):11–32, 1991. 4

[6] Hampapur, Arun e Ruud M. Bolle: *Comparison of distance measures for video copy detection*. Em *IEEE International Conference on Multimedia and Expo*. IEEE, 2001. 4

[7] Hsu, Wynne, S. T. Chua e H. H. Pung: *An integrated color-spatial approach to content-based image retrieval*. Em *ACM international conference on Multimedia*, páginas 305–313. ACM, 1995. 4

[8] Chang, E., J. Wang, C. Li e G. Wiederhold: *RIME: A replicated image detector for the world-wide web*. Relatório Técnico, Stanford InfoLab, 1998. 4

[9] Bhat, Dinkar N. e Shree K. Nayar: *Ordinal measures for visual correspondence*. Em *Computer Vision and Pattern Recognition*. IEEE, 1996. 5

[10] Bhat, Dinkar N. e Shree K. Nayar: *Ordinal measures for image correspondence*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(4):415–423, 1998. 5

[11] Skrepth, Champskud J. e Andreas Uhl: *Robust Hash Functions for Visual Data: An Experimental Comparison*, páginas 986–993. Springer Berlin Heidelberg, 2003. 6

[12] Wu, Mingni, Chiachen Lin e Chinchen Chang: *Image copy detection with rotation tolerance.* Em *International Conference Computational Intelligence and Security*, páginas 464–469, 2005. 7

[13] Zou, Fuhao, Xiaowei Li, Zhihua Xu, Hefei Ling e Ping Li: *Image copy detection with rotation and scaling tolerance.* Journal of Computer Research and Development, 46(8):1349–1356, 2009. 7

[14] Li, Li, Hehuan Xu e Chin Chen Chang: *Rotation invariant image copy detection using DCT domain.* Em *International Journal of innovative Computing, Information and Control*, volume 7, páginas 3633–3644, 2010. 7

[15] Baaziz, Nadia e Maxime Guinin: *Content-based image copy detection using dual signatures.* Em Elmaghraby, Adel e Dimitrios N. Serpanos (editores): *ISSPIT*, páginas 17–22. IEEE, 2011, ISBN 978-1-4673-0752-9. 7

[16] Zou, Xiaoxiang, Gaochao Li, Ming Chen, Shupeng Wang, Yongjian Wang e Guangjun Wu: *An improved method to image copy detection.* Em *IEEE 2nd International Conference on Cloud Computing and Intelligent Systems (CCIS))*, páginas 634–637, 2012. 7

[17] Tang, Zhenjun, Fan Yang, Liyan Huang e Xianquan Zhang: *Robust image hashing with dominant DCT coefficients.* Optik - International Journal for Light and Electron Optics, 125(18):5102–5107, 2014. 7

[18] Tang, Zhenjun, Huan Lao, Kai Liu e Xianquan Zhang: *Robust image hashing via DCT and LLE.* Computers & Security, 62:133–148, 2016. 8

[19] Kang, XiaoBing e ShengMin Wei: *An efficient approach to still image copy detection based on SVD and block partition for digital forensics.* Em *IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 3, páginas 457–461, 2009. 8

[20] Arnia, Fitri, Agustinus Ifan, Khairul Munadi, Massaki Fujiyoshi e Hitoshi Kiya: *Content based image copy detection based on sign of Wavelet coefficients.* Em *International Workshop on Advanced Image Technology*, volume 3, 2011. 8

[21] Zhou, Zhili, Xingming Sun, Xianyi Chen e Lina Tan: *A Fourier-Mellin Transform based feature for image copy detection with rotation simulation strategy.* Em *JDCTA (International Journal of Digital Content Technology and its Applications)*, volume 6, páginas 369–377, 2012. 8

[22] Zhou, Zhili, Xingming Sun, Xianyi Chen, Cheng Chang e Zhangjie Fu: *A novel signature based on the combination of global and local signatures for image copy detection.* IOSR Journal of Computer Engineering, 16(3):73–77, 2014. 8

[23] Zou, Fujao, Yunpeng Chen, Jingkuan Song, Ke Zhou, Yang Yang e Nicu Sebe: *Compact image fingerprint via multiple kernel hashing.* IEEE Transactions on Multimedia, 17(7):1006–1018, 2015. 9

[24] Hsieh, Shang Lin, Chun Che Chen e Chuan Ren Chen: *A novel approach to detecting duplicate images using multiple hash tables*. Multimedia Tools and Applications, 74(13):4947–4964, 2015. 9

[25] Hsieh, Shang Lin, I Ju Tsai, Chung Ping Yeh e Chia Ming Chang: *An image authentication scheme based on digital watermarking and image secret sharing*. Multimedia Tools and Applications, 52(2):597–619, 2011. 9

[26] Yan, Lingyu, Fuhao Zou, Rui Guo, Lianli Gao, Ke Zhou e Chunzhi Wang: *Feature aggregating hashing for image copy detection*. World Wide Web, 19(2):217–229, 2016. 9

[27] Baber, Junaid, Maheen Bakhtyar, Waheed Noor, Abdul Basit e Ihsan Ullah: *Performance enhancement of patch-based descriptors for image copy detection*. Em *(IJACSA) International Journal of Advanced Computer Science and Applications*, volume 7, páginas 449–456, 2016. 9

[28] Rousseeuw, P. J. e A. M. Leroy: *Robust Regression and Outlier Detection*. John Wiley & Sons, Inc., New York, NY, USA, 1987, ISBN 0-471-85233-3. 13

[29] Itseez: *Open source computer vision library*. https://github.com/itseez/opencv, 2015. 15

[30] Jegou, Herve, Matthijs Douze e Cordelia Schmid: *Hamming embedding and weak geometry consistency for large scale image search*. Em *Proceedings of the 10th European conference on Computer vision*, 2008. 15, 17, 18

[31] Stone, John E., David Gohara e Guochun Shi: *OpenCL: A parallel programming standard for heterogeneous computing systems*. IEEE Des. Test, 12(3):66–73, maio 2010, ISSN 0740-7475. 15

[32] NVIDIA Corporation: *NVIDIA CUDA C programming guide*, 2010. Version 3.2. 15

[33] Rijsbergen, C. J. Van: *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edição, 1979, ISBN 0408709294. 22

Anexo I

Exemplos de Imagens das Bases de Dados Utilizadas

I.1 Ucid.v2

(a) ucid00001 (b) ucid00002 (c) ucid00003 (d) ucid00004 (e) ucid00005

(f) ucid00006 (g) ucid00007 (h) ucid00008 (i) ucid00009 (j) ucid00010

Figura I.1: Exemplos de imagens em UCID.v2.

I.2 RAISE

(a) r0a10fe29t

(b) r0a18e454t

(c) r0a3c52a0t

(d) r0a2e85f0t

(e) r0a2ff882t

(f) r0a19cb85t

(g) r0a4ce555t

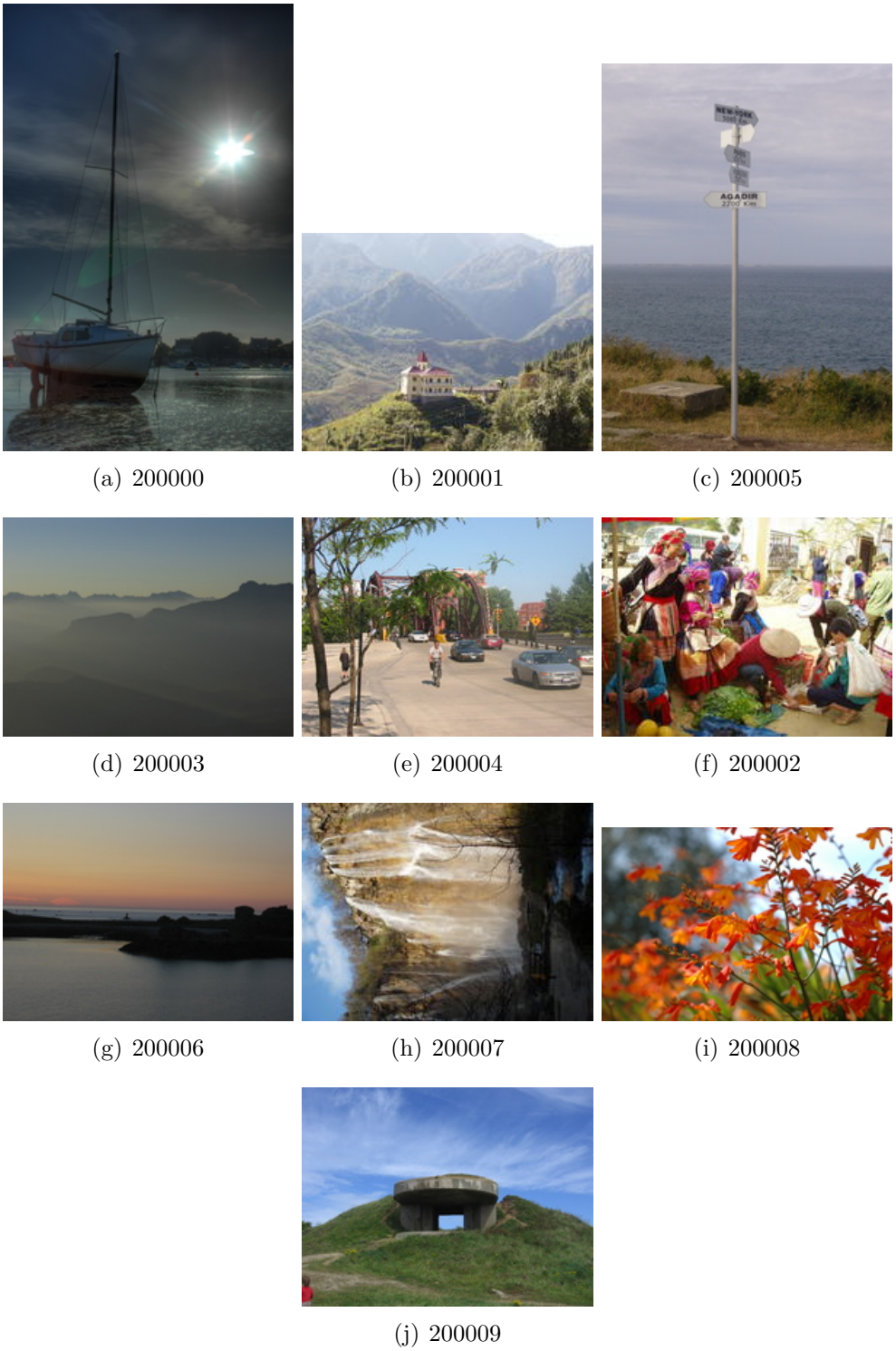
(h) r0a5d7cb6t

(i) r0a8b59bbt

(j) r0a8e5f8at

Figura I.2: Exemplos de imagens em RAISE.

I.3 Copydays



(a) 200000 (b) 200001 (c) 200005

(d) 200003 (e) 200004 (f) 200002

(g) 200006 (h) 200007 (i) 200008

(j) 200009

Figura I.3: Exemplos de imagens em Copydays.

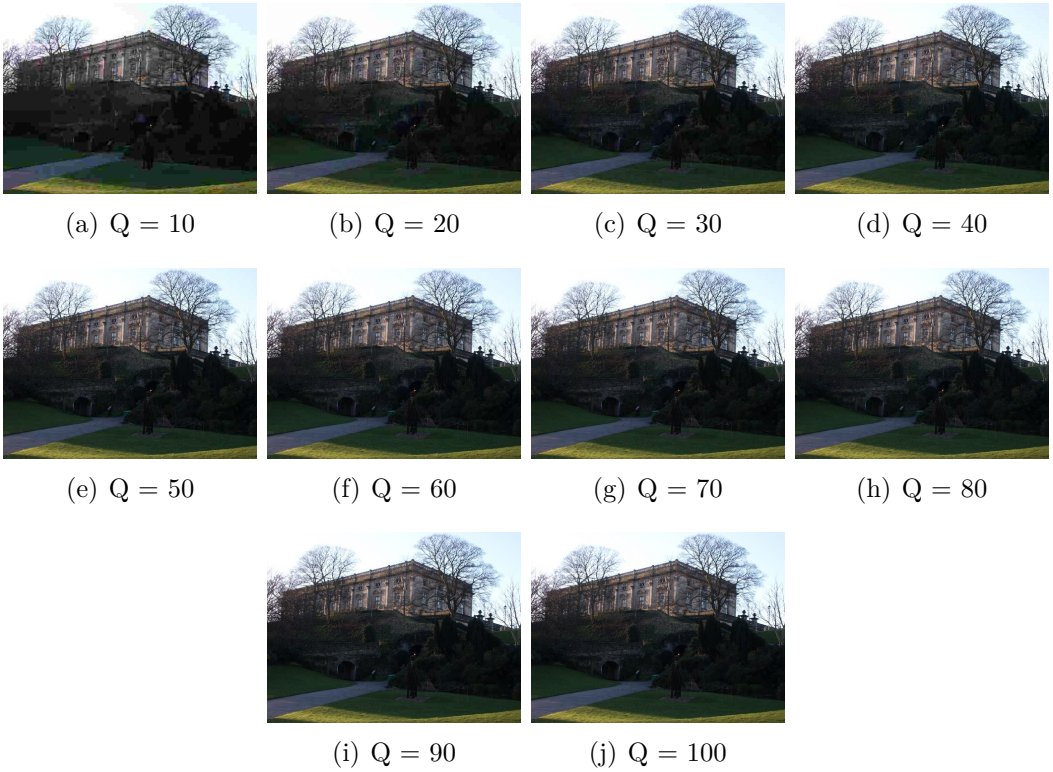
Anexo II

Exemplos de Cópias

II.1 Cópias Próprias

As cópias aqui mostradas são exemplos da imagem ucid00001 apresentada em Anexo I.

II.1.1 Compressão



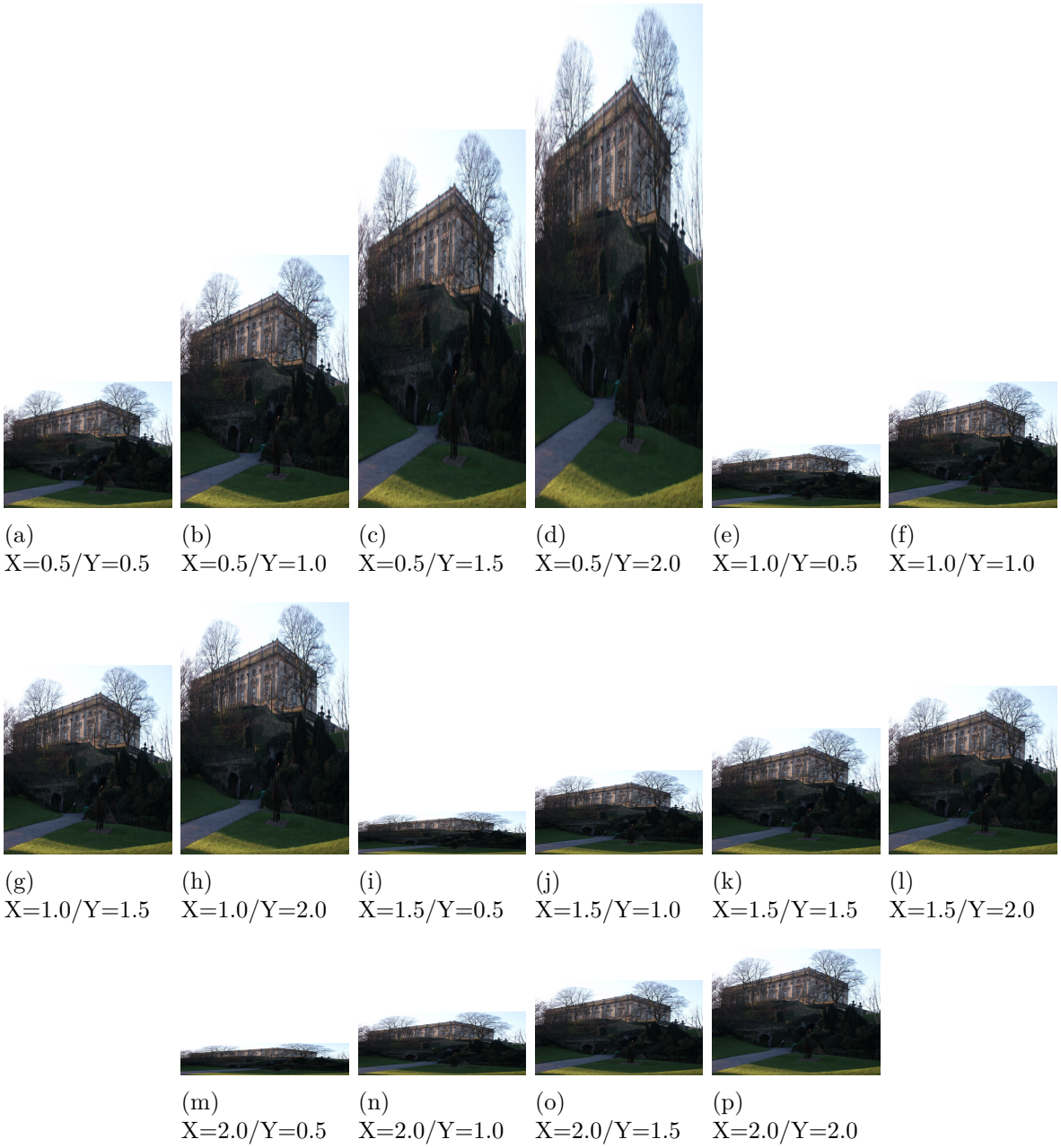
(a) Q = 10 (b) Q = 20 (c) Q = 30 (d) Q = 40

(e) Q = 50 (f) Q = 60 (g) Q = 70 (h) Q = 80

(i) Q = 90 (j) Q = 100

Figura II.1: Exemplos de compressão JPEG com fator de qualidade Q variando.

II.1.2 Redimensionamento



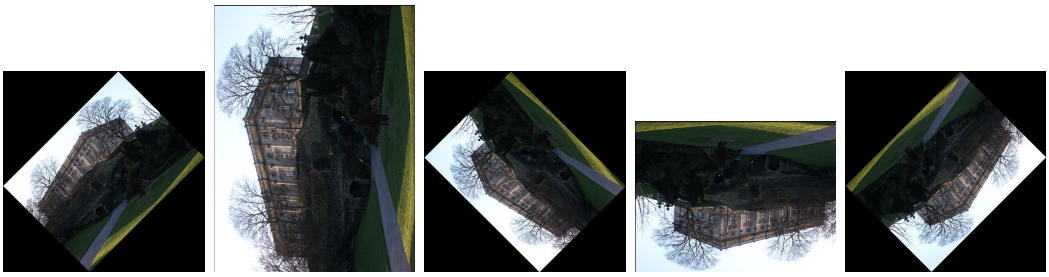
(a) X=0.5/Y=0.5 (b) X=0.5/Y=1.0 (c) X=0.5/Y=1.5 (d) X=0.5/Y=2.0 (e) X=1.0/Y=0.5 (f) X=1.0/Y=1.0

(g) X=1.0/Y=1.5 (h) X=1.0/Y=2.0 (i) X=1.5/Y=0.5 (j) X=1.5/Y=1.0 (k) X=1.5/Y=1.5 (l) X=1.5/Y=2.0

(m) X=2.0/Y=0.5 (n) X=2.0/Y=1.0 (o) X=2.0/Y=1.5 (p) X=2.0/Y=2.0

Figura II.2: Exemplos de redimensionamento com fator X representando a o fator que multiplica a largura e Y, o fator que multiplica a altura da imagem.

II.1.3 Rotação



(a) R = 45 (b) R = 90 (c) R = 135 (d) R = 180 (e) R = 225

(f) R = 270 (g) R = 315

Figura II.3: Exemplos de rotação, sendo R o valor da rotação em graus.

II.1.4 Espelhamento

(a) Horizontal (b) Vertical

Figura II.4: Exemplos de espelhamento horizontal e vertical.

II.1.5 Equalizadas

Figura II.5: Exemplo de histograma equalizado.

II.1.6 Escala de cinza

Figura II.6: Exemplo de escala de cinza.

II.1.7 Filtro por média aritmética

(a) N = 3 (b) N = 5 (c) N = 7 (d) N = 9

Figura II.7: Exemplos de filtro por média aritmética, utilizando um filtro de ordem N.

II.1.8 Filtro por mediana

(a) N = 3 (b) N = 5 (c) N = 7 (d) N = 9

Figura II.8: Exemplos de filtro por mediana, utilizando um filtro de ordem N.

II.1.9 *Sharpening*

Figura II.9: Exemplos de *sharpening*.

II.2 Cópias de Copydays

As cópias aqui mostradas são exemplos da imagem 200300 apresentada em Anexo I.

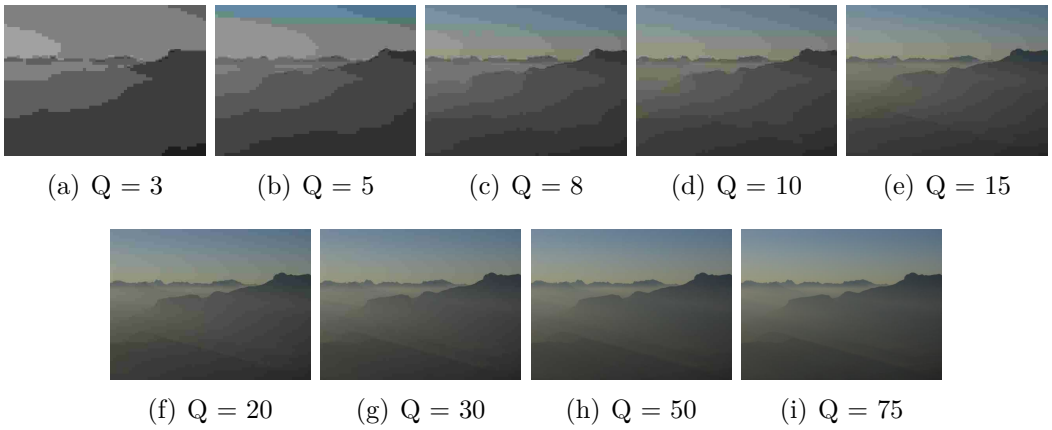
II.2.1 Corte

(a) K = 10 (b) K = 15 (c) K = 20 (d) K = 30 (e) K = 40

(f) K = 50 (g) K = 60 (h) K = 70 (i) K = 80

Figura II.10: Exemplos de corte com porcentagem K variando.

II.2.2 Compressão



(a) Q = 3 (b) Q = 5 (c) Q = 8 (d) Q = 10 (e) Q = 15

(f) Q = 20 (g) Q = 30 (h) Q = 50 (i) Q = 75

Figura II.11: Exemplos de compressão JPEG, sendo Q o fator de qualidade.

II.2.3 Ataques fortes

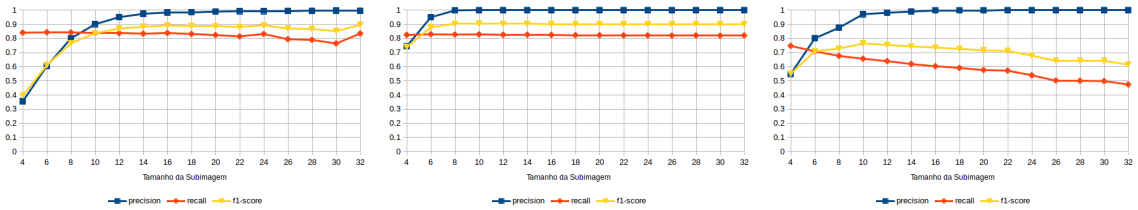
(a) Ataque forte 1 (b) Ataque forte 2 (c) Ataque forte 3

Figura II.12: Exemplos de ataques fortes em Copydays.

Anexo III

Resultados dos Ajustes

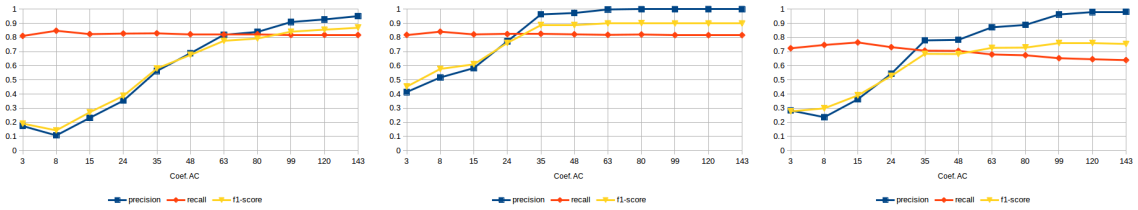
III.1 Divisão de blocos



(a) UCID.v2 (b) RAISE (c) Copydays

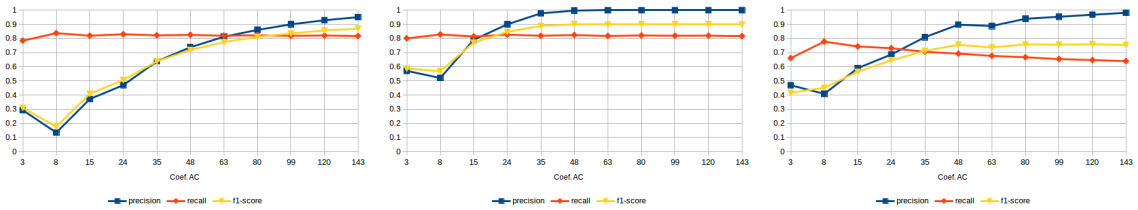
Figura III.1: *Precision*, *Recall* e *F1-Score* dos três bancos ao variar o tamanho da subimagem.

III.2 Extração de componentes AC

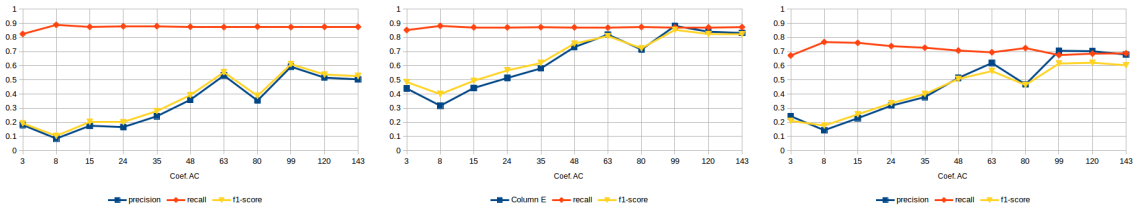


(a) UCID.v2/Próprio/Puro (b) RAISE/Próprio/Puro (c) Copydays/Próprio/Puro

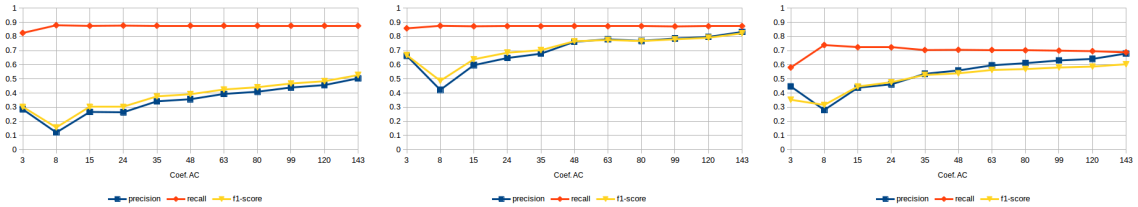
Figura III.2: *Precision*, *Recall* e *F1-Score* dos três bancos ao variar os parâmetros de extração dos coeficientes AC.



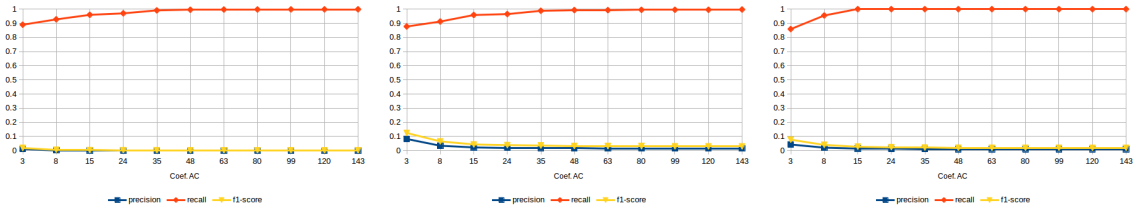
(a) UCID.v2/Kim/Puro (b) RAISE/Kim/Puro (c) Copydays/Kim/Puro



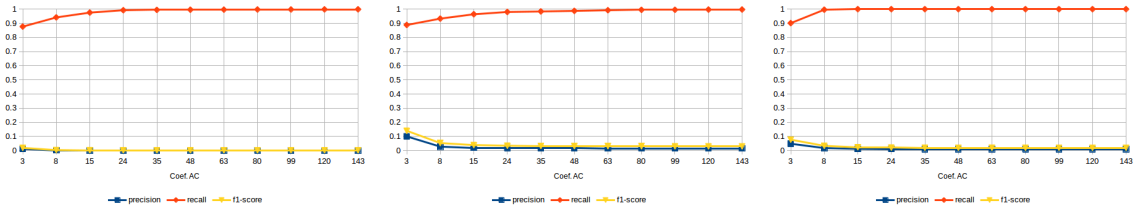
(d) UCID.v2/Próprio/Absoluto (e) RAISE/Próprio/Absoluto (f) Copydays/Próprio/Absoluto



(g) UCID.v2/Kim/Absoluto (h) RAISE/Kim/Absoluto (i) Copydays/Kim/Absoluto

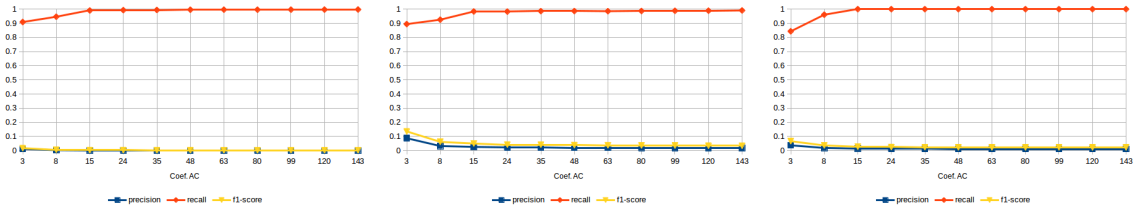


(j) UCID.v2/Próprio/Ordenado (k) RAISE/Próprio/Ordenado (l) Copydays/Próprio/Ordenado

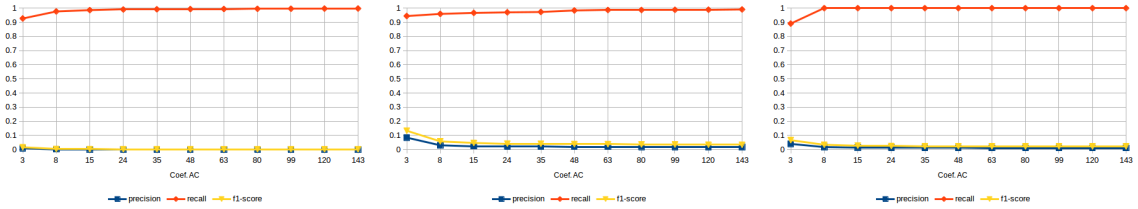


(m) UCID.v2/Kim/Ordenado (n) RAISE/Kim/Ordenado (o) Copydays/Kim/Ordenado

Figura III.3: Continuação: *Precision*, *Recall* e *F1-Score* dos três bancos ao variar os parâmetros de extração dos coeficientes AC.



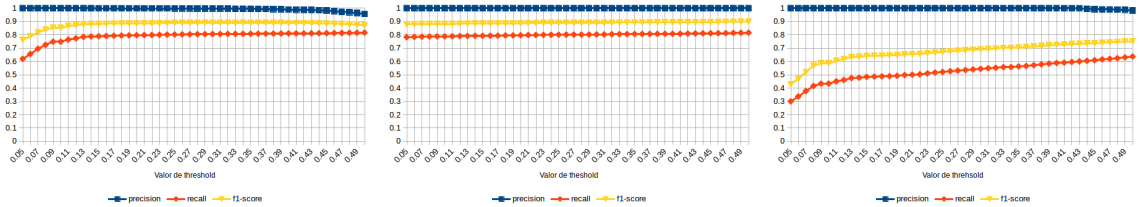
(a) UCID.v2/Próprio/Ambos (b) RAISE/Próprio/Ambos (c) Copydays/Próprio/Ambos



(d) UCID.v2/Kim/Ambos (e) RAISE/Kim/Ambos (f) Copydays/Kim/Ambos

Figura III.4: Continuação: *Precision*, *Recall* e *F1-Score* dos três bancos ao variar os parâmetros de extração dos coeficientes AC.

III.3 Determinação de *threshold*



(a) UCID.v2 (b) RAISE (c) Copydays

Figura III.5: *Precision*, *Recall* e *F1-Score* dos três bancos ao variar o valor do *threshold*.

Anexo IV

Código Final do Algoritmo Proposto

```
#include <iostream>
#include <math.h>
#include <fstream>
#include "opencv2/opencv.hpp"

#define QSIZE 12
#define ACS_DIM 12
#define ACS 143
#define THRESHOLD 0.32

using namespace std;
using namespace cv;

Mat getSubImage(Mat image){
    Mat y;
    image.convertTo(y, CV_32F);
    Mat avg(QSIZE,QSIZE,CV_32FC1);
    int sizeY = y.rows/QSIZE;
    int sizeX = y.cols/QSIZE;
    for(size_t i=0, k=0; i < y.rows && k<QSIZE; i+= sizeY, ++k){
        if(i+sizeY > y.rows) continue;
        float* avg_ptr = avg.ptr<float>(k);
        for(size_t j=0, l=0; j<y.cols && l<QSIZE; j += sizeX, ++l){
            if(j+sizeX > y.cols) continue;
            Mat roi = y(cv::Rect(j, i, sizeX, sizeY));
            avg_ptr[l] = (float)cv::mean(roi).val[0];
        }
    }
    return avg;
}
```

```
Mat ordinalMeasureAC(Mat image){
        Mat selected(1,ACS,CV_32F);
    int c=0;
    float* selected_ptr = selected.ptr<float>(0);
    for(size_t i=0; i<ACS_DIM; ++i){
        const float* image_ptr = image.ptr<float>(i);
        for(size_t j=0; j<ACS_DIM; ++j)
            if(i || j) selected_ptr[c++] = image_ptr[j];
    }
    return selected;
}

Mat processImage(Mat image){
        Mat sub = getSubImage(image);
        Mat dcted;
        dct(image, dcted);
        Mat acs = ordinalMeasureAC(dcted);
        return acs;
}

double getSM(Mat avComp1, Mat avComp2){
        float M = 0.0;
        for(size_t i=0; i < avComp1.rows; ++i){
                float* avComp1_ptr = avComp1.ptr<float>(i);
                float* avComp2_ptr = avComp2.ptr<float>(i);
                for(size_t j=0; j<avComp1.cols; ++j)
                        M += fabs(avComp1_ptr[j]
                        - avComp2_ptr[avComp1.cols-j-1]);
        }

        return norm(avComp1,avComp2,NORM_L1)/M;
}

int main(int argc, char** argv){
        if(argc != 3){
                cout << "Wrong␣Usage:␣<exec>␣<query␣image>␣<database␣path>"
                 << endl;
                return 0;
        }

        Mat image1 = imread(argv[1], CV_LOAD_IMAGE_GRAYSCALE);
        if(!image1.data){
                cout << "Ocorreu␣um␣erro␣ao␣carregar␣a␣imagem." << endl;
                return 1;
        }
```

```
        string path(argv[2]);
        ofstream file;
        file.open("result.txt", ios::out | ios::app);

        Mat avComp1 = processImage(image1);

        int c = 0;
        FileStorage fs(path+"parameters.xml", FileStorage::READ);
        if(!fs.isOpened()){
                cout << "Could not open parameters." << endl;
                exit(0);
        }
        while(true){
                try{
                        Mat avComp2(1,ACS,CV_32F);
                        string count = static_cast<ostringstream*>( &(ostringstream() <<
                        count = "imagem"+count;
                        FileNode node = fs[count];
                        string name = node["name"];
                        node["matriz"] >> avComp2;

                        if(name == "") break;

                        double n = getSM(avComp1, avComp2);
                        if(n < THRESHOLD) file << name << "\n";
                }catch(...){
                        cout << "Ocorreu um erro na leitura do arquivo de
                                database." << endl;
                        break;
                }
        }

        fs.release();
        file.close();
        return 0;
}
```