

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS  
CENTRO DE CIÊNCIAS EXATAS AMBIENTAIS E DE  
TECNOLOGIAS  
PROGRAMA DE PÓS-GRADUAÇÃO STRICTO-SENSU  
GERÊNCIA DE REDES DE TELECOMUNICAÇÕES

**ANDERSON DOS SANTOS SILVA TORRES**

**HASH PERCEPTIVO DE IMAGENS E SUA APLICAÇÃO  
NA IDENTIFICAÇÃO DE CÓPIA DE VÍDEOS**

**CAMPINAS**

**28 de Fevereiro de 2019**

**Anderson dos Santos Silva Torres**

**HASH PERCEPTIVO DE IMAGENS E SUA APLICAÇÃO  
NA IDENTIFICAÇÃO DE CÓPIA DE VÍDEOS**

Dissertação apresentada ao Centro de Ciências Exatas, Ambientais e de Tecnologias – CEATEC, da Pontifícia Universidade Católica – PUC – Campinas, como requisito à obtenção do título de Mestre em Gerência de Redes de Telecomunicações.

Orientador: Prof. Dr. Antonio Carlos Demanboro

Campinas


# ANDERSON DOS SANTOS SILVA TORRES

## “Hash Perceptivo de Imagens e sua Aplicação na Identificação de Cópia de Vídeos”

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Centro de Ciências Exatas, Ambientais e de Tecnologias da Pontifícia Universidade Católica de Campinas como requisito parcial para obtenção do título de Mestre em Gestão de Redes de Telecomunicações.

Área de Concentração: Engenharia Elétrica.  
Orientador: Prof. Dr. Antônio Carlos Demanboro

Dissertação defendida e aprovada em 28 de fevereiro de 2019 pela Comissão Examinadora constituída dos seguintes professores:



---

Prof. Dr. Antônio Carlos Demanboro

Orientador da Dissertação e Presidente da Comissão Examinadora  
Pontifícia Universidade Católica de Campinas



---

Prof. Dr. Eric Alberto de Mello Fagotto

Pontifícia Universidade Católica de Campinas



---

Prof. Dr. Rangel Arthur  
UNICAMP

Ficha catalográfica elaborada por Vanessa da Silveira CRB 8/8423  
Sistema de Bibliotecas e Informação - SBI - PUC-Campinas

005.741 Torres, Anderson dos Santos Silva.  
T693h Hash perceptivo de imagens e sua aplicação na identificação de cópia de vídeos / Anderson dos Santos Silva Torres.- Campinas: PUC-Campinas, 2019.  
117 f.: il.

Orientador: Antonio Carlos Demanboro.  
Dissertação (Mestrado em Gestão de Redes de Telecomunicações) - Centro de Ciências Exatas, Ambientais e de Tecnologias, Pontifícia Universidade Católica de Campinas, Campinas, 2019.  
Inclui bibliografia.

I. Hashing (Computação). 2. Processamento de imagens. 3. Computação. 4. Organização de arquivos (Computação). I. Demanboro, Antonio Carlos. II. Pontifícia Universidade Católica de Campinas. Centro de Ciências Exatas, Ambientais e de Tecnologias. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDD - 23. ed. 005.741

**Dedicatória.**

**Ao Deus que manteve seguro os meus caminhos, aos meus filhos, Thiago, Lucas e Nicole Torres que tornam os dias realmente bons. A minha bela e amada esposa, Irê, a quem serei sempre grato por tolerar meus desatinos.**

## **Agradecimentos**

**A minha família,**

**De quem as horas preciosas foram sacrificadas.**

**Ao Prof. Dr. Antonio Carlos Demanboro**

**Pela amizade, paciência e perspicácia, fundamentais para conclusão deste trabalho.**

**À Pontifícia Universidade Católica de Campinas,**

**Pelas concessão da bolsa no Programa de Mestrado, por ser um local de ponderação, estudo e conhecimento.**

**Aos colegas e companheiros da turma,**

**Pelo apoio e fraternidade.**

“Sorri quando a dor te torturar  
E a saudade atormentar  
Os teus dias tristonhos vazios  
Sorri quando tudo terminar  
Quando nada mais restar  
Do teu sonho encantador  
Sorri quando o sol perder a luz  
E sentires uma cruz  
Nos teus ombros cansados doridos  
Sorri vai mentindo a sua dor  
E ao notar que tu sorris  
Todo mundo irá supor  
Que és feliz”

*Versão brasileira da música “Smile”, escrita por João de Barro (Braguinha). A melodia foi composta por Charles Chaplin em 1936.*

## Resumo

TORRES, Anderson dos Santos Silva, **HASH PERCEPTIVO DE IMAGENS E SUA APLICAÇÃO NA IDENTIFICAÇÃO DE CÓPIA DE VÍDEO, 2018**. Dissertação de Conclusão de Curso para o Mestrado Profissional em Gerência de Redes de Telecomunicações.

Com o avanço da rede mundial de computadores, arquivos de vídeo e imagem são largamente compartilhados e consumidos por usuários de todo o mundo. Com isso, métodos para identificação desses arquivos surgiram como forma de preservar direitos autorais e de reprodução. A identificação com base no conteúdo ou *Hash Perceptivo* (*Perceptual Hashing*) é a técnica capaz de gerar um identificador numérico baseado nas características visuais de uma imagem. Com este identificador, é possível comparar e decidir se duas imagens são iguais, semelhantes, ou diferentes. O presente estudo tem como objetivo testar, por meio de ferramenta desenvolvida pelo autor, a aplicação de algoritmos distintos de *Hash Perceptivo* de imagens em arquivos de vídeo.

Propõe-se o uso de métodos conhecidos de hash perceptivo tais como *Average Hash*, e *Difference Hash*, *Perceptual Hash* e *Wavelet Hash*. Aplicou-se uma técnica de identificação utilizando um limiar (threshold) de similaridade por meio da distância de Hamming e a combinação dos algoritmos de *hash* perceptivo para investigação de potencial de cópia. Foi analisado o comportamento do método diante de ataques diversos e os resultados foram detalhados. Conclui-se que é possível utilizar algoritmos de *hash* perceptivo de imagens para identificação de cópia de vídeo, e obtém-se vantagens na eliminação de falsos positivos quando existe a combinação de mais de um deles preenchendo lacunas de desempenho e vulnerabilidades.

**Palavras-Chave:** Hash perceptivo, Marca D'água, Identificação de Vídeo



## Abstract

TORRES, Anderson dos Santos Silva. **IMAGE PERCEPTUAL HASH APPLIED FOR VIDEO COPY IDENTIFICATION, 2018.** Dissertation (Master in Telecommunication Network Management) - Pontifícia Universidade Católica de Campinas, Centro de Ciências Exatas, Ambientais e de Tecnologias,

*Campinas 2019.*

*With the event of the Internet, video and image files are widely shared and consumed by users from all over the world. Thus, methods to identify these files have emerged as a way to preserve intellectual and commercial rights. Content based identification or perceptual hashing is the technique capable of generating a numeric identifier from the image characteristics. With this identifier, it is possible compare and decide if two images are equal, similar or different. This study has as objective discuss the application of image perceptual hashing to identify video copies. It proposes the usage of known and public methods such as the Average and Difference Hash that are based on statistics of the image also Phash and Wavelet hash that are based on the image frequency.*

*An identification technique was applied using a Hamming distance similarity threshold and the combination of perceptual hash algorithms for video copy identification. The method was tested by applying several attacks to the candidate video and the results were properly detailed. It is possible to use perceptual hash algorithms for video copy identification, and there are benefits when there is a combination of more than one of them filling performance gaps and vulnerabilities and eliminating false positives*

**Keywords: Perceptual hashing, Watermarking, Video Identification**

## Lista de ilustrações

Figura 1 – Classificação de Funções <i>Hash</i> . . . . .	27
Figura 2 – Componentes básicos dos algoritmos de hash perceptivo . . . . .	32
Figura 3 – Diagrama de comparação entre hashes perceptivos . . . . .	33
Figura 4 – Arquitetura da DCT e DWT - Adaptado pelo Autor . . . . .	36
Figura 5 – Pontos de Extração de Características . . . . .	37
Figura 6 – Representações geradas a partir de uma imagem . . . . .	37
Figura 7 – Dois exemplos de sinal <i>x</i> , <i>y</i> para aplicação da correlação cruzada. .	43
Figura 8 – Gráfico mostrando a correlação entre os sinais <i>x</i> e <i>y</i> . . . . .	43
Figura 9 – Representação da composição de um arquivo de vídeo . . . . .	47
Figura 10 – Resumo dos métodos de identificação de vídeo com base no conteúdo	48
Figura 11 – Estrutura de Hash Perceptivo em Arquivos de Vídeos . . . . .	51
Figura 12 – Número de quadros de acordo com o tamanho do vídeo em função do tempo . . . . .	55
Figura 13 – Visualização expandida da extração de hash . . . . .	56
Figura 14 – Número de hashes de acordo com o tamanho do vídeo em função do tempo . . . . .	57
Figura 15 – Tabelas para armazenamento de metadados e hashes . . . . .	58
Figura 16 – Tabela de arquivos de vídeos de referência . . . . .	61
Figura 17 – Tabela com os vídeos candidatos . . . . .	62
Figura 18 – Ilustração de busca por cópia de vídeo . . . . .	64
Figura 19 – Informação sobre o computador utilizado para os testes. . . . .	65
Figura 20 – Lista de pastas de quadros . . . . .	68
Figura 21 – Gráfico com o número de quadros e tempo gasto no processamento dos valores hash . . . . .	71
Figura 22 – Arquivo CSV com resultado das comparações . . . . .	75
Figura 23 – Gráfico com os valores da comparação entre o vídeo candidato e as referências . . . . .	76
Figura 24 – Tabela com os valores da comparação entre o vídeo candidato e as referências . . . . .	76
Figura 25 – Valores abaixo do limiar de similaridade - aHash . . . . .	77
Figura 26 – Imagem 00001_0.04.jpg e valor aHash - 0000000000000000 . . . .	77
Figura 27 – Valores abaixo do limiar de similaridade - dHash . . . . .	78
Figura 28 – Imagem 01163_48.51.jpg e valor aHash - 70644cc4c4e6e4cc . . . .	78
Figura 29 – Valores abaixo do limiar de similaridade - pHash . . . . .	79
Figura 30 – Valores abaixo do limiar de similaridade - wHash . . . . .	80
Figura 31 – Imagem 00865_36.08.jpg e valor wHash - ceccdc6c6ce0c0 . . . .	80

Figura 32 – Gráfico com os valores da comparação entre o vídeo candidato e as referências após manipulação - Recorte . . . . .	82
Figura 33 – Tabela com os valores da comparação entre o vídeo candidato e as referências após manipulação - Recorte . . . . .	83
Figura 34 – Gráfico com os valores da comparação entre o vídeo candidato e as referências após manipulação - Resolução . . . . .	84
Figura 35 – Tabela com os valores da comparação entre o vídeo candidato e as referências após manipulação - Resolução . . . . .	84
Figura 36 – Gráfico com os valores da comparação entre o vídeo candidato e as referências após manipulação - Rotação 10 graus . . . . .	85
Figura 37 – Tabela com os valores da comparação entre o vídeo candidato e as referências após manipulação - Rotação 10 graus. . . . .	86
Figura 38 – Gráfico com os valores da comparação entre o vídeo candidato e as referências após manipulação - Rotação 90 graus . . . . .	87
Figura 39 – Tabela com os valores da comparação entre o vídeo candidato e as referências após manipulação - Rotação 90 graus. . . . .	87
Figura 40 – Gráfico com os valores da comparação entre o vídeo candidato e as referências após manipulação - Rotação 180 graus. . . . .	88
Figura 41 – Tabela com os valores da comparação entre o vídeo candidato e as referências após manipulação - Rotação 180 graus. . . . .	88

## Lista de tabelas

Tabela 1 – Lista de formatos e codecs de vídeos mais usuais . . . . .	19
Tabela 2 – Algoritmo MD5 aplicado na imagem de Lena Soderberg . . . . .	24
Tabela 3 – Distorções de Imagens . . . . .	25
Tabela 4 – Funções <i>hash</i> e equivalência científica . . . . .	38
Tabela 5 – Implementações públicas de funções de <i>hash</i> perceptivo . . . . .	38
Tabela 6 – Exemplos de Aplicação da Distância de Hamming . . . . .	40
Tabela 7 – Base de vídeos para os testes . . . . .	52
Tabela 8 – Descrição dos campos das tabelas candidateFile/ReferenceFile . .	59
Tabela 9 – Descrição dos campos das tabelas candidateHash/referenceHash .	59
Tabela 10 – Cálculo do Coeficiente e Percentual de Igualdade . . . . .	62
Tabela 11 – Simulação de cálculo de Percentual de Igualdade Final. . . . .	63
Tabela 12 – Tempo de execução - extração dos quadros . . . . .	67
Tabela 13 – Tempo de execução - Extração dos valores hash . . . . .	70
Tabela 14 – Tabela com tempo utilizado para comparação dos valores <i>hash</i> - Software e Banco de dados. . . . .	72
Tabela 15 – Tempo utilizado para comparação dos valores <i>hash</i> - Software e Banco de dados . . . . .	74
Tabela 16 – Manipulações aplicadas no vídeo candidato. . . . .	81

## Lista de abreviaturas e siglas

3GPP	3rd Generation Partnership Project
5G	Fifth Generation
AAC	Advanced Audio Encoding
AND	Soma os resultados de buscas de dois termos diferentes
ASCII	American Standard Code for Information Interchange
BER	Bit Error Rate
BI	Business Intelligence
BMP	Bit Map Picture
DCT	Discrete Cousine Transform
DFT	Discret Fourier Transform
DVD	Digital Versatile Disc (Disco Digital Versátil)
DWT	Discret Wavelet Transform
FLV	Flash Video
FPS	Frames per Second
HD	Hard Disk
IC	Imagem Candidata
ID	Identificação Digital
IR	Imagem de Referência
JPEG	Joint Photographic Experts Group
LAN	Local Area Network
LH	Low High
LL	Low Low
MAC	Message Authentication Code
MD5	Algoritmo de criptografia (message digests)
MDC	Modification Detection Code

MP3	Moving Pictures Experts Group audio layer 3.
NOT	Exclui um termo de uma determinada busca
P2P	peer-to-peer(ponto a ponto)
PCC	Peak Cross Correlation
PHP	Hypertext Preprocessor
PI	Porcentagem de Igualdade
PIL	Biblioteca Python Pillow
PNG	Portable Network Graphic
RGB	Red, Green, Blue
SHA	Secure Hash Algorithm
SQL	Structured Query Language

## Sumário

<b>1</b>	<b>Introdução</b>	<b>15</b>
<b>1.1</b>	<b>Considerações Iniciais</b>	<b>15</b>
<b>1.2</b>	<b>Transmissão de Vídeos na Rede Mundial de Computadores</b>	<b>18</b>
<b>1.3</b>	<b>Identificação de Cópia de Vídeos na Rede Mundial de Computadores</b>	<b>20</b>
<b>2</b>	<b>Revisão da Literatura</b>	<b>23</b>
<b>2.1</b>	<b>Integridade em Arquivos multimídia</b>	<b>23</b>
<b>2.2</b>	<b>Funções <i>Hash</i></b>	<b>25</b>
2.2.1	Hash com Chave e <i>Hash</i> Sem Chave	26
2.2.2	Considerações sobre Colisão em Funções <i>Hash</i>	27
<b>2.3</b>	<b><i>Hash</i> Perceptivo</b>	<b>30</b>
2.3.1	Estrutura do <i>Hash</i> Perceptivo	31
2.3.2	Características do <i>Hash</i> Perceptivo	33
2.3.3	Técnicas para Geração de <i>Hash</i> Perceptivo	34
2.3.3.1	Baseados em Estatísticas	34
2.3.3.2	Baseados em Relações	35
2.3.3.3	Baseados em Recursos de Baixo Nível	36
2.3.3.4	Representações Grosseiras ( <i>Coarse Representation</i> )	37
2.3.4	Implementações Conhecidas de <i>Hash</i> Perceptivo de Imagens	38
<b>2.4</b>	<b>Métodos Comparativos</b>	<b>39</b>
2.4.1	Distância de Hamming	40
2.4.2	Taxa de Erro de Bits (BER)	41
2.4.3	Correlação Cruzada ( <i>Cross Correlation</i> )	42
<b>2.5</b>	<b>Marca D'água Digital ou <i>Watermarking</i></b>	<b>44</b>
2.5.1	Correlação entre <i>Hash</i> Perceptivo e Marca D'água	45
<b>2.6</b>	<b>Composição dos Arquivos de Vídeo</b>	<b>46</b>
<b>2.7</b>	<b>Métodos de Identificação de Cópia de Vídeo</b>	<b>48</b>
<b>3</b>	<b>Materiais e Métodos</b>	<b>50</b>
<b>3.1</b>	<b>Biblioteca de Funções <i>Hash</i></b>	<b>50</b>
<b>3.2</b>	<b>Estrutura</b>	<b>51</b>
3.2.1	Seleção do Vídeo	52
3.2.2	Extração de Metadados	54
3.2.3	Extração de Quadros	54
3.2.4	Extração de <i>Hash</i>	56
3.2.5	Banco de Dados	57

3.2.6	Busca, Comparação e Identificação . . . . .	59
3.3	<b>Tempo de Execução . . . . .</b>	<b>65</b>
3.4	<b>Visualização dos Dados . . . . .</b>	<b>65</b>
4	<b>Resultados e Discussão . . . . .</b>	<b>67</b>
4.1	<b>Extração de Metadados e Quadros . . . . .</b>	<b>67</b>
4.1.1	Geração dos Valores <i>Hash</i> . . . . .	69
4.1.2	Comparando Valores <i>Hash</i> . . . . .	71
4.2	<b>Identificando Cópia de Vídeo . . . . .</b>	<b>75</b>
4.2.1	Sem Manipulações/Ataques no Vídeo . . . . .	75
4.2.2	Manipulações/Ataques no Vídeo . . . . .	81
5	<b>Discussão . . . . .</b>	<b>89</b>
6	<b>Conclusões . . . . .</b>	<b>91</b>
	<b>Referências . . . . .</b>	<b>93</b>
	 <b>APÊNDICES</b>	 <b>100</b>
	<b>APÊNDICE A – Código criado em Python demonstrando a utilização da biblioteca ImageHash para geração de valor hash . . . . .</b>	<b>101</b>
	<b>APÊNDICE B – Código em Python utilizado para extração dos quadros do vídeo . . . . .</b>	<b>102</b>
	<b>APÊNDICE C – Amostra do código utilizado para extração do valores hash a partir dos quadros do vídeo . . . . .</b>	<b>104</b>
	<b>APÊNDICE D – Código em Python que efetua o cálculo da Distância de Hamming entre valores hash . . . . .</b>	<b>107</b>
	<b>APÊNDICE E – Código em Python que efetua busca e comparações em memória . . . . .</b>	<b>108</b>
	<b>APÊNDICE F – Código em Python que efetua busca e comparações junto ao banco de dados . . . . .</b>	<b>111</b>
	<b>APÊNDICE G – Código em Python que efetua rotinas junto ao banco de dados . . . . .</b>	<b>113</b>



# 1 Introdução

## 1.1 Considerações Iniciais

De acordo com estudos recentes, um em cada dois usuários da rede mundial de computadores se envolvem em atividades classificadas como ilícitas (SOUTH. . . , 2016). O interesse dos usuários pode variar muito, mas geralmente transita nos mercados de jogos, músicas e vídeos. Com o avanço das tecnologias de telecomunicações, taxas cada vez maiores permitem que o acesso e compartilhamento de mídias digitais sejam rápidos e eficientes. Filmes e séries recém lançados estão disponíveis nos diversos repositórios especializados, poucas horas após seu lançamento. No mercado de esportes, por exemplo, é possível assistir partidas inteiras de futebol, que são compartilhadas quase que em tempo real.

A cópia, publicação e distribuição de conteúdo sem que os direitos de exploração financeira dos detentores da obra sejam preservados é o que se conhece como “pirataria” (KUMAR; MANIKANTA, 2013). Pessoas que fazem uso da pirataria ignoram as leis de propriedade intelectual bem como os direitos de autores, programadores, distribuidores e muitos outros, que dependem da exploração econômica desses ativos. A indústria audiovisual emprega esforços jurídicos e técnicos para impedir a proliferação desta ameaça ao seu modelo de negócio.

Métodos criptográficos são extremamente eficientes quando empregados na transmissão de dados e costumam ser bem aplicados na proteção de mídias de áudio e vídeo. É com estas ferramentas que os estúdios, produtoras e operadoras de TV e cinema transmitem o conteúdo de um para o outro. Mediante da criptografia o conteúdo é cifrado, podendo ser decifrado através de uma chave que é entregue ao destinatário que obteve os direitos de reprodução através de contratos e pagamentos equivalentes. Se por ventura alguém se apropriar indevidamente desse arquivo, mas não obtiver a chave que o decifra, a visualização não será possível. Infelizmente a criptografia não serve para proteger o conteúdo depois que ele é entregue ao usuário consumidor desses recursos. Não há como controlar como o consumidor manipula o conteúdo, que pode muito bem ter sido obtido de forma legítima, mas depois pode ser copiado e distribuído ilegalmente.

A indústria e indivíduos, lesados pelo compartilhamento desautorizado de suas mídias digitais, recorrem então a legislação e ao aparato jurídico na tentativa de impedir, prevenir ou mesmo punir os responsáveis pelo ilícito. No entanto, apenas através de tecnologias de identificação digital tais como *watermarking* ou *fingerprinting* é possível provar que a mídia distribuída de fato pertence àquele que reclama seus direitos.

*Watermarking* ou Marca D'água é a técnica ou prática de esconder uma mensa-

gem sobre uma imagem, áudio, vídeo ou outro tipo de mídia, dentro do próprio objeto. Por exemplo, ao observar uma nota de 10 (dez) reais brasileiros, sob a luz, percebe-se uma imagem do lado esquerdo da efígie da república. Este exemplo de *watermarking* é utilizado por muitos países para garantir a autenticidade das moedas em circulação. O que é importante salientar a respeito do *watermarking* é que a eficácia de sua utilização está baseada em não estar visível facilmente ao usuário, sendo que somente após a aplicação de técnicas de visualização específicas é que a marca fica aparente.

*Multimedia fingerprinting* ou *hash perceptivo* são os nomes dados à técnica de gerar, a partir de um arquivo de multimídia (áudio, vídeo) um identificador ou assinatura que seja único e inequívoco. Com este identificador é possível, a partir de uma base de dados de referência, identificar se o item verificado possui características que o classificam como cópia/réplica ou não. Para serem eficientes, os algoritmos de *perceptual hashing* precisam ser resistentes as modificações que o arquivo possa sofrer por conta da distribuição ou de manipulações intencionais utilizadas para evitar detecção. Escalabilidade é uma característica desejável na maioria dos sistemas e processos e indica a capacidade de atender o crescimento da carga de trabalho de maneira uniforme e a estar preparado para crescer. No caso dos algoritmos de *hash perceptivo*, ser escalável significa garantir que o tamanho do identificador *hash* seja grande o suficiente para que se evitem eventos de colisão. Em outras palavras, é dizer que o *hash perceptivo* não pode gerar identificadores iguais a partir de arquivos diferentes. Em aplicações em que a geração de *hash* é muito grande, deve-se garantir que tanto a base de dados quanto o tamanho do identificador gerado sejam suficientemente robustos para que a colisão não aconteça ou tenha sua probabilidade reduzida.

A grande vantagem dos sistemas de *hash perceptivo*, quando comparados com sistemas de *watermarking*, é que não é necessário inserir informações no arquivo de referência, preservando assim suas características originais. Consequentemente, os algoritmos de *hash perceptivo* podem ser aplicados em circunstâncias em que os arquivos já foram distribuídos (HADMI et al., 2012).

Algoritmos de *hash perceptivo* de imagens podem ser utilizados de maneira bastante criativa por empresas, pesquisadores e entusiastas. O algoritmo pode ser utilizado, por exemplo, em uma base de dados que informa o usuário se determinada imagem já existe na rede mundial de computadores e quais sítios a utilizam. Profissionais como fotógrafos, publicitários e artistas podem utilizar tal ferramenta para saber onde suas obras foram publicadas. Sítios de relacionamento podem utilizar *hash perceptivo* para controlar o teor das imagens publicadas em sua plataforma, suprimindo imagens de conteúdo inadequado. As universidades poderiam utilizar-se de uma base de *hash* de obras publicadas por seus alunos e compará-los, evitando plágio. Ou seja, é possível utilizar *hash perceptivo* em qualquer situação que exija a necessidade de

encontrar imagens semelhantes ou duplicadas.

As mídias digitais de vídeo, independentemente do método utilizado para compressão, armazenamento e encapsulamento, são, na verdade, compostas por uma sequência de imagens chamadas de quadros. Os quadros, se analisados individualmente, tem as características visuais únicas necessárias para aplicar as técnicas de *hash perceptivo* e obter uma base de dados rica para estudo.

O propósito desta dissertação é além de analisar algoritmos de *hash perceptivo* de imagens, verificar, através de um código criado pelo autor, como estes algoritmos se comportam juntos quando utilizados na identificação de vídeos. Neste estudo, foi utilizada a biblioteca de código aberto e livre em linguagem Python chamada ImageHash, na versão 4.0, disponível no endereço (<https://pypi.org/project/ImageHash/>). Dentro desta biblioteca encontram-se implementações de *hash perceptivos* - *Average Hash* (aHash) - (YANG; GU; NIU, 2006); *Difference Hash* (dHash) que é uma variação do aHash, mas utiliza os gradientes (variação entre os *pixels*) (DRMIC et al., 2017); *P Hash* (pHash) que é baseado na frequência da imagem e utiliza DCT (*Discrete Cosine Transform*) (ZAUNER, 2010); e finalmente o *Wavelet Hash* (wHash) que como o pHash faz uso da frequência, mas calcula as diferenças por meio do DWT (*Discrete Wavelet Transform*) (MONGA; EVANS, 2004).

Assim, foram utilizadas as técnicas identificadas por *aHash*, *pHash*, *dHash* e *wHash* como objetos de estudo deste trabalho e a análise de seu comportamento quando aplicadas em arquivos de vídeo.

Os objetivos específicos desta pesquisa são:

- Fazer um estudo das técnicas de *Hash Perceptivo* – *aHash*, *pHash*, *dHash* e *wHash*, de modo a obter o conhecimento necessário para realizar comparações e combinações entre elas;
- Aplicar os algoritmos de *hash perceptivo* presentes na biblioteca *ImageHash* 4.0 em arquivos de vídeo;
- Executar ataques nos arquivos de vídeo para verificar a eficácia dos algoritmos na identificação.

Esta dissertação está estruturada da seguinte maneira:

- O Capítulo 1 discorre brevemente sobre as formas de distribuição de arquivos de vídeo no mundo e a importância de tecnologias de identificação;
- No Capítulo 2 apresenta-se a fundamentação teórica, abordando os principais elementos, estruturas e características dos diversos tipos de *hash perceptivo*;

- O Capítulo 3 apresenta a metodologia empregada neste trabalho, detalhando materiais e os métodos utilizados;
- O Capítulo 4 é reservado para a apresentação dos resultados obtidos a partir dos testes realizados, bem como, observações a respeito dos desafios encontrados;
- As considerações adicionais, conclusão e indicações de trabalhos futuros possíveis estão nos capítulos 5 e 6.

## 1.2 Transmissão de Vídeos na Rede Mundial de Computadores

Artigo recente (AGÊNCIA O GLOBO, 2018/01) descreve os eventos peculiares que incentivaram o que hoje é conhecido como a primeira transmissão de vídeo em uma rede de computadores. Em 1991, uma câmera de vídeo analógica foi utilizada para transmitir a partir de uma sala da Universidade de Cambridge, no Reino Unido, se a cafeteira estava cheia ou não. Segundo o artigo a transmissão, precária para os padrões atuais, era de apenas 1 quadro por segundo, em escala de cinza e com resolução de  $128 \times 128$  pixels. Apenas em 1993, com o surgimento do navegador Mosaico, é que foi possível “conectar” a transmissão da cafeteira na rede mundial de computadores de maneira bastante rudimentar, uma vez que o navegador exibia apenas o quadro do momento da conexão e ficava estático a partir disto (MARTYN.JOHNSON, 1994). Quentin Stafford-Fraser, autor da façanha, não acreditava que a transmissão de uma cafeteira geraria tanto interesse e que muitos anos depois a transmissão de vídeos se tornaria a base da comunicação dos dias futuros.

Estima-se que mais de 70% dos usuários da rede mundial de computadores acessam transmissões de vídeo diariamente, por meio de seus aparelhos celulares, *tablets*, televisores inteligentes e computadores. O fenômeno é ainda mais surpreendente quando se considera o tempo que um usuário médio despende consumindo este tipo de mídia. Estima-se que 67% dos usuários de aparelhos celulares inteligentes consomem vídeos diariamente e cerca de 10% destes assistem conteúdo por mais de 3 horas. As fontes destes conteúdos, embora distintas, se concentram em redes sociais e plataformas especializadas tendo a base distribuída em 51% e 41% respectivamente (SRUOGINIS; WARREN, 2018). A plataforma de vídeos *online* Netflix, por exemplo, reporta um crescimento anual de 35% no número de horas assistidas pelos seus usuários, número este que se repete consistentemente desde 2009 (MCALONE, 2016). Outros estudos indicam que, no ano de 2019, 80% do consumo global da rede mundial de computadores será destinado à transmissão de vídeo (CISCO PUBLIC, 2018) e com o surgimento da tecnologia 5G estima-se que o crescimento do consumo de vídeo por meios móveis terá um incremento de 55% por ano até 2022 (JEJDLING, 2018).

Com um mercado desta magnitude e com projeções tão animadoras de crescimento, é perfeitamente normal observar o interesse igualmente crescente da indústria de vídeo. Estúdios de cinema e produtoras como Disney, Fox, HBO e mesmo gigantes da tecnologia como Google, Amazon e Netflix investem bilhões de dólares na aquisição e produção de novos títulos. O modelo de comercialização dos serviços também sofreu mudanças na última década, observando uma quebra de paradigma no abandono de tecnologias como o DVD e *Blue Ray*, para o modelo de assinaturas e a exploração de receita por meio de propagandas. É previsto que a receita gerada por *marketing* digital irá praticamente dobrar de tamanho em 2019, passando a ser um mercado de 109 bilhões de dólares (IRONPAPER, 2016/11).

Os arquivos de vídeos evoluíram e apresentam uma variedade de formatos, codificadores e decodificadores (*codecs*), métodos de proteção, compressão e tecnologias que permitem a rápida adaptabilidade necessária para o consumo *online* e móvel. Estas tecnologias são praticamente invisíveis a partir do ponto de vista de quem as consome. Respeitadas as exigências de reprodução, tais como resolução e capacidade de processamento, os sistemas compatíveis com estes formatos tornam a experiência do usuário alheia a todo o aparato tecnológico que o suporta. Lista-se a seguir, na Tabela 1, alguns formatos de vídeo, *codecs* e tecnologias mais usuais (ENCODING.COM, 2018).

**Tabela 1 – Lista de formatos e codecs de vídeos mais usuais**

<b>Formatos Recipientes (Container)</b>	<b>Codecs de Vídeo</b>	<b>Codecs de Áudio</b>	<b>Legendas</b>
MP4 (mp4, m4a, m4v, f4v, f4a, m4b, m4r, f4b, mov)	H.263	MP3	WEBVTT
3GP (3gp, 3gp2, 3g2, 3gpp, 3gpp2)	H.264	AAC	CEA-608/708
OGG (ogg, oga, ogv, ogx)	HEVC	HE-AAC	DFXP
WMV (wmv, wma, asf*)	MPEG4	AC3 (Dolby Digital)	SAMI

<b>Formatos Recipientes (Container)</b>	<b>Codecs de Vídeo</b>	<b>Codecs de Áudio</b>	<b>Legendas</b>
WEBM (webm)	Theora	EAC3 (Dolby Digital Plus)	SCC
FLV (flv)	3GP	Vorbis	SRT
AVI	Windows Media 8	WMA	TTML
QuickTime	Quicktime	PCM	3GPP
HDV	MPEG-4		
MXF (OP1a, OP-Atom)	VP8		
MPEG-TS (ts)	VP6		
MPEG-2 PS, MPEG-2 TS	MPEG1		
WAV, Broadcast WAV	MPEG2 (CableLabs)		
LXF, GXF	MPEG-TS		
VOB	MPEG-4		
	DNXHD		
	XDCAM		
	DV, DVCPRO, DVCPRO*, DVCPProHD		
	IMX, XDCAM HD, XDCAM HD422, XDCAM EX*		
	JPEG 2000		

Encoding.com (2018)

### 1.3 Identificação de Cópia de Vídeos na Rede Mundial de Computadores

O aumento crescente da oferta de banda larga, a redução dos preços de dispositivos de armazenamento, bem como a mudança no modelo de negócios, fizeram com que a popularidade dos serviços e plataformas de transmissão de vídeo tais como Youtube, Amazon Vídeo, HBO, Netflix e outros alcançassem até mesmo as classes sociais menos favorecidas. Isto não quer dizer que o compartilhamento ilegal de vídeos na rede mundial de computadores tenha desaparecido. Estima-se, de acordo com o Conselho Nacional de Combate a Pirataria (CNCP), que somente no Brasil as perdas

geradas por pirataria, contrabando e comércio ilegal de produtos e conteúdo são de 130 bilhões de reais por ano (G1.GLOBO.COM, 2017). A notícia também aponta que o número de acessos a um sítio dedicado à pirataria de conteúdo é cerca de 8 vezes maiores que o acesso ao Netflix no mesmo período. Entende-se que conteúdos na forma de vídeos tem um alto valor social e econômico e exercem, portanto, um papel importante na sociedade contemporânea.

P2P, Camcorder, *Cyberlockers*, *Smartbox*, *Playlists*, *UGC*s são nomes de tecnologias e formas de compartilhamento de conteúdo. P2P (*Peer to Peer*), por exemplo, tem causado danos a indústria de audiovisual desde seu surgimento (MITTAL, 2004) no início dos anos 2000 (MILOJICIC et al., 2002). Estima-se que métodos mais simples do ponto de vista tecnológico, porém igualmente danosos, como Camcorder, que é o ato de gravar e compartilhar filmes a partir de salas de cinema (HAJJ-AHMAD et al., 2017), ou UGC (*User Generated Content*) (MOENS; LI, 2014) que é capturar um conteúdo obtido nos lares (TV a cabo, DVD, B-Ray) e compartilhá-lo nas redes sociais e plataformas como Youtube e/ou Facebook produzirão juntos, perdas de 52 bilhões de dólares até 2022 (DTVE, 2017). Daí decorre o interesse da indústria de audiovisual investir em tecnologias e em formas de proteção de seus ativos. O Google, por exemplo, investiu cerca de 100 milhões de dólares em proteção de conteúdo dentro da plataforma do Youtube e afirma ter pago mais de 3 bilhões de dólares a detentores de direitos de reprodução (GOOGLE, 2018). Por meio da ferramenta de código fechado chamada de *Content ID*, a plataforma do Youtube identifica quando um vídeo ou música possui direitos autorais e notifica o dono da obra para lhe dar opções do que fazer. As opções podem ser desde a retirada imediata do conteúdo, a redirecionar os valores gerados com propaganda naquele vídeo para o seu devido dono. A plataforma social Facebook por meio da ferramenta proprietária *Rights Management*, de forma similar ajuda os detentores de direitos de autor e propriedade intelectual a identificar suas obras (FACEBOOK, 2018).

Tanto o sistema do Google quanto o sistema do Facebook utilizam ferramentas de verificação de integridade para autenticar se uma obra é similar ou igual a outra. Para isto, o detentor da obra entrega ao Youtube ou Facebook o arquivo, música ou vídeo do qual alega ser dono, para verificação. Este arquivo é então chamado de arquivo de referência e passa a compor uma base de dados que é utilizada na busca e comparação com os arquivos de vídeo e música disponibilizados pelos usuários destas plataformas.

É neste contexto, que esta dissertação desenvolveu e aplicou uma ferramenta de verificação de integridade de imagens e vídeos (híbrido), em código aberto, caracterizada de *Hash* Perceptivo de Imagens em arquivos de vídeos, visando a verificação independente de conteúdo. O sistema combina características de quatro algoritmos de

hash perceptivo por meio da utilização de um quociente de similaridade e com isto aproveita os pontos fortes de cada um destes para superar vulnerabilidades quando expostos a ataques.



## 2 Revisão da Literatura

### 2.1 Integridade em Arquivos multimídia

Com o avanço da rede mundial de computadores, o compartilhamento de arquivos multimídias tais como imagem e vídeo fazem parte do cotidiano dos usuários. Com isto, a necessidade de preservar a autenticidade, confidencialidade e a integridade destes arquivos torna-se importante para muitas áreas e especialmente jornalismo, fotografia, cinema e artes. Confidencialidade significa tornar ininteligível a leitura de um arquivo multimídia criptografado trocado entre entidades que não possuem uma chave que decifra tal conteúdo. Esquemas de criptografia utilizados para garantir a confidencialidade são conhecidos como métodos simétricos ou assimétricos. O método simétrico é aquele que utiliza a mesma chave para cifrar e decifrar o conteúdo e o assimétrico quando usa chaves distintas (MENEZES; OORSCHOT; VANSTONE, 1996). Autenticidade permite identificar o autor da mídia e que ele só é recebido ou consumido por aqueles que tem os direitos para tal. Integridade no campo das mídias digitais não se limita em apenas identificar se o conteúdo da mídia foi alterado, mas, se houve degradação no conteúdo (NIKOLAIDIS; PITAS, 2006).

Para verificar a integridade de dados, funções *Hash* como *Message Digest 5* (MD5) (SCIENCE; SECURITY, 1992) e *Secure Hash Algorithm* (SHA) (HANDSCHUH, 2011) são utilizadas (a discussão e definição de *hash* é feita no sub item 2.2). Estas funções são sensíveis à pequenas alterações de *bits* e, conseqüentemente, para verificar a integridade da mensagem basta verificar se seus bits foram alterados depois da transmissão (MENEZES; OORSCHOT; VANSTONE, 1996). No MD5, por exemplo, uma sequência de 128 *bits* (16 *bytes*) é gerada a partir de uma entrada arbitrária qualquer. No exemplo a seguir uma *string* ASCII (*American Standard Code for Information Interchange*) de texto a ser autenticado é dada como entrada na variável  $\$x$ .  $\$h$  é o valor *hash* a ser obtido pela função MD5 que recebe  $\$x$  como parâmetro. Ao final, um comando *echo* é executado para mostrar o conteúdo da variável  $\$h$  (neste caso o valor *hash*):

#### Código 2.1 – Exemplo de utilização do MD5

```
$x = "Olá Mundo";
$h = md5($x);
echo $h; // 7141e6862d77e5c2a8e2f60a77a153cd
```

Se for alterado ainda que seja um *bit* da entrada será obtido um *hash* com valor diferente. No exemplo a seguir o acento agudo do “Olá Mundo” foi retirado e o *hash* produzido é completamente diferente.

**Código 2.2 – Exemplo 2 de utilização do MD5**

```
$x = "Ola Mundo";
$h = md5($x);
echo $h; // 0ddc9b6af462ede614b74281e1e0f9db
```

Para exemplificar de maneira visual o uso de algoritmos *hash* para verificação de integridade, a figura de Lena Söderberg (WIKIPEDIA, 2019) é utilizada a seguir. No primeiro exemplo o *hash* MD5 calculado, a partir da imagem “original”. No segundo, o *hash* é calculado a partir da imagem de Lena com alterações visuais (o olho esquerdo está visivelmente mais escuro). Percebe-se que os *hash* obtidos são diferentes, embora as imagens sejam perceptivelmente a mesma.

**Tabela 2 – Algoritmo MD5 aplicado na imagem de Lena Söderberg**

Lena Söderberg - Original	Lena Söderberg - Modificada
	
MD5: 55e4d7a0f5ef8567dbb27a15079d636c	MD5: ed5cd2c8950c12e6a16d9f1869560508

Fonte: L.Weng; B.Preneel (2011).

O algoritmo SHA funciona de maneira semelhante, no entanto, produz um *hash* de 160 bits (20 bytes). Este tipo de criptografia tradicional não é indicado para objetos multimídias visto que o tamanho das mídias são em sua maioria de grande porte e o processamento exigido para consumir estes arquivos (uma vez que estejam criptografados) é alto. Um segundo ponto a ser considerado é que pequenas distorções são geralmente aceitáveis para percepção humana e a criptografia tradicional, como MD5, SHA, métodos simétricos e assimétricos são exigentes demais para imagem e vídeo. Certas distorções alteram de maneira significativa a ordem dos *bits*, mas não alteram a percepção que o usuário tem daquela mídia. Estas distorções são categorizadas como Manipulações que Preservam o Conteúdo e Manipulações que

Não Preservam o Conteúdo (HAN; CHU, 2010). A Tabela 3, mostra algumas das distorções possíveis.

**Tabela 3 – Distorções de Imagens**

<b>Manipulações que Preservam o Conteúdo.</b>	<b>Manipulações que Não Preservam o Conteúdo.</b>
Erros de Transmissão	Remoção de objetos
Adição de Ruídos	Movimento de elementos da imagem ou alteração de sua posição
Compressão	Adição de objetos
Redução de resolução	Modificação de características da imagem: cor, textura, estrutura, etc.
Dimensionamento	Modificação do fundo da imagem: Dia, Hora, Localização
Rotação	Modificação de luz; Sombras, etc.
Recorte	
Modificações de saturação, matiz e brilho	
Ajuste de contraste	
Distorção gama	

Fonte: Shuihua Han; Chao-Hsien Chu (2010).

Para superar o desafio de verificar a integridade de uma mídia com base no seu conteúdo, *hash* perceptivos são propostos. Estas funções de *hash* perceptivo calculam os valores *hash* com base em certas características, por vezes visuais, da mídia e são utilizados para verificar a “similaridade” de uma mídia em comparação com outra. Estas funções devem ser robustas o suficiente para garantir que manipulações que preservam o conteúdo não alterem o valor do *hash* (SEO et al., 2004). Deste modo, a integridade no campo das mídias digitais passa a ter um significado diferente do que é normalmente aceite, com *hash* perceptivos é possível verificar a similaridade de uma mídia e autenticá-la, ainda que tenha sofrido manipulações.

## 2.2 Funções Hash

Funções *hash* estão presentes na criptografia moderna e são geralmente utilizadas para verificação de integridade e autenticação de mensagens. Tem como função tomar uma mensagem de entrada, computar ou analisar o seu conteúdo e

por fim produzir um resultado de tamanho menor - algo como um resumo (string, ou inteiro) - chamado de *hash*. Em outras palavras o *hash* é a versão reduzida da mensagem que ele representa (OZSARI, 2003). Funções *Hash* são similares de maneira análoga ao que se conhece como impressão digital, pois devem ser únicas e identificam um indivíduo de maneira inequívoca. Deste modo, funções *hash* devem ter, em suas características, reduzidas ou eliminadas as hipóteses de se obter um valor igual para mensagens de entrada diferentes (MENEZES; OORSCHOT; VANSTONE, 1996); (OZSARI, 2003) .

Matematicamente, uma função *hash* é representada por  $H$ , e produz um valor *hash* representado por  $h$  a partir de um valor arbitrário de entrada  $x$  (MENEZES; OORSCHOT; VANSTONE, 1996), ou seja:

$$h = H(x) \quad (2.1)$$

Existem diversas categorias de funções *hash* que ao longo do tempo foram desenvolvidas para os fins de autenticação e verificação de integridade de mensagens. De maneira geral, as funções *hash* são categorizadas em dois tipos: Com Chaves (*Keyed*) e Sem Chaves (*Unkeyed*).

### 2.2.1 Hash com Chave e Hash Sem Chave

Uma função *hash* com chave (*keyed*) é utilizada para autenticação da origem da mensagem e são alternativamente chamados de Códigos de Autenticação de Mensagens (MAC - *Message Authentication Code*). Uma função *hash* com chave produz um *hash*  $h$  a partir de um valor arbitrário de entrada  $x$  e uma chave  $k$ :

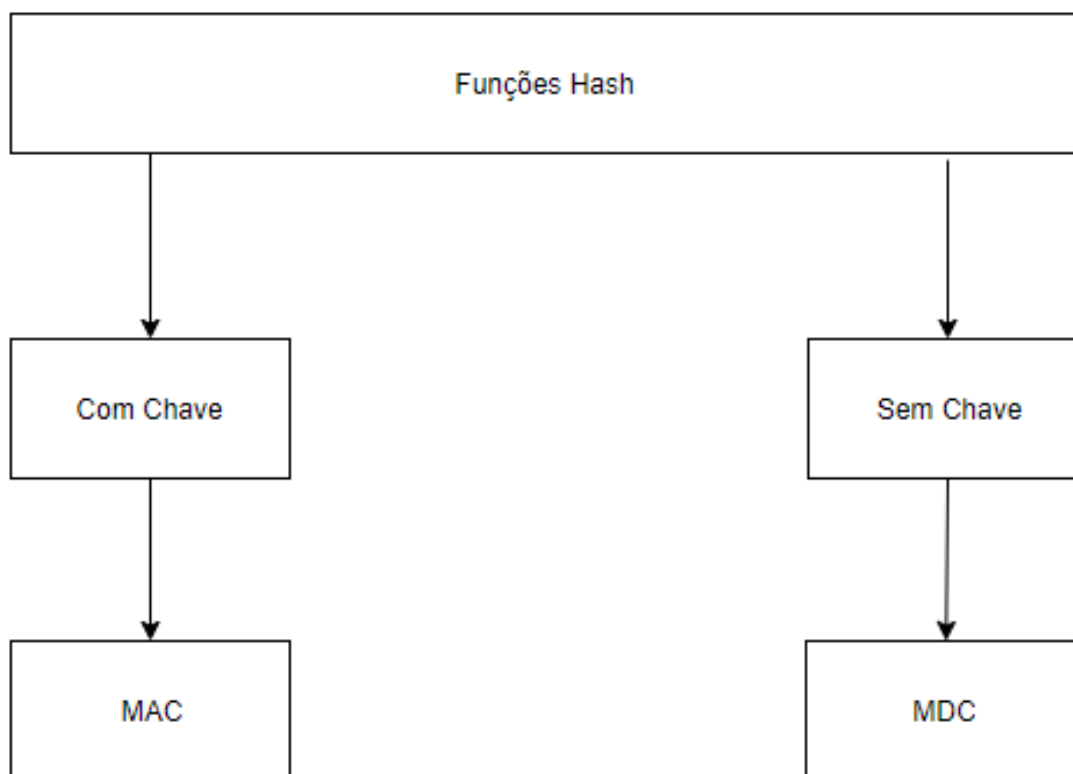
$$( \quad h = H(x, k) \quad (2.2) ) .$$

Em contra-partida, uma função *hash* sem chaves (*unkeyed*) é utilizada para verificar se o conteúdo da mensagem foi alterado. São também chamadas de Códigos de Detecção de Modificações (MDC - *Modification Detection Codes*) (MENEZES; OORSCHOT; VANSTONE, 1996). Uma função *hash* sem chave produz um *hash*  $h$  a partir de um valor arbitrário de entrada  $x$ ,

$$( \quad h = H(x) \quad (2.3) ) .$$

A Figura 1 apresenta a classificação das funções *Hash* (ALAHMAD; ALSHAIKHLI, 2013).

Figura 1 – Classificação de Funções Hash



Fonte: AlAhmad; Alshaiikli (2013).

### 2.2.2 Considerações sobre Colisão em Funções Hash

Na teoria da computação, colisão é descrita como o evento em que dois objetos ou registros recebem o mesmo identificador ou alocação na memória. A descrição também se aplica quando um algoritmo produz o mesmo valor *hash* para entradas diferentes. Um algoritmo considerado robusto deve ter como característica fundamental a independência de pares (*Pairwise independance*) conforme será mostrado na Seção 2.3.2, que de forma simplificada quer dizer que o *hash* gerado a partir de duas entradas distintas devem ser diferentes. As funções *hash* não possuem métodos que possibilitam a identificação dos itens de entrada, ou registro dos valores *hash* gerados a partir destes. Isto significa que, embora a característica de independência de pares seja algo desejável e extremamente necessária, a colisão é estatisticamente possível visto que haja um número ilimitado de entradas e um número limitado de saídas possíveis (PAAR;

PELZL, 2009, 298-300). A probabilidade de colisão está diretamente relacionada com a composição (carácteres) e o tamanho do valor *hash*. Quanto maior (em tamanho) o valor *hash* mais “difícil” se torna a colisão. O uso da palavra difícil e não impossível é adequado porque o crescimento do poder computacional torna os cálculos necessários para encontrar estes valores mais acessíveis.

Para ilustrar como funciona a colisão, considere-se uma função *hash* representada por  $H$ , e um valor de entrada qualquer, por exemplo, “Ola”, que como resultado gera um valor *hash*, 7 878 654, composto por 7 (sete) carácteres numéricos. Tendo a função equivalente:

$$H(Ola) = 7878654 \quad (2.4)$$

A colisão poderia acontecer quando, se para a mesma função *hash*  $H$  e uma entrada diferente, por exemplo, “Mundo”, o resultado gerado for igual ao que foi obtido para a entrada “Ola”.

$$H(Mundo) = 7878654 \quad (2.5)$$

No caso de *hash* perceptivos a colisão aconteceria se o mesmo valor *hash* for obtido a partir de imagens diferentes.

Na seção 2.6 explora-se como os arquivos de vídeo são formados e na seção 3.2.3 discute-se o número aproximado de quadros gerados de acordo com a taxa de reprodução e o tempo de duração do vídeo. Considerações sobre colisão de *hash* se tornam de extrema importância quando o armazenamento dos valores *hash* são necessários. Outro valor é necessário para tornar o cálculo da possibilidade de colisão possível, sendo este, o valor aproximado do tamanho da base de dados a ser utilizada.

A problemática da colisão pode ser ilustrada da seguinte maneira. Sendo  $N$  o valor total de saídas ou valores *hash* possíveis, e  $k$  um número aleatório de valores gerados, que são numéricos e menores que  $N$ . Qual a probabilidade de que dentre os múltiplos valores  $k$ , pelo menos dois deles sejam iguais? Ou melhor, qual a probabilidade de que todos os valores  $k$ , sejam únicos?

A abstração matemática desta questão pode ser resumida se apenas se subtrair o valor de  $N$ , por 1. Ou seja, dado um espaço de saídas ou valores *hash* possíveis  $N$  (pode ser qualquer valor inteiro), existem  $N - 1$  valores que são únicos e diferentes do primeiro item da lista. Portanto, a probabilidade de se gerar dois inteiros iguais é:

$$\frac{N - 1}{N} \quad (2.6)$$

Depois disto, haverá  $N - 2$  valores possíveis que são únicos e diferentes dos dois primeiros itens da lista. Isto quer dizer que a probabilidade de se gerar 3 valores inteiros iguais é:

$$\frac{N-1}{N} \times \frac{N-2}{N} \quad (2.7)$$

Como os algoritmos de *hash* produzem valores aleatórios dentro do espaço  $N$ , os quais são chamados de  $k$ , é possível calcular a probabilidade de se ter apenas valores únicos em  $k$ , da seguinte forma:

$$\frac{N-1}{N} \times \frac{N-2}{N} \times \dots \times \frac{N-(k-2)}{N} \times \frac{N-(k-1)}{N} \quad (2.8)$$

A expressão anterior pode ainda ser simplificada e ter resultados semelhantes da seguinte forma:

$$e^{-\frac{k(k-1)}{2N}} \quad (2.9)$$

Ao subtrair o resultado da função por 1, obtem-se a resposta para a pergunta original - “Qual a probabilidade de que todos os valores  $k$ , sejam únicos?” - que é o mesmo que obter a probabilidade de colisão (PAAR; PELZL, 2009, 299-302), (MENEZES; OORSCHOT; VANSTONE, 1996, 369) tendo então:

$$1 - e^{-\frac{k(k-1)}{2N}} \quad (2.10)$$

Cálculos de grande magnitude são realizados por computadores e mesmo um *hash* relativamente pequeno de 32 bits ou  $N = 2^{32}$  representa um universo de 4.294.967.296 (4.2 bilhões) de entradas possíveis. Através de um algoritmo escrito pelo autor, é possível testar a teoria apresentada nesta seção e experimentar qual o tamanho ideal para o valor *hash* e o tamanho da base de dados (vídeos). No código 2.3, é testada a probabilidade de colisão com um *hash* de 64 bits.

**Código 2.3 – Código em Python para calcular probabilidade de colisão de hash - Criado pelo autor**

```
#!/usr/bin/python

#title           :collision.py
#description     :Calculates collision probability
    from a give hash space:
#author         :Anderson Torres
#date          :20181124
#version       :1.0
```

```

#usage          :python collision.py
#notes         :
#python_version :2.6.6
#
=====

import threading
probUnique = 1.0          #Constante, probabilidade de se ter
                          dois valores iguais.

def domath(n,k):
    t = threading.Thread(target=collision, args=(n,k))
    t.start()

def collision(N,k):
    #Teoria http://preshing.com/20110504/hash-collision-
    probabilities/
    global probUnique     #Para modificar a variável global

    probUnique = probUnique * (N - (k - 1)) / N
    probability = probUnique - 1
    probPercentage = abs(round(probability * 100,2))
    print('Registro %s tem %s ou %s %% chances de
          colisao' % (k, probability, probPercentage))

N = 18446744073709551616          #Universo de hash possiveis
    - 2^64 or 16^16
for k in xrange(1, 5000000000):    #Numero de hash gerados - 5
    Billion
    domath(N,k)

```

Depois de aplicar o algoritmo e testar a probabilidade de colisão, é possível afirmar que independente do tamanho do *hash*, a probabilidade para uma colisão atinge 50% quando se divide o expoente por 2. Assim, a probabilidade de se ter dois números iguais para uma base de dados com 256 posições atinge 50% (ou mais) ao ter cerca de 16 ( $2^4$ ) valores gerados, que é exatamente a metade do expoente de 256 ( $2^8$ ). A expressão a seguir é uma maneira simplificada de expressar este cálculo.

$$pc = 2^{\frac{x}{2}} \quad (2.11)$$

### 2.3 Hash Perceptivo

*Hash* perceptivos ou *Multimídia Fingerprinting* são funções que propõe identificar similaridades em arquivos multimídias com base no conteúdo, de modo que alterações e distorções não significativas não alterem o valor do *hash*. Nesta seção,



explora-se a estrutura, classificações e características necessárias para compor um algoritmo robusto.

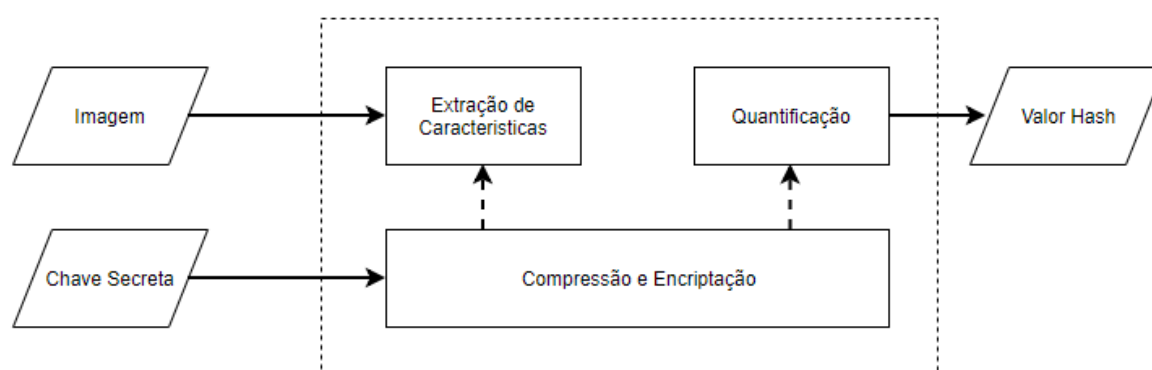
### 2.3.1 Estrutura do *Hash* Perceptivo

Nos algoritmos de *hash* perceptivos é possível dividir a construção do valor *hash* em quatro estágios ou fases que são:

- Transformação
- Extração de Características
- Quantificação
- Compressão e Encriptação

No estágio da transformação a imagem é modificada, ou transformada em um sub-produto da imagem de origem. Estas modificações intencionais podem ser de caráter espacial (alteração de cor ou tamanho) ou no domínio da frequência (*Discret Cosine Transform* ou *Discret Wavelet Transform*). Este passo é importante para que as informações necessárias para os cálculos da próxima fase sejam obtidos apenas com base nos pixels ou nos coeficientes da frequência. No segundo estágio, características únicas da imagem (linhas, pontos recursos de baixo nível, etc) são extraídas para construir os vetores necessários para o cálculo do *hash*. No estágio da quantificação, o *hash* é convertido de um valor binário para um valor do tipo *string*. É nesta fase que a probabilidade de colisão é reduzida e se aumenta a robustez do algoritmo. No último passo, o *hash* é encriptado e comprimido para uma versão final e reduzida. Como os algoritmos de *hash* perceptivos são utilizados para autenticação e identificação de similaridade, os passos de compressão e encriptação são geralmente ignorados. O valor *hash* obtido no estágio de quantificação é suficiente para armazenamento e atende os propósitos de autenticação básica. Outra razão que torna os estágios de compressão e encriptação inconvenientes é que é difícil prever o comportamento proveniente das características da imagem depois de submetê-la a manipulações (HADMI et al., 2012). Na Figura 2, apresentam-se os componentes básicos dos algoritmos de *hash* perceptivo (WENG; PRENEEL, 2007) .

Figura 2 – Componentes básicos dos algoritmos de hash perceptivo

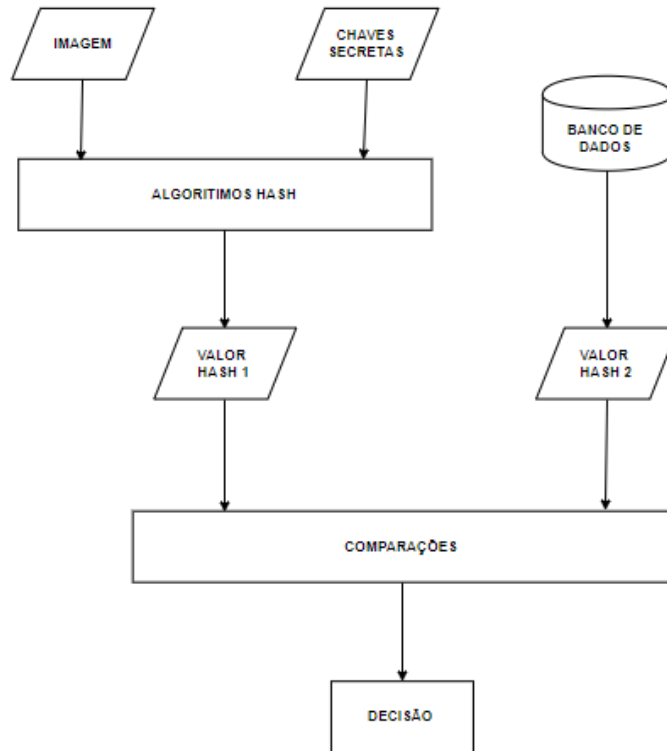


Fonte: Weng; Preneel (2007).

A utilização do *hash* perceptivo é útil em aplicações como identificação de imagens em bases de dados, verificação de propriedade intelectual, detecção de plágio ou ainda otimização de bases de dados evitando assim a inclusão de objetos duplicados. Tais aplicações ou uso só são possíveis quando se adiciona o elemento “banco de dados” à estrutura do *hash* perceptivo. É evidente que esta adição não se refere a geração ou produção do *hash*, mas ao seu armazenamento para consulta posterior. Armazenamento, indexação e busca dos *hash* devem ser igualmente eficientes e robustos e técnicas como árvore k-d (árvore k-dimensional) podem ser utilizadas (ROBINSON, 1981). O estudo detalhado a respeito das categorias de bancos de dados e indexação não será abordado neste trabalho, basta dizer, que todos os bancos de dados modernos incluem indexação eficiente. O armazenamento de *hash* será discutido na seções 3.2.5 e 3.3 .

Depois de obter os hash a partir da imagem de referência e da imagem candidata à verificação, um método de comparação para estes valores deve ser empregado. Uma análise, sobre os métodos comparativos, é dada no sub-item 2.4 deste trabalho.

A Figura 3 ilustra a estrutura de *hash* perceptivo incluindo os elementos do banco de dados e de comparação (WENG; PRENEEL, 2011).

Figura 3 – Diagrama de comparação entre *hash* perceptivos

Adaptado pelo autor Fonte: Weng; Preneel, (2011).

### 2.3.2 Características do *Hash* Perceptivo

Para que os algoritmos de *hash* perceptivo sejam robustos e resistentes a ataques, duas entradas são necessárias. Uma Imagem  $I$  e uma chave secreta  $k$  e produzir um vetor binário curto  $\vec{h} = H_k(I)$  a partir de  $\{0, 1\}^n$ , ( $n$  sendo o número de *bits*).

Certas propriedades são exigidas das funções de *hash* perceptivo, por exemplo: o valor *hash* gerado a partir de imagens visualmente idênticas deve ter alta a probabilidade de ser o mesmo. Do mesmo modo, o valor *hash* gerado a partir de imagens visualmente diferentes deve ter alta a probabilidade de ser diferente. As propriedades requeridas de um algoritmo de *hash* perceptivo são, Aleatoriedade, Independência de pares e Invariância. São resumidas a seguir (KOZAT; VENKATESAN; MIHCAK, 2004):

1) Aleatoriedade (*Randomization*): Para uma entrada  $I$ , o valor *hash* deve ser distribuído de maneira uniforme entre todas as saídas possíveis de  $2^n$ :

$$\forall h \in \{0, 1\}^n, Pr\{H_k(I) = \vec{h}\} \approx 2^{-n} \quad (2.12)$$

2) Independência de pares (*Pairwise Independance*): O *hash* gerado a partir de duas imagens visualmente diferentes ( $I_1$  e  $I_2$ ) devem ser aproximadamente inde-

pendentes. Em outras palavras, tanto o tamanho do hash quanto os caracteres que o compõe devem ser construídos de tal forma que sejam resistentes a colisão:

$$\forall h_1, h_2 \in \{0, 1\}^n, Pr\{H_k(I_1) = \vec{h}_1 | H_k(I_2) = \vec{h}_2\} \approx Pr\{H_k(I_1) = \vec{h}_1\} \quad (2.13)$$

3) Invariância (*Invariance*): Para todos os distúrbios aceitáveis, a saída da função *hash* deve permanecer aproximadamente invariável. Sendo  $I$  e  $I'$  imagens visualmente similares, então:

$$\forall h \in \{0, 1\}^n, Pr\{H_k(I) = \vec{h}\} \approx Pr\{H_k(I') = \vec{h}\} \quad (2.14)$$

O termo “distúrbios aceitáveis” não é preciso, porém o que é geralmente aceito é que duas imagens são similares quando não há alterações visuais perceptíveis para o olhar humano (KOZAT; VENKATESAN; MIHCAK, 2004).

### 2.3.3 Técnicas para Geração de *Hash* Perceptivo

Os estudos relacionados ao *hash* perceptivo de imagens centralizam os esforços na fase de extração de características e podem ser classificadas de acordo com as técnicas que utilizam. As técnicas podem ser divididas (mas não se limitam) em Estatísticas, Relações, Recurso de Baixo Nível e Representações Grosseiras (*Coarse Representation*) (BHATTACHARJEE; KUTTER, 1998); (HADMI et al., 2012) comentadas a seguir.

#### 2.3.3.1 Baseados em Estatísticas

Este grupo de algoritmo extrai as informações para a geração do *hash* a partir de estatísticas, para o cálculo do valor médio dos pixels, obtendo sua variação e seu histograma (KHELIFI; JIANG, 2011); (KHELIFI; JIANG, 2010); (VENKATESAN et al., 2000). Neste sentido, (YANG; GU; NIU, 2006), propõe os seguintes passos para obter o valor *hash*:

- Passo 1: Normalizar a imagem original em um tamanho predefinido;
- Passo2: Dividir a imagem normalizada  $I$ , em blocos  $\{I_1, I_2, \dots, I_n\}$  que não se sobrepõem, dos quais  $n$  é igual ao número de blocos e o tamanho final do valor *hash*;
- Passo 3: Encriptar o índice da sequência de blocos  $\{I_1, I_2, \dots, I_n\}$  usando uma chave secreta  $k$  para obter uma sequência de blocos com ordenação diferente  $\{I'_1, I'_2, \dots, I'_n\}$ ;
- Passo 4: Calcular a sequência do valor médio  $\{M_1, M_2, \dots, M_n\}$  a partir de blocos correspondente  $\{I'_1, I'_2, \dots, I'_n\}$  e obter o valor médio de  $M_d$ , sendo:

$$M_d = (M_i)(i = 1, 2, \dots, N) \quad (2.15)$$

- Passo 5: Normalizar a sequência de valor médio em uma forma binária e obter o valor *hash*  $h$ , sendo:

$$h(i) = \begin{cases} 0, & M_i < M_d \\ 1 & M_i \geq M_d \end{cases} \quad (2.16)$$

### 2.3.3.2 Baseados em Relações

Este grupo extrai o valor *hash* a partir da relação invariável de coeficientes da transformada de cosseno (*Discrete Cousine Transform - DCT*) ou da transformada em ondeleta (*Discrete Wavelet Transform - DWT*) (LU; LIAO, 2003; LIN; CHANG, 2001).

A DCT é baseada na transformada de Fourier (DFT) porém soma apenas as funções (sequência de pontos finitos) cossenos e é frequentemente usada em processamento de sinal e imagem. Existem diversas variantes DCT, no entanto, a mais utilizada é a de tipo 2 ou DCT-II. A fórmula dessa transformada para uma matriz (ou seja uma imagem) de tamanho  $N \times M$  é:

$$F(u, v) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_0^N \sum_0^M \Lambda(i) \cdot \Lambda(j) \cdot \cos \left[ \frac{\pi \cdot u}{2 \cdot N} (2i + 1) \right] \cos \left[ \frac{\pi \cdot v}{2 \cdot M} (2j + 1) \right] \cdot f(i, j) \quad (2.17)$$

A DWT é utilizada para separar a imagem de um *pixel*. É frequentemente utilizada para processamento de sinal e imagens e especialmente para compressão com baixa perda. Os métodos baseados em DWT realizam filtros na imagem com o fim de remover os *pixels* considerados redundantes e transforma os que sobram para o domínio da frequência. Os blocos na DWT são então gerados, *low-low* (LL, produz uma aproximação da imagem), *low-high* (LH, horizontal na frequência alta), *high-low* (HL, vertical na frequência alta) e *high-high* (HH, diagonal na frequência alta) (MUTHU; RANI, 2017). A transformada DWT consiste em identificar os parâmetros  $c_k$  e  $d_{j,k}$ ,  $k \in \mathbb{N}$ ,  $j \in \mathbb{N}$  da equação:

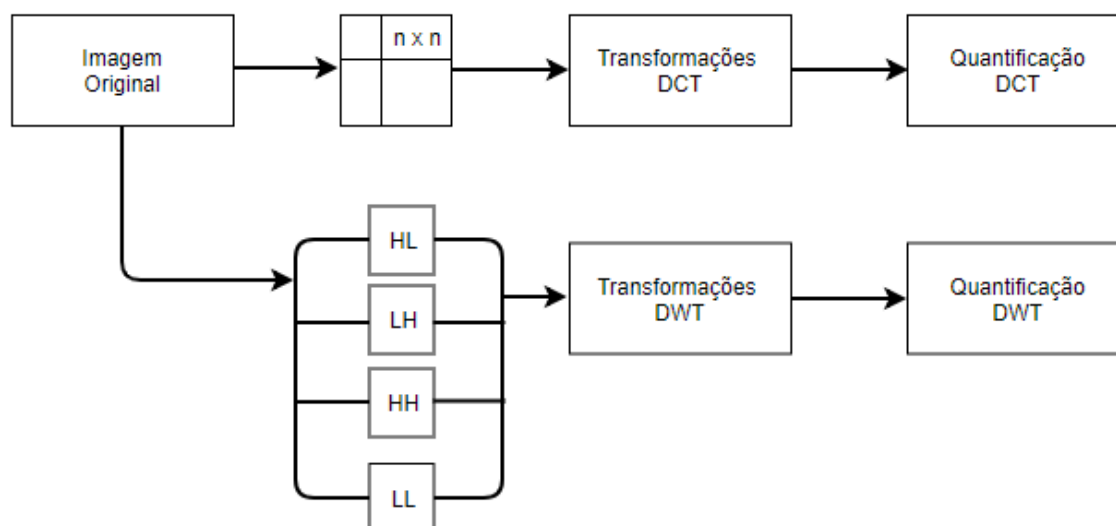
$$f(t) = \sum_{k=-\infty}^{\infty} c_k \phi(t - k) + \sum_{k=-\infty}^{\infty} \sum_{j=0}^{\infty} d_{j,k} \psi(2^j t - k), \quad (2.18)$$

de modo que  $\Phi(t)$  e  $\psi(t)$  são as funções conhecidas, respectivamente, como ondeleta pai e ondeleta mãe.

Tanto DCT quanto DWT normalizam a imagem em blocos para extração dos coeficientes. Estes métodos transformam a imagem em um bloco de proporções iguais para então realizar os cálculos no domínio da frequência.

Após passar para o domínio da frequência seja no método DCT ou no DWT, o valor *hash* é gerado a partir destes coeficientes (Figura 4) (JOSHUA; ARRIVUKANNAMMA; SATHIASEELAN, 2016).

Figura 4 – Arquitetura da DCT e DWT - Adaptado pelo autor.



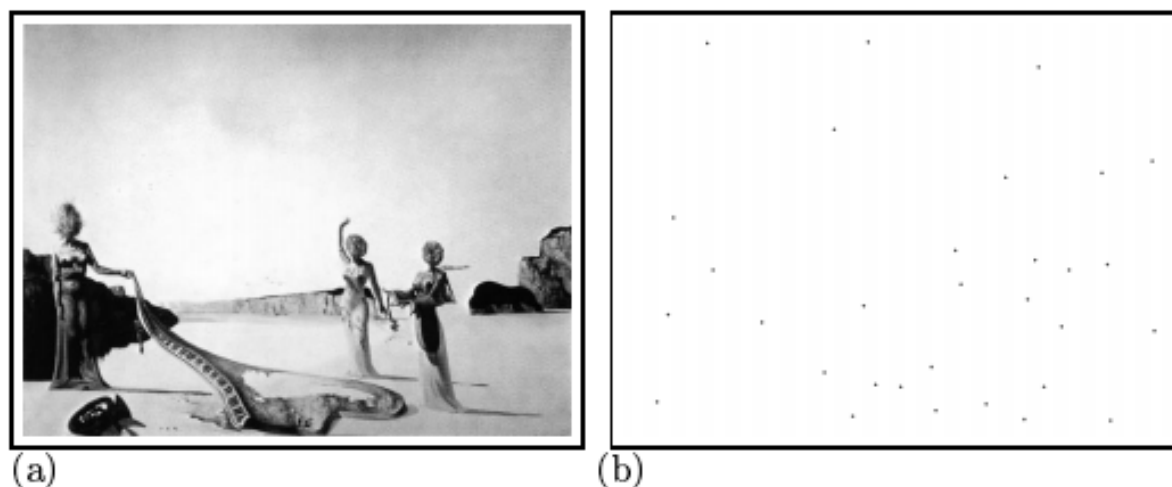
Fonte: Joshua; Arrivukannamma; Sathiaselalan (2016).

### 2.3.3.3 Baseados em Recursos de Baixo Nível

Este método, primeiro realiza a transformada DCT ou a DWT da imagem original e utiliza o próprio coeficiente para gerar o valor *hash*. Neste método são extraídas características interessantes a percepção como registro da imagem, correspondência de movimento, e até mesmo reconhecimento de faces humanas. Conforme descrito por (BHATTACHARJEE; KUTTER, 1998), a extração de características é baseada no modelo de interação em escala que detecta resoluções, linhas em uma determinada orientação e os pontos finais destas linhas (MONGA; EVANS, 2004).

A vantagem de se utilizar recursos de baixo nível é que eles são resistentes a transformações como mudança de escala, rotação, ruídos, borrões e compressão. Os cálculos resultantes da extração destas características de baixo nível são então divididos em linhas e colunas e seus valores utilizados para geração do valor *hash*. A Figura 5 apresenta pontos de extração de características.

Figura 5 – Pontos de Extração de Características

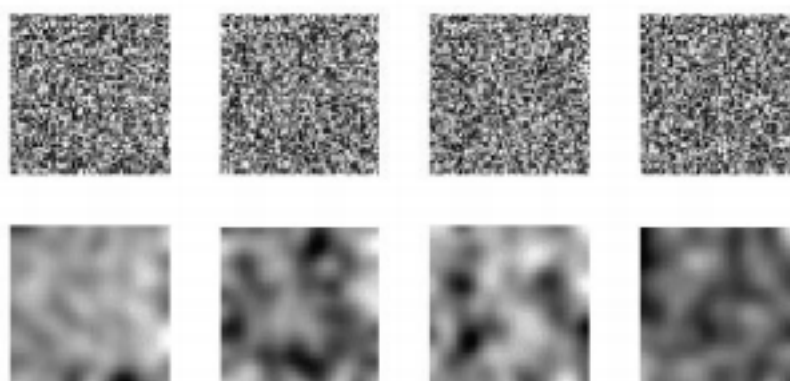


Fonte: S. Bhataclerjee; M Kutter (1998).

#### 2.3.3.4 Representações Grosseiras (*Coarse Representation*)

Este método tem por nome grosseiro (tradução direta de *coarse*) porque calcula o valor *hash* com base não na imagem propriamente dita, mas em sua representação (KOZAT; VENKATESAN; MIHCAK, 2004). Alguns autores sugerem o uso da frequência a partir da transformada de Fourier (FRIDRICH; GOLJAN, 2000) outros sugerem calcular o *hash* por meio da distribuição espacial, a partir de pontos significativos da imagem (SWAMINATHAN; MAO; WU, 2006). A Figura 6, mostra uma “representação grosseira” gerada a partir de 4 imagens distintas e é com base nestas representações que são gerados os valores *hash*.

Figura 6 – Representações geradas a partir de uma imagem



Fonte: Fridrich, J; Goljan, M (2000)

### 2.3.4 Implementações Conhecidas de *Hash* Perceptivo de Imagens

Existem poucas implementações públicas dos algoritmos de *hash* perceptivo disponível para utilização e estudo sendo apenas duas que agregam várias funções em uma única biblioteca. A biblioteca ImageHash ((BUSCHNER, 2016), disponível no endereço <https://github.com/JohannesBuchner/imagehash>, escrita em Python, e atualmente na versão 4.0, contém as implementações das funções **aHash**, **pHash**, **dHash** e **wHash** e foi utilizada para este estudo. As funções *hash* incluídas na biblioteca ImageHash em Python são equivalentes ao que foi estudado nas seções 2.3.3 até 2.3.6 e a Tabela 4, mostra a equivalência.

**Tabela 4 – Funções *hash* e equivalência científica**

Implementação ImageHash	Nome	Método Científico
aHash	<i>Average Hash</i>	Baseado em Estatísticas
pHash	<i>Perceptual Hash</i>	Baseado em Relações
dHash	<i>Difference Hash</i>	Baseado em Estatísticas
wHash	<i>Wavelet Hash</i>	Baseado em Relações

Fonte: Próprio autor

A Tabela 5 lista algumas das implementações públicas existentes. Estas implementações públicas das funções de *hash* perceptivo contém uma ou outra das muitas funções existentes.

**Tabela 5 – Implementações públicas de funções de *hash* perceptivo**

Nome	Função	Disponível em
BlockHash.io	<i>Average Hash</i>	<a href="http://blockhash.io/">http://blockhash.io/</a>
PHASH	<i>Perceptual Hash</i>	<a href="http://www.phash.org/">http://www.phash.org/</a>
ImageMagick	<i>Perceptual Hash</i>	<a href="https://www.imagemagick.org/">https://www.imagemagick.org/</a>
ImageHash	<i>Average Hash, Difference Hash, BlockHash, Perceptual Hash</i>	<a href="https://github.com/jenssegers/imagehash">https://github.com/jenssegers/imagehash</a>
ImageHash	<i>Average Hash, Difference Hash, Perceptual Hash, Wavelet Hash</i>	<a href="https://pypi.org/project/ImageHash/">https://pypi.org/project/ImageHash/</a>

Fonte: Próprio autor



A biblioteca ImageHash (<https://pypi.org/project/ImageHash/>) escrita em Python, além de possuir as vantagens inerentes da linguagem de programação que foi escrita, também é mais versátil uma vez que pode ser facilmente integrada a aplicações multiplataforma. Seus resultados podem ser mostrados nos sistemas Linux de maneira instantânea, sendo assim mais indicada para este estudo. Outras vantagens de se utilizar a biblioteca em Python incluem a possibilidade de executar os códigos através de computação distribuída e paralela.

Percebe-se que existe uma implementação homônima, ImageHash (SEGERS, 2014) mostrada na Tabela 5 que difere, entre outras coisas, na linguagem em que foi escrita, PHP. Esta biblioteca serve para aplicações voltadas para à *internet* e funções cliente/servidor. A biblioteca feita em PHP proporciona, já em seu código, as funções de comparação e distância de *Hamming*, bem como conversão de base que são úteis para o armazenamento no banco de dados. A implementação desta versão da ImageHash é mais demorada e requer a escrita de muito mais código para ser funcional e ter seus resultados demonstrados e estudados. A distância de *Hamming* é discutida na Seção 2.4.1.

Os valores gerados a partir das funções *hash* implementadas na biblioteca ImageHash, são de sessenta e quatro (64) bits, representados por uma string hexadecimal de 16 caracteres que vai de “0000000000000000” até “FFFFFFFFFFFFFFFF”. Isto quer dizer que o número de *hash* possíveis é de  $2^{64}$  ou 18.446.744.073.709.551.616 (dezoito quintilhões).

## 2.4 Métodos Comparativos

Funções de *hash* perceptivos são utilizadas para verificar similaridades entre imagens. Os valores obtidos a partir das funções de *hash* perceptivo na fase de quantificação são representados por *strings* (ASCII) ou *bits*. Esses valores não indicam a similaridade do objeto, mas representam o coeficiente de suas características visuais. Para verificar a similaridade de uma imagem com outra é necessário empregar métodos comparativos tais como Distância de Hamming (*Hamming Distance*) (HAMMING, 1950), Taxa de Erro de Bits (BER - *Bit Error Rate*) (BREED, 2003), e Correlação Cruzada (*Peak of Cross Correlation*) (BRACEWELL, 1965). Os dois primeiros métodos são classificados como métricas de distância e o último como métrica de similaridade (CHEN; MA; ZHANG, 2009). As métricas de distância são utilizadas na área de telecomunicações para verificação de erros de transmissões entre emissor e receptor. Também são utilizadas para buscas em banco de dados (SAHINALP et al., 2003), comparação de palavras (CALUDE; SALOMAA; YU, 2002) e gráficos (TORSELLO; ROWE; PELILLO, 2005). Métricas de similaridade são usadas na área médica para comparação de sequência de proteínas (SMITH; WATERMAN, 1981) e para avaliar a importância de

atributos em cálculos binários, bem como geração de entropia. As particularidades de cada métrica será discutida nos sub-itens a seguir.

### 2.4.1 Distância de Hamming

O conceito “Distância de Hamming” foi introduzido na teoria da computação em 1950 por Richard Hamming no artigo intitulado *Error detecting and error correcting codes*. Este conceito é utilizado para calcular o número de posições que duas *strings* (do mesmo tamanho) diferem entre si. Pode também ser interpretada como o número de substituições necessárias para transformar uma string em outra, sendo estas substituições chamadas de erros. As strings, objetos de comparação, podem ser elementos de sistemas numéricos ou letras. A Tabela 6 mostra alguns exemplos de aplicações da distância de Hamming.

**Tabela 6 – Exemplos de Aplicação da Distância de Hamming - Adaptado pelo autor.**

String 1	String 2	Distância de Hamming
<b>bolsa</b>	<b>louca</b>	3
123456789	133456779	2
101101	101001	1

Fonte: Zauner (2010).

Os elementos mostrados na tabela 6 são de sistemas diferentes (alfabeto, decimal e binário). A distância de Hamming é muito usada para detecção de erros por que não só encontra se há erros na transmissão, mas porque também identifica a posição do bit ou elemento com erro. Por exemplo, observa-se a sequência 101101 mostrada na tabela 6 (terceira linha), e a sequência 101001, assim temos  $A = 101101$  e  $B = 101001$ . Intuitivamente é fácil identificar que a distância de Hamming é de 1, por que o elemento na posição 3 (começando pelo zero, da esquerda para direita) da sequência A é diferente do elemento da posição 3 da sequência B. A distância de Hamming é descrita da seguinte forma (HAMMING, 1950):

Seja A um alfabeto de tamanho finito.  $x = (x_1, \dots, x_n)$  denota uma string de tamanho uniforme, enquanto  $x \in A$ . O mesmo é verdade para  $y = (y_1, \dots, y_n)$ . Então a distância de hamming  $\Delta$  entre x e y é definida como:

$$\Delta_n(x, y) = \sum_{x_i \neq y_i} 1, i = 1, \dots, n. \quad (2.19)$$

(SWAMINATHAN; MAO; WU, 2006) propõe a função da distância de Hamming normalizada  $\Delta_n$  como:

$$\Delta_n(x, y) = \frac{1}{n} \sum_{x_i \neq y_i} 1, i = 1, \dots, n. \quad (2.20)$$

A Equação 2.12 consiste em gerar uma nova sequência de bits C, que marca as posições nas quais os bits de A e B diferem entre si. Assim, utilizam-se zeros para marcar os bits que são iguais e 1 para marcar os bits que são diferentes. Em outras palavras, basta aplicar o operador XOR (OU Exclusivo) entre os números A e B. Logo C é 000100. Finalmente para obter a distância de Hamming basta contar a quantidade de números 1 que aparecem na sequência C, neste caso, apenas um.

A aplicação da distância de Hamming nos *hash* perceptivos se dá a partir de um limiar ou patamar (*threshold*) escolhido. Este limiar estará associado ao número de bits que forma o valor do *hash* perceptivo (WENG; PRENEEL, 2011). As duas imagens sendo comparadas são consideradas similares se a distância estiver abaixo deste limiar.

Existe ainda uma derivação da distância de Hamming chamada de Porcentagem de Igualdade (PI) sugerida por (VENKATESAN et al., 2000), indicando que imagens perceptualmente similares devem ter o PI alto ( $\approx 100\%$ ) e imagens perceptualmente diferentes devem ter o PI baixo ( $\approx 0\%$ ). A porcentagem de Igualdade é definida da seguinte maneira:

$$PI = 100 \cdot \Delta_n. \quad (2.21)$$

#### 2.4.2 Taxa de Erro de Bits (BER)

A taxa de erro de bit (BER) ou taxa de *bits* errados é uma importante ferramenta de detecção de erros utilizada com bastante frequência na transmissão de dados ou para indicar a qualidade de enlace de telecomunicações. Este indicador é utilizado para expressar o número de bits com erro na recepção a partir do total de *bits* enviados na transmissão. Em outras palavras BER é o número de *bits* errados identificados em comparação com o número de *bits* transmitidos (BREED, 2003).

$$BER = (\text{Número de } \textit{bits} \text{ errados}) / (\text{Número de bits transmitidos})$$

Na área de telecomunicações, BER faz uma comparação dos bits transmitidos com os recebidos, computando o número de erros e taxa (no tempo) de erros dos mesmos. Uma característica conhecida das medições de BER para telecomunicações é que o número de bits transmitidos é gigantesco e os valores usados para expressar BER são dados em porcentagem ou em potência de 10. Por exemplo, para medir a qualidade

de transmissão em redes locais LAN (*Local Area Network*) de *ethernet* a 10Mbps, é esperado um valor inferior a um bit errado para um bilhão de bits transmitidos, ou uma BER de  $10^{-9}$ . Para redes em banda larga, espera-se erro de um bit de erro em um trilhão de bits transmitidos ou BER de  $10^{-12}$ . Para redes mais velozes, como as dos sistemas ópticos a tolerância de BER é ainda menor, sendo 1 *bit* errado para  $10^{-15}$  *bits* transmitidos.

No caso de *hash* perceptivos, não há transmissão de dados, logo a taxa de erros no tempo não se faz necessária, assim os valores utilizados para expressar BER são menores.

Uma vez que a BER é utilizada para verificação de erros em bits, é necessário utilizar o valor *hash* ainda em sua forma binária. Conforme descrito por (YANG; GU; NIU, 2006), pode-se também utilizar a BER para comparar dois valores *hash* gerados pela mesma função sendo definido da seguinte forma:

Considerando a forma binária do hash perceptivo, o número de bits errados  $i$  normalizado pelo número de bits do hash perceptivo  $n$ , descreve a distância entre dois hash perceptivos.

$$\rho = \frac{i}{n} \quad (2.22)$$

visto que  $i \in \{0, 1, 2, \dots, n\}$  e  $0 \leq \rho \leq 1$ .

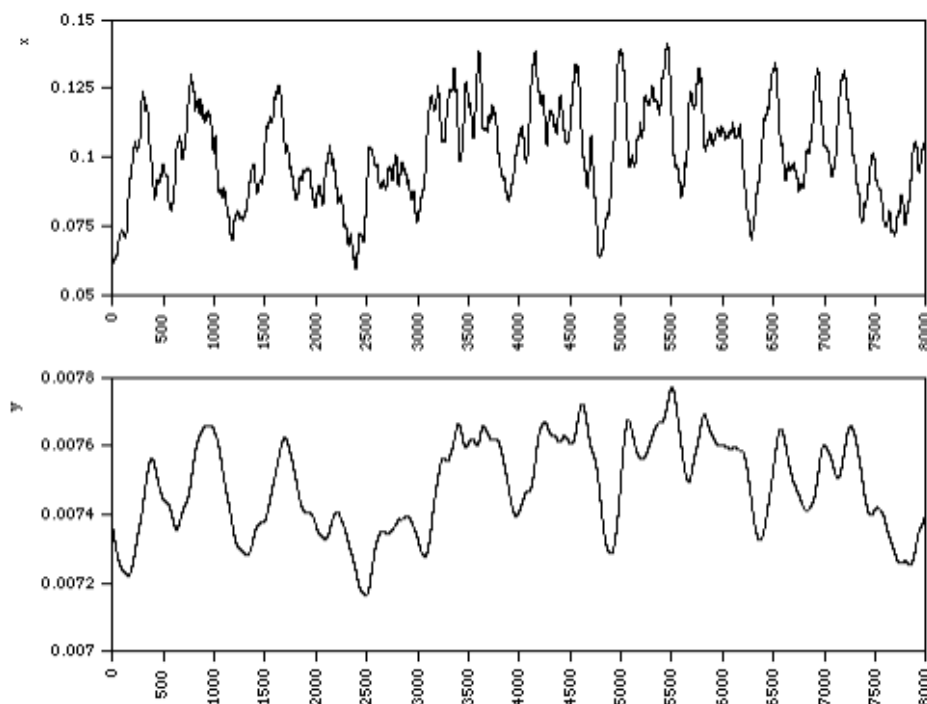
O “número de bits errados” descrito por (YANG; GU; NIU, 2006) é, na verdade, o número de bits diferentes resultantes do cálculo da distância de Hamming. Este valor é então dividido pelo número de *bits* que compõe o valor *hash*. O resultado da equação proposta vai variar entre 0 e 1. Isto quer dizer que quanto menor for a BER, maior a probabilidade dos objetos verificados (neste caso as imagens) serem idênticos (ZAUNER, 2010).

### 2.4.3 Correlação Cruzada (*Cross Correlation*)

Correlação cruzada é classificada como uma métrica de similaridade, e ao contrário das métricas de distância mencionadas nos sub-itens 2.4.1 e 2.4.2, que medem as diferenças de posições a partir de dois valores, esta métrica utiliza os *pixels* da imagem para fazer a comparação. Esta função é fortemente ligada com o termo Convolução, que a partir da sobreposição de duas funções (sinais), produz uma terceira (integral de superposição). A correlação cruzada aplicada para análise de similaridade de imagens, ao invés de “somar” os sinais, compara um sinal com outro por meio da soma dos pixels. Quando (e se) a imagem ou espaço sendo analisado for similar, os valores são somados, aumentando assim o coeficiente exponencialmente e causando

“picos” (FISHER; OLIVER, 1998). O exemplo da Figura 7 mostra dois sinais a serem comparados pela correlação cruzada. A Figura 8 mostra o resultado (BOURKE, 1996):

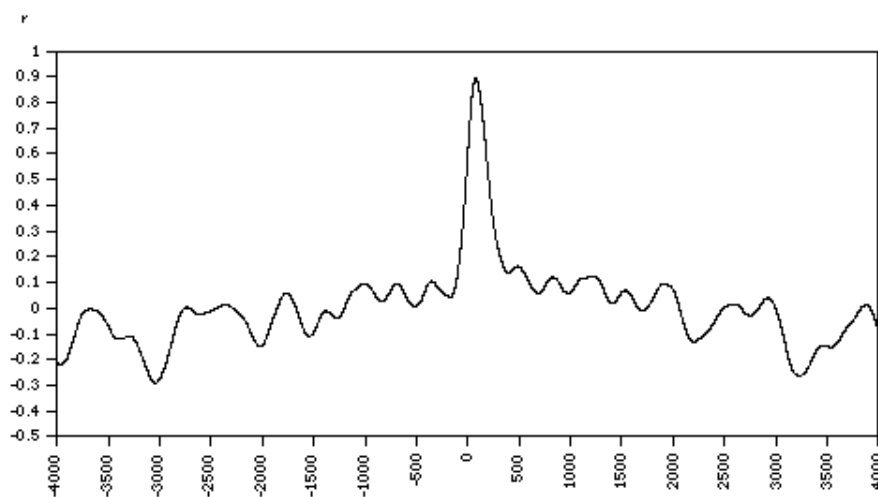
**Figura 7 – Dois exemplos de sinal x, y para aplicação da correlação cruzada.**



Fonte: P.Bourke (1996)

Pode-se observar uma forte “correlação” no ponto 4000 das duas imagens, como mostrado na Figura 8 a seguir.

**Figura 8 – Gráfico mostrando a correlação entre os sinais x e y.**



Fonte: P.Bourke (1996)

O cálculo da correlação cruzada é definido da seguinte forma (ZAUNER, 2010):

Considerando duas séries  $x(t)$  e  $y(t)$ , a função  $r_{xy}(T)$  descreve a concorrência destes dois sinais com relação ao deslocamento no tempo  $T$ .

$$r_{xy} = \sum_{m=-\infty}^{\infty} x[t]y[t+T]dt \quad (2.23)$$

Pode-se argumentar que a correlação cruzada pode ser utilizada para a autenticação de imagens inutilizando-se assim a necessidade dos *hash* perceptivos. No entanto, o que se observa é que a correlação cruzada é bastante sensível as alterações do sinal como, por exemplo, ruídos na imagem, rotação e compressão. Logo, ela não se mostra eficaz como método para autenticação com base na percepção visual de objetos com potencial de alterações. Outro ponto a ser observado é que, como a avaliação da correlação cruzada é feita pixel a pixel, o processo pode ser demorado e custoso em recursos computacionais (FISHER; OLIVER, 1998). O que se propõe então é a utilização da correlação cruzada na verificação de similaridade entre os valores *hash*.

## 2.5 Marca D'água Digital ou *Watermarking*

Marca d'água digital é um método utilizado para transmitir informações associadas à arquivos multimídias (áudio, imagens, vídeo, texto) incorporando-as no próprio conteúdo. A marca d'água digital é inserida através de modificações leves e comumente imperceptíveis na mídia original. Quando adequadamente projetada, a marca d'água digital pode resistir à degradação de conteúdo e permite que seus criadores recuperem as informações inseridas de forma legível e a partir de qualquer formato da mídia (KALKER; HAITSMAN; OOSTVEEN, 2001). A marca pode ser inserida em imagens através de técnicas similares com as de *Hash* perceptivo nos domínios da frequência, cores e gradiente. Em arquivos de vídeo a marca pode ser inserida nos quadros ou nos quadros chave (VURAL; BARAKLI, 2015). Em arquivos de áudio um som distinto reproduzido em frequência imperceptível pelo ouvido humano pode ser inserido e considerado como marca d'água digital (WU et al., 2005). Ainda é possível adicionar marca d'água digital em programas e textos. No caso de texto a marca pode ser inserida no nível da impressão ou de forma semântica, quando a organização das palavras e uso de expressões constituem a assinatura da obra (REZA et al., 2012).

O surgimento da Marca d'água digital em 1995 revelou-se uma área de bastante interesse para o mercado de audiovisual (ABDEL-MOTTALEB; KRISHNAMACHARI; VAITHILINGAM, 1999). Uma análise da literatura disponível mostra que a maioria dos trabalhos tem imagens como alvo (KASKALIS; PITAS, 1995), (COX et al., 1997). No entanto, a produção de artigos relacionados à áudio e vídeo passou a crescer tendo como

foco a proteção de propriedade intelectual (REZA et al., 2012). Essa observação explica o grande interesse no aspecto de segurança da marca d'água, e um grande número de publicações, portanto, focaliza em contramedidas e ataques intencionais (KUTTER; VOLOSHYNOVSKIY; VOLOSHYNOVSKIY, 2000), (VOLOSHYNOVSKIY et al., 2001), (DEGUILLAME; CSURKA; PUN, 2001).

### 2.5.1 Correlação entre *Hash* Perceptivo e Marca D'água

Nesta seção observam-se as similaridades e diferenças entre *Hash* Perceptivo e Marca d'água digital. Em termos gerais os pontos que podem ser observados entre as duas tecnologias são os de robustez, impacto, ambiente, mensagem, probabilidade de colisão, e aplicabilidade, a saber:

- Robustez pode ser definida como a habilidade de identificar dois objetos como perceptivamente similares. Para os algoritmos de *hash* perceptivo, robustez, significa que o valor *hash* gerado seja resistente à alteração e ataques no objeto, no caso da marca d'água digital robustez significa que a marca persiste à compressões e codificações dos diversos formatos de mídia (FRIDRICH; GOLJAN, 2000).
- Marca d'água digital tem impacto direto no conteúdo e ainda que de maneira imperceptível altera a mídia original, isto é chamado de técnica ativa. *Hash* Perceptivo, em contrapartida, não altera de forma alguma o conteúdo do objeto e é uma técnica passiva. Esta diferença é o que faz com que técnicas de *hash* perceptivo sejam preferíveis por artistas e estúdios (KALKER; HAITSMAN; OOSTVEEN, 2001).
- *Hash* perceptivo necessita obrigatoriamente de uma base de referência composta de valores gerados previamente e disponíveis para comparação. Este ambiente conectado é o que difere *hash* perceptivo das técnicas de marca d'água uma vez que o segundo não necessita de uma conexão com um banco de dados e é operado de maneira autônoma e independente necessitando apenas da mídia da qual se deseja extrair a marca (HADMI et al., 2012).
- O canal de transmissão para aplicação de marca d'água permite a inserção de mensagens de tamanho variável, considerando-se apenas os limites de tamanho disponível. Isto significa que uma mídia digital pode carregar diferentes mensagens a partir de algoritmos distintos. *Hash* perceptivo, está associado ao conteúdo da mídia e carrega consigo apenas uma informação ou mensagem, que é o valor *hash* final (BRIN; DAVIS; GARCÍA-MOLINA, 1995).

- Tanto marca d'água digital quanto *hash* perceptivo para que sejam robustos e eficientes devem ter independência de pares conforme visto na Seção 2.3.2. Isto quer dizer que não se deve ter na mesma imagem duas marcas d'água idênticas ou dois valores *hash* iguais a partir de imagens diferentes. Outro fator interessante existente nas técnicas de *hash* perceptivo é que a decisão a respeito do fator de similaridade é uma questão subjetiva, situação que não existe em marca d'água digital (KALKER; HAITSMÁ; OOSTVEEN, 2001).
- A aplicabilidade do *hash* perceptivo é muito superior às técnicas de marca d'água digital. Marca d'água necessita ter o conteúdo da mídia alterado, fazendo com que seja impossível aplicá-lo nos objetos que já foram distribuídos. A identificação da marca também adiciona desafios, já que não só os donos do conteúdo precisam desejar e autorizar a inserção da marca d'água em suas mídias, como os distribuidores também precisam permitir a monitoração e análise para a busca da mesma. Estes desafios não existem em técnicas de *hash* perceptivo que pode ser aplicado em qualquer obra sem a necessidade de participação de terceiros (KALKER; HAITSMÁ; OOSTVEEN, 2001).

## 2.6 Composição dos Arquivos de Vídeo

Um filme, vídeo ou animação é composto por centenas/milhares de imagens, chamados de quadros, que são movimentados em uma determinada ordem e velocidade. Durante a exibição do vídeo cada uma destas imagens é mostrada durante um período curto de tempo, uma fração de segundo (  $1/24$ ,  $1/25$ ,  $1/30$ ,  $1/60$ ) que, por virtude do fenômeno chamado Persistência Retiniana, existente na visão humana, “cola” os quadros um-a-um dando a percepção de uma única ação e movimento. Um objeto colocado à esquerda num quadro e aparecendo à direita no quadro seguinte, cria a ilusão de que o objeto se desloca da esquerda para a direita. O termo “quadros por segundo”, representado por fps (*frames per second*) se refere a frequência (taxa) com que os quadros do vídeo são mostrados na tela. O sistema visual humano pode entender ou processar de 10 a 12 quadros por segundo e identificá-los como imagens únicas. Frequências mais altas são percebidas como movimento (READ; MEYER, 2000).

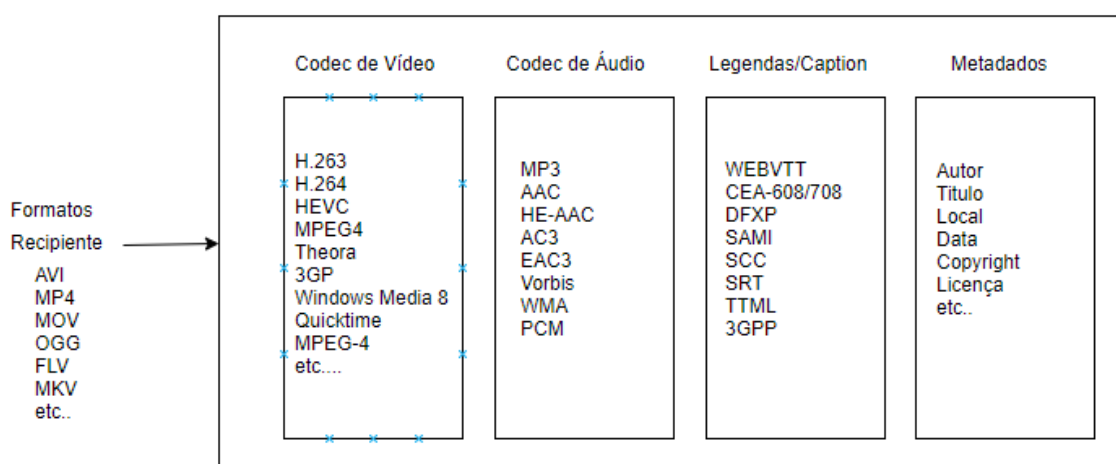
Quando observado de perto o arquivo de vídeo é formado por um recipiente ou *container*, *codecs* (áudio e vídeo), legendas e metadados. O *container* é um meta-arquivo que especifica como diferentes tipos de arquivo e dados co-existem em um mesmo arquivo de computador. No caso de arquivos de vídeo, o *container* descreve como o vídeo, áudio a legenda e outras informações devem ser executadas no computador. O uso de um determinado *container* em detrimento de outro se dá por razões relacionadas a popularidade entre os usuários, tamanho do arquivo gerado, funcio-



nalidades suportadas pelo *container* tais como capítulos, legendas, metadados ou ainda funcionalidades que facilitam sua transmissão (VERDOLIVA, 2016). A motivação para o uso do container é a de prover ao usuário um único arquivo para execução, considerando que vídeo e áudio são executados por programas diferentes. Embora o container carregue consigo informações a respeito dos arquivos que ele contém, o container em si não executa ou possui a inteligência para decodificar o vídeo e o áudio, isto é realizado por um programa chamado de *Codec*.

*Codec* é uma abreviação de codificação e decodificação. São os *codecs* quem possuem os algoritmos que fazem a compressão e descompressão do vídeo, sendo que os mais avançados executam esta tarefa sem que haja perda de dados no processo. Este processo é necessário para transformar ou representar um dado analógico em seu respectivo formato digital. Por exemplo, nos caso de vídeo, a conversão de cores (Chroma) e Luminância (Luma) para o espaço de cores digital *YCbCr* ou *RGB* (KONDOZ, 2009). O mesmo se dá para os codecs de áudio, que tem como objetivo transformar os dados analógicos do som no menor número possível de bits, preservando a fidelidade do sinal (NOWAK; ZABIEROWSKI, 2011). Adicionalmente, o vídeo pode ser composto por arquivos de texto, para representar as legendas e metadados com informações relevantes para identificação do arquivo como autor, data, tamanho, local etc. A Figura 9 representa esquematicamente como um arquivo de vídeo é composto .

**Figura 9 – Representação da composição de um arquivo de vídeo**



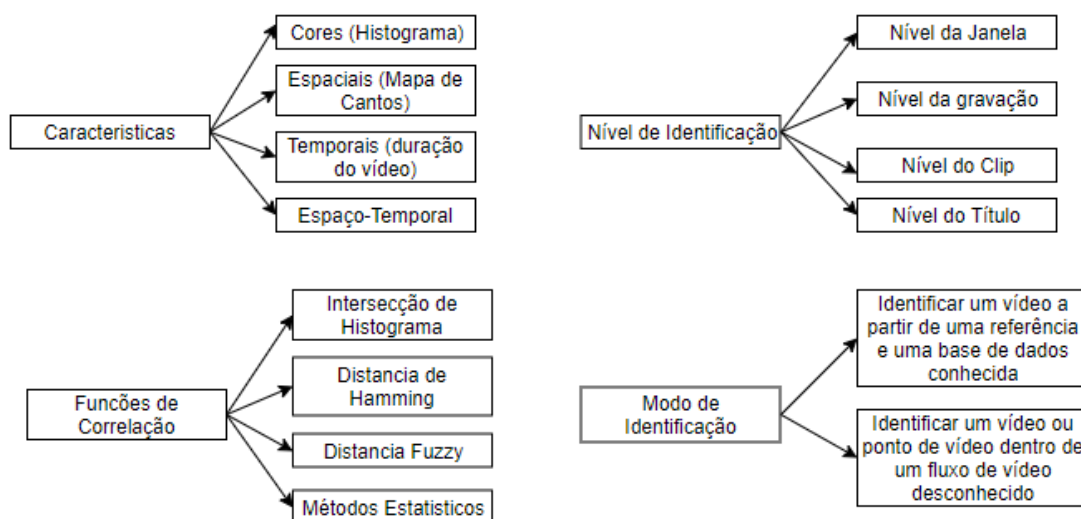
Fonte: w3.org (2018)

Para executar o estudo proposto neste trabalho, será necessário concentrar-se na porção dos quadros do vídeo, não sendo necessário considerar os codecs, áudio, legendas e metadados.

## 2.7 Métodos de Identificação de Cópia de Vídeo

Identificação de vídeo com base no conteúdo (*Content Based Copy Retrieval*) CBCR, *Video Perceptual Hashing* ou *Vídeo Fingerprinting* (YANG; SUN; TIAN, 2003), são alguns dos nomes dado às técnicas de identificação de cópia de vídeo. Identificação de vídeo assim como os métodos para identificação de imagens são áreas que atraem bastante interesse da comunidade técnica especialmente considerando os números relacionados aos ganhos financeiros provenientes do mercado de vídeo e as perdas geradas por meio da pirataria (JOLY; BUISSON; FRÉLICOT, 2007). De maneira resumida estes métodos e técnicas são categorizados e diferenciados de acordo com suas características de identificação, funções de correlação, níveis e modos de indentificação (granularidade) (YANG; SUN; TIAN, 2003) como mostrado na Figura 10.

**Figura 10 – Resumo dos métodos de identificação de vídeo com base no conteúdo**



Fonte: Yang, Sun, Tian (2003).

Similarmente com o que foi apresentado na Seção 2.3.1 sobre *Hash* perceptivo de imagens, a extração de características é um passo importante na identificação de vídeos. Informações como cores, resolução, tempo do vídeo, número de quadros, dentre outras, são utilizadas para geração do valor *hash* para vídeos. No entanto, o desafio encontrado na identificação e autenticação de vídeo é que existe um modo ou cenário diferente quando comparado com a identificação de imagens. A cópia a ser identificada pode se encontrar dentro de um fluxo, como no caso de transmissões ao vivo ou *broadcast* e não em uma base estática como é o caso das imagens. Nestes casos, a existência da cópia é volátil e limitada às condições temporais impostas por quem a detêm. Por exemplo, pode-se supor que um determinado jogo de futebol

(Corinthians x São Paulo), que deveria estar limitado somente às fontes conhecidas, é retransmitido sem autorização de seus donos em alguma plataforma da internet. Este jogo tem começo, meio e fim, portanto sua transmissão e identificação estão sujeitas ao tempo de duração do jogo (90 minutos). Isto leva à outro ponto de consideração que é o nível de identificação, que propõe ao usuário qual o ponto ou forma na qual se deseja identificar o vídeo - através de seu título, clips (grupos de quadros), formato de gravação, janela (tempo) e assim por diante (JOLY; BUISSON; FRÉLICOT, 2007; YANG; SUN; TIAN, 2003).

Em virtude destas características, a identificação de vídeo e os estudos de *hash* perceptivo para vídeos são bem mais complexos e estão divididos em duas classes, a saber, àqueles que utilizam os quadros do vídeo para a geração do valor *hash* e os que utilizam o vídeo inteiro (SU; HUANG; GAO, 2009).

Os que utilizam dos quadros do vídeo o fazem para executarem as operações de extração de características que geralmente estão associadas às variações de cores, gradiente e frequência (METZNER et al., 2011), (LEE; YOO, 2008). Outros ainda extraem a partir dos quadros, o *Key Frames* (Quadros Chave), e a partir deste realizam a extração das características do vídeo (MAANI; TSAFTARIS; KATSAGGELOS, 2008), (LEE; YOO; KALKER, 2009). *Quadro Chaves* são os que apresentam diferenças nas ações de um personagem ou cena, uma mudança crucial no movimento. A outra porção dos artigos estudados não fazem a extração dos quadros para geração do valor *hash* mas utilizam o vídeo em sua plenitude. Para isto executam cálculos DCT, transformações baseadas em luminância e geração aleatória de resumos/amostras do vídeo (COSKUN; SANKUR; MEMON, 2006), (CHEUNG; ZAKHOR, 2003). A metodologia proposta neste trabalho é apresentada a seguir.

### 3 Materiais e Métodos

Nas seções anteriores, revisou-se a teoria relacionada aos *hash* perceptivos, métodos comparativos e as implementações de *hash* perceptivo públicas conhecidas. Também revisou-se a composição de arquivos de vídeos, bem como os algoritmos específicos existentes para geração de *hash* a partir desse tipo de mídia. Nesse capítulo, apresentam-se as técnicas e métodos que foram utilizados para a aplicação de algoritmos de *hash* perceptivo de imagens, em arquivos de vídeo. Para implementação do estudo proposto nesta dissertação, um programa foi criado pelo autor e está disponível de forma pública no endereço <https://github.com/TorresAndy/4Hash>. Os códigos gerados podem ser executados em sistemas Windows e Linux, no entanto, por questões de afinidade do autor, os exemplos, amostras e testes foram executados em sistema Linux Debian (9). Mais informações a respeito do programa e seus detalhes serão explorados nas seções subsequentes.

#### 3.1 Biblioteca de Funções Hash

Conforme descrito na Seção 2.3.4, a biblioteca ImageHash escrita em Python (<https://pypi.org/project/ImageHash/>) e PHP possuem similaridades, mas não são exatamente iguais. O processo de decisão entre a biblioteca escrita em linguagem Python e outra escrita em linguagem PHP se baseia na facilidade de instalação e testes no ambiente Linux. Verifica-se que a visualização de saída de comandos é mais amigável quando se utiliza a linguagem Python. Por esta razão, foi selecionada a biblioteca escrita em Python ImageHash, na versão 4.0. Um valor *hash* foi gerado para cada quadro do vídeo de referência e de comparação, sendo mostrado a seguir o resumo da versão utilizada, extraído do sistema Linux utilizado para os testes.

##### Código 3.1 – Resumo da biblioteca ImageHash instalada em sistema Linux

```
$ pip show imagehash
Name: ImageHash
Version: 4.0
Summary: Image Hashing library
Home-page: https://github.com/JohannesBuchner/imagehash
Author: Johannes Buchner
Author-email: buchner.johannes@gmx.at
License: BSD 2-clause (see LICENSE file)
Location: /usr/local/lib/python2.7/dist-packages
Requires: pywavelets, numpy, scipy, six, pillow
```

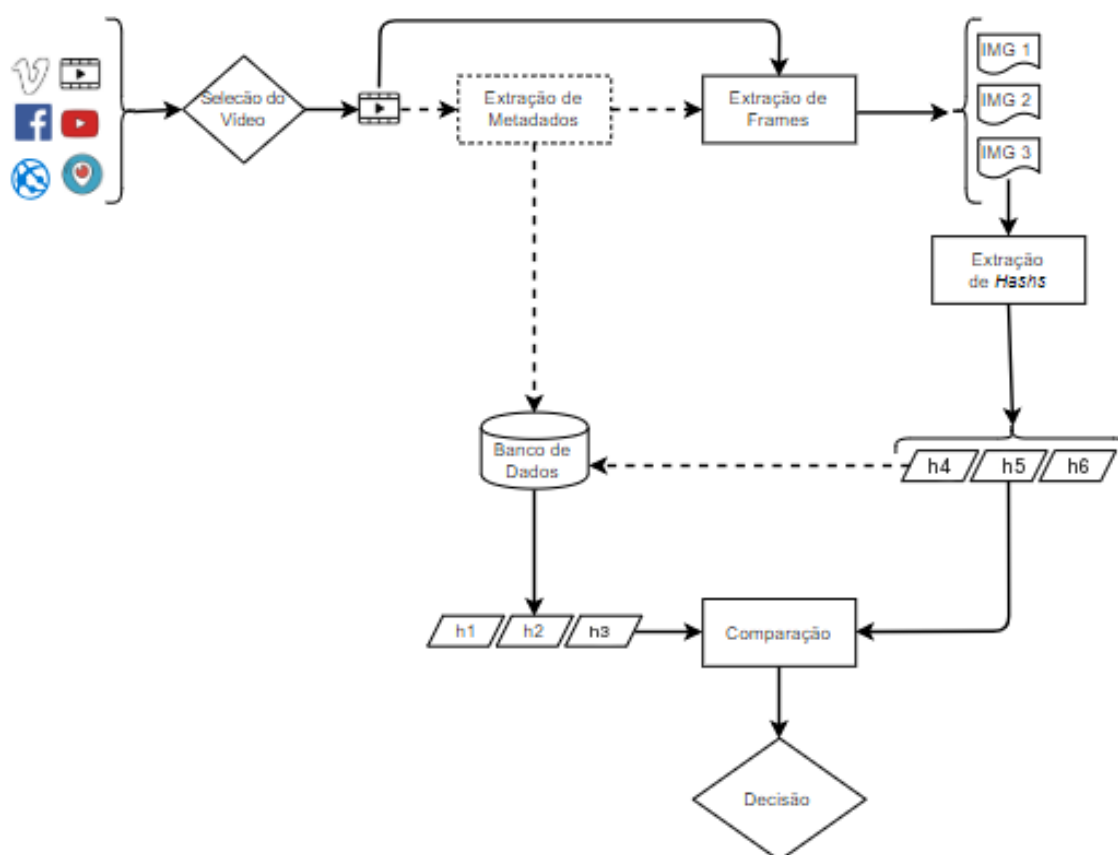
Nota-se que a última linha do comando refere-se às dependências da biblioteca ImageHash, sendo os programas *pywavelets*, *numpy*, *scipy*, *six* e *pillow*, automaticamente instalados pelo sistema Linux.

A biblioteca ImageHash tem implementado os algoritmos aHash, dHash, pHash e wHash, que recebem como parâmetro um objeto (image) e retornam o valor *hash* como resultado, guardada na variável (h). O código (Apêndice A.1), criado pelo autor, mostra como executar e receber o valor *hash* utilizando a biblioteca ImageHash em Python.

### 3.2 Estrutura

Os elementos propostos neste trabalho, vistos a partir da perspectiva de *hash* perceptivos de imagens para identificação de vídeos, são parecidos com os mostrados na Seção 2.3. Porém, é necessário adicionar alguns processos que se fazem úteis na identificação dos objetos. A Figura 11 (criada pelo autor), a seguir, mostra de maneira visual a estrutura completa, considerando os processos adicionais.

Figura 11 – Estrutura de Hash Perceptivo em Arquivos de Vídeos.



Fonte: Próprio autor

A seguir, aborda-se o papel de cada um dos passos da estrutura proposta de *hash* perceptivo para vídeos.

### 3.2.1 Seleção do Vídeo

Parte-se do pressuposto que o usuário interessado em utilizar *hash* perceptivo tenha posse do arquivo de referência, não sendo necessário o emprego de mecanismos de buscas ou investigação. A seleção do vídeo de referência deve atender apenas aos critérios estabelecidos pelo proprietário. Pode-se esperar um único arquivo como referência ou uma biblioteca de vídeos de onde serão gerados os *hash* para comparação posterior. O arquivo de comparação, no entanto, pode ser obtido de várias fontes diferentes, de onde se supõe que a reprodução esteja sendo feita de forma não autorizada. As fontes possíveis para se obter o arquivo de referência podem ser sítios *web*, *blogs*, redes sociais, plataformas digitais, redes *peer-to-peer* (ponto-a-ponto) e outros. Neste trabalho, foram utilizados uma série de vídeos curtos, chamados de *trailers*, disponíveis de forma gratuita no sítio web <http://www.davestrailerpage.co.uk>.

Os vídeos são oferecidos de forma gratuita, em vários formatos, tamanhos e tempos de execução. Na Tabela 7, mostra-se os vídeos utilizados e as informações (metadados) que serão guardadas no banco de dados.

**Tabela 7 – Base de vídeos para os testes**

Nome do Arquivo	Resolução	Duração (segundos)	Quadros	Execução (fps)
almost-almost-famous-trailer-1-ca_1a_h720p.mov	1280×720	102.75	2440	23.98
aquaman-trailer-3_h720p.mov	1280×544	150.08	3599	23.98
beyond-white-space-trailer-1_h720p.mov	1280×544	100.71	2415	23.98
bounty-killer-trailer-1_h720p.mov	1280×544	97.29	2333	23.98
captainamericacivilwar-tlr2_h720p.mov	1280×544	144.62	3468	23.98

Nome do Arquivo	Resolução	Duração (segundos)	Quadros	Execução (fps)
creed_trailer_2_txd_stereo_gc_h720p1280x720	1280×720	153.09	3671	23.98
dead-in-a-week-trailer-1_h720p.mov	1280×688	132.11	3168	23.98
dumbo-trailer-2_h720p.mov	1280×688	140.83	3377	23.98
early-man-trailer-2_h720p.mov	1280×720	141.12	3384	23.98
ferdinand-trailer-1_h720p.mov	1280×544	150.96	3620	23.98
insideout-usca-15sectease_h720p.mov	1280×720	16.81	403	23.98
kungfupanda3-tlr2_h720p.mov	1280×544	150.46	3611	23.98
mad_max_fury_road_trailer_2-720p.mov	1280×720	153.13	3672	23.98
moana-trailer-2_h720p.mov	1280×544	151.04	3622	23.98
once-upon-a-deadpool-trailer-1_h720p.mov	1280×544	60.01	1439	23.98
pokemon-detective-pikachu-trailer-1_h720p.mov	1280×544	148.33	3557	23.98
secretlifeofpets-tlr2_h720p.mov	1280×688	168.77	4047	23.98
the-vanishing-trailer-1_h720p.mov	1280×544	152.84	3665	23.98
wall-e-tlr2_h720p.mov	1280×544	150.28	4504	23.97
wonder-woman-trailer-3_h720p.mov	1280×544	145.12	3480	23.98

Criado pelo autor

Para este estudo, utilizou-se um valor de *hash* de sessenta e quatro (64) bits, conforme mencionado na seção 2.3.4. Considerando a discussão apresentada na seção 2.2.2, o tamanho da base de referência deverá ser de  $2^{\frac{64}{2}}$  antes que haja a primeira colisão. Ou seja, a base de referência poderá ter com segurança  $2^{32}$  registros ou 4.294.967.296 (4 bilhões de valores *hash*).

Na prática, considerando o tempo de reprodução médio de um filme popular (1h e 30m) executado a 25 quadros por segundo (540000), o número de filmes possíveis para armazenagem de valores *hash* e análise seria de aproximadamente 7950, o que é mais que suficiente para os testes apresentados.

### 3.2.2 Extração de Metadados

Não costuma ser boa prática armazenar arquivos de vídeo dentro do banco de dados, visto que o tamanho da mídia pode alcançar dezenas de *Gigabytes*. É necessário armazenar informações que ajudem a identificar os objetos no momento da comparação, tais como localização, data, tamanho, nome, título, *hash* de integridade e outros. Os metadados tem importância para evitar que comparações do mesmo arquivo sejam executadas. Embora a maioria dos metadados sobre o arquivo de comparação sejam opcionais, pode haver a necessidade de utilizar certas informações como a fonte do arquivo, *link*, data, hora da obtenção, tamanho, títulos e outros, com o fito de utilizá-los como prova jurídica em casos de litígio. Para isto o programa OpenCV - *Open Source Computer Vision Library* (<https://opencv.org/>), que é uma ferramenta de código aberto e multi-plataforma utilizada para analisar, gravar, converter e manipular áudio e vídeo, foi utilizado. Neste trabalho, foram armazenados os seguintes metadados a respeito dos arquivos de vídeo de referência e candidato:

- Local do arquivo
- Nome do arquivo
- *Hash* SHA do vídeo
- Resolução do vídeo
- Duração do vídeo
- Número de quadros
- Quadros por segundo

### 3.2.3 Extração de Quadros

Para realização desta dissertação, foi necessário extrair os quadros do vídeo de referência e também do vídeo de comparação e o programa OpenCV também foi utilizado para esta parte do processo. O número de quadros por segundo, utilizados na composição de vídeos, está relacionado com o tipo de audiência e tecnologia de visualização. No cinema, por exemplo, costuma-se utilizar 24 quadros por segundo como padrão. Em transmissões para televisão utiliza-se de 25 a 30 quadros por segundo. Tecnologias mais recentes usadas em produções de alta definição (HD, UHD, 4K) e produções de jogos (videogame) utilizam até 60 quadros por segundo. É possível simular o número de imagens resultantes do uso do programa OpenCV com a seguinte fórmula:



$$I \approx Qt \quad (3.1)$$

visto que  $I$  é número de imagens,  $Q$  é o número de quadros por segundo e  $t$ , o tempo total do arquivo de vídeo. Assim, se para o teste for utilizado um vídeo com duração de dez segundos e taxa de 24 quadros por segundo, obtem-se 240 imagens para serem analisadas. A Figura 12, a seguir, mostra a quantidade teórica de imagens extraídas a partir dos quadros do vídeo, considerando as taxas de reprodução mais comuns.

**Figura 12 – Número de quadros de acordo com o tamanho do vídeo em função do tempo**

Tempo de Reprodução			Quadros por Segundo					
Segundos	Minutos	Horas	12	24	25	30	48	60
			Número de quadros (imagens) gerados					
10	-	-	120	240	250	300	480	600
60	-	-	720	1440	1500	1800	2880	3600
600	10	-	7200	14400	15000	18000	28800	36000
1200	20	-	14400	28800	30000	36000	57600	72000
1800	30	-	21600	43200	45000	54000	86400	108000
3600	60	1	43200	86400	90000	108000	172800	216000
7200	120	2	86400	172800	180000	216000	345600	432000
9000	150	2,5	108000	216000	225000	270000	432000	540000

Fonte: Próprio autor

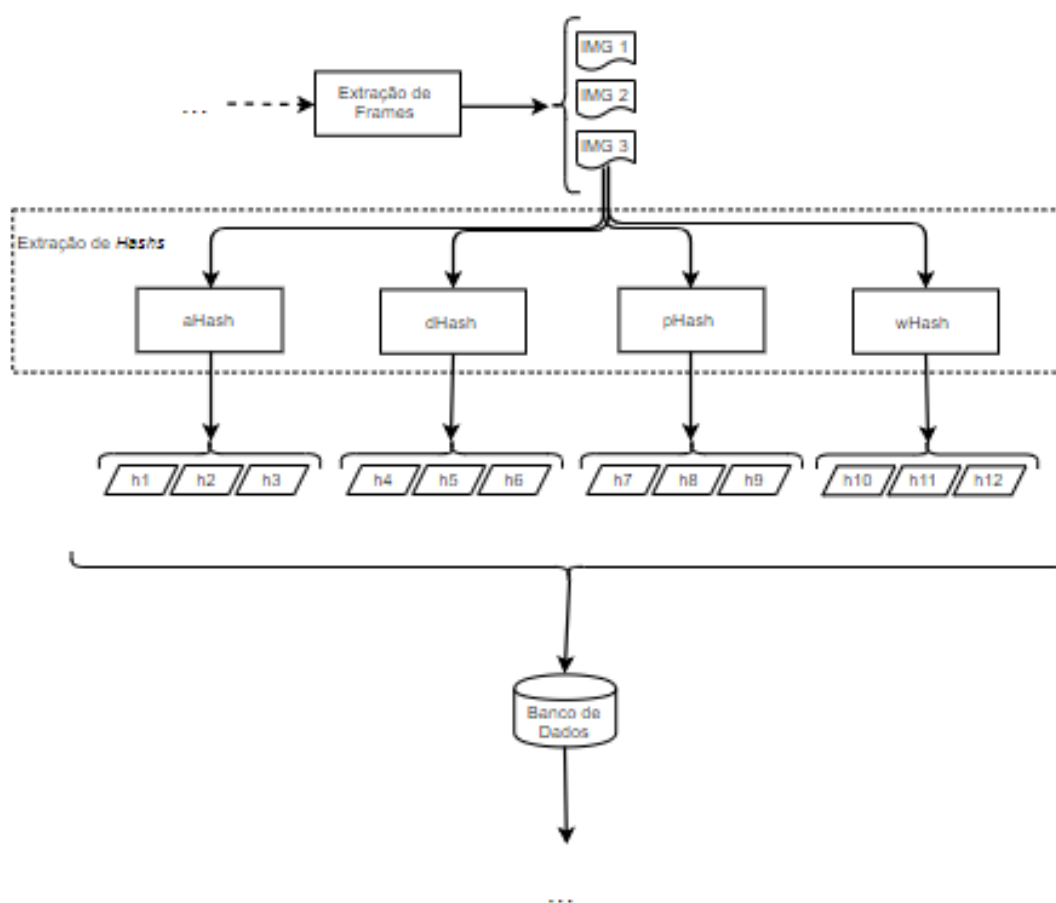
A escolha da extensão ou tipo de arquivo gerado, a partir da extração de quadros, tem impacto direto no desempenho do *software* OpenCV, espaço em disco e consequentemente no algoritmo de *hash* perceptivo. Para esta dissertação, foram realizados testes com as extensões BMP (*Bit Map Image File*), JPEG (*Joint Photographics Experts Group*) e PNG (*Portable Network Graphics*). Por questões de praticidade e facilidades de manipulação, JPEG é a extensão utilizada neste estudo. O código (Apêndice B.1), criado pelo autor, executa a extração dos quadros de acordo com as informações do vídeo e os guarda em uma pasta para o processamento dos respectivos valores *hash*.

O código possui um contador de tempo para verificação do desempenho da extração dos quadros. Estes valores são mostrados na seção de resultados deste estudo. Os quadros, após serem extraídos, são guardados em uma pasta com nome de um identificador de seu vídeo. Este identificador nada mais é do que um valor *hash* criado por meio da função SHA1 (LINDENBERG; WIRT, 2005), contendo 160 bits, representado por um número hexadecimal de 40 dígitos. Estas imagens ou quadros são então usados como parâmetros para extração dos valores *hash*.

### 3.2.4 Extração de Hash

A biblioteca Python ImageHash na versão 4.0 foi utilizada e contém as implementações **aHash**, **pHash**, **dHash** e **wHash**, conforme visto na Seção 3.1. Na Figura 11, a extração dos *hash* é simplificada em apenas um componente, porém é útil salientar que o objetivo é combinar os resultados gerados a partir dos quatro algoritmos (simultâneo ou sequencial) em um único coeficiente de similaridade. Ou seja, foi gerado a partir de cada quadro, quatro valores *hash* distintos e armazenados no banco de dados. Os detalhes sobre a busca, identificação e comparação dos *hash* são apresentados na Seção 3.4. Na Figura 13, o elemento “Extração de Hash” é expandido conforme mostrado a seguir.

Figura 13 – Visualização expandida da extração de hash



Fonte: Próprio autor

Observa-se que o número de quadros, conforme mostrado na Seção 3.2.3, tem relação direta com o número de *hash* gerados. Neste caso, multiplica-se o número de quadros total pelo número de funções *hash* utilizada (quatro). A Figura 14 mostra a quantidade estimada de *hash* a serem produzidos a partir dos quadros, em função do

tempo do vídeo.

**Figura 14 – Número de *hash* de acordo com o tamanho do vídeo em função do tempo**

Tempo de Reprodução			Quadros por Segundo					
Segundos	Minutos	Horas	12	24	25	30	48	60
			Número de Hashes					
10	-	-	480	960	1000	1200	1920	2400
60	-	-	2880	5760	6000	7200	11520	14400
600	10	-	28800	57600	60000	72000	115200	144000
1200	20	-	57600	115200	120000	144000	230400	288000
1800	30	-	86400	172800	180000	216000	345600	432000
3600	60	1	172800	345600	360000	432000	691200	864000
7200	120	2	345600	691200	720000	864000	1382400	1728000
9000	150	2,5	432000	864000	900000	1080000	1728000	2160000

Fonte: Próprio autor

Um programa (Apêndice C.1) escrito em Python foi criado pelo autor para que, a partir dos quadros recém armazenados, possa gerar os respectivos valores *hash* e guarda-los no banco de dados.

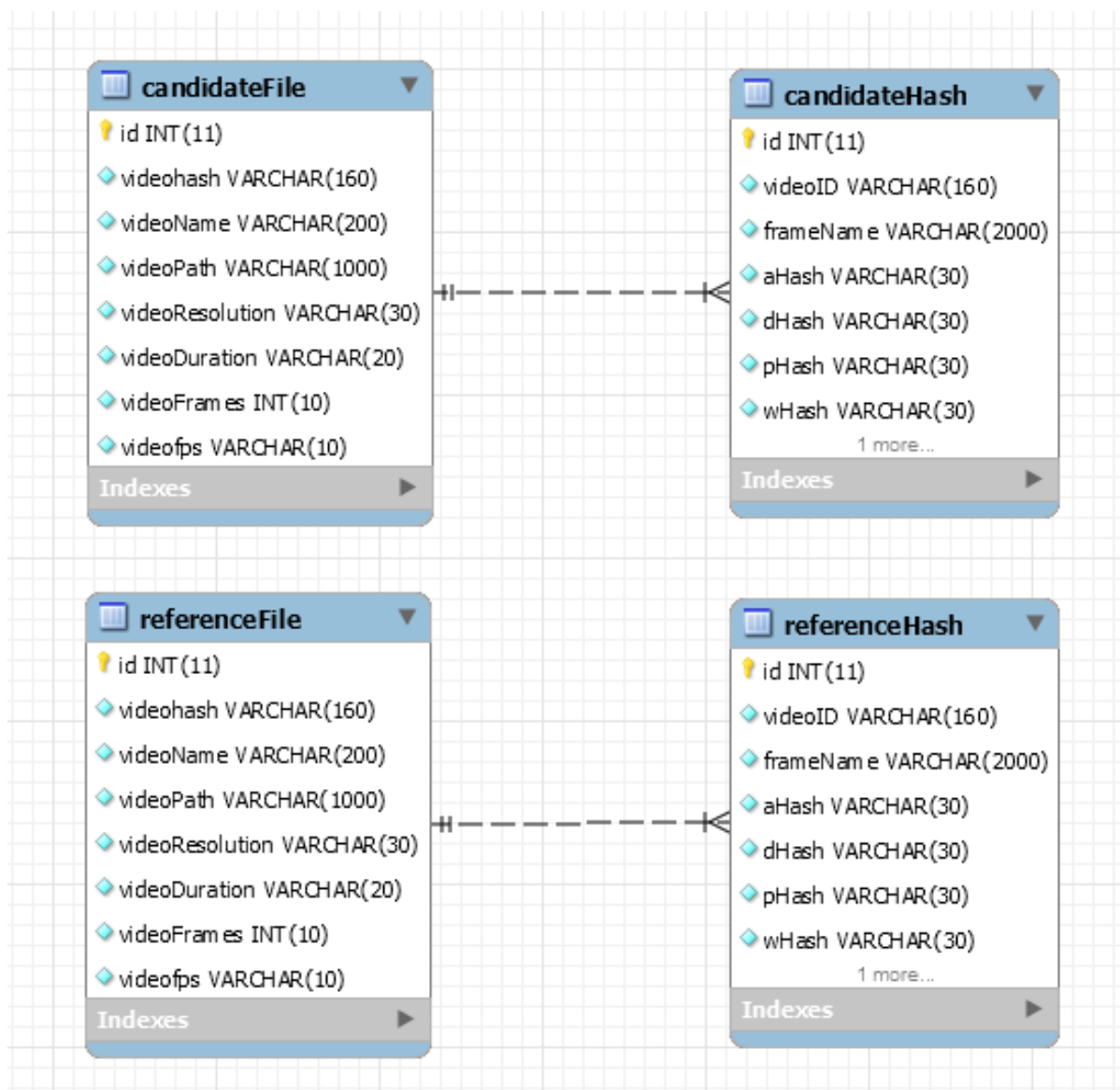
### 3.2.5 Banco de Dados

O banco de dados MySQL (<https://www.mysql.com/>) foi utilizado para armazenar informações e metadados sobre o arquivo de referência, arquivo de comparação, quadros extraídos e *hash* gerados. Para melhor organização, múltiplas tabelas foram utilizadas com os nomes:

- candidateFile - Para armazenar metadados a respeito do arquivo com potencial de cópia
- candidateHash - Para armazenar os valores *hash* gerados a partir do arquivo candidato
- referenceFile - Para armazenar metadados a respeito do arquivo de referência
- referenceHash - Para armazenar os valores *hash* gerados a partir do arquivo de referência

A Figura 15, mostra o que foi aplicado no banco de dados.

Figura 15 – Tabelas do banco de dados



Fonte: Próprio autor

Os campos das tabelas candidateFile e ReferenceFile são explicados na Tabela 8.

**Tabela 8 – Descrição dos campos das tabelas candidateFile/ReferenceFile**

Nome do campo	Descrição do campo
videoHash	Identificador do vídeo - SHA1
videoName	Nome do arquivo de vídeo
videoPath	Local de armazenamento do vídeo
videoResolution	Resolução do vídeo - x/x
videoDuration	Tempo de reprodução do vídeo em segundos
videoFrames	Número de frames existentes
videoFPS	Reprodução em quadros por segundos (FPS)

Fonte: Próprio autor

Os campos das tabelas candidateHash e referenceHash são explicados na tabela 9.

**Tabela 9 – Descrição dos campos das tabelas candidateHash/referenceHash**

Nome do campo	Descrição do campo
videoID	Identificador do vídeo - SHA1 (o mesmo que o videoHash da tabela anterior)
frameName	Nome e local do quadro do vídeo representado por [pasta]/[videoHash]/00000_0.00.jpg. Os primeiros 5 algarismos representam o número (ordem) do quadro e os 3 últimos representam o tempo ou local do quadro.
aHash	Valor hash do quadro, gerado pelo aHash
dHash	Valor hash do quadro, gerado pelo dHash
pHash	Valor hash do quadro, gerado pelo pHash
wHash	Valor hash do quadro, gerado pelo wHash

Fonte: Próprio autor

### 3.2.6 Busca, Comparação e Identificação

Foi observado até aqui que o número de quadros e conseqüentemente de valores *hash* gerados a partir de um vídeo é alto, e quanto maior o tempo de reprodução, maior o número de quadros. No método apresentado, há dois grupos de valores *hash* a

serem analisados e comparados. O primeiro sendo o valor de referência, representando o vídeo do qual se deseja localizar e o segundo os valores candidatos, representando os vídeos com potencial de cópia. Isto quer dizer que para cada arquivo de referência que se deseja buscar, deve-se efetuar a busca e comparação de todos os elementos da base de candidatos. Isto é chamado de busca linear ou força bruta. O número de operações está diretamente relacionado com o número de quadros a serem verificados. Visto que pode se obter múltiplos candidatos para comparação, cálculo este que é representado por  $N \times N$  ou  $bR \times bC$  (*base de referência x base de candidatos*), o algoritmo de busca deve ser suficientemente rápido para que torne o método eficaz. O banco de dados possui características de indexação e otimização de busca que agilizam muito o processo e estas foram utilizadas quando a extração ocorreu utilizando o MySQL. Quando utilizado programa criado pelo autor, no entanto, a busca é feita de maneira linear ou “Força Bruta”.

A comparação entre um valor *hash* e outro podem ser feitas de maneira simultânea ao utilizar-se funções nativas do banco de dados como BIT\_COUNT para a contagem do bits e o operador  $\oplus$  (OU Exclusivo, XOR).

Essas operações existentes no banco de dados equivalem aos cálculos de distância de Hamming observados na seção 2.4.1. A função BIT\_COUNT retorna o número de bit de um valor binário passado como parâmetro. No caso dos valores *hash*, apresenta-se uma *string* de 64 bits representados por uma *string* de 16 caracteres. A função BIT\_COUNT “transforma” estes caracteres em bits novamente e o operador  $\oplus$  é utilizado para comparar um valor *hash* com o outro por meio do cálculo XOR. Visto que 64 é o número de bits do valor *hash*, obter distância  $D$  próximo de 64 equivale a valores *hash* completamente diferentes e obter distância  $D$  próximo a zero equivale à valores *hash* similares. O cálculo utilizado foi:

$$D = (h1 \oplus h2) \quad (3.2)$$

O símbolo no banco de dados MySQL que indica a função XOR é o “^”. O código 3.2, mostrado a seguir, contém a porção do comando utilizado para o cálculo da distância de Hamming dos quatro valores *hash*.

### Código 3.2 – Comandos do MYSQL para cálculo da distância de hamming

```
bit_count(cast(conv(candidateHash.aHash,16,10) as UNSIGNED) ^
cast(conv(referenceHash.aHash,16,10) as UNSIGNED) ) as
adistance
bit_count(cast(conv(candidateHash.dHash,16,10) as UNSIGNED) ^
cast(conv(referenceHash.dHash,16,10) as UNSIGNED) ) as
ddistance,
```

```

bit_count(cast(conv(candidateHash.pHash,16,10) as UNSIGNED) ^
           cast(conv(referenceHash.pHash,16,10) as UNSIGNED) ) as
           pdistance,
bit_count(cast(conv(candidateHash.wHash,16,10) as UNSIGNED) ^
           cast(conv(referenceHash.wHash,16,10) as UNSIGNED) ) as
           wdistance
    
```

Alternativamente, o cálculo de distância de Hamming pode ser feito por meio de um programa em Python escrito pelo autor, que recebe os valores *hash* do banco de dados e os submete às operações matemáticas necessárias. Neste caso, o procedimento utilizado converte os caracteres *hash* para uma *string* binária de 64 posições e utiliza os caracteres da *string*, comparando-os um a um, mostrado no Apêndice D.1.

Durante a comparação, seja pelo banco de dados ou pelo programa Python, dois valores *hash* são então declarados similares se o acumulado da distância de Hamming dos valores *hash* estiverem abaixo de um certo coeficiente de similaridade ou limiar (*threshold*)  $T$  definido pelo autor. Os cálculos foram executados para cada valor *hash* da base de referência, em busca e comparação com os valores da base de vídeos candidata. Um único quadro não é suficiente para que a identificação seja finalizada. Por exemplo, o arquivo de vídeo intitulado “dumbo-trailer-2” presente na base de referência, possui 3377 quadros (Figura 16), que foram propriamente extraídos conforme descrito na Seção 3.2.3. Na base de candidatos existe um arquivo com potencial “pirata” com 1439 quadros (Figura 17). Fossem estes os únicos arquivos no banco de dados, o número de comparações seria de aproximadamente 4.859.503 (quatro milhões) e a consulta demoraria alguns minutos. No entanto, espera-se que a base de referência e candidatos em potencial seja muito maior que apenas um arquivo de vídeo e seus respectivos quadros, logo o tempo de execução deve também ser ponderado. Os valores relacionados ao tempo gasto em cada operação é mostrado na seção que trata dos resultados.

Figura 16 – Tabela de arquivos de vídeos de referência

id	videohash	videoName	videoPath	videoResolution	videoDuration	videoFrames	videofps
12	0fe94230ba7d21ce8ccd456fac5f3ea05d9fafc6	movies/sample.mkv	frames/0fe94230ba7d21ce8ccd456fac5f3ea05...	1280.0/544.0	59.97	1438	23.98
13	414436ad786a68786cecf1606b825d41cbfb0b48	movies/beyond-white-space-trailer-1_h720p.mov	frames/414436ad786a68786cecf1606b825d41c...	1280.0/544.0	100.71	2415	23.98
14	d0946cdcd79a149d0e1dc47b5ea8772967e80a	movies/bounty-killer-trailer-1_h720p.mov	frames/d0946cdcd79a149d0e1dc47b5ea8772...	1280.0/544.0	97.29	2333	23.98
15	da55c1612ebc1ee2a4597b99260f95deaaaf20b4e	movies/once-upon-a-deadpool-trailer-1_h720p....	frames/da55c1612ebc1ee2a4597b99260f95de...	1280.0/544.0	60.01	1439	23.98
16	d12468f2267508f2ead97aabb07ad395414b434a	movies/page-18-movie-4.mkv	frames/d12468f2267508f2ead97aabb07ad395...	480.0/270.0	216.04	5401	25.0
17	51df0d276ce3a5e0bf2b05794d4e48144374c201	movies/secret-life-of-pets-2-trailer-2_h720p.mov	frames/51df0d276ce3a5e0bf2b05794d4e48144...	1280.0/688.0	84.65	2030	23.98
18	8729f260da851604ac2e8bedfaa3fb0b70e87073	movies/small.mkv	frames/8729f260da851604ac2e8bedfaa3fb0b7...	560.0/320.0	5.53	166	30.0
19	f8d03dabfdb94d791e21440559d0517702b1f78	movies/the-vanishing-trailer-1_h720p.mov	frames/f8d03dabfdb94d791e21440559d05177...	1280.0/544.0	152.84	3665	23.98
20	b69cc024ad7f793f4088f5cd6c7a26fb15530ac2	movies/almost-almost-famous-trailer-1-ca_h720...	frames/b69cc024ad7f793f4088f5cd6c7a26fb15...	1280.0/720.0	101.75	2440	23.98
21	58dd862f8dbff12ec2f3da6cb3f91b5cc294b2c1	movies/dead-in-a-week-trailer-1_h720p.mov	frames/58dd862f8dbff12ec2f3da6cb3f91b5cc2...	1280.0/688.0	132.11	3168	23.98
22	f9a668966deb3653f50a68c680e1608e5980743	movies/inolemon-detective-nikachu-trailer-1_h7...	frames/f9a668966deb3653f50a68c680e1608e...	1280.0/544.0	148.33	3557	23.98
23	e526091e2b52b2e116c836cd8973ac3464f697b6	movies/dumbo-trailer-2_h720p.mov	frames/e526091e2b52b2e116c836cd8973ac34...	1280.0/688.0	140.83	3377	23.98
24	8034eac68ef7b46d439e054000abc31f11614e4	movies/aquamant-trailer-3_h720p.mov	frames/8034eac68ef7b46d439e054000abc31f1...	1280.0/544.0	150.08	3599	23.98

Fonte: Próprio autor

Figura 17 – Tabela com os vídeos candidatos

id	videohash	videoName	videoPath	videoResolution	videoDuration	videoFrames	videofps
10	da55c1612ebc1ee2a4597b99260f95deaaf20b4e	movies/once-upon-a-deadpool-trailer-1_h720p....	frames/da55c1612ebc1ee2a4597b99260f95de...	1280.0/544.0	60.01	1439	23.98
11	4cfad87b6c2950cd842a4cf60ea3dea08bf3ecb	movies/shazam-extended-tv-spot_h720p.mov	frames/4cfad87b6c2950cd842a4cf60ea3dea08...	1280.0/544.0	53.5	1283	23.98

Fonte: Próprio autor

Observa-se, a partir deste ponto, o número de quadros que são iguais ou menores que o coeficiente de similaridade ou limiar (*threshold*) T. Pode-se definir o *threshold* com o valor 10 (mais próximo de zero) e verifica-se quantos quadros tem sua distância de Hamming equivalente abaixo deste valor. Denominou-se este número encontrado de Potencial de Igualdade ou PI na seção 2.4.1. Ilustra-se, por meio da Tabela 10, os coeficiente ou limiar T, possíveis.

Tabela 10 – Cálculo do Coeficiente e Percentual de Igualdade

Distância de Hamming	% de Igualdade
0	100
5	92,18
10	84,37
15	76,56
20	68,75
25	60,93
30	53,12
35	45,31
40	37,5
45	29,68
50	21,87
55	14,06
60	6,25
64	0

Criado pelo autor

Na Tabela 10 é possível observar que, a partir de uma distância de hamming abaixo de 15, o percentual de igualdade é maior que 70%. 85% de similaridade é alcançado com valores da distância de hamming abaixo de 10. Neste trabalho, optou-se por adotar um limiar de 13 que corresponde a um PI de 80%. Até aqui foi considerado



o uso de apenas um valor *hash* comparado com o outro (referência/candidato), e seu respectivo limiar (threshold). A proposta deste estudo é verificar o uso de não apenas um algoritmo de *hash* perceptivo, mas quatro deles, combinados, por isto o limiar de similaridade é aplicado igualmente para todos os algoritmos. Para cada conjunto de valores dos algoritmos *hash*, conta-se os valores que estão abaixo do limiar, para cada algoritmo e calcula-se a média ponderada entre eles.

A atribuição de pesos para cada algoritmo de *hash* perceptivo obrigaria o autor a identificar características únicas que distinguem um algoritmo do outro. Fatores, como velocidade, invariância e aleatoriedade teriam de ser observados, bem como o número de falsos positivos e negativos de cada um. Estudos desta natureza já foram realizados, e é possível avaliar estas características de forma quantitativa. No entanto, o que se observa é que não há consenso sobre o uso de um algoritmo em detrimento de outro, mas há indicações do uso combinado deles (ZAUNER, 2010), que sugere que os pesos podem ser iguais. Somente após a execução dos primeiros testes e a partir dos resultados gerados será possível observar como cada um deles se comporta em virtude dos ataques realizados. Com isto pesos diferentes poderão ser utilizados e uma ponderação mais “justa” aplicada.

Com o cálculo da média ponderada, obtem-se o que o autor chama de PIF (Potencial de Igualdade Final). A saber:

$$PIF = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i}, \quad (3.3)$$

que pode ser formalizado da seguinte maneira:

$$PIF = \frac{p_{i_1} x_1 + p_{i_2} x_2 + p_{i_3} x_3 + p_{i_4} x_4}{x_1 + x_2 + x_3 + x_4} \quad (3.4)$$

Por exemplo, após executar as comparações, pode-se obter os seguintes valores a partir dos valores *hash*, Tabela 11 :

**Tabela 11 – Simulação de cálculo de Percentual de Igualdade Final.**

Algoritmos Hash	Contagem dos valores abaixo do limiar	Peso
aHash	59 793	0,25

Algoritmos Hash	Contagem dos valores abaixo do limiar	Peso
dHash	34 516	0,25
pHash	13 267	0,25
wHash	44 935	0,25

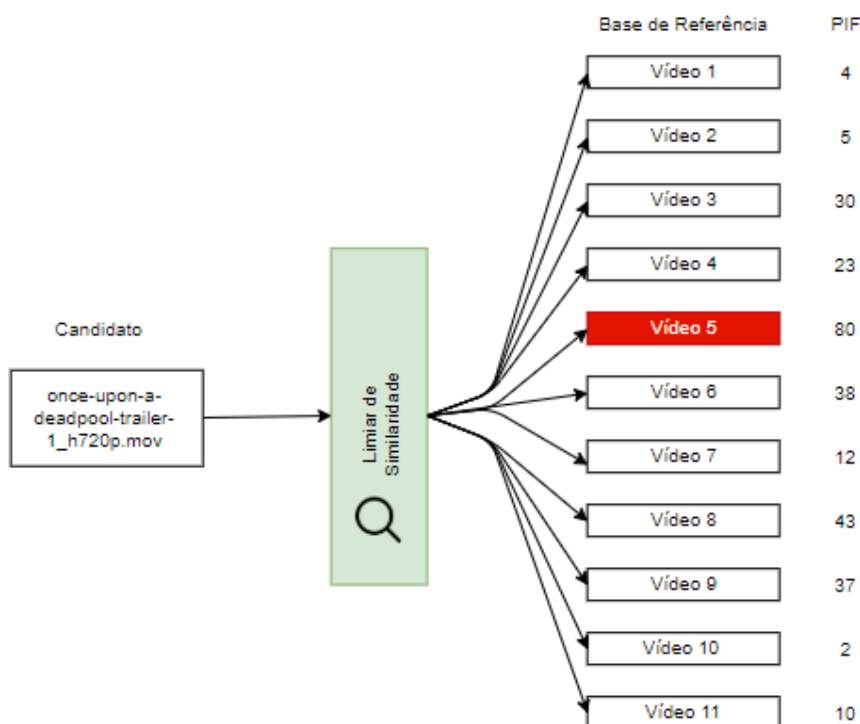
Fonte: Próprio autor

Tem-se, então:

$$PIF = 59793 \times 0,25 + 34516 \times 0,25 + 13267 \times 0,25 + 44935 \times 0,25 / 1 = 37\,306,5$$

Depois de ter comparado o vídeo candidato com todos os vídeos da base de referência, comparou-se o valor PIF de cada um deles para determinar qual dos vídeos tem maior potencial de cópia, conforme mostrado na Figura 18.

Figura 18 – Ilustração de busca por cópia de vídeo



Criado pelo autor

Além disso, foram executados ataques que preservam o conteúdo, como alteração da resolução, orientação, recortes conforme explicado na Seção 2.1. Estes ataques foram escolhidos de forma arbitrária pelo autor, com base em sua própria experiência e compreendem as formas mais comuns de manipulações utilizadas na distribuição de conteúdo na internet. Evidentemente existem outras formas de ataque

de vídeo, que não foram tratadas neste estudo tais como espelhamento, inserção de outros objetos, sobreposição de quadros e outros.

Por fim, calculou-se o tamanho da base de dados considerando uma possível colisão dos valores *hash* gerados, discutidos na seção 4.3.

### 3.3 Tempo de Execução

Um ponto que é de extrema importância para a compreensão dos resultados deste estudo é que os valores expressos nas tabelas, a seguir, estão limitados à capacidade de processamento, armazenamento e memória do computador, bem como a eficiência do código utilizado. Por exemplo, para extração dos quadros, técnicas de multi-processamento (multi threading) foram utilizadas, o que permite aproveitar ao máximo da capacidade do processador da máquina. Já para a geração dos *hash* e inserção dos valores no banco de dados, tais técnicas não foram possíveis e o impacto no desempenho é observado. Assim, os resultados e números expressos neste estudo estão condicionados à capacidade do processador da máquina de testes. Depois de testados e validados os códigos utilizados para os testes, omitiu-se a visualização das rotinas na tela do computador, para poupar recursos de processamento e não comprometer os resultados e medições. Para os testes, utilizou-se um computador de porte simples com 2 processadores, cuja especificação pode ser vista na Figura 19.

Figura 19 – Informação sobre o computador utilizado para os testes.

```
cpu: Intel(R) Celeron(R) CPU 1037U @ 1.80GHz, 1800 MHz
      Intel(R) Celeron(R) CPU 1037U @ 1.80GHz, 1800 MHz
monitor: Philco TV
graphics card: Intel 3rd Gen Core processor Graphics Controller
sound: Intel 7 Series/C216 Chipset Family High Definition Audio Controller
storage: Intel 7 Series Chipset Family 4-port SATA Controller [IDE mode]
          Intel 7 Series Chipset Family 2-port SATA Controller [IDE mode]
network: Realtek RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller
          Realtek RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller
network interface: eth0 Ethernet network interface
                  eth1 Ethernet network interface
                  lo Loopback network interface
                  br0 Ethernet network interface
disk: /dev/sda Generic External
      /dev/sdb SanDisk SSD i110
```

Fonte: Próprio autor

### 3.4 Visualização dos Dados

Ferramentas de visualização de dados foram utilizadas para facilitar a produção de gráficos e estatísticas a respeito do estudo foco deste trabalho. A escolha do

programa ideal exigiu tempo e pesquisa uma vez que ele teve de satisfazer requisitos de desempenho e capacidade de execução na ordem dos milhares e milhares de linhas, bem como produzir gráficos que ilustrassem corretamente os resultados. A comparação de todos os quadros de um vídeo candidato contra todos os quadros de um vídeo de referência, mesmo de curta duração, gerou milhares de linhas como resultado. Por isto o programa também deve ser capaz de, se necessário, conectar-se diretamente na base de dados de escolha e realizar as consultas, poupando assim recursos de processamento e memória. Foi necessário avaliar, previamente, as seguintes ferramentas:

- R (<https://www.r-project.org/>)
- Gnuplot (<http://www.gnuplot.info/>)
- Tableau (<https://www.tableau.com/>)
- Microsoft Power BI (<https://powerbi.microsoft.com/pt-br/>)
- Qlik Sense Desktop (<https://www.qlik.com/us>)

Depois de avaliar cada uma destas, optou-se por utilizar o programa Tableau (<https://www.tableau.com/>) para a construção dos gráficos e análise dos resultados.

## 4 Resultados e Discussão

Neste capítulo, apresentam-se os resultados obtidos a partir de testes e simulações executados pelo autor.

### 4.1 Extração de Metadados e Quadros

Embora a extração dos metadados e a extração dos quadros sejam descritos como passos distintos nas Seções 3.2.2 e 3.2.3, estes passos são realizados por meio do mesmo programa (OpenCV) e mesmo código (C.1). Pode-se dizer que esta é a parte mais simples e rápida de todo o processo, uma vez que a coleta dos metadados é feita em alguns milissegundos e a extração e armazenamentos dos quadros dos vídeos utilizados nos testes duraram alguns poucos segundos. O programa OpenCV possui chamadas específicas que trazem tais metadados de forma simples e rápida. A Tabela 12 contém os valores obtidos durante a extração dos metadados e quadros do vídeo. Vale observar que este passo do processo utilizou-se de técnicas de *multi-threading* ou multi-processamento para explorar todo o potencial de execução do computador. Isto foi possível por que não tem importância a ordem de gravação dos arquivos no disco, uma vez que cada um deles possui um identificador único.

**Tabela 12 – Tempo de execução - Extração dos quadros**

Nome do Arquivo	Quadros	Tempo - Segundos
almost-almost-famous-trailer-1-ca_h720p.mov	2440	00:20
aquaman-trailer-3_h720p.mov	3599	00:25
beyond-white-space-trailer-1_h720p.mov	2415	00:15
bounty-killer-trailer-1_h720p.mov	2333	00:16
captainamericacivilwar-tlr2_h720p.mov	3468	00:23
creed_trailer_2_txd_stereo_gc_h720p.mov	3671	00:30
dead-in-a-week-trailer-1_h720p.mov	3168	00:24
dumbo-trailer-2_h720p.mov	3377	00:27

Nome do Arquivo	Quadros	Tempo - Segundos
early-man-trailer-2_h720p.mov	3384	00:27
ferdinand-trailer-1_h720p.mov	3620	00:23
insideout-usca-15sectease_h720p.mov	403	00:02
kungfupanda3-tlr2_h720p.mov	3611	00:24
mad_max_fury_road_trailer_2-720p.mov	3672	00:31
moana-trailer-2_h720p.mov	3622	00:25
once-upon-a-deadpool-trailer-1_h720p.mov	1439	00:10
pokemon-detective-pikachu-trailer-1_h720p.mov	3557	00:24
secretlifeofpets-tlr2_h720p.mov	4047	00:32
the-vanishing-trailer-1_h720p.mov	3665	00:23
wall-e-tlr2_h720p.mov	4504	00:26
wonder-woman-trailer-3_h720p.mov	3480	00:25

Fonte: Próprio autor

A Figura 20 a seguir, mostra a lista de pastas criadas para o armazenamento dos quadros. Estas pastas recebem o mesmo código SHA1, usado para identificar o vídeo no banco de dados.

**Figura 20 – Lista de pastas de quadros**

02e05beac521e22352e1e0496b5a11ec4c1bd7a2	2019-01-22 ...	labuser	labuser
0fe94230ba7d21ce8ccd456fac5f3ea05d9fafc6	2019-01-12 ...	labuser	labuser
12c0a09e50e337a5abafac718b6ab14c89b8272f	2019-01-12 ...	labuser	labuser
2785c762f300c56ec7d95ae225a0eee10d23493a	2019-01-22 ...	labuser	labuser
414436ad786a68786cecf1606b825d41cbfb0b48	2019-01-22 ...	labuser	labuser
58dd862f8dbff12ec2f3da6cb3f91b5cc294b2c1	2019-01-22 ...	labuser	labuser
6badcf0f6201b793b394cb9047c662474c68e900	2019-01-22 ...	labuser	labuser
7462dfd9a5951f0117d92667b43cd43b481913b0	2019-01-22 ...	labuser	labuser
79fd711ddc3cbd78253d11068dc94d4adaa05d7a	2019-01-22 ...	labuser	labuser
7ff39b45320dea91686c1de4076e218b919dff92	2019-01-22 ...	labuser	labuser
8034eac68ef7b46d439e054000abc31f11f614e4	2019-01-22 ...	labuser	labuser
8731c13b90a230c274eec9a165eb519b24be7890	2019-01-22 ...	labuser	labuser
88239e25ea0f2b0db322a504a7e1560666e617fe	2019-01-22 ...	labuser	labuser
98fb9abee3a7a3f7a6ec4f24d0f1a7495134925c	2019-01-22 ...	labuser	labuser
b69cc024ad7f793f4088f5cd6c7a26fb15530ac2	2019-01-22 ...	labuser	labuser
c5e0737e1955ecacae435b73b2b12455917f1ac0	2019-01-22 ...	labuser	labuser
d0946cdcd79a149d0e1dc47b5eaf8772967e80a	2019-01-22 ...	labuser	labuser
da55c1612ebc1ee2a4597b99260f95deaaf20b4e	2019-01-22 ...	labuser	labuser
e526091e2b52b2e116c836cd8973ac3464f697b6	2019-01-22 ...	labuser	labuser
f8d03dabfdb94d791e21440559d0517702bb17f8	2019-01-22 ...	labuser	labuser
f8faa85990c23fee69c32524c76b1d009629a3ab	2019-01-22 ...	labuser	labuser
f98668966deb36535f50a68c680e1608e59807d3	2019-01-22 ...	labuser	labuser

Fonte: Próprio autor

#### 4.1.1 Geração dos Valores *Hash*

Para geração dos valores *hash*, cada uma das funções *aHash*, *dHash*, *pHash* e *wHash* são chamadas de forma simultânea e recebem como parâmetro cada um dos quadros do vídeo de referência e candidato (Figura 21). O código 4.1, criado pelo autor, exhibe a aplicação da biblioteca ImageHash simplificada.

##### Código 4.1 – Exemplo de implementação do ImageHash 4.0 - Criado pelo autor

```
#!/usr/bin/python
from PIL import Image
import imagehash

referencia = imagehash.average_hash(Image.open('test.png'))
print(referencia)
>> d879f8f89b1bbf
comparacao = imagehash.average_hash(Image.open('other.bmp'))
print(comparacao)
>> ffff3720200fff
print(referencia == comparacao)
False
```

A partir deste ponto, cada linha ou conjunto de *hash* referentes aos quadros do vídeo é armazenada no banco de dados. Por esta razão, as técnicas de *multithreading* não foram utilizadas para prevenir a inserção de valores fora de ordem na tabela. É importante salientar que cada método de *hash* perceptivo utiliza uma técnica diferente para o processo de extração de características, que entre outras coisas envolvem a normalização da imagem em um tamanho pré definido, que no caso do *aHash* significa reduzi-la para uma versão de  $8 \times 8$  bits, conforme descrito na seção 2.3. Cada um dos algoritmos de *hash* perceptivo presente na biblioteca ImageHash 4.0 em Python executam a extração das características e cálculos para geração do valor *hash* na memória do computador e isto tem impacto no tempo de execução. A Tabela 13, contém o tempo referente a extração dos valores *hash* para cada quadro dos vídeos utilizados para os testes.

**Tabela 13 – Tempo de execução - Extração dos valores hash**

Nome do Arquivo	Quadros	Tempo - Minutos
almost-almost-famous-trailer-1-ca_h720p.mov	2440	3:54
aquaman-trailer-3_h720p.mov	3599	4:42
beyond-white-space-trailer-1_h720p.mov	2415	2:55
bounty-killer-trailer-1_h720p.mov	2333	3:03
captainamericacivilwar-tlr2_h720p.mov	3468	4:24
creed_trailer_2_txd_stereo_gc_h720p.mov	3671	5:38
dead-in-a-week-trailer-1_h720p.mov	3168	4:48
dumbo-trailer-2_h720p.mov	3377	5:09
early-man-trailer-2_h720p.mov	3384	5:36
ferdinand-trailer-1_h720p.mov	3620	4:41
insideout-usca-15sectease_h720p.mov	403	0:34
kungfupanda3-tlr2_h720p.mov	3611	4:48
mad_max_fury_road_trailer_2-720p.mov	3672	5:43
moana-trailer-2_h720p.mov	3622	4:46
once-upon-a-deadpool-trailer-1_h720p.mov	1439	1:55
pokemon-detective-pikachu-trailer-1_h720p.mov	3557	4:33
secretlifeofpets-tlr2_h720p.mov	4047	6:25
the-vanishing-trailer-1_h720p.mov	3665	4:30
wall-e-tlr2_h720p.mov	4504	5:30

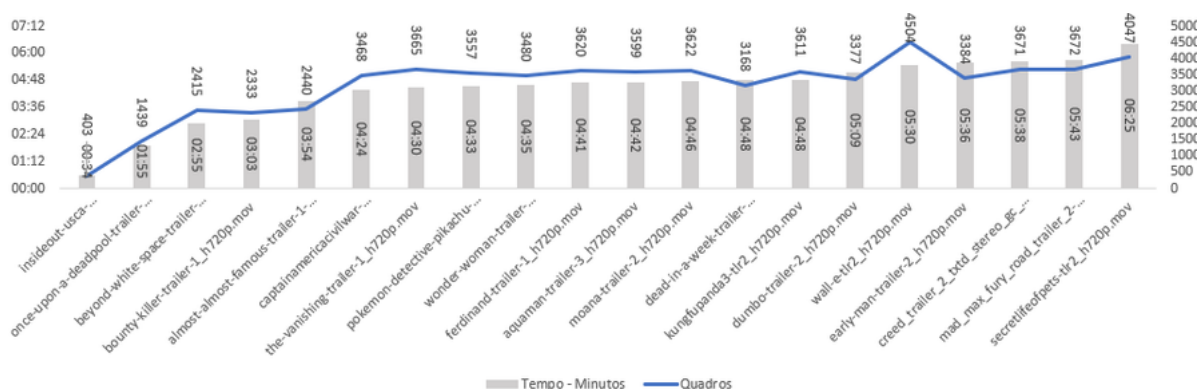


Nome do Arquivo	Quadros	Tempo - Minutos
wonder-woman-trailer-3_h720p.mov	3480	4:35

Fonte: Próprio autor

Observa-se, na Figura 21, que a relação número de quadros e tempo de geração de *hash* é consistente, sem apresentar grandes anomalias. O tempo de execução médio gasto para extração dos valores *hash* é de 4:24 (quatro minutos e vinte quatro segundos) contra um número médio de quadros de 3174. Isto quer dizer que o tempo de processamento é de 12 quadros por segundo. Um filme comum tem cerca de 518 400 quadros (Seção 3.2.3), e se tivesse os valores *hash* calculados a esta velocidade, o processo levaria em torno de 12 horas.

**Figura 21 – Gráfico com o número de quadros e tempo gasto no processamento dos valores hash**



Fonte: Próprio autor

#### 4.1.2 Comparando Valores Hash

Para comparar os valores *hash*, utilizou-se duas maneiras distintas, a primeira utilizando o próprio banco de dados, e a outra por meio de programa em Python criado pelo autor, como indicado na Seção 3.2.6.

A busca (*query*) por meio da força bruta, calculando os valores individualmente, sem os filtros do limiar ou *threshold* no banco de dados é muito mais rápido do que fazendo por meio de código escrito pelo autor. As linhas retornadas devem ser escritas em um arquivo (csv) para permitir a produção de gráficos para análise posterior. Este passo adicional, feito por meio de chamadas em Python (Apêndice F.1 e G.1), traz complicações enormes à operação, uma vez que todos os resultados retornados pelo banco de dados são armazenados na memória do computador de uma única vez, fazendo com o sistema retornasse erros de execução. Por estas questões, a busca cega, sem parâmetros de limiar, foi executada no banco de dados apenas para análise

de desempenho. Não foi possível gravar a saída em arquivo de texto para a análise posterior.

Já o código criado pelo autor, armazena apenas o número total de linhas referentes aos quadros de cada vídeo na memória, e o cálculo da distância de Hamming é feito durante a execução do programa (Apêndice E.1), um a um, sem prejuízos de esgotamento de espaço de memória com operações matemáticas ou armazenamento dos resultados em arquivo de texto. Portanto, embora a geração dos valores seja mais rápida pelo banco de dados, o programa criado pelo autor se mostrou mais flexível para manipular os resultados em arquivos de texto.

A Tabela 14, a seguir, compara as operações utilizando os dois métodos e o tempo de execução de cada um.

**Tabela 14 – Tabela com tempo utilizado para comparação dos valores *hash* - Software e Banco de dados.**

Vídeo Candidato	Video Referência	Comparações	Programa	Banco de Dados (query/fetching)
once-upon-a-deadpool-trailer-1_h720p.mov	almost-almost-famous-trailer-1-ca_h720p.mov	3511160	7:56 min	0.0/97.11 seg
once-upon-a-deadpool-trailer-1_h720p.mov	aquaman-trailer-3_h720p.mov	5178961	11:44 min	1.3/118 seg
once-upon-a-deadpool-trailer-1_h720p.mov	once-upon-a-deadpool-trailer-1_h720p.mov	2070721	4:39 min	21.5/45.6 seg
once-upon-a-deadpool-trailer-1_h720p.mov	beyond-white-space-trailer-1_h720p.mov	3475185	7:52	3.2/90.6 seg

Vídeo Candidato	Video Referência	Comparações	Programa	Banco de Dados (query/fetching)
once-upon-a-deadpool-trailer-1_h720p.mov	bounty-killer-trailer-1_h720p.mov	3357187	7:36	4.43/86.6 seg
once-upon-a-deadpool-trailer-1_h720p.mov	captainamericacivilwar-tlr2_h720p.mov	4990452	15:02	5.67/118.9 seg

Fonte: Próprio autor

Percebe-se que mesmo tendo o tempo incrementado pela apresentação das linhas na tela (*fetching*), a operação feita pelo banco de dados é muito mais rápida (cerca de 5 vezes) do que a executada via *software*. A limitação de não conseguir escrever os resultados em arquivos de texto nas operações provenientes do banco de dados é resolvida por meio de sistemas de visualização de dados que se conectam diretamente no banco para criação dos gráficos, conforme descritos na Seção 3.4.

Ao indicar o limiar de similaridade (13) nas consultas, o desempenho aumenta consideravelmente, fazendo com que haja economia de tempo, recursos e espaço em disco. No caso do banco de dados, a melhora no desempenho é observada no processo de visualização (*fetching*) dos dados que é reduzido pela metade (Tabela 15). Já com as comparações realizadas pelo programa, embora ainda tenha que executar todos os cálculos e selecionar apenas os que estão abaixo ou igual ao limiar, o tempo da operação foi reduzido também pela metade.

**Tabela 15 – Tempo utilizado para comparação dos valores *hash*- Software e Banco de dados**

Vídeo Candidato	Video Referência	Comparações	Programa	Banco de Dados (query/fetching)
once-upon-a-deadpool-trailer-1_h720p.mov	almost-almost-famous-trailer-1-ca_h720p.mov	3511160	4:15 min	0.0/43.37 seg
once-upon-a-deadpool-trailer-1_h720p.mov	aquaman-trailer-3_h720p.mov	5178961	6:17 min	1.7/44.9 seg
once-upon-a-deadpool-trailer-1_h720p.mov	once-upon-a-deadpool-trailer-1_h720p.mov	2070721	2:36 min	22.5/18.6 seg
once-upon-a-deadpool-trailer-1_h720p.mov	beyond-white-space-trailer-1_h720p.mov	3475185	4:13 min	3.2/39.9 seg
once-upon-a-deadpool-trailer-1_h720p.mov	bounty-killer-trailer-1_h720p.mov	3357187	4:04 min	4.78/38.6 seg
once-upon-a-deadpool-trailer-1_h720p.mov	captainamericacivilwar-tlr2_h720p.mov	4990452	6:04 min	6.37/39.7 seg

Fonte: Próprio autor

Em ambos os casos o produto gerado para análise é uma lista, separado por vírgula (CSV) com as informações dos vídeos, quadros e valores *hash* sendo comparados (Figura 22).

Figura 22 – Arquivo CSV com resultado das comparações

```

loop;loop;:c_videoId;:c_videoId;:c_entryId;:c_entryId;:c_hash;:r_hash;:distance;:hash
1;1; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43418; '0000000000000000'; '0000000000000000'; 0; ahash
1;1; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43418; '0000000000000000'; '0000000000000000'; 0; dhash
1;1; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43418; '0000000000000000'; '0000000000000000'; 0; phash
1;1; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43418; '0000000000000000'; '0000000000000000'; 0; whash
1;2; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43419; '0000000000000000'; '0000000000000000'; 0; ahash
1;2; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43419; '0000000000000000'; '0000000000000000'; 0; dhash
1;2; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43419; '0000000000000000'; '0000000000000000'; 0; phash
1;2; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43419; '0000000000000000'; '0000000000000000'; 0; whash
1;3; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43420; '0000000000000000'; '0000000000000000'; 0; ahash
1;3; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43420; '0000000000000000'; '0000000000000000'; 0; dhash
1;3; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43420; '0000000000000000'; '0000000000000000'; 0; phash
1;3; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43420; '0000000000000000'; '0000000000000000'; 0; whash
1;4; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43421; '0000000000000000'; '0000000000000000'; 0; ahash
1;4; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43421; '0000000000000000'; '0000000000000000'; 0; dhash
1;4; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43421; '0000000000000000'; '0000000000000000'; 0; phash
1;4; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43421; '0000000000000000'; '0000000000000000'; 0; whash
1;5; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43422; '0000000000000000'; '0000000000000000'; 0; ahash
1;5; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43422; '0000000000000000'; '0000000000000000'; 0; dhash
1;5; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43422; '0000000000000000'; '0000000000000000'; 0; phash
1;5; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43422; '0000000000000000'; '0000000000000000'; 0; whash
1;6; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43423; '0000000000000000'; '0000000000000000'; 0; ahash
1;6; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43423; '0000000000000000'; '0000000000000000'; 0; dhash
1;6; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43423; '0000000000000000'; '0000000000000000'; 0; phash
1;6; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43423; '0000000000000000'; '0000000000000000'; 0; whash
1;7; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43424; '0000000000000000'; '0000000000000000'; 0; ahash
1;7; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43424; '0000000000000000'; '0000000000000000'; 0; dhash
1;7; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43424; '0000000000000000'; '0000000000000000'; 0; phash
1;7; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43424; '0000000000000000'; '0000000000000000'; 0; whash
1;8; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43425; '0000000000000000'; '0000000000000000'; 0; ahash
1;8; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43425; '0000000000000000'; '0000000000000000'; 0; dhash
1;8; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43425; '0000000000000000'; '0000000000000000'; 0; phash
1;8; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43425; '0000000000000000'; '0000000000000000'; 0; whash
1;9; 'da55c1612ebc1ee2a4597b99260f95deaaaf20b4e'; 'b69cc024ad7f793f4088f5cd6c7a26fb15530ac2'; 10969; 43426; '0000000000000000'; '0000000000000000'; 0; ahash

```

Fonte: Próprio autor

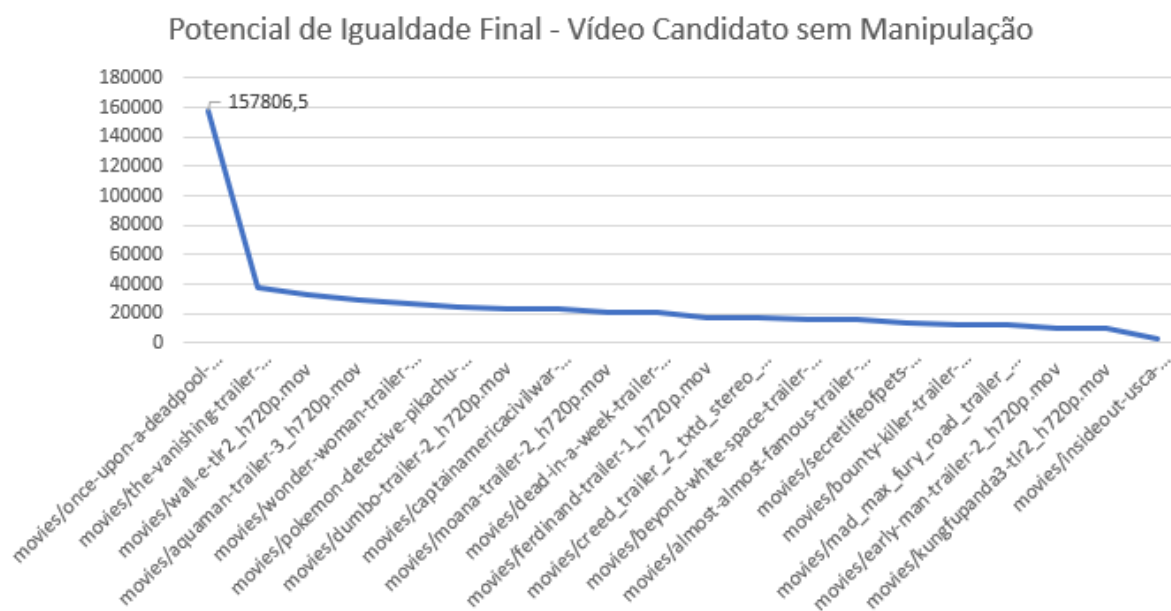
## 4.2 Identificando Cópia de Vídeo

Feita a execução e escolha da melhor ferramenta para comparar os valores *hash* gerados, utiliza-se análise estatística para identificar quais dos vídeos na base de referência é similar ao vídeo utilizado como candidato. Nos testes utilizou-se o arquivo intitulado “once-upon-a-deadpool-trailer-1\_h720p.mov”, extraíram-se os quadros, geraram-se os valores *hash* de seus quadros e obtiveram-se os valores da distância Hamming. A busca e comparação se deu por meio de busca linear, ou força bruta, varrendo o banco de dados e gerando a comparação de cada um dos valores *hash*. Para o primeiro teste, o mesmo arquivo e seus respectivos *hash* foram colocados na base de referência e na base de candidatos, sem manipulações. Nos testes subsequentes, o arquivo “once-upon-a-deadpool-trailer-1\_h720p.mov” foi submetido a ataques, incluindo modificações na resolução, orientação e realizando *cropping* (recortes). Este sub-produto do vídeo foi colocado na base de candidato e verificado contra a base de referência. As seções a seguir tratam dos resultados obtidos.

### 4.2.1 Sem Manipulações/Ataques no Vídeo

Foi possível observar que a base de referência continha de fato o arquivo “once-upon-a-deadpool-trailer-1\_h720p.mov” e que o número de registros abaixo do limiar é maior que os dos outros arquivos. Na Figura 23, é mostrado o número de registros encontrados e na Figura 24 os números relacionados.

Figura 23 – Gráfico com os valores da comparação entre o vídeo candidato e as referências



Fonte: Próprio autor

Figura 24 – Tabela com os valores da comparação entre o vídeo candidato e as referências

Candidato	Referência	aHash	dHash	pHash	Whash	PIF
movies/once-upon-a-deadpool-trailer-1_h720p.mov	movies/once-upon-a-deadpool-trailer-1_h720p.mov	215121	125313	83163	207629	157806,5
	movies/the-vanishing-trailer-1_h720p.mov	54225	34831	13083	48831	37742,5
	movies/wall-e-tlr2_h720p.mov	65312	17473	5970	42144	32724,75
	movies/aquaman-trailer-3_h720p.mov	52404	11033	6379	47187	29250,75
	movies/wonder-woman-trailer-3_h720p.mov	47685	16527	6786	38691	27422,25
	movies/pokemon-detective-pikachu-trailer-1_h720p.mov	47455	12091	3748	35862	24789
	movies/dumbo-trailer-2_h720p.mov	40312	13414	6483	32352	23140,25
	movies/captainamericivilwar-tlr2_h720p.mov	32416	19274	10009	30390	23022,25
	movies/moana-trailer-2_h720p.mov	28979	14014	8215	31815	20755,75
	movies/dead-in-a-week-trailer-1_h720p.mov	35856	18765	5764	22391	20694
	movies/ferdinand-trailer-1_h720p.mov	29003	12623	3927	23556	17277,25
	movies/creed_trailer_2_txtd_stereo_gc_h720p.mov	23085	22613	11161	10496	16838,75
	movies/beyond-white-space-trailer-1_h720p.mov	25909	11250	4103	21770	15758
	movies/almost-almost-famous-trailer-1-ca_h720p.mov	24931	11563	4344	21124	15490,5
	movies/secretlifeofpets-tlr2_h720p.mov	19247	12202	2250	22017	13929
	movies/bounty-killer-trailer-1_h720p.mov	18944	12826	3253	15969	12748
	movies/mad_max_fury_road_trailer_2-720p.mov	15828	15191	8550	8904	12118,25
	movies/early-man-trailer-2_h720p.mov	19504	5635	2748	13204	10272,75
	movies/kungfupanda3-tlr2_h720p.mov	17331	9016	2989	11567	10225,75
	movies/insideout-usca-15sectease_h720p.mov	6062	4720	51	1239	3018

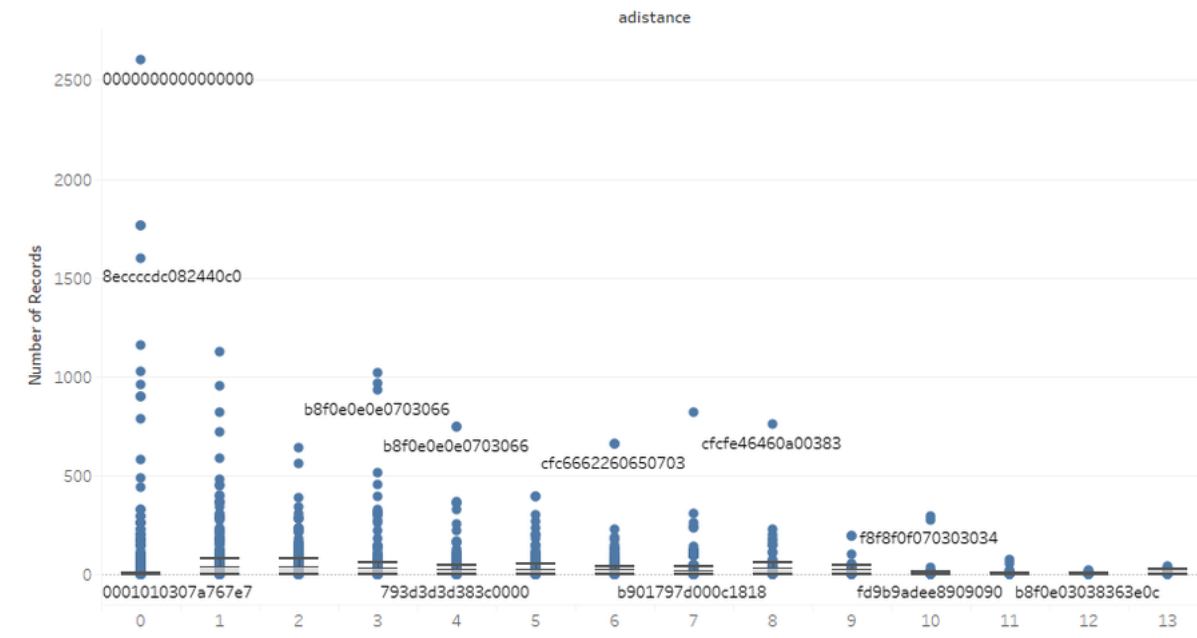
Fonte: Próprio autor

Analisando os dados de cada valor *hash*, individualmente, é possível ver a distribuição da distância de Hamming de cada um deles. As Figuras 25, 27, 29 e 30 mostram esta distribuição para os algoritmos *ahash*, *dhash*, *phash* e *whash*, respectivamente.

Distribuição da distância de Hamming a partir do algoritmo *aHash*:

**Figura 25 – Valores abaixo do limiar de similaridade - aHash**

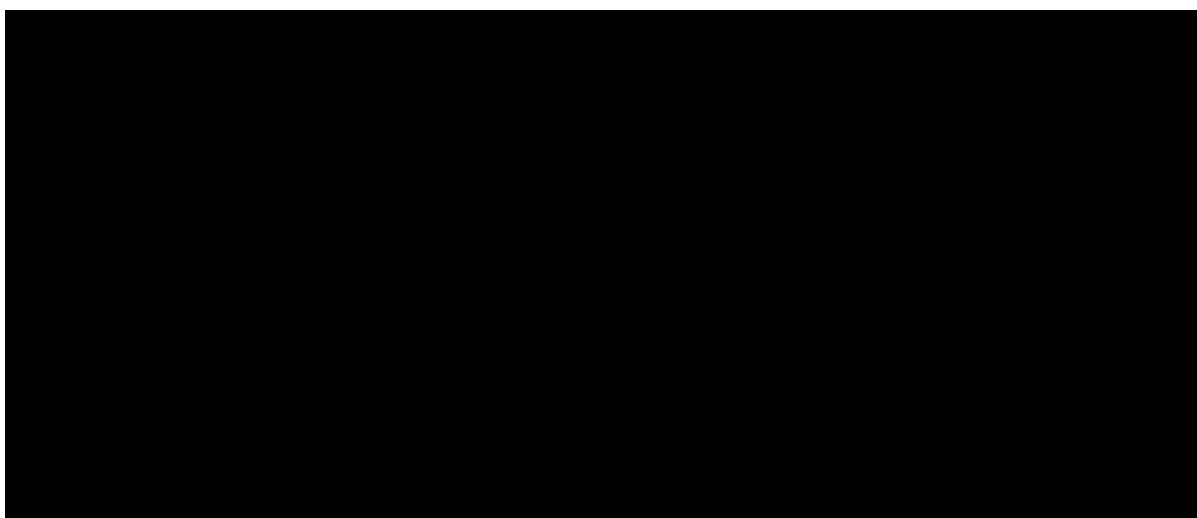
Comparação once-upon-a-deadpool-trailer-1\_h720p.mov X once-upon-a-deadpool-trailer-1\_h720p.mov



Fonte: Próprio autor

O pico observado na Figura 25, com contagem superior a 2500, refere-se a valores *hash* "0000000000000000" resultante de quadros escuros como mostrado na Figura 26.

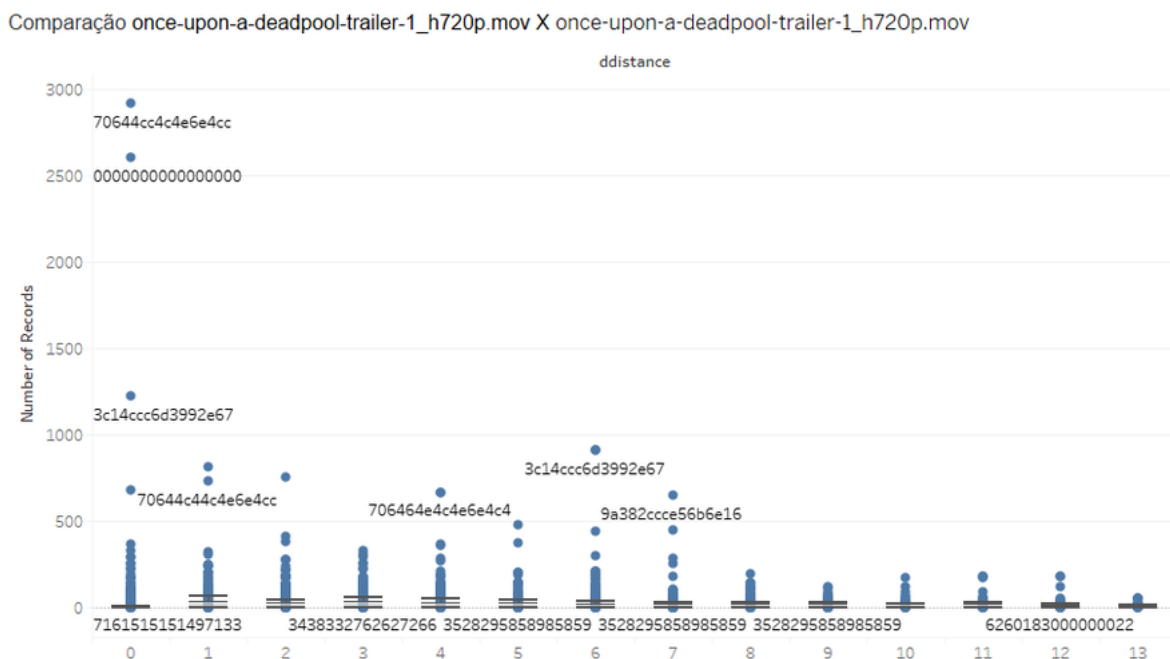
**Figura 26 – Imagem 00001\_0.04.jpg e valor aHash - 0000000000000000**



Fonte: Próprio autor

Distribuição da distância de Hamming a partir do algoritmo dHash, Figura 27:

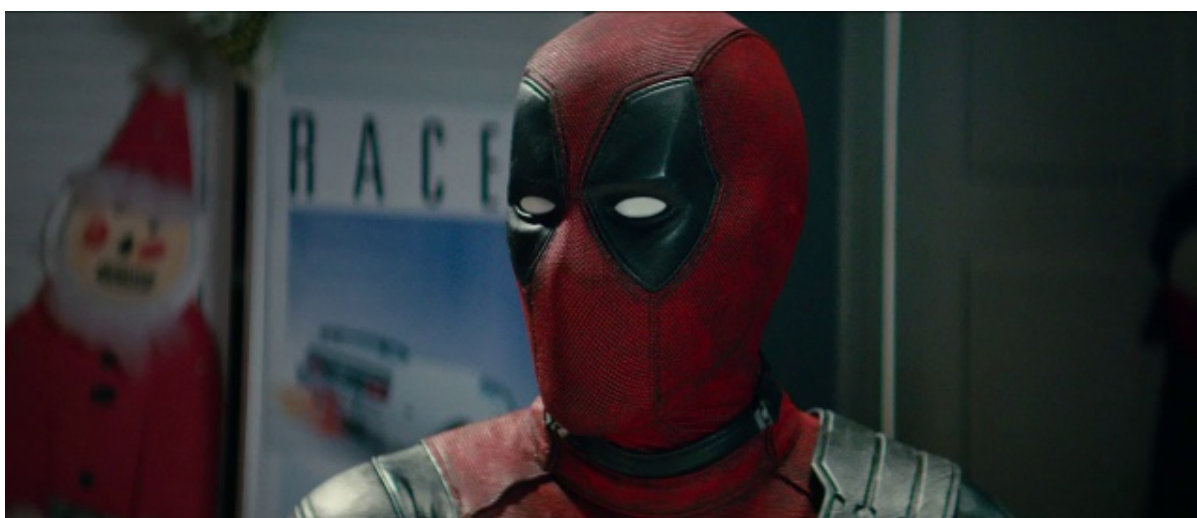
**Figura 27 – Valores abaixo do limiar de similaridade - dHash**



Fonte: Próprio autor

O pico observado na Figura 27, com contagem próximo à 2800 refere-se a quadros como o mostrado na Figura 28. Esta imagem é parte de um *take* de 2 segundos, com cerca de 48 quadros praticamente iguais.

**Figura 28 – Imagem 01163\_48.51.jpg e valor aHash - 70644cc4c4e6e4cc**



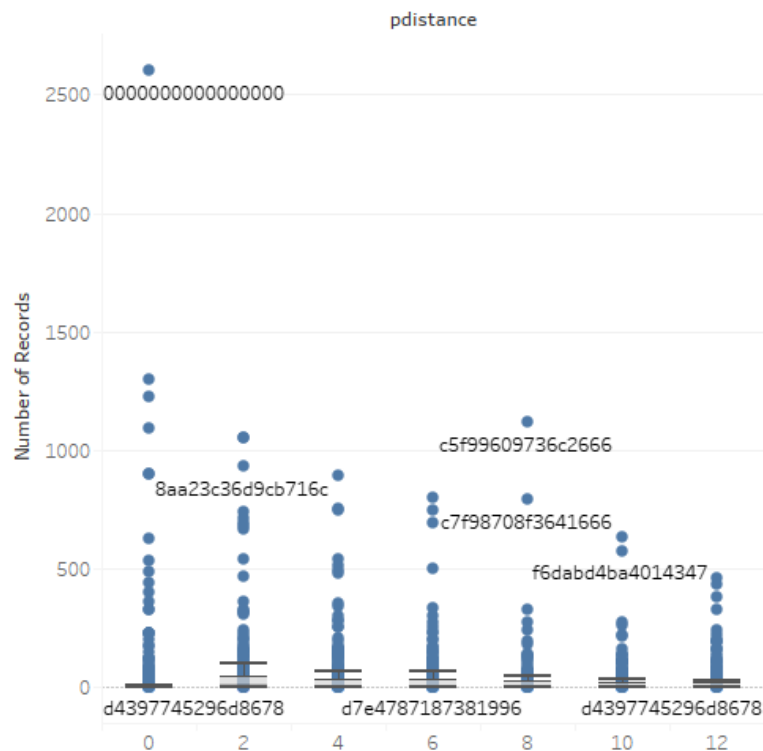
Fonte: Próprio autor



Distribuição da distância de Hamming a partir do algoritmo *pHash*, Figura 29:

**Figura 29 – Valores abaixo do limiar de similaridade - *pHash***

Comparação once-upon-a-deadpool-trailer-1\_h720p.mov X once-upon-a-deadpool-trailer-1\_h720p.mov



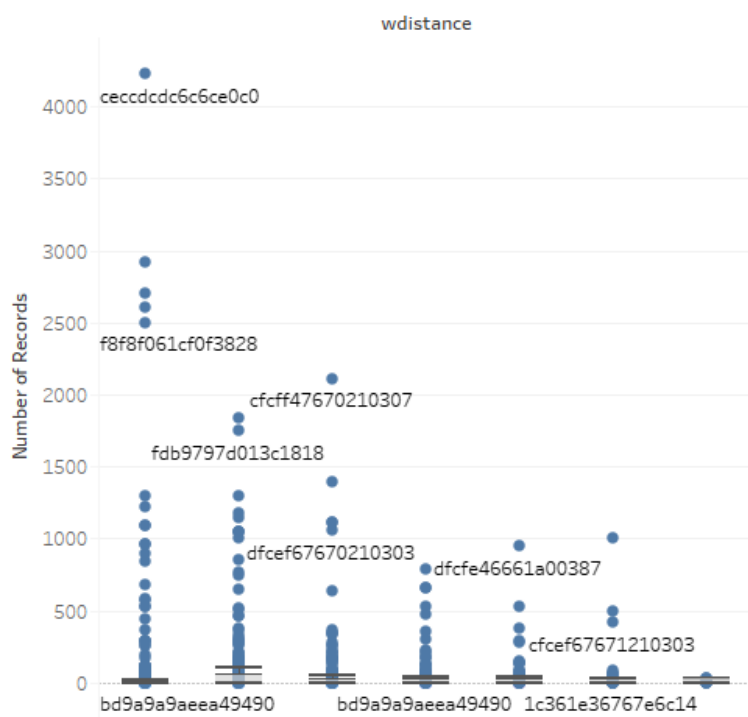
Fonte: Próprio autor

O pico observado na Figura 29, com contagem superior a 2500 refere-se a valores *hash* “000000000000000000” resultante de quadros escuros como o que foi mostrado na Figura 26.

Distribuição da distância de Hamming a partir do algoritmo *wHash*.

**Figura 30 – Valores abaixo do limiar de similaridade - wHash**

Comparação once-upon-a-deadpool-trailer-1\_h720p.mov X once-upon-a-deadpool-trailer-1\_h720p.mov



Fonte: Próprio autor

O pico observado na Figura 30, com contagem superior a 4000 refere-se a valores *hash* “ceccdc6c6ce0c0” resultante de imagens como a Figura 31. Esta imagem é parte de um *take* de 2 segundos, com cerca de 48 quadros praticamente iguais.

**Figura 31 – Imagem 00865\_36.08.jpg e valor wHash - ceccdc6c6ce0c0**

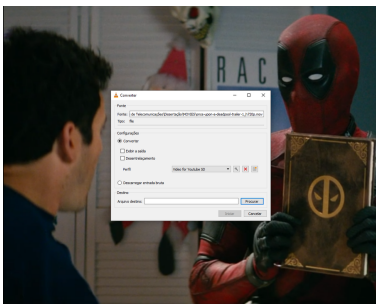
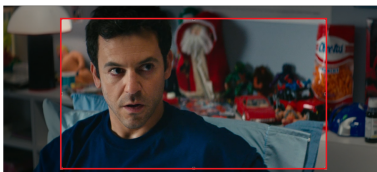



Fonte: Próprio autor

#### 4.2.2 Manipulações/Ataques no Vídeo

Para testar a robustez dos algoritmos de *hash* perceptivo, foram feitas manipulações intencionais no arquivo de vídeo e aplicou-se o novamente a extração dos quadros, geração dos valores *hash* e comparações junto ao banco de dados. O vídeo “once-upon-a-deadpool-trailer-1\_h720p.mov” foi submetido às alterações que preservam o conteúdo assim como descrito na seção 2.1. A ideia é simular o que poderia ser feito nas plataformas de compartilhamento de vídeo.

**Tabela 16 – Manipulações aplicadas no vídeo candidato.**

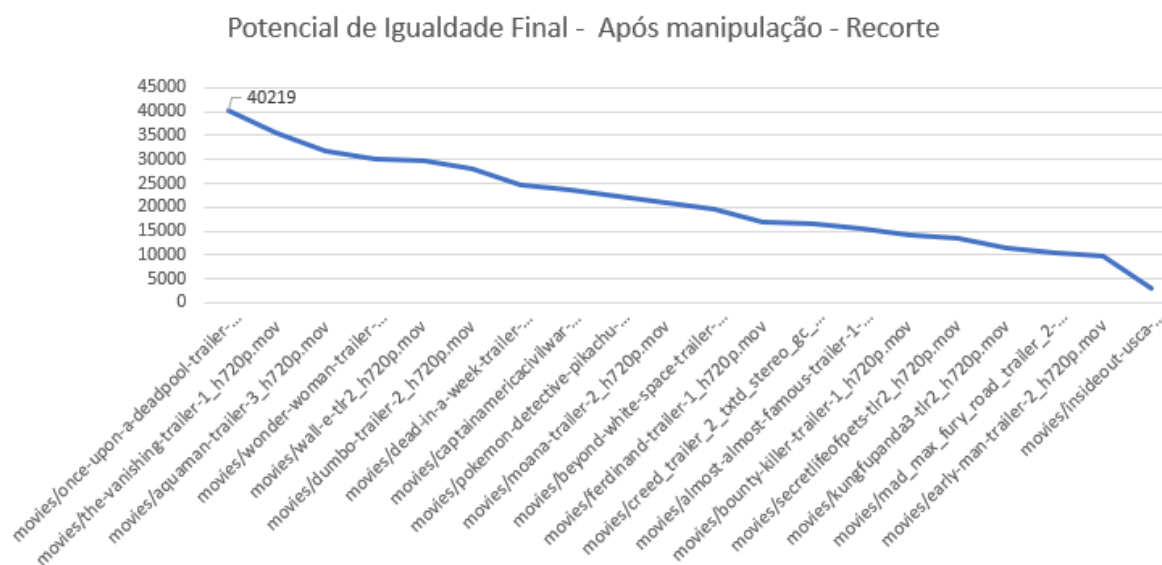
Manipulação	Exemplo	Descrição
Redução da Resolução		Vídeo convertido para uma versão de qualidade inferior - 640x480
Recorte		Vídeo recortado com redução de 30% de seu tamanho original
Rotação 10º		Vídeo rotacionado em 10º

Manipulação	Exemplo	Descrição
Rotação 90°		Video rotacionado em 90°
Rotação 180°		Video rotacionado em 180.

• **Recorte**

O recorte efetuado no vídeo reduziu a área da imagem em aproximadamente 30%, e isto teve um impacto visível no número de registros abaixo do limiar. A redução no número de registros encontrados foi de cerca de 75%. Mesmo com uma redução tão drástica nos valores, ainda foi possível identificar o registro com potencial de cópia, como mostrado na Figura 32.

**Figura 32 – Gráfico com os valores da comparação entre o vídeo candidato e as referências após manipulação - Recorte**



Fonte: Próprio autor

A figura 33 mostra a vantagem de se utilizar os valores *hash* combinados para a obtenção do valor PIF. Observa-se (em vermelho) que para os algoritmos *dHash*

e *pHash*, se utilizados individualmente, não seriam capazes de resistir ao ataque de recorte, uma vez que retornaram valores que eliminariam a possibilidade de identificação do vídeo. Os valores retornados por estes algoritmos sozinhos, sugerem o arquivo “the-vanishing-trailer-1\_h720p.mov” como referência equivalente ao arquivo “once-upon-a-deadpool-trailer-1\_h720p.mov”, o que não é correto. Não fosse isto suficiente, o vídeo correto é colocado na última posição segundo o cálculo *dHash* e em décimo sexto segundo o *pHash*. No entanto, quando utilizados juntos, as características fortes e robustez de um algoritmo compensa a possível vulnerabilidade do outro.

**Figura 33 – Tabela com os valores da comparação entre o vídeo candidato e as referências após manipulação - Recorte**

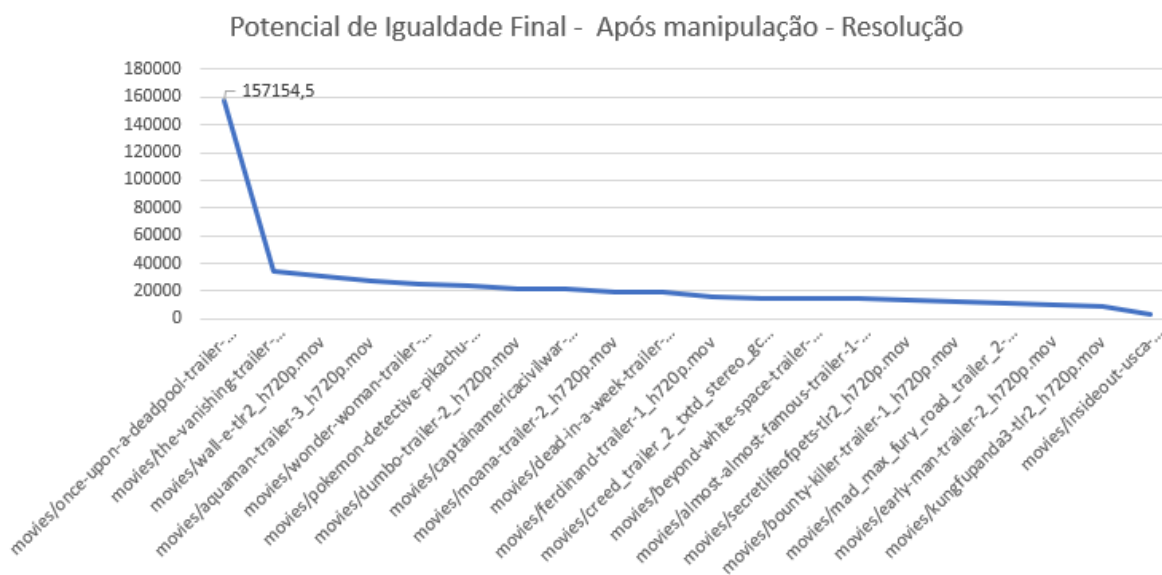
Candidato	Referência	aHash	dHash	pHash	WHash	PIF
movies/deadpool_cropped.mp4	movies/once-upon-a-deadpool-trailer-1_h720p.mov	79622	4372	3070	73812	40219
	movies/the-vanishing-trailer-1_h720p.mov	56440	29580	13302	43147	35617,25
	movies/aquaman-trailer-3_h720p.mov	44742	8398	6440	67339	31729,75
	movies/wonder-woman-trailer-3_h720p.mov	44697	11292	6784	57229	30000,5
	movies/wall-e-trl2_h720p.mov	60308	15090	5967	37303	29667
	movies/dumbo-trailer-2_h720p.mov	40728	10981	6649	53469	27956,75
	movies/dead-in-a-week-trailer-1_h720p.mov	41722	14262	5763	37584	24832,75
	movies/captainamericacivilwar-trl2_h720p.mov	35106	15706	10012	34442	23816,5
	movies/pokemon-detective-pikachu-trailer-1_h720p.mov	42423	8358	3723	34253	22189,25
	movies/moana-trailer-2_h720p.mov	24894	10635	8301	40005	20958,75
	movies/beyond-white-space-trailer-1_h720p.mov	34762	9217	4263	30663	19726,25
	movies/ferdinand-trailer-1_h720p.mov	21698	11008	4004	30999	16927,25
	movies/creed_trailer_2_txd_stereo_gc_h720p.mov	27667	14434	11118	13032	16562,75
	movies/almost-almost-famous-trailer-1-ca_h720p.mov	23719	9301	4343	24603	15491,5
	movies/bounty-killer-trailer-1_h720p.mov	22306	10644	3221	20355	14131,5
	movies/secretlifeofpets-trl2_h720p.mov	18681	11240	2252	21911	13521
movies/kungfupanda3-trl2_h720p.mov	18188	8075	3014	17276	11638,25	
movies/mad_max_fury_road_trailer_2-720p.mov	12336	12181	8579	8745	10460,25	
movies/early-man-trailer-2_h720p.mov	20011	4436	2795	11801	9760,75	
movies/insideout-usca-15sectease_h720p.mov	6033	4701	51	1235	3005	

Criado pelo autor

#### • Redução da Resolução

Neste teste, reduziu-se a resolução do vídeo candidato de 1280×544 para 640×480. A identificação foi positiva e com pouca alteração no número de registros abaixo do limiar de similaridade. Houve uma alteração de cerca de 0,41% no número de registros encontrados (Figura 34).

**Figura 34 – Gráfico com os valores da comparação entre o vídeo candidato e as referências após manipulação - Resolução**



Fonte: Próprio autor

Ao analisar os valores para cada algoritmos *hash*, na figura 35, nota-se que todos eles foram resistentes a este ataque, mantendo o vídeo correto na primeira posição.

**Figura 35 – Tabela com os valores da comparação entre o vídeo candidato e as referências após manipulação - Resolução**

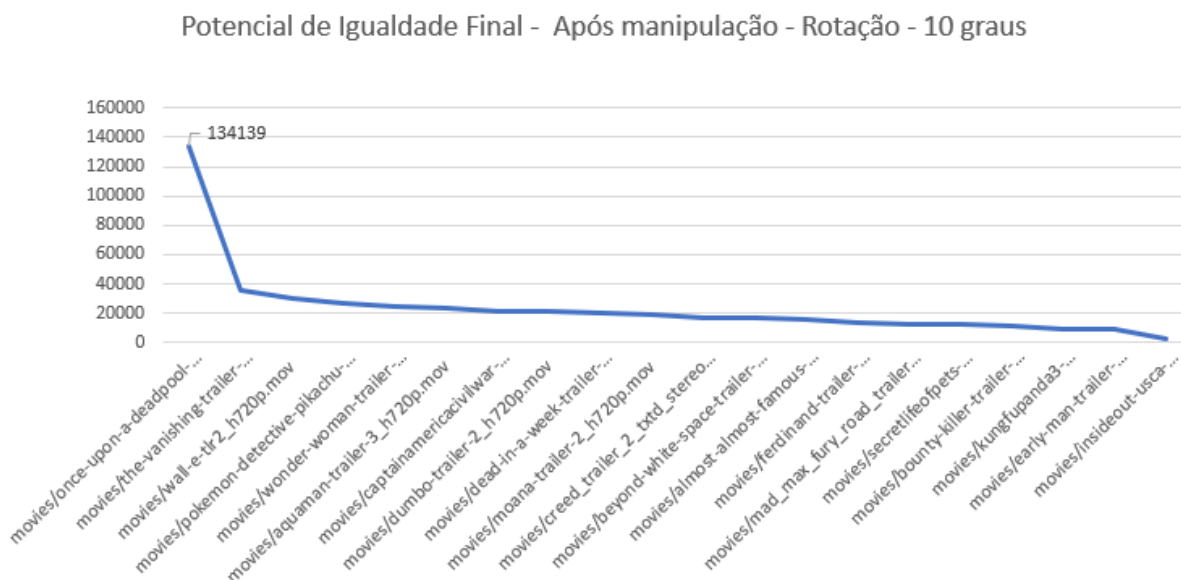
Candidato	Referência	aHash	dHash	pHash	WHash	PIF
/mnt/HD1/movies/deadpool_resize d_sd.mov	movies/once-upon-a-deadpool-trailer-1_h720p.mov	213967	123600	83136	207915	157154,5
	movies/the-vanishing-trailer-1_h720p.mov	49919	30884	11548	46221	34643
	movies/wall-e-trl2_h720p.mov	60655	15507	5268	40257	30421,75
	movies/aquaman-trailer-3_h720p.mov	50281	9950	5630	45229	27772,5
	movies/wonder-woman-trailer-3_h720p.mov	45052	14856	5986	36160	25513,5
	movies/pokemon-detective-pikachu-trailer-1_h720p.mov	46359	10986	3315	34385	23761,25
	movies/dumbo-trailer-2_h720p.mov	37530	11732	5736	30462	21365
	movies/captainamericacivilwar-trl2_h720p.mov	30238	17012	8834	28450	21133,5
	movies/moana-trailer-2_h720p.mov	27415	12461	7249	30110	19308,75
	movies/dead-in-a-week-trailer-1_h720p.mov	33258	17188	5086	20505	19009,25
	movies/ferdinand-trailer-1_h720p.mov	27745	11122	3465	22512	16211
	movies/creed_trailer_2_txtd_stereo_gc_h720p.mov	21001	20682	9844	9275	15200,5
	movies/beyond-white-space-trailer-1_h720p.mov	25294	10048	3628	21424	15098,5
	movies/almost-almost-famous-trailer-1-ca_h720p.mov	23986	10348	3830	20443	14651,75
	movies/secretlifeofpets-trl2_h720p.mov	18754	10920	1989	21941	13401
	movies/bounty-killer-trailer-1_h720p.mov	17795	11343	2876	15285	11824,75
	movies/mad_max_fury_road_trailer_2-720p.mov	14900	13428	7541	7898	10941,75
	movies/early-man-trailer-2_h720p.mov	18025	5041	2396	12685	9536,75
	movies/kungfupanda3-trl2_h720p.mov	16325	7966	2648	10798	9434,25
	movies/insideout-usca-15sectease_h720p.mov	5348	4164	45	1217	2693,5

Fonte: Próprio autor

• **Rotação de 10°, 90° e 180°**

Ataque de rotação são comuns no compartilhamento de vídeos na *internet*, porém existem um limite aceitável para a reprodução da mídia. Situações como as do uso de *Camcorder* como fonte, por exemplo, impõe inclinações da câmera. Compartilhamento da TV, por exemplo, por meio de plataformas como Periscope (<https://www.pscp.tv/>) comumente apresentam rotações superiores a 10 graus, porém a experiência é desagradável. Os testes executados neste trabalho foram com rotações de 10, 90 e 180 graus, sentido horário. Observou-se que o método se mostrou eficaz na identificação do vídeo com rotação em 10 graus, com pouca variação dos valores obtidos (~15%), Figura 36 e Figura 37.

**Figura 36 – Gráfico com os valores da comparação entre o vídeo candidato e as referências após manipulação - Rotação 10 graus**



Fonte: Próprio autor

**Figura 37 – Tabela com os valores da comparação entre o vídeo candidato e as referências após manipulação - Rotação 10 graus.**

Candidato	Referência	aHash	dHash	pHash	WHash	PIF
movies/deadpool_flipped_10.mov	movies/once-upon-a-deadpool-trailer-1_h720p.mov	186382	120124	56590	173460	134139
	movies/the-vanishing-trailer-1_h720p.mov	55137	36309	13058	40048	36138
	movies/wall-e-trl2_h720p.mov	65933	16524	5985	34194	30659
	movies/pokemon-detective-pikachu-trailer-1_h720p.mov	51805	15270	3748	36007	26707,5
	movies/wonder-woman-trailer-3_h720p.mov	47719	16245	6747	26524	24308,75
	movies/aquaman-trailer-3_h720p.mov	43120	10359	6376	35398	23813,25
	movies/captainamericacivilwar-trl2_h720p.mov	33086	17607	10008	26944	21911,25
	movies/dumbo-trailer-2_h720p.mov	39599	12561	6477	27580	21554,25
	movies/dead-in-a-week-trailer-1_h720p.mov	38889	18228	5764	19522	20600,75
	movies/moana-trailer-2_h720p.mov	30837	12906	8217	22951	18727,75
	movies/creed_trailer_2_txtd_stereo_gc_h720p.mov	24640	21752	11121	11135	17162
	movies/beyond-white-space-trailer-1_h720p.mov	27883	10838	4043	23186	16487,5
	movies/almost-almost-famous-trailer-1-ca_h720p.mov	26512	11079	4335	20143	15517,25
	movies/ferdinand-trailer-1_h720p.mov	20067	11802	3930	20137	13984
	movies/mad_max_fury_road_trailer_2-720p.mov	17281	13790	8536	10514	12530,25
	movies/secretlifeofpets-trl2_h720p.mov	18999	11687	2246	16717	12412,25
	movies/bounty-killer-trailer-1_h720p.mov	17954	11900	3225	14482	11890,25
	movies/kungfupanda3-trl2_h720p.mov	17668	8526	2963	9284	9610,25
	movies/early-man-trailer-2_h720p.mov	17458	5425	2706	9936	8881,25
	movies/insideout-usca-15sectease_h720p.mov	6098	4711	51	967	2956,75

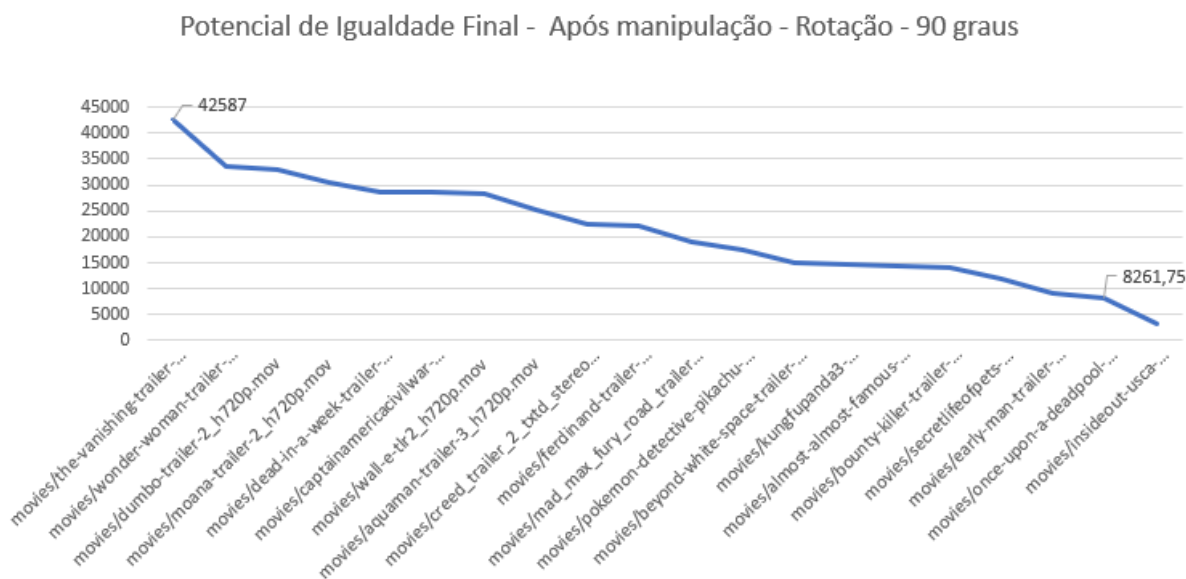
Fonte: Próprio autor

Não foi possível encontrar o vídeo candidato na base de referência utilizando os algoritmos de *hash* perceptivo (Figura 38 e 40), sob o ataque de rotação de 90 e 180 graus. Isto é explicado pelo fato dos algoritmos existentes na biblioteca ImageHash 4.0 serem baseados em estatísticas e relações, conforme explicado na seção 2.3. Isto quer dizer que utilizam, entre outras coisas, o pixel e sua posição (x,y) para geração do valor *hash*, fazendo com que *aHash*, *dHash*, *pHash* e *WHash* sejam pouco resistentes às transformações espacial agressiva.

O vídeo sugerido como cópia, durante o ataque de 90 graus foi o “the-vanishing-trailer-1\_h720p.mov” tendo o arquivo “once-upon-a-deadpool-trailer-1\_h720p.mov” na penúltima posição (Figura 39).



**Figura 38 – Gráfico com os valores da comparação entre o vídeo candidato e as referências após manipulação - Rotação 90 graus**



Fonte: Próprio autor

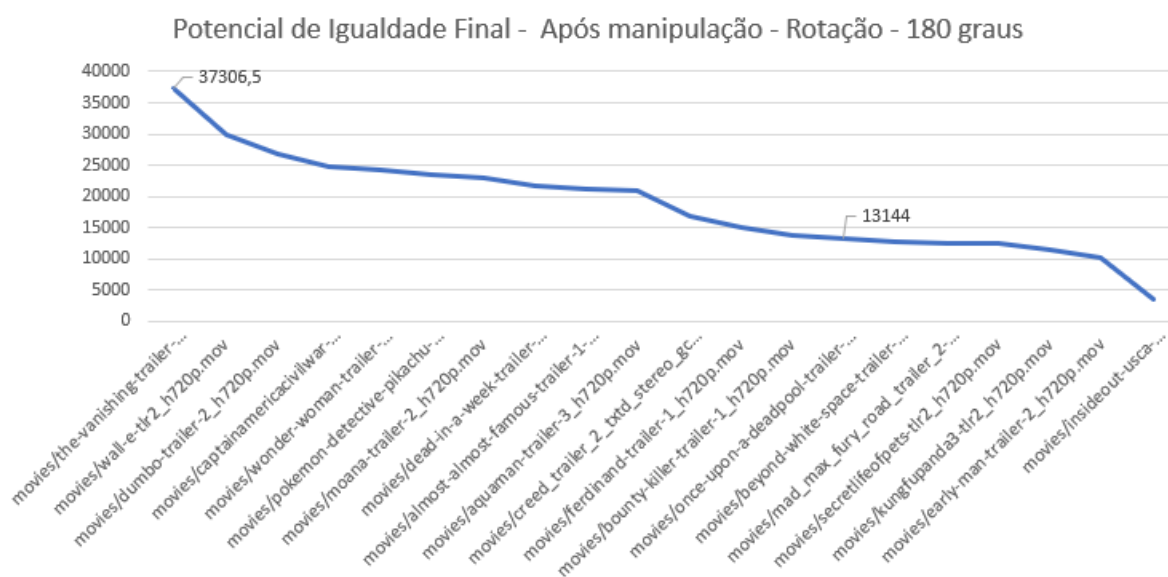
**Figura 39 – Tabela com os valores da comparação entre o vídeo candidato e as referências após manipulação - Rotação 90 graus.**

Candidato	Referência	aHash	dHash	pHash	WHash	PIF
movies/deadpool_flipped_90.mov	movies/the-vanishing-trailer-1_h720p.mov	53075	51292	13114	52867	42587
	movies/wonder-woman-trailer-3_h720p.mov	55316	23291	7044	49187	33709,5
	movies/dumbo-trailer-2_h720p.mov	44816	21971	6494	58836	33029,25
	movies/moana-trailer-2_h720p.mov	44946	23112	8267	45868	30548,25
	movies/dead-in-a-week-trailer-1_h720p.mov	52157	22536	5768	34352	28703,25
	movies/captainamericivilwar-tlr2_h720p.mov	35536	31840	10026	36962	28591
	movies/wall-e-tlr2_h720p.mov	58519	25336	5972	23482	28327,25
	movies/aquaman-trailer-3_h720p.mov	34140	17380	6410	42985	25228,75
	movies/creed_trailer_2_txtd_stereo_gc_h720p.mov	31288	31422	11347	15507	22391
	movies/ferdinand-trailer-1_h720p.mov	32674	18724	3928	33059	22096,25
	movies/mad_max_fury_road_trailer_2-720p.mov	25331	25072	8524	17823	19187,5
	movies/pokemon-detective-pikachu-trailer-1_h720p.mov	27743	15538	3811	22513	17401,25
	movies/beyond-white-space-trailer-1_h720p.mov	19169	17153	4063	20008	15098,25
	movies/kungfupanda3-tlr2_h720p.mov	28359	13673	2959	13888	14719,75
	movies/almost-almost-famous-trailer-1-ca_h720p.mov	18454	18240	4338	17047	14519,75
	movies/bounty-killer-trailer-1_h720p.mov	19621	16834	3218	16222	13973,75
	movies/secretlifeofpets-tlr2_h720p.mov	14271	16113	2245	15112	11935,25
	movies/early-man-trailer-2_h720p.mov	15745	8896	2720	8946	9076,75
	movies/once-upon-a-deadpool-trailer-1_h720p.mov	11891	8285	2601	10270	8261,75
	movies/insideout-usca-15sectease_h720p.mov	6019	6029	51	751	3212,5

Fonte: Próprio autor

O vídeo sugerido como cópia, durante o ataque de 180 graus foi o “the-vanishing-trailer-1\_h720p.mov” tendo o arquivo “once-upon-a-deadpool-trailer-1\_h720p.mov” na posição 14, Figura 41.

**Figura 40 – Gráfico com os valores da comparação entre o vídeo candidato e as referências após manipulação - Rotação 180 graus.**



Fonte: Próprio autor

**Figura 41 – Tabela com os valores da comparação entre o vídeo candidato e as referências após manipulação - Rotação 180 graus.**

Candidato	Referência	aHash	dHash	pHash	Whash	PIF
movies/deadpool_flipped.mov	movies/the-vanishing-trailer-1_..._h720p.mov	56508	34516	13267	44935	37306,5
	movies/wall-e-tlr2_..._h720p.mov	59793	24857	6315	28628	29898,25
	movies/dumbo-trailer-2_..._h720p.mov	49950	14592	6482	35752	26694
	movies/captainamericacivilwar-tlr2_..._h720p.mov	38584	19395	10016	30960	24738,75
	movies/wonder-woman-trailer-3_..._h720p.mov	37541	17007	6734	35776	24264,5
	movies/pokemon-detective-pikachu-trailer-1_..._h720p.mov	42108	13214	3755	34694	23442,75
	movies/moana-trailer-2_..._h720p.mov	35592	15563	8225	32965	23086,25
	movies/dead-in-a-week-trailer-1_..._h720p.mov	38064	17265	5971	25730	21757,5
	movies/almost-almost-famous-trailer-1-ca_..._h720p.mov	35510	12368	4396	32063	21084,25
	movies/aquaman-trailer-3_..._h720p.mov	34462	11286	6377	31120	20811,25
	movies/creed_trailer_2_txtd_stereo_gc_..._h720p.mov	23339	21991	11162	11233	16931,25
	movies/ferdinand-trailer-1_..._h720p.mov	23867	13219	3927	18998	15002,75
	movies/bounty-killer-trailer-1_..._h720p.mov	19532	13136	3229	18817	13678,5
	movies/once-upon-a-deadpool-trailer-1_..._h720p.mov	14215	20206	2601	15554	13144
	movies/beyond-white-space-trailer-1_..._h720p.mov	16649	12226	4114	18142	12782,75
	movies/mad_max_fury_road_trailer_2_..._h720p.mov	17331	14912	8532	8894	12417,25
	movies/secretlifeofpets-tlr2_..._h720p.mov	18196	11181	2247	17978	12400,5
	movies/kungfupanda3-tlr2_..._h720p.mov	19959	9234	2964	13536	11423,25
movies/early-man-trailer-2_..._h720p.mov	21787	5812	2708	10837	10286	
movies/insideout-usca-15sectease_..._h720p.mov	6090	4953	51	2505	3399,75	

Fonte: Próprio autor

## 5 Discussão

Considera-se que o ambiente proposto está longe de simular situações de uso real. O computador utilizado, e mesmo as técnicas de busca e comparação, poderiam ser mais eficientes e se beneficiar de multi-processamento, computação distribuída, inteligência de máquina e outras tecnologias mais avançadas. Recursos computacionais robustos são necessários para execução da escrita em disco e consulta no banco de dados. Durante os testes, a execução da consulta de 1 vídeo candidato contra 20 de referência duraram cerca de 40 minutos, utilizando os recursos e funções do próprio banco de dados e cerca de 2 horas, utilizando o programa escrito pelo autor para os cálculos.

O tamanho da base de vídeos de referência, por exemplo, limita a verificação de falsos positivos e falsos negativos, uma vez que a obtenção de algo próximo a realidade é impraticável sem recorrer a um base de dados maior - que é o que justamente o método procurou evitar. Além disso, os testes foram executados com vídeos de duração curta (em torno de 60 segundos), tendo como consequência a redução drástica do número de quadros a serem analisados.

O coeficiente de similaridade pode ser alterado com base nos resultados e comportamento de cada algoritmo em virtude dos ataques à que foram submetidos. Isto quer dizer que pesos iguais, talvez não reflitam de forma justa o comportamento de cada algoritmo, que, como visto produzem resultados distintos.

Ponderações a respeito do número mínimo de quadros necessários para uma identificação positiva não fizeram parte do escopo do estudo, mas é possível observar que quadros escuros, ou *takes* de longa duração tem impacto no número de valores *hash* e consequentemente no cálculo de similaridade. Parte do problema está relacionado ao uso de busca linear ou força bruta, que testa todos os quadros um a um, gerando um número de comparações provavelmente desnecessárias. Neste sentido, pré-processamento e análise dos quadros poderiam ser realizados, possibilitando ações como a identificação de *keyframes*, ou mesmo o descarte de frames considerados repetidos. A partir disto, seria possível definir o número mínimo de quadros necessários para identificação de um vídeo, excluindo aqueles que causam ruídos na métrica.

Relevante notar a baixa eficiência dos algoritmos de *hash* perceptivo quando o vídeo é submetido à modificação de carácter espacial. Ao rotacionar os quadros em 90 e 180 graus, não foi possível identificá-lo como cópia. Talvez, 90 e 180 graus possam ser manipulações exageradas, uma vez que é pouco provável, embora não impossível, que as plataformas de compartilhamento ou mesmo o usuário comum, submetam arquivos, com alta demanda e popularidade de cabeça para baixo. Rotações

superiores a 10 graus oferecem uma experiência desagradável àquele que consome o vídeo, portanto tais testes não foram executados.

## 6 Conclusões

Este trabalho demonstrou a possibilidade de identificação de cópia de vídeo por meio de *hash* perceptivo de imagens e apresentou os resultados obtidos através dos testes que foram realizados.

Utilizou-se ferramentas existentes como as presentes na biblioteca OpenCV, MySQL, a biblioteca em Python ImageHash 4.0, bem como programa desenvolvido pelo autor, para execução dos passos propostos.

Propôs-se uma técnica de identificação utilizando um coeficiente de similaridade ou limiar (threshold) e a combinação dos algoritmos de *hash* perceptivo de imagens (*aHash*, *dHash*, *pHash* e *wHash*) para investigação de potencial de cópia.

Testes a respeito do número de quadros, tempo de execução e desempenho foram detalhados, bem como a submissão dos vídeos a ataques que preservam o conteúdo. Foi analisado o comportamento do método diante destes ataques e os resultados foram, salvo as rotações de 90° e 180° graus, satisfatórios.

Conclui-se que é possível utilizar algoritmo de *hash* perceptivo de imagens para identificação de cópia de vídeo, no entanto, a combinação de mais de um deles preenche lacunas de desempenho e vulnerabilidades.

Foi mostrado que o método exige recursos de memória, processamento e técnicas avançadas de programação e consultas no banco de dados, para que se torne viável em situações de uso cotidiano.

A partir do cálculo de colisão observou-se que com o método proposto neste trabalho seria possível endereçar cerca de 7950 arquivos de vídeos e seus respectivos valores *hash*, antes de ocorrer a primeira colisão. O número pode ser considerado baixo para o uso em aplicações comerciais, que podem ter catálogos com quantidade de itens superior a 30000.

A técnica proposta é, portanto, candidata a ser utilizada como parte de ferramentas de identificação, ao incorporar sistemas de detecção de fraude de imagens e vídeos, bem como uso cível e jurídico, caso utilizado como dispositivo forense.

Destaca-se que o combate à cópias ilegais cumpre ainda um papel social ao empregar esforços na preservação de direitos individuais e coletivos, retorno de investimentos, geração de valor e até mesmo sustentabilidade e criação de oportunidades de empregos.

Sugere-se que trabalhos futuros possam incluir a utilização de inteligência artificial para identificação de quadros repetidos e/ou *key frames*, ou computação paralela, com o fito de aumentar a desempenho e cálculo dos resultados. Pode-se

ainda explorar a inserção de outros algoritmos resistentes a ataques, para adicionar robustez ao sistema. E mesmo número e características dos ataques, bem como os pesos atribuídos para geração do coeficiente de similaridade podem ser revistos para garantir a cobertura do maior número possível de vulnerabilidades e maior eficiência.

## Referências

- ABDEL-MOTTALEB, M.; KRISHNAMACHARI, S.; VAITHILINGAM, G. Signature-based image identification. *Proceedings of SPIE - The International Society for Optical Engineering*, v. 1, n. 3845, p. 22 – 28, Nov 1999. Disponível em: <[https://www.researchgate.net/publication/244441813\\_Signature-based\\_image\\_identification](https://www.researchgate.net/publication/244441813_Signature-based_image_identification)>. Acesso em: 04/12/2018.
- AGÊNCIA O GLOBO. *O pioneiro das transmissões de vídeo pela internet*. 2018/01. Disponível em: <<https://epocanegocios.globo.com/Tecnologia/noticia/2018/01/o-pioneiro-das-transmissoes-de-video-pela-internet.html>>. Acesso em: 03/12/2018.
- ALAHMAD, M. A.; ALSHAIKHLI, I. F. Broad View of Cryptographic Hash Functions. *ISI Journal*, ISI Journal, v. 1, n. -, p. 4 – 10, Feb 2013. Disponível em: <[https://www.researchgate.net/publication/261875331\\_AIAhmad\\_M\\_A\\_and\\_I\\_F\\_Alshaiikli\\_2013\\_Broad\\_View\\_of\\_Cryptographic\\_Hash\\_Functions\\_International\\_Journal\\_of\\_Computer\\_Science\\_Issues\\_journal\\_ISI\\_Journal\\_104](https://www.researchgate.net/publication/261875331_AIAhmad_M_A_and_I_F_Alshaiikli_2013_Broad_View_of_Cryptographic_Hash_Functions_International_Journal_of_Computer_Science_Issues_journal_ISI_Journal_104)>. Acesso em: 04/12/2018.
- BHATTACHARJEE, S.; KUTTER, M. Compression Tolerant Image Authentication. *International Conference on Image Processing*, IEEE, 1998.
- BOURKE, P. *AutoCorrelation – 2D Pattern Identification*. 1996. ONLINE. Disponível em: <<http://paulbourke.net/miscellaneous/correlate/>>. Acesso em: 04/12/2018.
- BRACEWELL, R. “Pentagram Notation for Cross Correlation.” *The Fourier Transform and Its Applications*. New York: McGraw-Hill, p. 46 – 243, 1965.
- BREED, G. Bit Error Rate: Fundamental Concepts and Measurement Issues. *High Frequency Electronics*, Summit TechnicalMedia LLC, 2003.
- BRIN, S.; DAVIS, J.; GARCÍA-MOLINA, H. Copy detection mechanisms for digital documents. In: . [S.l.: s.n.], 1995.
- BUSCHNER, J. *ImageHash*. 2016. Disponível em: <<https://pypi.org/project/ImageHash/>>. Acesso em: 10/06/2018.
- CALUDE, C.; SALOMAA, K.; YU, S. Additive Distances and Quasi-Distances Between Words. *J. UCS*, v. 8, n. 2, p. 141 – 152, 2002. Disponível em: <<http://dx.doi.org/10.3217/jucs-008-02-0141>>.
- CHEN, S.; MA, B.; ZHANG, K. On the similarity metric and the distance metric. *Theor. Comput. Sci.*, v. 410, n. 24-25, p. 2365 – 2376, 2009. Disponível em: <<http://dx.doi.org/10.1016/j.tcs.2009.02.023>>.
- CHEUNG, S. S.; ZAKHOR, A. Efficient video similarity measurement with video signature. *IEEE Trans. Circuits Syst. Video Techn.*, v. 13, n. 1, p. 59 – 74, 2003. Disponível em: <<http://dx.doi.org/10.1109/TCSVT.2002.808080>>.
- CISCO PUBLIC. *Cisco Visual Networking Index: Forecast and Trends, 2017–2022*. [S.l.], 2018. Disponível em: <<https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf>>. Acesso em: 04/12/2018.

COSKUN, B.; SANKUR, B.; MEMON, N. D. Spatio-Temporal Transform Based Video Hashing. *IEEE Trans. Multimedia*, v. 8, n. 6, p. 1190 – 1208, 2006. Disponível em: <<http://dx.doi.org/10.1109/TMM.2006.884614>>.

COX, I. J. et al. Secure spread spectrum watermarking for multimedia. *IEEE Trans. Image Processing*, v. 6, n. 12, p. 1673 – 1687, 1997. Disponível em: <<http://dx.doi.org/10.1109/83.650120>>.

DEGUILLAME, F.; CSURKA, G.; PUN, T. Countermeasures for Unintentional and Intentional Video Watermarking Attacks. *Proceedings of SPIE 3971, Security and Watermarking of Multimedia Content II*, v. 3971, n. -, p. 346 – 357, Jan 2001.

DRMIC, A. et al. Evaluating robustness of perceptual image hashing algorithms. In: . [S.l.: s.n.], 2017.

DTVE. *Piracy to cost TV and film industry US\$52bn by 2022*. 2017. ONLINE. Disponível em: <<https://www.digitaltveurope.com/2017/10/30/piracy-to-cost-tv-and-film-industry-us52bn-by-2022/>>. Acesso em: 04/12/2018.

ENCODING.COM. *Most popular encoding formats*. 2018. ONLINE. Disponível em: <<https://www.encoding.com/formats/>>. Acesso em: 04/12/2018.

FACEBOOK. *Primeiros passos com o Rights Manager*. 2018. ONLINE. Disponível em: <<https://www.facebook.com/help/publisher/1824313947806360>>. Acesso em: 04/12/2018.

FISHER, R. B.; OLIVER, P. Multi-Variate Cross-Correlation and Image Matching. *Proceedings of the 6th British machine vision conference 1995*, v. 1,2, 1998.

FRIDRICH, J.; GOLJAN, M. Robust hash functions for digital watermarking. *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'00)*, IEEE Computer Society, Washington,DC, v. 1, n. 1, p. 178 – 183, 2000.

G1.GLOBO.COM. *Brasil perde R\$ 130 bilhões por ano com pirataria, contrabando e comércio ilegal, aponta estudo*. Brasília: [s.n.], 2017. ONLINE. Disponível em: <<https://g1.globo.com/economia/noticia/brasil-perde-r-130-bilhoes-por-ano-com-pirataria-contrabando-e-comercio-ilegal-aponta-estudo.ghtml>>. Acesso em: 04/12/2018.

GOOGLE. *How Google Fights Piracy*. 2018. ONLINE. Disponível em: <[https://www.blog.google/documents/25/GO806\\_Google\\_FightsPiracy\\_eReader\\_final.pdf](https://www.blog.google/documents/25/GO806_Google_FightsPiracy_eReader_final.pdf)>. Acesso em: 04/12/2018.

HADMI, A. et al. Perceptual Image Hashing. *Watermarking, Dr. Mithun Das Gupta (Ed.)*, v. 2, 2012. ISSN 978-953-51-0619-7. Disponível em: <<http://www.intechopen.com/books/watermarking-volume-2>>.

HAJJ-AHMAD, A. et al. Flicker Forensics for Camcorder Piracy. *IEEE Transactions on Information Forensics and Security*, IEEE, v. 12, n. 1, p. 89 – 100, Jan 2017. ISSN 1556-6013. Disponível em: <<https://ieeexplore.ieee.org/document/7553576/authors>>. Acesso em: 04/12/2018.



HAMMING, R. W. Error detecting and error correcting codes. *The Bell System Technical Journal*, 1950.

HAN, S.; CHU, C. Content-based image authentication: current status, issues, and challenges. *Int. J. Inf. Sec.*, v. 9, n. 1, p. 19 – 32, 2010. Disponível em: <<http://dx.doi.org/10.1007/s10207-009-0093-2>>.

HANDSCHUH, H. SHA-0, SHA-1, SHA-2 (Secure Hash Algorithm). In: *Encyclopedia of Cryptography and Security (2nd Ed.)*. [s.n.], 2011. p. 1190 – 1193. Disponível em: <[http://dx.doi.org/10.1007/978-1-4419-5906-5\\_615](http://dx.doi.org/10.1007/978-1-4419-5906-5_615)>.

IRONPAPER. *Online Marketing Trends 2016 to 2017*. 2016/11. ONLINE. Disponível em: <<https://www.ironpaper.com/webintel/articles/online-marketing-trends-2016-2017/>>. Acesso em: 04/12/2018.

JEJDLING, F. *Ericsson Mobility Report - November 2018*. [S.l.], 2018. Disponível em: <<https://www.ericsson.com/assets/local/mobility-report/documents/2018/ericsson-mobility-report-november-2018.pdf>>. Acesso em: 04/12/2018.

JOLY, A.; BUISSON, O.; FRÉLICOT, C. Content-Based Copy Retrieval Using Distortion-Based Probabilistic Similarity Search. *IEEE Trans. Multimedia*, v. 9, n. 2, p. 293 – 306, 2007. Disponível em: <<http://dx.doi.org/10.1109/TMM.2006.886278>>.

JOSHUA, T. P.; ARRIVUKANNAMMA, M.; SATHIASEELAN, J. G. R. Comparison of DCT and DWT Image Compression. *Conference Proceedings, IEEE*, -, n. -, p. – – –, - 2016. Disponível em: <<https://www.semanticscholar.org/paper/Comparison-of-DCT-and-DWT-Image-Compression-Joshua-Arrivukannamma/100c1136e6402bac95324c46a59697059d44d4ee?navId=citing-papers>>. Acesso em: 04/12/2018.

KALKER, T.; HAITSMA, J.; OOSTVEEN, J. Issues with Digital Watermarking and Perceptual Hashing. *Proceedings of SPIE - The International Society for Optical Engineering*, v. 1, n. 1, p. 189 – 197, Nov 2001. Disponível em: <[https://www.researchgate.net/publication/244433134\\_Issues\\_with\\_digital\\_watermarking\\_and\\_perceptual\\_hashing](https://www.researchgate.net/publication/244433134_Issues_with_digital_watermarking_and_perceptual_hashing)>. Acesso em: 04/12/2018.

KASKALIS, T. H.; PITAS, I. Applying Signatures on Digital Images. *Workshop on Nonlinear Signal Processing*, IEEE, Thessaloniki, Greece, -, n. -, p. 460 – 463, Oct 1995. Disponível em: <[https://www.researchgate.net/publication/27383612\\_Applying\\_Signatures\\_on\\_Digital\\_Images](https://www.researchgate.net/publication/27383612_Applying_Signatures_on_Digital_Images)>. Acesso em: 04/12/2018.

KHELIFI, F.; JIANG, J. Analysis of the Security of Perceptual Image Hashing Based on Non-Negative Matrix Factorization. *IEEE Signal Process. Lett.*, v. 17, n. 1, p. 43 – 46, 2010. Disponível em: <<http://dx.doi.org/10.1109/LSP.2009.2032451>>.

KHELIFI, F.; JIANG, J. K -NN Regression to Improve Statistical Feature Extraction for Texture Retrieval. *IEEE Trans. Image Processing*, v. 20, n. 1, p. 293 – 298, 2011. Disponível em: <<http://dx.doi.org/10.1109/TIP.2010.2052277>>.

KONDOZ, A. *Visual Media Coding and Transmission*. 1. ed. West Sussex, United Kingdom: Wiley & Sons Ltd, 2009. v. 1. (1, v. 1). ISBN 9780470740576. Disponível em: <<https://onlinelibrary.wiley.com/doi/book/10.1002/9780470740644>>. Acesso em: 04/12/2018.

- KOZAT, S.; VENKATESAN, R.; MIHCAK, M. Robust perceptual image hashing via matrix invariants. In: . [S.l.: s.n.], 2004.
- KUMAR, G. S.; MANIKANTA, G. A novel framework for video content infringement detection and prevention. *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, IEEE, Mysore, India, v. 1, n. 1, p. 424 – 429, Aug 2013. ISSN 978-1-4673-6217-7. Disponível em: <<https://ieeexplore.ieee.org/document/6637209>>. Acesso em: 04/12/2018.
- KUTTER, M.; VOLOSHYNOVSKIY, S.; VOLOSHYNOVSKIY, S. Watermark Copy Attack. *Proceedings of SPIE 3971, Security and Watermarking of Multimedia Content II*, v. 3971, n. -, p. 371 – 380, Jan 2000.
- LEE, S.; YOO, C. D. Robust video fingerprinting based on 2D-OPCA of affine covariant regions. In: . [S.l.: s.n.], 2008.
- LEE, S.; YOO, C. D.; KALKER, T. Robust video fingerprinting based on symmetric pairwise boosting. 2009.
- LIN, C.; CHANG, S. A robust image authentication method distinguishing JPEG compression from malicious manipulation. *IEEE Trans. Circuits Syst. Video Techn.*, v. 11, n. 2, p. 153 – 168, 2001. Disponível em: <<http://dx.doi.org/10.1109/76.905982>>.
- LINDENBERG, C.; WIRT, K. SHA1, RSA, PSS and more. *Archive of Formal Proofs*, v. 2005, 2005. Formal proof development. Disponível em: <<https://www.isa-afp.org/entries/RSAPSS.shtml>>.
- LU, C.; LIAO, H. M. Structural digital signature for image authentication: an incidental distortion resistant scheme. *IEEE Trans. Multimedia*, v. 5, n. 2, p. 161 – 173, 2003. Disponível em: <<http://dx.doi.org/10.1109/TMM.2003.811621>>.
- MAANI, E.; TSAFTARIS, S. A.; KATSAGGELOS, A. K. Local feature extraction for video copy detection in a database. In: . [S.l.: s.n.], 2008.
- MARTYN.JOHNSON. *The Trojan Room Coffee Machine*. 1994. ONLINE. Disponível em: <<https://www.cl.cam.ac.uk/coffee/coffee.html>>. Acesso em: 04/12/2018.
- MCALONE, N. *People became even more addicted to Netflix in 2015, according to Goldman Sachs*. 2016. ONLINE. Disponível em: <<https://www.businessinsider.com/subscribers-spent-more-time-per-person-watching-netflix-in-2015-2016-1>>. Acesso em: 04/12/2018.
- MENEZES, A. J.; OORSCHOT, P. C. van; VANSTONE, S. A. *Handbook of applied cryptography*. Boca Raton: CRC Press, 1996. ISBN 9780849385230.
- METZNER, J. et al. Closed claims' analysis. v. 25, n. 2, p. 263 – 276, 2011. Disponível em: <<http://dx.doi.org/10.1016/j.bpa.2011.02.007>>.
- MILOJICIC, D. S. et al. Peer-to-Peer Computing. *HP Laboratories Palo Alto*, Hewlett-Packard Company, Palo Alto, -, n. -, p. – – –, Mar 2002. Disponível em: <<http://www.cs.ucsb.edu/~almeroth/classes/F02.276/papers/p2p.pdf>>. Acesso em: 04/12/2018.

- MITTAL, R. P2P Networks: Online Piracy of Music, Films and Computer Software. *Journal of Intellectual Property Rights*, v. 9, n. -, p. 440 – 461, Sep 2004. Disponível em: <<http://nopr.niscair.res.in/bitstream/123456789/4884/1/JIPR%209%285%29%20440-461.pdf>>. Acesso em: 04/12/2018.
- MOENS, M.; LI, J. Mining User Generated Content and Its Applications. In: *Mining User Generated Content*. [s.n.], 2014. p. 3 – 17. Disponível em: <<http://www.crcnetbase.com/doi/abs/10.1201/b16413-3>>.
- MONGA, V.; EVANS, B. L. Robust perceptual image hashing using feature points. In: . [S.l.: s.n.], 2004.
- MUTHU, R.; RANI, C. Perceptual Hashing For Efficient Fingerprint Based Identification. *Advanced Computing and Communication Systems (ICACCS)*, 2017.
- NIKOLAIDIS, N.; PITAS, J. image and video fingerprinting for digital rights management of multimedia data. 2006.
- NOWAK, N.; ZABIEROWSKI, W. Methods of sound data compression — Comparison of different standards. In: ELETRÔNICO, 2011, Polyana-Svalyava, Ukraine. *2011 11th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*. Polyana-Svalyava, Ukraine, 2011. p. – – –. ISBN 978-966-2191-17-2. Disponível em: <<https://ieeexplore.ieee.org/document/5744526>>. Acesso em: 04/12/2018.
- OZSARI, T. A Hash of Hash Functions. *CoRR*, cs.CR/0310033, 2003. Disponível em: <<http://arxiv.org/abs/cs.CR/0310033>>.
- PAAR, C.; PELZL, J. *Understanding Cryptography: A Textbook for Students and Practitioners*. [S.l.]: Springer Berlin Heidelberg, 2009. ISBN 9783642041013.
- READ, P.; MEYER, M. *Restoration of Motion Picture Film*. 1. ed. Butterworth-Heinemann, 2000. ISBN 9780750627931. Disponível em: <[https://books.google.com.br/books?id=jzbUUL0xJAEC&pg=PA24&redir\\_esc=y#v=onepage&q&f=false](https://books.google.com.br/books?id=jzbUUL0xJAEC&pg=PA24&redir_esc=y#v=onepage&q&f=false)>. Acesso em: 04/12/2018.
- REZA, M. S. et al. An Approach of Digital Image Copyright Protection by Using Watermarking Technology. *CoRR*, abs/1205.6229, 2012. Disponível em: <<http://arxiv.org/abs/1205.6229>>.
- ROBINSON, J. The k-d-b-tree: A search structure for large multidimensional dynamic indexing. *Proc. ACM SIGMOD Int. Conf. Management Data*, p. 10 – 18, 1981.
- SAHINALP, S. et al. Distance based indexing for string proximity search. *Proceedings of the 19th International Conference on Data Engineering*, p. 125 – 136, 2003.
- SCIENCE, M. L. for C.; SECURITY, I. R. D. *The MD5 Message-Digest Algorithm*. 1992.
- SEGERS, J. *ImageHash*. 2014. Disponível em: <<https://github.com/jenssegers/imagehash>>. Acesso em: 16/06/2018.
- SEO, J. S. et al. A robust image fingerprinting system using the Radon transform. *Sig. Proc.: Image Comm.*, v. 19, n. 4, p. 325 – 339, 2004. Disponível em: <<http://dx.doi.org/10.1016/j.image.2003.12.001>>.

- SMITH, T.; WATERMAN, M. Comparison of biosequences. *Advances in Applied Mathematics* 2, p. 482 – 489, 1981.
- SOUTH America Television Piracy Landscape For Alianza Contra la Piratería de Televisión Paga. 2016.
- SRUOGINIS, K.; WARREN, J. *Live Video Streaming-A Global Perspective*. [S.l.], 2018. Disponível em: <<https://www.iab.com/wp-content/uploads/2018/06/IAB-Live-Video-Streaming-Trends.pdf>>. Acesso em: 04/12/2018.
- SU, X.; HUANG, T.; GAO, W. Robust video fingerprinting based on visual attention regions. In: . [S.l.: s.n.], 2009.
- SWAMINATHAN, A.; MAO, Y.; WU, M. Robust and secure image hashing. *IEEE Transactions on Information Forensics and Security*, 1(2), p. 215 – 230, 2006.
- TORSELLO, A.; ROWE, D. H.; PELILLO, M. Polynomial-Time Metrics for Attributed Trees. *IEEE Trans. Pattern Anal. Mach. Intell.*, v. 27, n. 7, p. 1087 – 1099, 2005. Disponível em: <<http://doi.ieeecomputersociety.org/10.1109/TPAMI.2005.146>>.
- VENKATESAN, R. et al. Robust Image Hashing. *Proceedings of the International Conference on Image Processing (ICIP)*, v. 3, p. 664 – 666, September 2000.
- VERDOLIVA, L. Handbook of Digital Forensics of Multimedia Data and Devices [Book Reviews]. *IEEE Signal Process. Mag.*, v. 33, n. 1, p. 164 – 165, 2016. Disponível em: <<http://dx.doi.org/10.1109/MSP.2015.2488018>>.
- VOLOSHYNOVSKIY, S. et al. Attack modeling: Towards a Second Generation Benchmark. *Signal Processing, Special Issue: Information Theoretic Issues in Digital Watermarking*, v. 81, n. 6, p. 1177 – 1214, Jun 2001.
- VURAL, C.; BARAKLI, B. Reversible video watermarking using motion-compensated frame interpolation error expansion. *Signal, Image and Video Processing*, v. 9, n. 7, p. 1613 – 1623, 2015. Disponível em: <<http://dx.doi.org/10.1007/s11760-014-0618-7>>.
- WENG, L.; PRENEEL, B. ATTACKING SOME PERCEPTUAL IMAGE HASH ALGORITHMS. *2007 IEEE International Conference on Multimedia and Expo*, IEEE, Beijing, China, -, n. -, p. – – –, Aug 2007. Disponível em: <<https://ieeexplore.ieee.org/document/4284791?arnumber=4284791>>.
- WENG, L.; PRENEEL, B. A Secure Perceptual Hash Algorithm for Image Content Authentication. *CMS'11 Proceedings of the 12th IFIP TC 6/TC 11 international conference on Communications and multimedia security*, p. 108 – 121, October 2011.
- WIKIPEDIA. *Lena Söderberg*. 2019. ONLINE. Disponível em: <[https://en.wikipedia.org/wiki/Lena\\_S%C3%B6derberg](https://en.wikipedia.org/wiki/Lena_S%C3%B6derberg)>. Acesso em: 04/12/2018.
- WU, S. et al. Efficiently self-synchronized audio watermarking for assured audio data transmission. *TBC*, v. 51, n. 1, p. 69 – 76, 2005. Disponível em: <<http://dx.doi.org/10.1109/TBC.2004.838265>>.
- YANG, B.; GU, F.; NIU, X. Block mean value based image perceptual hashing. In: . [S.l.: s.n.], 2006.

YANG, X.; SUN, Q.; TIAN, Q. Content-based video identification: A survey. In: . [S.l.: s.n.], 2003.

ZAUNER, C. Implementation and Benchmarking of Perceptual Image Hash Functions. 2010.

## **Apêndices**

## APÊNDICE A – Código criado em Python demonstrando a utilização da biblioteca ImageHash para geração de valor hash

**Código A.1 – Código criado em Python demonstrando a utilização da biblioteca ImageHash para geração de valor hash**

```
#!/usr/bin/python

#title           :perceptive.py
#description     :Generate hash values from image
#                given as parameter
#author         :Anderson Torres
#date           :20181124
#version        :1.0
#usage          :python perceptive.py
#notes          :
#python_version :2.6.6
#
```

=====

```
from PIL import Image
import imagehash

class hash:
    def __init__(self):
        pass

    def ahash(self, image):
        h = imagehash.average_hash(Image.open(image))
        return h

    def dhash(self, image):
        h = imagehash.dhash(Image.open(image))
        return h

    def phash(self, image):
        h = imagehash.phash(Image.open(image))
        return h

    def whash(self, image):
        h = imagehash.whash(Image.open(image))
        return h
```

## APÊNDICE B – Código em Python utilizado para extração dos quadros do vídeo

Código B.1 – Código em python para extração dos quadros do vídeo

```
#!/usr/bin/python

#title           :frames.py
#description     :Extract frames from videos passed as
    argument:
#author         :Anderson Torres
#date           :20181124
#version        :1.0
#usage          :called within class
#notes          :
#python_version :2.6.6
#
=====

import time
import cv2
import threading
import imutils

class extraction:

    def __init__(self,location,video):
        self.location = location
        self.cap = cv2.VideoCapture(video)
        self.video_length = int(self.cap.get(cv2.CAP_PROP_FRAME
            _COUNT)) - 1
        self.video_fps = round(self.cap.get(cv2.CAP_PROP_FPS)
            ,2)
        self.video_time = round((self.video_length / self.video
            _fps),2)
        self.video_resolution = str(self.cap.get(cv2.CAP_PROP_
            FRAME_WIDTH)) + "/" + str(self.cap.get(cv2.CAP_PROP_
            FRAME_HEIGHT ))

    def saveImages(self,imagenname,frame):
        t = threading.Thread(target=cv2.imwrite,args=(
            imagenname,frame,))
        t.start()

    def info(self):
        print (" [INFO] ")
        print (" - Video Reproduction time: %s seconds" ) %(self.
            video_time)
        print (" - Frame Rate: %s/fps" ) %(self.video_fps)
        print (" - Number of frames: %s" ) %(self.video_length)
```



```
print (" - Video Resolution: %s" % (self.video_
      resolution))

def extract(self):
    # Log the time
    time_start = time.time()
    count = 0

    # Start converting the video
    print (" [INFO] ")
    print (" - Starting frames extraction ")

    while self.cap.isOpened():
        ret, frame = self.cap.read() #Extracting Frames
        frame_time = round((self.cap.get(cv2.CAP_PROP_POS_
            MSEC)/1000),2)
        frame_name = self.location + "/%#05d_%s.jpg" % (count
            +1,frame_time)
        frame_small = imutils.resize(frame, width=640) #
            Resizing the frame to save space and processing
        # Calling function to save images in folder
        self.saveImages(frame_name,frame_small)
        count = count + 1
        # If there are no more frames left
        video_length = int(self.cap.get(cv2.CAP_PROP_FRAME_
            COUNT)) - 1
        if (count > (video_length - 1)):
            # Log the time again
            time_end = time.time()
```

## APÊNDICE C – Amostra do código utilizado para extração do valores hash a partir dos quadros do vídeo

**Código C.1 – Amostra do código utilizado para extração do valores hash a partir dos quadros do vídeo**

```
#!/usr/bin/python

#title          :generate.py
#description    :Extract video frames and generates
                hashes using ImageHash functions:
#author        :Anderson Torres
#date          :20181124
#version       :1.0
#usage         :python generate.py [-h] --movie
                MOVIE --type TYPE
#notes         :
#python_version :2.6.6
#
=====

from frames import extraction
from perceptive import hash
from database.additems import additems

import sys
import os
import time
import hashlib
import json

import argparse

class generate:
    """ Main class that call the perceptual hash methods """

    def __init__(self,video):
        """ Initiating the class
        Parameters
        -----
        video: object
            The actual videofile to be processed

        """

        self.video = video
        self.frameslocation = ''

    def location(self):
```

```

""" Generate the folder based on the video file sha1
    string """

hash_object = hashlib.sha1(self.video)
self.videosha = hash_object.hexdigest() # sha1 hash to
    distinguish the video file and make it unique in
    the database
self.frameslocation = "frames/" + self.videosha

try:
    print ("[INFO]")
    print (" - Fingerprinting process started")
    print (" - Frames folder created with name: %s" % (
        self.videosha)
        os.mkdir(self.frameslocation)

except OSError:
    pass

def framesextraction(self):
    """ Calls the method that will actually extract the
        video frames """
    self.location()
    self.v = extraction(self.frameslocation, self.video)
    self.v.info() # Prints the video
        information
    self.v.extract() # The actual frame
        extraction
    print ("[INFO]")
    print (" - Frames extraction completed")

...

...

def hashes(self, videotype):
    """ Print the hashes for Generate a list with the
        frames files names """
    """ Generate a list with the frames files names """
    print ("[INFO]")
    print (" - Generating hashes..\n")
    d = additems()
    files = self.images()
    time.sleep(5)
    h = hash()
    for file in sorted(files):

        ahash = str(h.ahash(file))
        dhash = str(h.dhash(file))
        phash = str(h.phash(file))
        whash = str(h.whash(file))

        if videotype == 'reference':

```

```
d.referenceHash(self.videosha, file, ahash,
                dhash, phash, whash)
print ("%s %s %s %s %s %s %s") %(videotype,
    self.videosha, file, ahash, dhash, phash,
    whash)

if videotype == 'candidate':
    d.candidateHash(self.videosha, file, ahash,
                   dhash, phash, whash)
    print ("%s %s %s %s %s %s %s") %(videotype,
        self.videosha, file, ahash, dhash, phash,
        whash)

d.dbclose()
```

## APÊNDICE D – Código em Python que efetua o cálculo da Distância de Hamming entre valores hash

**Código D.1 – Código em Python que efetua o cálculo da Distância de Hamming entre valores hash**

```
#!/usr/bin/python

#title           :hamming.py
#description      :Calculates hamming distance from 2
                  hash values strings of the same size.
#author          :Anderson Torres
#date            :20181124
#version         :1.0
#usage           :python hamming.py
#notes          :
#python_version :2.6.6
#
=====

import binascii

class hamming:

    def __init__(self):
        pass

    def transform(self, hexvalue):
        integer = int(hexvalue, 16)
        binvalue = format(integer, '0>64b')
        return binvalue

    def distance(self, a, b):
        lengthofa = len(a)
        lengthofb = len(b)
        if lengthofa != lengthofb:
            return False
        count = 0
        for i in range (0, lengthofa):
            if a[i] != b[i]:
                count += 1
        return count
```

## APÊNDICE E – Código em Python que efetua busca e comparações em memória

### Código E.1 – Código em Python que efetua busca e comparações em memória

```
#!/usr/bin/python
#title :compare_linear.py
#description :Compare Perceptual Hashes
#author :Anderson Torres
#date :20181124
#version :1.0
#usage :python compare_linear.py
#notes :
#python_version :2.6.6
#=====

from database.finditems import finditems
from hamming.hamming import hamming
from progressbar import ProgressBar, Timer, Counter,
    Percentage, Bar
import time
import timing
import codecs

class compare:

    def __init__(self):
        self.a = finditems()

    def getValues(self, candID, candName, refeID, refeName):

        try:
            print("Processing %s| %s" % (candName, refeName))
            filename = candID + "_x_" + refeID + ".csv"
            chashes = self.a.findCandidateHashes(candID)
            rhashes = self.a.findReferenceHashes(refeID)
            self.getSimilar(chashes, rhashes, filename)
        except:
            print("Something wrong happened")

    def getSimilar(self, chashes, rhashes, filename):
        results = "/mnt/HD1/comparacoes/software/" + filename

        h = hamming()
        f = codecs.open(results, "a", 'utf-8-sig')
        f.write("loop;loop1;c_videoid;r_videoid;c_entryid;r_
            entryid;c_hash;r_hash;distance;hash\n")

        loop = 0
        loop1 = 0
```

```

loop2 = 0

widgets = ['Processed: ', Counter(), ' lines (', Timer
           ( ), ')', Percentage(), Bar()]
pbar = ProgressBar(widgets=widgets)

for candidates in pbar(chashes):
    loop = loop + 1
    c_entryid = str(candidates[0])
    c_videoid = str(candidates[1])
    c_ahash = str(candidates[2])
    c_dhash = str(candidates[3])
    c_phash = str(candidates[4])
    c_whash = str(candidates[5])
    for references in rashes:
        loop1 = loop1 + 1
        r_entryid = str(references[0])
        r_videoid = str(references[1])
        r_ahash = str(references[2])
        r_dhash = str(references[3])
        r_phash = str(references[4])
        r_whash = str(references[5])

        adistance = h.distance(h.transform(c_ahash), h.
                               transform(r_ahash))
        ddistance = h.distance(h.transform(c_dhash), h.
                               transform(r_dhash))
        pdistance = h.distance(h.transform(c_phash), h.
                               transform(r_phash))
        wdistance = h.distance(h.transform(c_whash), h.
                               transform(r_whash))
        if adistance <= 13:
            f.write("%s;%s; '%s'; '%s'; %s;%s; '%s'; '%s'; %s
                    ; ahash\n" %(loop, loop1, c_videoid, r_
                                videoid, c_entryid, r_entryid, c_ahash, r_
                                ahash, adistance))
        if ddistance <= 13:
            f.write("%s;%s; '%s'; '%s'; %s;%s; '%s'; '%s'; %s
                    ; dhash\n" %(loop, loop1, c_videoid, r_
                                videoid, c_entryid, r_entryid, c_dhash, r_
                                dhash, ddistance))
        if pdistance <= 13:
            f.write("%s;%s; '%s'; '%s'; %s;%s; '%s'; '%s'; %s
                    ; phash\n" %(loop, loop1, c_videoid, r_
                                videoid, c_entryid, r_entryid, c_phash, r_
                                phash, pdistance))
        if wdistance <= 13:
            f.write("%s;%s; '%s'; '%s'; %s;%s; '%s'; '%s'; %s
                    ; whash\n" %(loop, loop1, c_videoid, r_
                                videoid, c_entryid, r_entryid, c_whash, r_
                                whash, wdistance))

    f.close()

def getFiles(self):

```





## APÊNDICE F – Código em Python que efetua busca e comparações junto ao banco de dados

### Código F.1 – Código em Python que efetua busca e comparações junto ao banco de dados

```
#!/usr/bin/python
#title :compare_database.py
#description :Compare Perceptual Hashes
#author :Anderson Torres
#date :20181124
#version :1.0
#usage :python compare_database.py
#notes :
#python_version :2.6.6
#
=====

from database.finditems import finditems
from progressbar import ProgressBar, Timer, Counter,
    Percentage, Bar
import timing
import time
import codecs
import csv

class compare:
    def __init__(self):
        self.a = finditems()

    def getValues(self, candID, candName, refeID, refeName):

        try:
            a = self.a.aHash(candID, refeID)
            d = self.a.dHash(candID, refeID)
            p = self.a.pHash(candID, refeID)
            w = self.a.wHash(candID, refeID)
            for hash in a:
                ahash = str(hash[0])
            for hash in d:
                dhash = str(hash[0])
            for hash in p:
                phash = str(hash[0])
            for hash in w:
                whash = str(hash[0])

            print("%s| %s | %s | %s | %s | %s" % (candName,
                refeName, ahash, dhash, phash, whash))

        except:
            print("[ERROR] Something wrong happened")
```



## APÊNDICE G – Código em Python que efetua rotinas junto ao banco de dados

**Código G.1 – Código em Python que efetua rotinas junto ao banco de dados**

```
#!/usr/bin/python
#title :findiitems.py
#description :Class to find values in database
#author :Anderson Torres
#date :20181124
#version :1.0
#usage :called within class
#notes :
#python_version :2.6.6
#
=====

import mysql.connector
from dbconnect import dbconnect

class finditems:

    def __init__(self):
        self.a = dbconnect()
        self.db = self.a.connect()
        self.cursor = self.db.cursor()
        print (" [INFO] ")
        print (" - database Connected")

    def findCandidateID(self, filename):
        self.cursor.execute("SELECT videohash FROM
            candidateFile WHERE videoName = %s", (filename,))
        records = self.cursor.fetchall()
        for row in records:
            candidateID = row[0]
        return candidateID

    def findCandidateHashes(self, filename):
        self.cursor.execute("SELECT candidateHash.id, videoID,
            aHash, dHash, pHash, wHash FROM candidateHash,
            candidateFile "
            "WHERE candidateHash.videoID =
            candidateFile.videohash "
            "AND candidateFile.videohash = %
            s", (filename,))
        records = self.cursor.fetchall()
        if records:
            return records

    def findReferenceHashes(self, filename):
```

```

self.cursor.execute("SELECT referenceHash.id,videoID ,
                    aHash,dHash,pHash,wHash FROM referenceHash ,
                    referenceFile "
                    "WHERE referenceHash.videoID =
                    referenceFile.videohash "
                    "AND referenceFile.videohash = %s
                    ",(filename,))
records = self.cursor.fetchall()
if records:
    return records

def findCandidateFiles(self):
    self.cursor.execute("SELECT candidateFile.videohash ,
                        candidateFile.videoName FROM candidateFile")
    records = self.cursor.fetchall()
    if records:
        return records

def findReferenceFiles(self):
    self.cursor.execute("SELECT referenceFile.videohash ,
                        referenceFile.videoName FROM referenceFile")
    records = self.cursor.fetchall()
    if records:
        return records

def aHash(self,candidate,reference):
    self.cursor.execute("SELECT COUNT(A.adistance) as
                        totalvalues "
                        "FROM (SELECT bit_count(cast(conv
                        (candidateHash.aHash,16,10) as
                        UNSIGNED) ^ cast(conv(
                        referenceHash.aHash,16,10)as
                        UNSIGNED) ) as adistance "
                        "FROM candidateFile ,
                        candidateHash, referenceFile,
                        referenceHash "
                        "WHERE candidateFile.videohash =
                        candidateHash.videoID "
                        "AND referenceFile.videohash =
                        referenceHash.videoID "
                        "AND candidateFile.videohash = %s
                        "
                        "AND referenceFile.videohash = %s
                        ) as A "
                        "WHERE A.adistance <= 13",(
                        candidate,reference,))
    records = self.cursor.fetchall()
    if records:
        return records

def dHash(self,candidate,reference):
    self.cursor.execute("SELECT COUNT(A.ddistance) as
                        totalvalues "

```

```

        "FROM (SELECT bit_count(cast(conv
            (candidateHash.dHash,16,10) as
            UNSIGNED) ^ cast(conv(
            referenceHash.dHash,16,10)as
            UNSIGNED) ) as ddistance "
        "FROM candidateFile,
            candidateHash, referenceFile,
            referenceHash "
        "WHERE candidateFile.videohash =
            candidateHash.videoID "
        "AND referenceFile.videohash =
            referenceHash.videoID "
        "AND candidateFile.videohash = %s
            "
        "AND referenceFile.videohash = %s
            ) as A "
        "WHERE A.ddistance <= 13", (
            candidate,reference,))
    records = self.cursor.fetchall()
    if records:
        return records

def pHash(self, candidate, reference):
    self.cursor.execute("SELECT COUNT(A.pdistance) as
        totalvalues "
        "FROM (SELECT bit_count(cast(conv
            (candidateHash.pHash,16,10) as
            UNSIGNED) ^ cast(conv(
            referenceHash.pHash,16,10)as
            UNSIGNED) ) as pdistance "
        "FROM candidateFile,
            candidateHash, referenceFile,
            referenceHash "
        "WHERE candidateFile.videohash =
            candidateHash.videoID "
        "AND referenceFile.videohash =
            referenceHash.videoID "
        "AND candidateFile.videohash = %s
            "
        "AND referenceFile.videohash = %s
            ) as A "
        "WHERE A.pdistance <= 13", (
            candidate,reference,))
    records = self.cursor.fetchall()
    if records:
        return records

def wHash(self, candidate, reference):
    self.cursor.execute("SELECT COUNT(A.wdistance) as
        totalvalues "
        "FROM (SELECT bit_count(cast(conv
            (candidateHash.wHash,16,10) as
            UNSIGNED) ^ cast(conv(
            referenceHash.wHash,16,10)as

```

```
        UNSIGNED) ) as wdistance "
"FROM candidateFile,
    candidateHash, referenceFile,
    referenceHash "
"WHERE candidateFile.videohash =
    candidateHash.videoID "
"AND referenceFile.videohash =
    referenceHash.videoID "
"AND candidateFile.videohash = %s
    "
"AND referenceFile.videohash = %s
    ) as A "
"WHERE A.wdistance <= 13", (
    candidate,reference,))
records = self.cursor.fetchall()
if records:
    return records

def dbclose(self):
    if self.db:
        self.db.close()
        print (" [INFO] ")
        print (" - database Closed")
```