UNIVERSIDADE FEDERAL DO PAMPA

Giulliano Lyra Paz

OpenVT: Aplicação Escalável de Código Aberto para Busca de Imagens Semelhantes Utilizando Árvore de Vocabulário

Alegrete
2018

Giulliano Lyra Paz

OpenVT: Aplicação Escalável de Código Aberto para Busca de Imagens Semelhantes Utilizando Árvore de Vocabulário

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Marcelo Resende Thielo

Alegrete
2018

Giulliano Lyra Paz

OpenVT: Aplicação Escalável de Código Aberto para Busca de Imagens Semelhantes Utilizando Árvore de Vocabulário

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em 28 de Junho de 2018.
Banca examinadora:

Prof. Dr. Marcelo Resende Thielo
Orientador
UNIPAMPA

Prof.ª Ma. Alice Fonseca Finger
UNIPAMPA

Prof. Me. Eliezer Soares Flores
UNIPAMPA

Ao meus pais que, durante toda minha vida,
apoiaram-me e não mediram esforços para que
eu chegasse a esta etapa da minha vida.

AGRADECIMENTOS

À minha família, especialmente aos meus pais, os quais foram meus maiores incentivadores e batalharam para proporcionar a melhor educação aos seus filhos. Agradeço à minha companheira por todo conforto, incentivo e compreensão, sendo uma das principais responsáveis por toda minha disciplina e dedicação a este trabalho. Aos colegas e parceiros que, por anos, estiveram juntos a mim nesta batalha para me tornar um cientista. Agradeço aos mestres que encontrei durante meu percurso acadêmico, os quais dedicam seus dias a ensinar e compartilhar todo seu conhecimento. Um agradecimento especial ao meu professor orientador pela instrução, pelas conversas e por ter confiado um tema tão notável a mim.

“Ninguém que é curioso é idiota. As pessoas que não fazem perguntas permanecem ignorantes para o resto de suas vidas.”
Neil DeGrasse Tyson

RESUMO

Recuperação de imagens baseada em conteúdo (CBIR) é a aplicação de técnicas da visão computacional ao problema de recuperação de imagens. “Baseada em conteúdo” significa que informações contidas nas imagens – características – são utilizadas para realizar as buscas. Esse tipo de sistema de busca pode ser utilizado para inúmeros fins, como verificar se imagens estão sendo utilizadas sem autorização, localizar pessoas e objetos em filmes e vídeos e buscar por pontos de referência, como imóveis e pontos turísticos. As aplicações CBIR comumente encontradas utilizam algoritmos patenteados ou possuem seus códigos fechados. Pensando nisto, os objetivos deste trabalho são: a construção de um motor de busca por imagem, o qual seja escalável para uma grande quantidade de imagens e que seja robusto às diferentes condições que uma imagem pode se encontrar, como, por exemplo, escala, rotação e iluminação; além de verificar a possibilidade de se utilizar apenas métodos livres, resultando em uma aplicação de código aberto com desempenho satisfatório, buscando fomentar e auxiliar futuros desenvolvimentos. Tendo eficiência e escalabilidade como principais características, o método proposto por Nister e Stewenius (2006) – árvore de vocabulário – foi definido como base para este trabalho. Para alcançar os objetivos propostos, foi feito um levantamento de trabalhos relacionados a fim de descobrir os principais ganhos e desafios de se utilizar árvores de vocabulário como estrutura base. Após investigar e especificar as funcionalidades, foi realizado o desenvolvimento da aplicação proposta, visando eficiência, agilidade e flexibilidade. As três principais tarefas nas quais este trabalho concentrou-se foram: extração de características de imagens, construção da árvore de vocabulário e sistema de pontuação. Para avaliar a eficiência da aplicação resultante deste trabalho, esta foi comparada com algoritmos patenteados e o estado da arte em reconhecimento de imagens. Por fim, foi possível concluir que aplicações de qualidade e eficientes podem ser criadas utilizando apenas métodos livres e que árvores de vocabulário são estruturas adaptáveis e flexíveis, com aplicações diversas e ótimo desempenho em buscas.

Palavras-chave: Árvore de Vocabulário. Escalabilidade. Código Aberto. CBIR. OpenVT.

ABSTRACT

Content-based image retrieval (CBIR) is the application of computer vision techniques to the image recovery problem. "Content-based" means that information contained in the images - features - is used to perform the searches. This type of search system can be used for numerous purposes, such as verifying that images are being used without authorization, locating people and objects in movies and videos, and searching for landmarks such as real estate and sights. Commonly encountered CBIR applications use patented algorithms or have their codes closed. With this in mind, the objectives of this work are: the construction of an image search engine, which is scalable to a lot of images and is also robust to the different conditions that an image can find, such as scale, rotation and lighting; besides verify the possibility of using only open source methods, resulting in an open source application with satisfactory performance, seeking to foster and help future developments. Thinking about efficiency and scalability as main features, the method proposed by Nister e Stewenius (2006) - vocabulary tree - was defined as the basis for this work. In order to reach the proposed objectives, a survey was made of related works in order to discover the main gains and challenges of using vocabulary trees as a base structure. After investigating and specifying the functionalities, the proposed application was developed, aiming for efficiency, agility and flexibility. The three main tasks that this work focused on were: image feature extraction, building of the vocabulary tree and score system. To evaluate the efficiency of the application resulting from this work, this was compared with patented algorithms and the state-of-the-art in image recognition. Finally, it was possible to conclude that quality and efficient applications can be created using only open source methods and that vocabulary trees are adaptable and flexible structures with diverse applications and excellent search performance.

Key-words: Vocabulary Tree. Scalability. Open Source. CBIR. OpenVT.

LISTA DE FIGURAS

Figura 1 – Tipos de consultas. 26
Figura 2 – Motor de busca por imagem. 28
Figura 3 – Ilustração da extração dos descritores. 33
Figura 4 – Pontos de interesse encontrados pelo ORB 35
Figura 5 – Atualização do valor dos centroides do K-Means. 38
Figura 6 – Ilustração do funcionamento dos arquivos invertidos. 40
Figura 7 – Construção da árvore de vocabulário 41
Figura 8 – Ilustração visual dos descritores nos nodos da árvore de vocabulário . . 42
Figura 9 – Ilustração da relevância dos nodos para o sistema de pontuação. 43
Figura 10 – Ilustração da árvore de vocabulário com arquivos invertidos. 44
Figura 11 – Histograma da imagem de consulta. 45
Figura 12 – Visão geral da aplicação. 53
Figura 13 – Armazenamento das imagens e seus descritores. 55
Figura 14 – Vetor de descritores. 57
Figura 15 – Relação entre nodos e arquivos invertidos. 59
Figura 16 – Consulta usando árvore de vocabulário. 60
Figura 17 – OpenVT para 400 imagens com 50 mil folhas. 61
Figura 18 – OpenVT para 273 imagens com 10 mil folhas. 62
Figura 19 – Estratégia para avaliar a precisão da aplicação. 63
Figura 20 – Comparação entre os métodos para cálculo de similaridade entre histogramas. 65
Figura 21 – Uso da memória RAM para 1.400 imagens com 160 mil folhas. 71
Figura 22 – Busca por imagens semelhantes a um violão com a OpenVT. 81
Figura 23 – Busca por imagens semelhantes à Torre de Pisa com a OpenVT. 83
Figura 24 – Busca por imagens semelhantes a um girassol com a OpenVT. 85

LISTA DE TABELAS

Tabela 1 – Trabalhos Relacionados 49
Tabela 2 – OpenVT com algoritmos livres vs OpenVT com algoritmos patenteados 66
Tabela 3 – OpenVT vs modelos pré-treinados do Keras 69
Tabela 4 – OpenVT vs método original 70

LISTA DE SIGLAS

BOW Conjunto de palavras (BOW do inglês *Bag-Of-Words*)

CBIR Recuperação de imagens baseada em conteúdo (CBIR do inglês *Content-Based Image Retrieval*)

CNN Redes Neurais Convolucionais (CNN do inglês *Convolutional Neural Networks*)

IDF Frequência inversa de um termo em documentos (IDF do inglês *Inverse Document Frequency*)

OpenCV Biblioteca de visão computacional de código aberto (OpenCV do inglês *Open Computer Vision*)

OpenVT Aplicação de código aberto utilizando árvore de vocabulário (OpenVT do inglês *Open Vocabulary Tree*)

SGBD Sistema de gerenciamento de banco de dados

TF Frequência de um termo em um documento (TF do inglês *Term Frequency*)

TF-IDF Frequência de um termo – frequência inversa de um termo em documentos (TF-IDF do inglês *Term Frequency–Inverse Document Frequency*)

VocabTree Árvore de Vocabulário (VocabTree do inglês *Vocabulary Tree*)

LISTA DE SÍMBOLOS

D	Quantidade total de documentos presentes em uma base de dados
d_{ij}	TF-IDF da imagem j sobre nodo i, onde $j \in J$ e $i \in I$
D_p	Quantidade de documentos que a palavra p aparece
d_{tz}	TF-IDF da imagem z sobre o nodo t, onde $z \in Z$ e $t \in T$
d_z	Histograma do caminho percorrido pelos descritores da imagem z pela árvore de vocabulário
DF	Valor estatístico que reflete a proporção entre a quantidade de descritores e a quantidade de folhas
I	Conjunto de nodos da árvore de vocabulário
i	Nodo da árvore de vocabulário, onde $i \in I$
J	Conjunto de imagens da base de dados
j	Imagem da base de dados, onde $j \in J$
k	Quantidade de grupos em que um conjuntos de dados é *clusterizado*
L	Nível limite para a construção da árvore de vocabulário
m_{ij}	Quantidade de descritores da imagem j que passaram pelo nodo i
N	Quantidade total de imagens presentes na base de dados
N_i	Quantidade de imagens da base de dados com caminho através do nodo i
n_t	Quantidade de descritores da imagem de consulta que passaram pelo nodo t, onde $t \in T$
nd	Quantidade total de descritores extraídos de um conjunto de imagens
p	Palavra presente em um documento
q	Histograma do caminho percorrido pelos descritores da imagem de consulta pela árvore de vocabulário
q_t	TF-IDF da imagem de consulta em um nodo t, onde $t \in T$
s_z	Pontuação total da imagem z em relação à imagem de consulta
T	Subconjunto de nodos da árvore de vocabulário, os quais tiveram pelo menos um descritor da imagem de consulta passando por eles

t	Nodo da árvore de vocabulário, onde $t \in T$
w_i	Peso do nodo i, onde $i \in I$
w_t	Peso do nodo t, onde $t \in T$
Z	Subconjunto de imagens que tiveram seus descritores passando por pelo menos um nodo t
z	Imagem da base de dados, onde $z \in Z$

SUMÁRIO

1 INTRODUÇÃO 25
1.1 Motivação 27
1.2 Objetivos 27
1.3 Organização do documento 28

2 FUNDAMENTAÇÃO TEÓRICA 31
2.1 Visão Computacional 31
2.2 Reconhecimento de Padrões em Imagens 31
2.2.1 Detecção de Regiões 32
2.2.2 Extração de Descritores 33
2.3 OpenCV 34
2.4 Python 36
2.5 Clustering 38
2.6 Recuperação de Informação Baseada em Texto 39
2.7 Árvore de Vocabulário 40
2.7.1 Construção da Árvore de Vocabulário 41
2.7.2 Sistema de Pontuação e Consulta com a Árvore de Vocabulário 42

3 TRABALHOS RELACIONADOS 47
3.1 Protocolo de Pesquisa 47
3.2 Resultados da Pesquisa 48
3.2.1 Algoritmos utilizados para a extração de descritores 50
3.2.2 Método utilizado para a classificação dos descritores 50
3.2.3 Estratégia adotada para o cálculo da pontuação 50
3.3 Considerações do Capítulo 51

4 DESENVOLVIMENTO DO TRABALHO 53
4.1 Extração dos descritores das imagens 54
4.2 Algoritmos de Clustering 56
4.3 Construção da árvore de vocabulário 58
4.4 Consulta com árvore de vocabulário 59
4.5 Aplicação web para OpenVT 60

5 RESULTADOS 63
5.1 Métodos de comparação entre histogramas 64
5.2 OpenVT com algoritmos livres vs OpenVT com algoritmos patenteados 65
5.3 OpenVT vs Redes Neurais Convolucionais 67
5.4 OpenVT vs método original 69

5.5 Análise da utilização de memória 70

6 CONSIDERAÇÕES FINAIS 73

REFERÊNCIAS 75

APÊNDICES 79

APÊNDICE A – BUSCA POR IMAGENS SEMELHANTES A UM VIOLÃO COM A OPENVT 81

APÊNDICE B – BUSCA POR IMAGENS SEMELHANTES À TORRE DE PISA COM A OPENVT . . 83

APÊNDICE C – BUSCA POR IMAGENS SEMELHANTES A UM GIRASSOL COM A OPENVT . . . 85

1 INTRODUÇÃO

Com a popularização de dispositivos digitais equipados com câmeras e a massificação de acesso à Internet, houve um aumento considerável de conteúdo baseado em imagens disponíveis na Web. Estima-se que, em 2021, 82% do tráfego da Internet serão vídeos, um aumento de 73% comparado a 2016 (CISCO, 2017). O ubíquo acesso à Internet e aos seus conteúdos visuais lançou uma luz sobre aplicações emergentes baseadas em busca e recuperação de imagens.

Normalmente, em um sistema de busca, as informações são indexadas por palavras-chave e são comparadas com nomes de arquivos, conteúdo ou contexto. Entretanto, esse tipo de sistema tende a perder qualidade em buscas por palavras-chave com significados amplos ou ambíguos, ou quando uma dada consulta é muito comum, resultando em conteúdos irrelevantes. Ainda, pode haver dificuldade em pesquisar por algo que não pode ser nomeado, ou o termo correto para descrevê-lo não é conhecido. Descrição textual pode ser ineficiente para buscas por conteúdos visuais.

Segundo Zhou, Li e Tian (2017), a busca eficiente por imagens tem sido explorada desde os anos 90, a qual ainda chama a atenção da comunidade de visão computacional (seção 2.1) – subárea da ciência da computação responsável por emular a visão humana – graças ao advento da computação escalável e de novas técnicas emergentes.

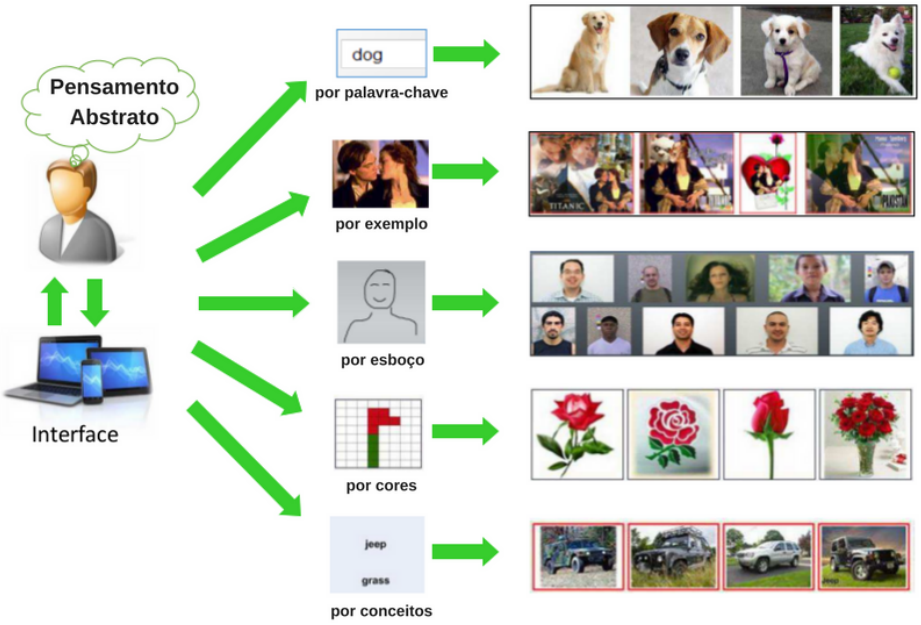
Recuperação de imagens baseada em conteúdo (CBIR) é a aplicação de técnicas da visão computacional a fim de extrair informações e características únicas de imagens. Estas características são utilizadas como forma de comparação entre imagens a fim de mensurar suas similaridades. Há três principais questões em CBIR, as quais são: como representar uma imagem, como organizar essas representações e como utilizar essas representações para mensurar similaridades. Outra questão relevante é ter de lidar com imagens com diferentes pontos de vista, ruídos e diferentes tamanhos (ZHOU; LI; TIAN, 2017).

Há um ditado popular que diz que “uma imagem vale mais do que mil palavras”, entretanto não é trivial identificar essas “palavras” em uma imagem. Outro desafio é organizar uma base de dados com uma enorme quantidade de dados para uma identificação rápida e relevante para uma dada consulta.

Conforme Zhou, Li e Tian (2017), a qualidade de uma chave de consulta tem um impacto significante no resultado final. Uma boa e específica consulta pode reduzir enormemente a dificuldade e aumentar a relevância em uma busca. Há vários tipos de consultas (Figura 1), como consulta por imagem de exemplo, por esboço, por mapa de cores, etc. Cada tipo oferece diferentes resultados.

O tipo de consulta mais intuitivo é a consulta por imagem de exemplo, na qual o usuário utiliza uma imagem de exemplo para encontrar imagens similares ou de melhor qualidade. Esse tipo de busca pode ser utilizado para verificar a existência de logomarcas parecidas, verificar se imagens estão sendo utilizadas em páginas web sem permissão, ou

Figura 1 – Tipos de consultas.



Fonte: Adaptado de Zhou, Li e Tian (2017).

até buscar por logos de organizações terroristas em imagens e vídeos (ZHOU; LI; TIAN, 2017). As aplicações são diversas.

Pensando nas possibilidades desse tipo de aplicação, tem-se como objetivo a criação de um motor de busca por imagem de código aberto[1], buscando contribuir com futuros desenvolvimentos e novas aplicações para esse tipo de busca. Este trabalho é largamente inspirado por Nister e Stewenius (2006), que apresentam uma estrutura hierárquica e escalável para quantização e busca eficiente, denominada árvore de vocabulário (VocabTree) (seção 2.7), além de um sistema de pontuação para eleger as imagens mais semelhantes à dada consulta.

A implementação que este trabalho objetiva desenvolver possui duas fases principais. A primeira é a fase de treinamento, a qual serve para ler as imagens de treinamento, extrair suas características, armazená-las na base de dados e construir a estrutura de indexação (árvore de vocabulário) e a fase de busca, a qual recebe a consulta por imagem de exemplo e propaga seus descritores pela estrutura, calculando pontuações e ranqueando as imagens por semelhança.

Este capítulo apresenta na seção 1.1 a motivação do desenvolvimento do trabalho. Na seção 1.2 estão os objetivos gerais e específicos deste trabalho. Por fim, na seção 1.3, é apresentada uma visão geral sobre os demais capítulos.

[1] Disponível em <https://opensource.org/>

1.1 Motivação

Recuperação de informação é de suma importância na computação. Máquinas computacionais estão constantemente armazenando e recuperando informações de alguma base de dados. A Google[2], empresa que virou referência com seu motor de busca, provê um serviço que permite aos usuários buscar por páginas e informações a partir de imagens (CBIR), chamado Google Imagens[3], além de retornar imagens semelhantes à imagem de entrada.

Esse tipo de aplicação é denominada motor de busca por imagem. Algumas aplicações semelhantes encontradas na web são o *TinEye*[4], o qual busca páginas web que contenham imagens semelhantes à imagem de consulta e o *ImageBrief*[5], o qual retorna imagens semelhantes presentes em sua base de dados. Além dessas, há uma lista[6] de aplicações comerciais e livres semelhantes disponíveis.

Além de buscar imagens semelhantes, há diversas aplicações para esse tipo de serviço. Localizar casas, edifícios e pontos turísticos por fotos, localizar pessoas e objetos em vídeos e filmes e procurar por produtos em lojas virtuais são alguns exemplos.

Devido às possibilidades, a criação de um motor de busca por imagem de código aberto contribuiria para o desenvolvimento de novas soluções. Por esse tipo de solução ser completamente dependente das imagens presentes na sua base de dados, uma grande quantidade de imagens deve ser armazenada. Ainda, dependendo da aplicação, novas imagens deverão ser adicionadas constantemente à base de dados. Escalabilidade e eficiência na busca são componentes de suma importância.

A utilização de árvore de vocabulário neste trabalho foi definida pelo prestígio dessa estrutura no campo de visão computacional. O trabalho de Nister e Stewenius (2006) tornou-se referência pela eficiência e escalabilidade que sua estrutura proposta apresenta, sendo citado por mais de 2 mil trabalhos, segundo a base de dados *Scopus*[7].

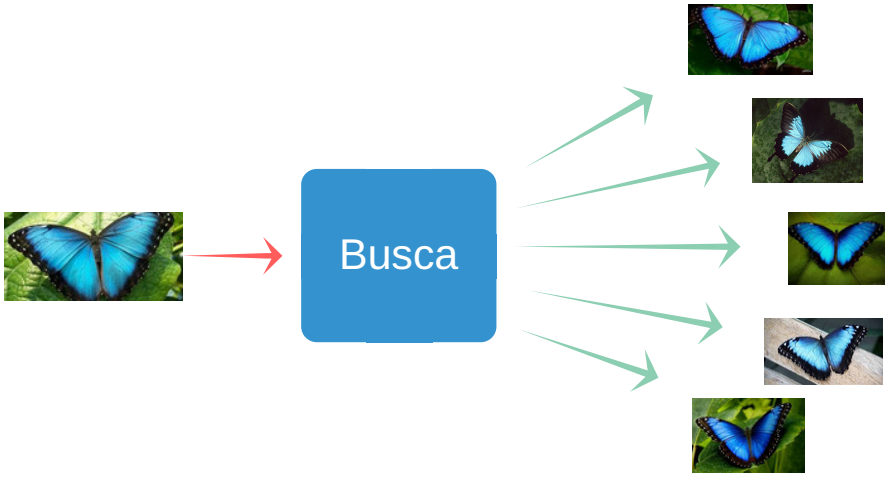
1.2 Objetivos

Tem-se como objetivo geral a construção de um motor de busca por imagem, o qual seja escalável para uma grande quantidade de imagens e que seja robusto às diferentes condições que uma imagem pode se encontrar, como diferentes pontos de vista, orientação, ruídos, tamanho e condições de iluminação, utilizando apenas métodos livres, resultando em uma aplicação de código aberto. Esse motor de busca por imagem deve receber consultas por imagens de exemplo e retornar as imagens semelhantes, ou que contenham

[2] Disponível em <http://www.google.com>
[3] Disponível em <https://www.google.com.br/imghp?hl=en&tab=wi>
[4] Disponível em <https://www.tineye.com/>
[5] Disponível em <http://www.imagebrief.com/home#/>
[6] Disponível em <https://wikivisually.com/wiki/List_of_CBIR_engines>
[7] Disponível em <https://www.scopus.com/>

objetos em comum, presentes na base de dados, como mostra a Figura 2.

Figura 2 – Motor de busca por imagem.



Fonte: Elaborado pelo autor.

Como objetivos específicos, podem ser citados os seguintes:

1. utilizar biblioteca de visão computacional OpenCV, junto à linguagem de programação Python para o desenvolvimento da aplicação;
2. desenvolver aplicação CBIR de código aberto que receba consultas por imagens de exemplo e retorne ao usuário imagens semelhantes;
3. desenvolver aplicação que mantenha um desempenho satisfatório mesmo para uma grande quantidade de imagens;
4. desenvolver aplicação inspirada no trabalho de Nister e Stewenius (2006), com qualidade e desempenho semelhantes, utilizando somente métodos e algoritmos de código aberto.

1.3 Organização do documento

- Capítulo 2 **Fundamentação Teórica**: conceitos ligados ao trabalho apresentado são abordados do ponto de vista de alguns autores e pesquisadores;
- Capítulo 3 **Trabalhos Relacionados**: é relatada a aplicação do mapeamento sistemático para investigação do estado da arte em aplicações escaláveis de busca de imagens semelhantes utilizando árvores de vocabulário como estrutura de indexação;

- Capítulo 4 **Desenvolvimento do Trabalho**: são apresentados os métodos, algoritmos e implementações que compõem a aplicação resultante deste trabalho;
- Capítulo 5 **Resultados**: são apresentados os resultados obtidos a partir da comparação da aplicação resultante deste trabalho com métodos patenteados, redes neurais artificiais e o método original apresentado por Nister e Stewenius (2006);
- Capítulo 6 **Considerações Finais**: contém as considerações finais sobre o trabalho realizado, pesquisas e resultados.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os fundamentos teóricos necessários para o entendimento deste trabalho. O capítulo está organizado da seguinte forma: na seção 2.1 é introduzido o conceito de visão computacional; na seção 2.2 é introduzido o conceito de reconhecimento de padrões em imagens; na seção 2.3 é apresentada a biblioteca de visão computacional OpenCV; na seção 2.4 é apresentada a linguagem de programação Python; na seção 2.5 é apresentado o conceito de *clustering*; na seção 2.6 é introduzido o conceito de recuperação de informação baseada em texto; por fim, na seção 2.7 é apresentada a estrutura proposta por Nister e Stewenius (2006), chamada árvore de vocabulário.

2.1 Visão Computacional

Visão computacional é uma área interdisciplinar responsável por emular a visão humana, a qual abrange aspectos de inteligência artificial, aprendizado de máquina, robótica, biologia e principalmente ciência da computação. A visão computacional processa e busca padrões e significados em imagens obtidas por uma câmera digital, assim como o cérebro com imagens obtidas através dos olhos. Além de reconhecer objetos e pessoas em imagens, o que auxilia enormemente em biometria e computação forense, a visão computacional tem inúmeras outras aplicações, como verificar qualidade de alimentos, reconhecer objetos astronômicos a partir de imagens capturadas por telescópios, "dar visão" a carros autônomos, robôs e drones, traduzir mensagens em tempo real a partir de dispositivos de captura de imagens e realizar buscas por imagens de exemplo (NIXON; AGUADO, 2012).

Não é clara a fronteira entre o processamento de imagens e visão computacional. Pode-se dizer que processamento de imagens é um processo onde a entrada do sistema é uma imagem e a saída é um conjunto de valores numéricos, que podem ou não compor uma outra imagem. A visão computacional procura emular a visão humana, portanto também possui como entrada uma imagem e a saída é uma interpretação da imagem como um todo, ou parcialmente. Em suma, processamento de imagens tem como objetivo modificar ou melhorar as condições em que imagens se encontram, enquanto que visão computacional busca dar sentido, compreender, o que dada imagem representa. A visão computacional frequentemente utiliza o processamento de imagens como ferramenta para tratar e melhorar imagens, facilitando a sua compreensão (MARENGONI; STRINGHINI, 2009).

2.2 Reconhecimento de Padrões em Imagens

Para os seres vivos, principalmente seres humanos, reconhecer padrões é algo inato. Crianças em suas primeiras horas de vida já assimilam padrões, como o rosto e voz da mãe. Reconhecimento de padrões é uma habilidade fundamental nos seres humanos, estando diretamente conectada com a memória, sentidos e pensamentos. Reconhecimento

de padrões é um processo que depende diretamente de conhecimento e experiências adquiridas. Geralmente, reconhecimento de padrões se refere a um processo que recebe um sinal como estímulo (informação) e compara com padrões existentes na memória de longo prazo, categorizando o estímulo dentre as classes conhecidas pelo indivíduo. Em outras palavras, o reconhecimento de padrões depende diretamente do conhecimento e experiência já adquiridos (PI et al., 2008). Assim como humanos, o conhecimento necessário para os computadores reconhecerem padrões está diretamente ligado aos modelos, classes e dados conhecidos e armazenados por este.

Como introduz Chen (2015) em seu livro, há duas principais abordagens no campo de reconhecimento de padrões. A primeira abordagem, comumente utilizada em compiladores, representa padrões utilizando estruturas de características. A principal ideia dessa abordagem é representar padrões em forma de *strings*, árvores, grafos e um conjunto de estruturas como uma linguagem formal. A segunda representa padrões utilizando descritores, os quais são valores discretos que representam características extraídas de um conjunto de dados.

Tratando-se de imagens, há dois principais tipos de descritores: os globais e os locais. Os descritores globais descrevem uma imagem por inteiro, considerando cores, texturas e formas. Esse tipo de descritor torna-se atrativo para aplicações moderadas, pois representa imagens de forma muito compacta, geralmente com apenas um descritor. Descritores locais descrevem características únicas de imagens, como curvas, bordas e cores. Estes geralmente são extraídos em duas fases. Inicialmente é realizada a detecção de regiões e pontos de interesse, então são realizados os cálculos e extração dos descritores (LISIN et al., 2005).

O estado da arte em extração de descritores são os descritores gerados pelo Keras[1]. Keras é uma API de redes neurais artificiais escrita em Python com foco em rápido desenvolvimento e experimentação. O Keras possui modelos de aprendizagem profunda[2] (*deep learning models*) pré-treinados, os quais podem ser utilizados para predição, classificação e extração de descritores (FRANKY, 2018).

2.2.1 Detecção de Regiões

Uma região é um grupo de pixels conectados em uma imagem que compartilham propriedades em comum, como escala de cinza, cor, tamanho e forma (MALLICK, 2015).

Detecção de regiões é um dos problemas mais estudados da visão computacional, sendo usado em muitas aplicações, como detecção de objetos, *matching* e segmentação. Extração de regiões tem como principal propósito a redução da quantidade de dados a serem tratados, mantendo apenas as informações mais importantes sobre a imagem (RIEMENSCHNEIDER; DONOSER; BISCHOF, 2009).

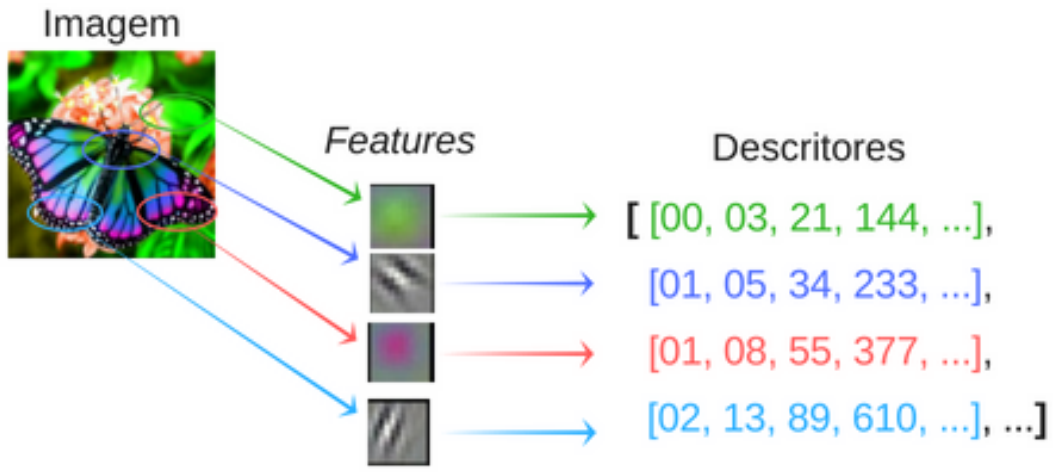
[1] Disponível em <https://keras.io/>
[2] Disponível em <https://keras.io/applications/>

Um exemplo de algoritmo de detecção de regiões e pontos de interesse é o FAST (ROSTEN et al., 2010), o qual utiliza técnicas de aprendizagem de máquina para detectar cantos, bordas e regiões. Em seus trabalhos, Sivic e Zisserman (2003) e Nister e Stewenius (2006) utilizam o MSER (*Maximally Stable Extremal Regions*) (MATAS et al., 2004), um detector de regiões que busca estabelecer correspondência entre duas, ou mais imagens, independente do ponto de vista (*viewpoint*), iluminação e rotação. Este método introduz um novo conjunto de regiões, chamado *extremal regions*.

2.2.2 Extração de Descritores

Matching entre imagens é um aspecto fundamental de muitos problemas na visão computacional, incluindo reconhecimentos de objetos ou cenas e rastreamento de movimentos (LOWE, 2004). Descritores são valores discretos que expressam características visuais de uma imagem, como formas, bordas, cores, texturas e cantos. A Figura 3 ilustra o processo de extração de descritores, onde estes são gerados a partir da aplicação de métodos matemáticos sobre características (*features*) únicas encontradas em uma imagem. Grandes quantidades de descritores podem ser extraídos facilmente de imagens, entretanto o valor desses descritores pode sofrer alterações devido às diversas condições que imagens podem ser encontradas.

Figura 3 – Ilustração da extração dos descritores.



Fonte: Elaborado pelo autor.

A busca por correspondências discretas entre imagens pode ser dividida em 3 passos principais. Primeiro, pontos de interesse – utilizando detector de regiões – são selecionados em localizações distintas na imagem, como cantos, regiões e T-junções. A propriedade mais valiosa para um detector de pontos de interesse é a repetibilidade. Se o detector encontrar, de forma confiável, um mesmo ponto, em diferentes condições de

visualização, este ponto é um sério candidato a ser um ponto de interesse. Em seguida, a proximidade de cada ponto de interesse é representada por um descritor (vetor de características). Este descritor deve ser distintivo e, ao mesmo tempo, robusto a ruídos, erros de detecção e deformações geométricas e fotométricas. Finalmente, os descritores são combinados entre imagens diferentes. A correspondência é, muitas vezes, baseada em uma distância entre os vetores, como as distâncias *Mahalanobis*[3], *Hamming*[4], *Bhattacharyya*[5] ou *Euclidiana*[6]. A dimensão do descritor tem um impacto direto no tempo que isso leva e, portanto, é desejável um número menor de dimensões (BAY; TUYTELAARS; GOOL, 2006).

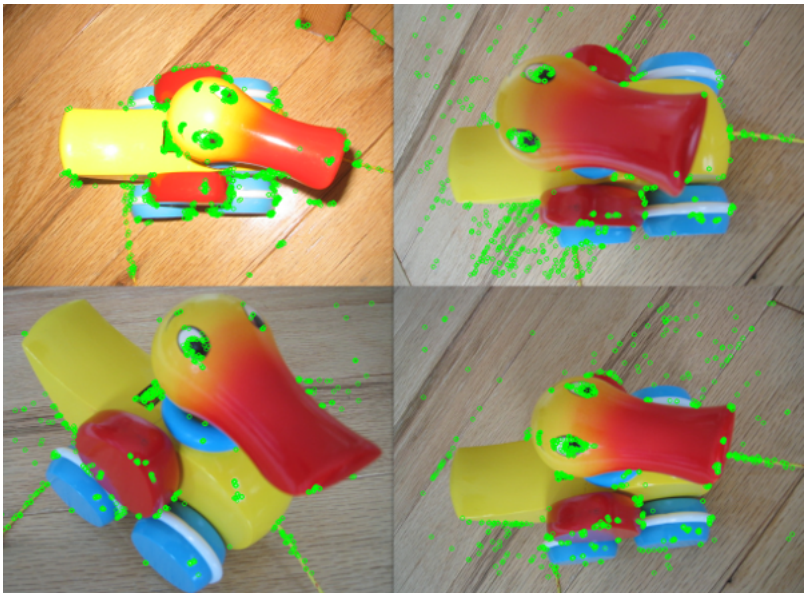
Em seu trabalho, Lowe (2004) apresenta seu método que extrai descritores invariantes à rotação, ponto de vista, iluminação e escala, diminuindo a possibilidade de erros em *matchings* entre imagens devido a ruído, má qualidade das imagens e oclusão. Em razão dessas características, inúmeros trabalhos optam por utilizar o SIFT[7] (*Scale-Invariant Keypoints*) como detector e extrator de características, como Nister e Stewenius (2006) e Sivic e Zisserman (2003). Compartilhando das mesmas características e robustez do SIFT, o SURF[8] (*Speeded Up Robust Features*) (BAY; TUYTELAARS; GOOL, 2006) surgiu propondo um maior desempenho e velocidade na extração dos descritores. Além de SIFT e SURF, algumas alternativas surgiram, como FAST (ROSTEN et al., 2010), BRIEF (CALONDER et al., 2010), BRISK (LEUTENEGGER; CHLI; SIEGWART, 2011), KAZE (ALCANTARILLA; BARTOLI; DAVISON, 2012) e ORB (RUBLEE et al., 2011) (fusão do FAST com o BRIEF), sendo BRIEF, SIFT e SURF algoritmos patenteados. A Figura 4 é a captura de tela da execução do ORB para quatro imagens semelhantes, sendo possível visualizar o desempenho do ORB em encontrar pontos de interesses robustos à rotação, iluminação e pontos de vistas distintos.

2.3 OpenCV

OpenCV é uma biblioteca de visão computacional[9] de código aberto[10]. A biblioteca é escrita em C e C++ e é executada em Linux, Windows, Mac OS, Android, além de outros sistemas operacionais. Há desenvolvimento ativo em interfaces para Python, Ruby, Matlab e outras linguagens. O OpenCV dispõe de mais de 500 métodos que abrangem muitas áreas da visão computacional. Pelo fato da visão computacional e aprendizagem de máquina serem bem próximas, o OpenCV ainda provê uma biblioteca de aprendizado de máquina de propósito geral (BRADSKI; KAEHLER, 2008).

[3] Disponível em <http://www.statisticshowto.com/mahalanobis-distance/>
[4] Disponível em <http://www.maths.manchester.ac.uk/~pas/code/notes/part2.pdf>
[5] Disponível em <http://www.cse.yorku.ca/~kosta/CompVis_Notes/bhattacharyya.pdf>
[6] Disponível em <https://www.cut-the-knot.org/pythagoras/DistanceFormula.shtml>
[7] Disponível em <https://docs.opencv.org/trunk/d5/d3c/classcv_1_1xfeatures2d_1_1SIFT.html>
[8] Disponível em <https://docs.opencv.org/trunk/d5/df7/classcv_1_1xfeatures2d_1_1SURF.html>
[9] Disponível em <https://sourceforge.net/projects/opencvlibrary/>
[10] Disponível em <https://opensource.org/>

Figura 4 – Pontos de interesse encontrados pelo ORB



Fonte: Elaborado pelo autor.

O OpenCV foi idealizado com o objetivo de tornar a visão computacional acessível a usuários e programadores. Desenvolvido para prover eficiência computacional, a biblioteca possui suporte para programação em multi-core e otimização em interfaces Intel (MARENGONI; STRINGHINI, 2009).

A licença de código aberto para o OpenCV[11] foi estruturada de modo que se pode desenvolver um produto comercial usando todo ou parte das funcionalidades disponíveis na biblioteca. Não há obrigação de que o produto contenha código aberto ou que seja desenvolvido para domínio público, embora estas características sejam desejáveis. Em partes, por causa desses termos liberais de licenciamento, há uma grande comunidade de usuários que inclui pessoas de grandes empresas como IBM, Microsoft, Intel, SONY, Siemens e Google, e centros de pesquisa como Stanford, MIT , CMU, Cambridge e INRIA. Existe um fórum do Yahoo Groups[12] onde os usuários podem postar perguntas e discussão com quase 50 mil membros.

OpenCV é popular em todo o mundo com grandes comunidades de usuários na China, Japão, Rússia, Europa e Israel. Desde a sua versão *alpha* em Janeiro de 1999, o OpenCV tem sido utilizado em muitas aplicações, produtos e esforços de investigação. Entre essas aplicações estão incluídas: costura de imagens em mapas, alinhamento de digitalização de imagem, redução de ruído de imagens médicas, análise de objetos, sistemas de segurança e detecção de intrusão, sistemas automáticos de monitoramento e segurança, sistemas de inspeção de fabricação, calibração de câmera, aplicações militares e aéreos não

[11] Disponível em <https://opencv.org/license.html>
[12] Disponível em <https://groups.yahoo.com/neo/groups/OpenCV/info>

tripulados, veículos subaquáticos e terrestres. Também tem sido usado no reconhecimento de som e música, onde técnicas de reconhecimento de visão são aplicadas às imagens de espectrograma de som (BRADSKI; KAEHLER, 2016).

O OpenCV é estruturado em módulos, o que significa que o pacote inclui várias bibliotecas compartilhadas ou estáticas. Alguns dos módulos disponíveis são[13]:

- **core**: um módulo compacto que define basicamente estruturas de dados, incluindo o denso vetor Mat multi-dimensional e funções básicas usadas em outros módulos;
- **imgproc**: um módulo de processamento de imagens que inclui filtragem não linear de imagem, transformações geométricas de imagens (redimensionamento, distorção afim e perspectiva, remapeamento genérico baseado em tabelas), conversão de cores, histogramas e mais;
- **video**: um módulo de análise de vídeo que inclui estimativa de movimento, subtração de fundo e algoritmos de rastreamento de objetos;
- **calib3d**: algoritmos básicos de geometria de visão múltipla, calibração de câmera única e estéreo, estimativa de pose de objeto, algoritmos de correspondência estéreo e elementos de reconstrução 3D;
- **features2d**: algoritmos detectores de características e *blobs*, extratores de descritores e correspondentes de descritores, como FAST, MSER, ORB e BRISK;
- **xfeatures2d**: módulo extra que contém algoritmos *non-free* e experimentais de extração e detecção de características e descritores de imagens, como SIFT, SURF e BRIEF;
- **objdetect**: detecção de objetos e instâncias de uma classe predefinida. Por exemplo: olhos, rostos, pessoas, carros e mais;
- **highgui**: uma interface fácil de usar para recursos de UI (*user interfaces*) simples;
- **imgcodecs**: auxilia a ler e escrever imagens no disco ou na memória.

2.4 Python

Python[14] é uma linguagem de programação de alto nível, multi-paradigma e de propósito geral. Criada por Guido Van Rossum em 1991, é uma linguagem de script que tem como filosofia simplicidade, legibilidade e uma sintaxe que permita que os programadores possam expressar conceitos em poucas linhas de código. Comparada com outras linguagens de programação, como C e C++, Python é considerada lenta, pois

[13] Disponível em <https://docs.opencv.org/master/>
[14] Disponível em <https://www.python.org/>

esta é executada por um interpretador, ao invés de ser compilada para código nativo de máquina. Entretanto, Python tem uma importante característica que permite utilizar códigos C e C++ como módulos do Python (*wrapping*), quando necessário. Isso provê vantagens como: o código continua rápido e eficiente como um código originalmente escrito em C/C++ e fácil para codificar em Python (GORELICK; OZSVALD, 2014). O OpenCV-Python[15] funciona desta forma, utilizando implementações do OpenCV originalmente feitas em C++. Quando a biblioteca OpenCV é utilizada no Python, basicamente são códigos C++ que estão sendo executados (MORDVINTSEV; K., 2013).

Outra alternativa à lentidão do Python é utilizar o Cython. Segundo Smith (2015), Cython pode ser descrito como: uma linguagem de programação que combina a linguagem de programação Python com o sistema de tipagem estática do C e C++; e um compilador que traduz códigos Cython para códigos C/C++ eficientes, que então são compilados e utilizados como módulos externos do Python.

O poder do Cython se deve à forma a qual este combina o Python e C/C++. O Cython faz o desenvolvedor sentir-se programando em Python, com todo o conforto que o Python oferece, porém com um desempenho próximo a códigos C/C++. Cython pode ser entendido como uma linguagem intermediária entre C/C++ e Python (SMITH, 2015).

Entretanto, Cython ainda possui suas limitações. Há dependências com inúmeros módulos Python, o que faz com que Cython seja utilizado mais comumente em tarefas específicas que necessitam de um melhor desempenho, e não em todo o projeto.

Além de Python ser uma linguagem fácil de se utilizar, ela provê uma grande quantidade de módulos que, somados aos já existentes no OpenCV, proveem um ambiente poderoso aos desenvolvedores, como:

- **Numpy**[16]: módulo altamente otimizado para computação científica que suporta operações com matrizes multidimensionais, álgebra linear e números randômicos, integrado com códigos C++ e Fortran (todas as estruturas do tipo *array* do OpenCV foram convertidas para o Numpy);

- **Sklearn**[17]: ferramenta simples para aprendizado de máquina e mineração e análise de dados. Provê classes para classificação, regressão, *clustering*, seleção de modelos, normalização e extração de características;

- **Matplotlib**[18]: produz figuras de qualidade para visualização. Pode-se criar gráficos de erro e dispersão e histogramas, além de ler, mostrar e escrever imagens.

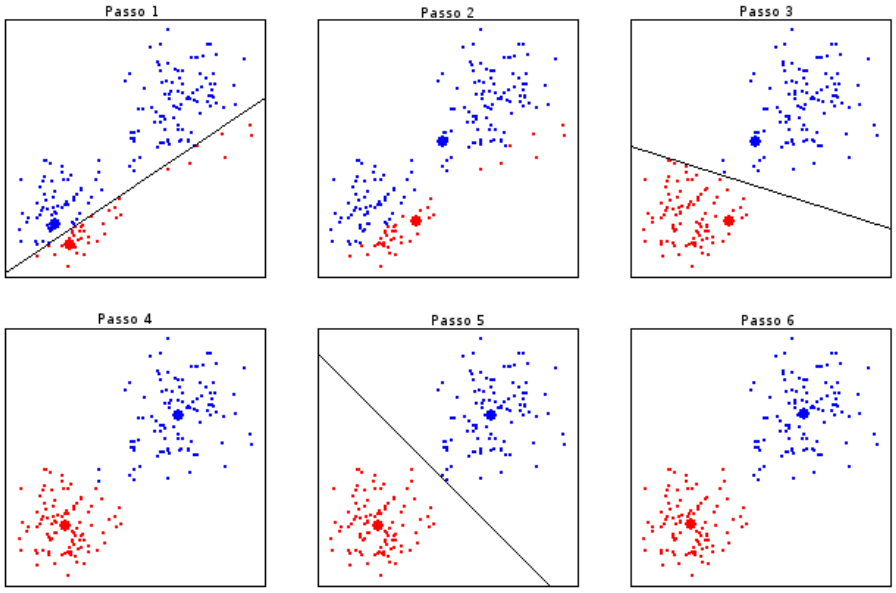
[15] Disponível em <https://opencv-python-tutroals.readthedocs.io/en/latest/>
[16] Disponível em <http://www.numpy.org/>
[17] Disponível em <http://scikit-learn.org/stable/>
[18] Disponível em <https://matplotlib.org/>

2.5 Clustering

Clustering é um processo de classificação não-supervisionada. Dado um conjunto de dados não-nomeados ou desconhecidos, este é particionado em grupos, baseando-se nas características de cada dado (ABBAS, 2008). Em outras palavras, é a divisão de um conjunto de dados em grupos que compartilham similaridades entre si. *K-Medians* (ANDERSON et al., 2006), *K-Medoids* (CAO; YANG, 2010), *K-Means* (YADAV; SHARMA, 2013) e *K-Majority* (GRANA; BORGHESANI; CUCCHIARA, 2013) são alguns exemplos de algoritmos de *clustering*.

Inicialmente proposto por MacQueen (1967), *K-Means* é um não-supervisionado, não-determinístico, numérico e interativo método de *clustering*. Cada grupo é representado pelo valor médio (centroide) dos objetos pertencentes ao grupo. O algoritmo consiste em duas fases. Primeiramente os k centroides são iniciados aleatoriamente – ou com algum algoritmo de inicialização, então, cada objeto do conjunto de dados é associado ao centroide mais próximo. A distância Euclidiana é utilizada para mensurar a distância entre os centroides e cada objeto do conjunto de dados (YADAV; SHARMA, 2013). A cada iteração do algoritmo, os centroides são ajustados até que se estabeleçam os grupos. A Figura 5 ilustra, passo a passo, a atualização dos centroides do *K-Means*.

Figura 5 – Atualização do valor dos centroides do K-Means.



Fonte: Adaptado de Madushan (2017).

Quando o conjunto de dados é composto de *features* binárias ou palavras, a distância Euclidiana e o *K-Means* não são escolhas válidas, já que estes dados não pertencem ao espaço Euclidiano. Utilizar o *K-Medoids* (CAO; YANG, 2010) seria uma alternativa.

Entretanto, buscando evitar os demasiados cálculos que são requeridos pelo *K-Medoids*, Grana, Borghesani e Cucchiara (2013) criaram o *K-Majority*. Este utiliza a distância de Hamming para calcular as diferenças entre vetores binários (quantidade de diferentes bits em suas respectivas posições) e um sistema de votos para definir os centroides, onde cada elemento do conjunto de dados vota em 0 ou 1 para uma respectiva posição.

A primeira parte do *K-Majority* assemelha-se aos demais algoritmos de *clustering*, onde são gerados k centroides aleatoriamente – ou escolhidos aleatoriamente dentro do conjuntos de dados. Utilizando a distância de Hamming, cada elemento do conjunto de dados é associado ao centroide mais semelhante. Em cada iteração, os elementos votam em 0 ou 1 para cada posição do vetor binário, atualizando os centroides até que estes não mudem mais e os grupos se estabeleçam. Comparado com os demais algoritmos de *clustering*, o *K-Majority* apresenta vantagens, como fácil implementação e velocidade. Enquanto outros algoritmos necessitam de cálculos complexos, o sistema de votos do *K-Majority* e a distância de Hamming funcionam apenas com operações básicas (GRANA; BORGHESANI; CUCCHIARA, 2013).

2.6 Recuperação de Informação Baseada em Texto

O método de recuperação de informação baseada em texto (*text-based informational retrieval*) vem sendo extensamente estudado na Ciência da Computação e é largamente utilizado em recuperação de informação. Em abordagens baseadas em representação por conjunto de palavras (BOW), os documentos são transformados em uma lista de palavras. A partir desta lista, são retiradas as palavras mais comuns, como “e” e “a”, as quais são pouco discriminantes. Essas palavras são denominadas *stop list*. Após a *stop list* ser retirada do BOW, são gerados arquivos invertidos (*inverted files*), os quais armazenam a localização e o quão importante uma palavra é para a recuperação de um documento (TF-IDF), sendo utilizados para a classificação e eficiência de busca por documentos em grandes quantidades de dados (PATTANIYIL; ZANIBBI, 2014).

O TF-IDF é um valor estatístico calculado sobre todo o conjunto de palavras, sendo a multiplicação de dois valores. O primeiro valor (TF) é a frequência que uma palavra aparece em um determinado documento, enquanto que o segundo valor (IDF) reflete a quantidade de documentos que essa palavra aparece, sendo calculado pela fórmula:

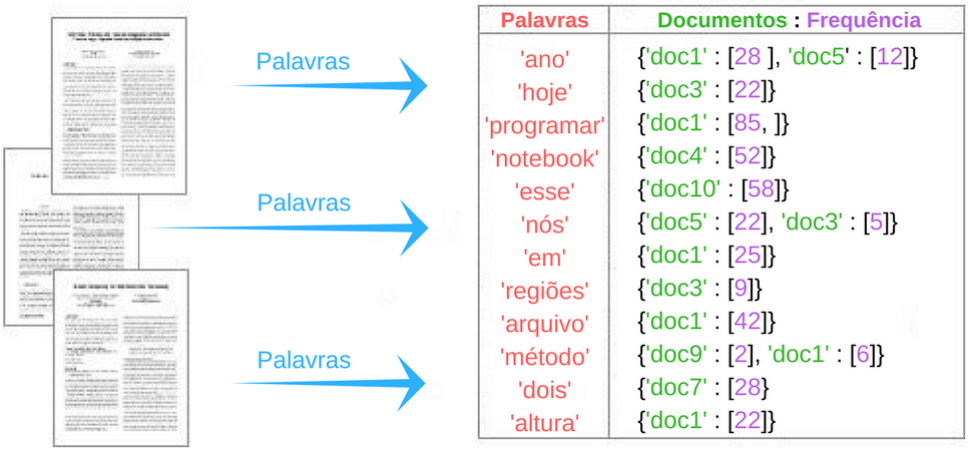
$$IDF(p) = \ln \frac{D}{D_p} \tag{2.1}$$

onde D é a quantidade total de documentos e D_p é a quantidade de documentos que a palavra p aparece. Com isso, palavras menos discriminantes são penalizadas.

Segundo Catena, Macdonald e Ounis (2014), estruturas eficientes de arquivos invertidos são a chave para a rápida resposta à consultas em motores de busca. Documentos normalmente são armazenados como uma lista de palavras. Arquivos invertidos, também

conhecidos como índices invertidos, invertem isso, armazenando, para cada palavra, a lista de documentos em que essa palavra aparece. Há muitas variações desse método, onde pode-se armazenar o TF-IDF e as localizações em que tal palavra aparece em um documento (MAHAPATRA; BISWAS, 2011). A Figura 6 mostra uma ilustração do funcionamento dos arquivos invertidos.

Figura 6 – Ilustração do funcionamento dos arquivos invertidos.



Fonte: Elaborado pelo autor.

Em seu trabalho, Sivic e Zisserman (2003) exploram as possibilidades de utilização de BOWs, TF-IDF e arquivos invertidos para reformular a classificação de imagens e reconhecimento de objetos. Em essência, isso requer uma analogia de descritores (subseção 2.2.2) extraídos de imagens como palavras visuais. Descritores assumirão o papel das palavras e as imagens assumirão o papel dos documentos, criando um "vocabulário visual".

2.7 Árvore de Vocabulário

Inspirados por Sivic e Zisserman (2003), Nister e Stewenius (2006) propõem um TF-IDF (seção 2.6) hierárquico para calcular a pontuação entre a imagem de consulta e as imagens da base de dados. As palavras visuais são definidas e quantizadas hierarquicamente, formando uma árvore de vocabulário (VocabTree), também conhecida como árvore visual. Isso provê mais eficiência na pesquisa por palavras visuais, o que possibilita a utilização de um vasto vocabulário.

Esse método segue quatro passos principais, os quais são:

1. Indexar hierarquicamente – em uma árvore – os descritores extraídos das imagens da base de dados usando *k-means* (seção 2.5);

2. Utilizar arquivos invertidos para armazenar os TF-IDF das imagens que tiveram seus descritores (pelo menos um) passando por dado nodo;

3. Para dada imagem de consulta, calcular a pontuação de cada imagem com nodos em comum em seus caminhos de descida;

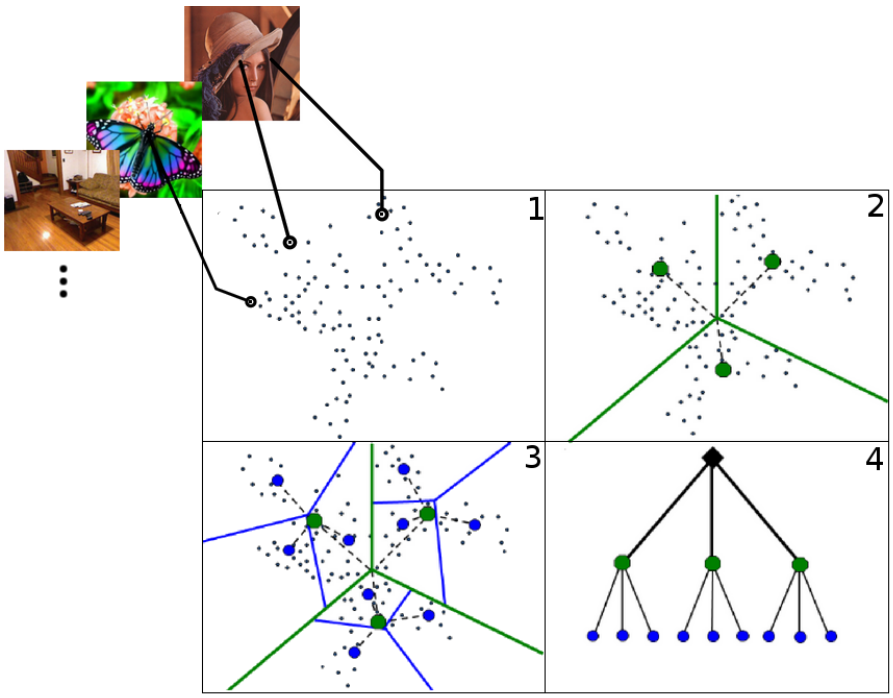
4. Encontrar imagens com melhores pontuações na base de dados.

Entre as contribuições que a utilização de VocabTrees proporcionam, pode-se citar que esse esquema define diretamente a quantização dos descritores, estando a quantização e a indexação completamente integradas.

2.7.1 Construção da Árvore de Vocabulário

Em sua fase de treinamento, as imagens do conjunto de treinamento são lidas, uma por uma, seus descritores são extraídos e armazenados em um vetor (vetor de descritores). Este vetor é composto pelos descritores de todas as imagens do conjunto de treinamento, os quais ficam misturados e sem distinção. Uma imagem pode possuir centenas, ou até milhares, de descritores.

Figura 7 – Construção da árvore de vocabulário



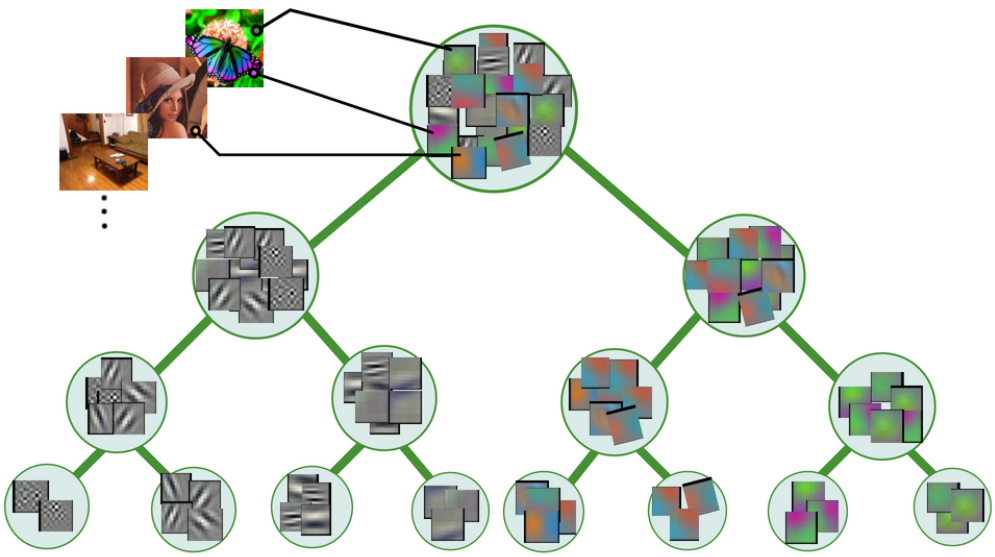
Fonte: Adaptado de Mars (2014).

Inicialmente, na raiz da árvore, o vetor de descritores é *clusterizado* (seção 2.5) e separado em k grupos. Cada um dos k grupos são passados para os k filhos do nodo

raiz. Este processo é, então, aplicado recursivamente para cada grupo de descritores, em cada nodo, construindo a VocabTree. Quando a árvore alcançar o nível limite L predeterminado, a construção termina. O parâmetro k também define o grau da árvore, ou seja, define a quantidade de filhos que cada nodo terá (exceto nodos folha).

Na Figura 7, é ilustrado o processo de construção da VocabTree, onde k é igual a 3. No quadrante 1, os descritores são representados em um espaço Euclidiano 2D, sendo possível visualizar mais claramente suas relações. No quadrante 2, é executado o algoritmo de *clustering* e são encontrados os k centroides (pontos verdes), separando o vetor de descritores em k grupos. No quadrante 3, repete-se o passo anterior para cada um dos grupos, encontrando os centroides (pontos azuis) e separando-os em novos grupos. Ao fim da descida, quando chegar ao nível limite L, a árvore é formada (quadrante 4) e os descritores semelhantes ficam em uma mesma folha da VocabTree.

Figura 8 – Ilustração visual dos descritores nos nodos da árvore de vocabulário



Fonte: Elaborado pelo autor.

Cada descritor é um vetor numérico (ou binário) que representa uma característica visual de uma imagem. A Figura 8 ilustra uma árvore de vocabulário já contruída, com k igual a 2 e L igual a 3 (a raiz não é considerada), representando os descritores em suas formas visuais. A ilustração busca facilitar o entendimento de como é realizada a *clusterização* e, ao fim da construção da VocabTree, como descritores semelhantes ficam em uma mesma folha.

2.7.2 Sistema de Pontuação e Consulta com a Árvore de Vocabulário

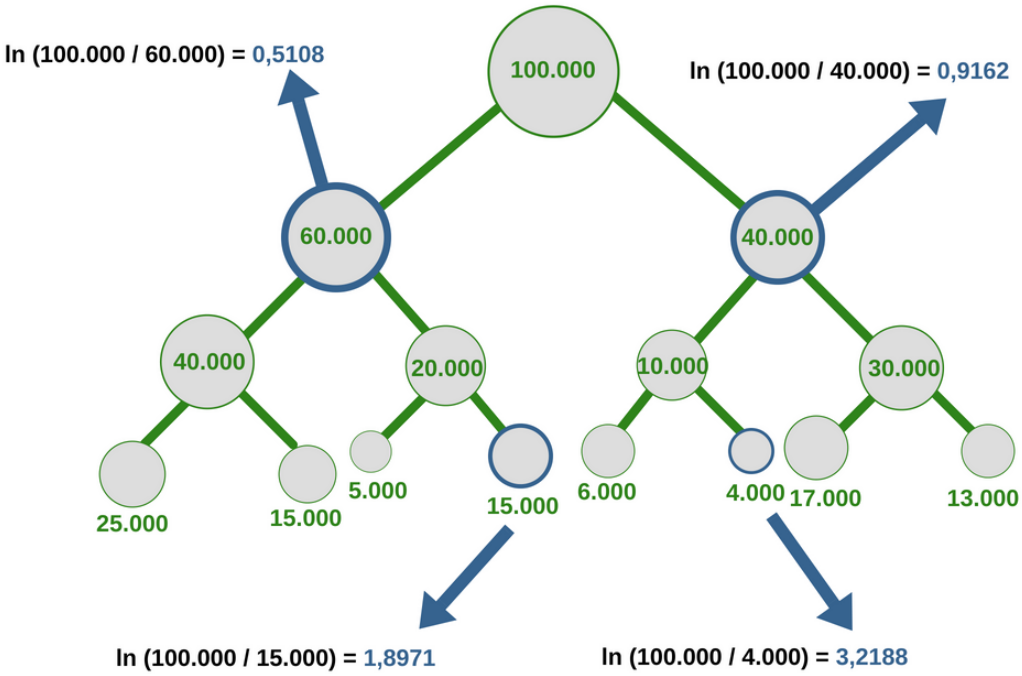
Durante a construção da VocabTree, são realizados os cálculos de todos os pesos e estatísticas relativas às imagens presentes na base de dados, buscando agilizar ao máximo

o processo de busca. Cada nodo i da VocabTree recebe um peso w_i o qual é dado por:

$$w_i = \ln \frac{N}{N_i} \tag{2.2}$$

onde $i \in I$, o qual é o conjunto com os índices de todos os nodos da VocabTree, N é a quantidade total de imagens na base de dados e N_i é a quantidade de imagens da base de dados que teve pelo menos um descritor passando pelo nodo i. O cálculo deste peso visa penalizar nodos muito comuns. Quanto mais imagens têm seus descritores passando por um nodo, menos discriminante este nodo é, tendo um peso menos relevante para a pontuação. Na Figura 9, é possível visualizar os pesos e a relevância dos nodos para o sistema de pontuação.

Figura 9 – Ilustração da relevância dos nodos para o sistema de pontuação.



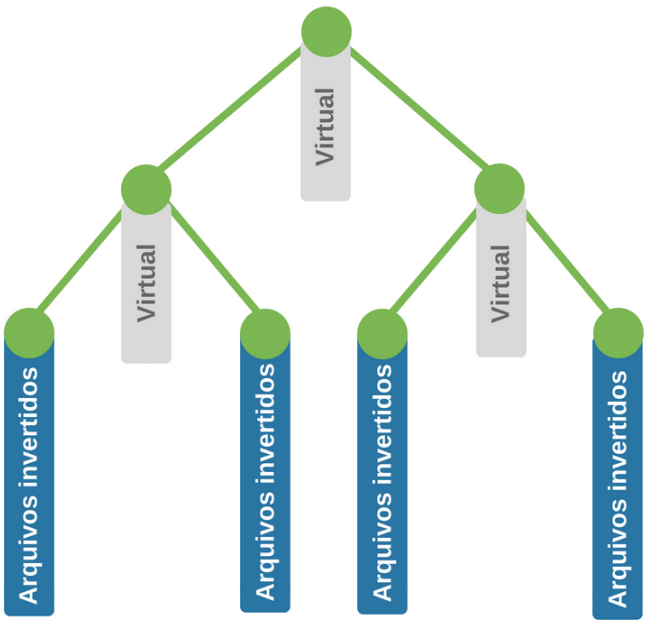
Fonte: Elaborado pelo autor.

Como os descritores são propagados da raiz às folhas, quanto mais próximo das folhas um nodo estiver, menos descritores esse nodo receberá do seu nodo pai. A definição do peso da Equação 2.2 implica que, quanto mais próximo às folhas, maior tende a ser o peso desse nodo para o sistema de pontuação.

Ainda durante a construção da VocabTree, para cada imagem $j \in J$, onde J é o conjunto com os índices das imagens presentes na base de dados, são calculados seus d_{ij}, sendo

$$d_{ij} = m_{ij} w_i \tag{2.3}$$

Figura 10 – Ilustração da árvore de vocabulário com arquivos invertidos.



Fonte: Elaborado pelo autor

onde m_{ij} é a quantidade de descritores da imagem j que passaram pelo nodo i (TF), w_i é o peso do nodo i (IDF) e d_{ij} é o TF-IDF da imagem j para cada nodo i. É importante ressaltar que d_{ij} será zero, se nenhum descritor da imagem j passar pelo nodo i.

Para manter um sistema de pontuação eficiente em bases de dados muito grandes, são utilizados arquivos invertidos (seção 2.6), assim cada nodo da VocabTree é associado com um arquivo invertido, o qual armazena os identificadores das imagens as quais passam por esse nodo, assim como seus d_{ij}.

Nodos folha são ligados diretamente aos arquivos invertidos, enquanto que os nodos não-folha são apenas concatenações dos arquivos invertidos dos nodos folha. Essa abordagem é utilizada para economizar memória. A Figura 10 ilustra como a VocabTree é entendida, onde nodos não-folha são vistos como sendo virtuais e nodos folha são vistos contendo os arquivos invertidos.

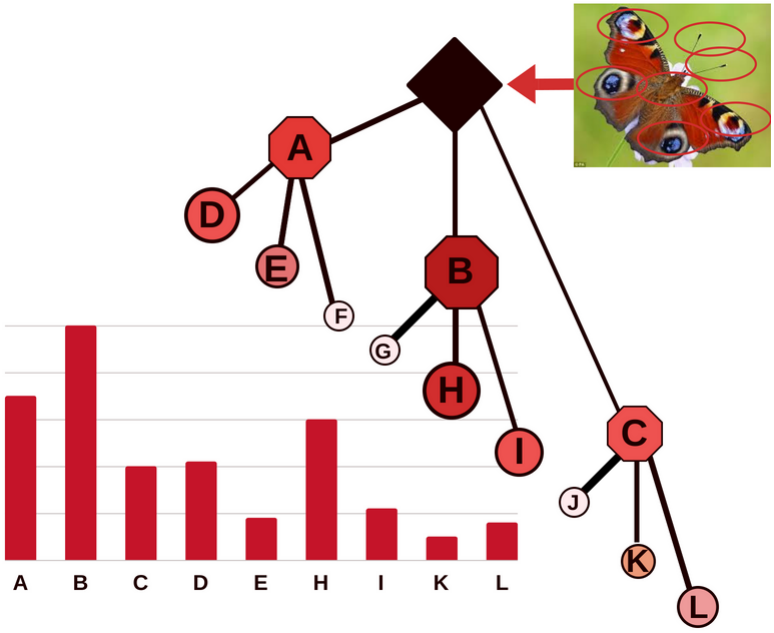
Durante a fase de busca, uma imagem é recebida como consulta, seus descritores são propagados pela VocabTree, comparando-os com cada centroide, em cada nodo. A semelhança entre as imagens é dada pela semelhança do conjunto de trajetórias de descida de seus descritores pela VocabTree. Esses caminhos são codificados em histogramas que são comparados e geram um único valor escalar que mensura a diferença entre as imagens.

Durante a descida dos descritores da imagem de consulta pela VocabTree, são calculados seus q_t, para cada nodo t, sendo

$$q_t = n_t w_t \tag{2.4}$$

onde $t \in T$, o qual é o subconjunto de nodos pelos quais a imagem de consulta teve seus descritores passando, sendo $T \subset I$. n_t é a quantidade de descritores da imagem de consulta que passaram pelo nodo t (TF), w_t é o peso do nodo t (IDF) e q_t é o TF-IDF da imagem de consulta para cada nodo t. Ao fim da descida dos descritores da imagem de consulta, seus q_t são compilados em um histograma q, com $q = (q_{t_1}, q_{t_2}, ..., q_{t_n})$, onde $(t_1, t_2, ..., t_n) = T$. Nodos, os quais não tiveram descritores da imagem de consulta passando por eles, são desconsiderados.

Figura 11 – Histograma da imagem de consulta.



Fonte: Elaborado pelo autor.

A Figura 11 ilustra a constituição do histograma da imagem de consulta. Os nodos F, G e J não fazem parte do histograma, pois nenhum descritor da imagem de consulta passou por eles.

Para o cálculo da pontuação, apenas imagens que tiveram seus descritores passando em pelo menos um dos nodos do conjunto T serão consideradas. Com isso, um subconjunto Z de imagens possivelmente semelhantes são consideradas, sendo $Z \subset J$, reduzindo a quantidade de imagens a serem comparadas. Assim, para cada imagem $z \in Z$, seus d_{tz} (Equação 2.3) são recuperados dos arquivos invertidos e compilados em um histograma d_z, com $d_z = (d_{t_1 z}, d_{t_2 z}, ..., d_{t_n z})$, onde $(t_1, t_2, ..., t_n) = T$.

Por fim, a pontuação para cada imagem z é dada pela diferença normalizada entre os histogramas q e d_z, sendo

$$s_z(q, d_z) = || \frac{q}{||q||} - \frac{d_z}{||d_z||} || \tag{2.5}$$

onde, quanto menor s_z, mais semelhantes a imagem de consulta e a imagem z são.

3 TRABALHOS RELACIONADOS

Neste capítulo, é apresentado o mapeamento sistemático da literatura realizado para avaliar motores de busca robustos e escaláveis que utilizem árvores de vocabulário como estrutura base. Esse mapeamento foi executado conforme o guia proposto por Petersen et al. (2008). A seguir, na seção 3.1, é apresentado o protocolo de pesquisa do mapeamento. Na seção 3.2 são apresentados os resultados obtidos na pesquisa e, por fim, na seção 3.3 são apresentadas as considerações do capítulo.

3.1 Protocolo de Pesquisa

O propósito deste mapeamento sistemático é inspecionar o estado da arte em motores de busca por imagens, utilizando métodos baseados em árvores de vocabulário, os quais escalam eficientemente para grandes quantidades de objetos e são robustos à rotação, ponto de vista, iluminação, tamanho e oclusão. O primeiro passo é a definição do escopo do mapeamento, ao qual se dá a definição de um conjunto de pesquisa. O escopo foi definido de forma a descobrir quais técnicas de extração e indexação de descritores de imagens são utilizados e como essa informação é tratada, além de descobrir quais desafios e benefícios que esse tipo de aplicação proporciona. Consolidando essas dúvidas levantadas, as seguintes questões foram definidas:

- **Q1**: Quais algoritmos são utilizados para a extração de descritores?
- **Q2**: Qual método é utilizado para a classificação dos descritores?
- **Q3**: Qual estratégia é adotada para o cálculo da pontuação?

O processo de busca foi estabelecido a partir da definição da base de dados. O *Scopus*[1] foi utilizado como base de dados por indexar outras bases de pesquisa como: *IEEE (Institute of Electrical and Electronics Engineers) Xplore Digital Library*[2], *ACM (Association Computing Machinery) Digital Library*[3] , *ScienceDirect*[4] e *Springer*[5]. A seguinte chave de pesquisa foi utilizada como forma de busca avançada:

– *TITLE-ABS-KEY ((“scale invariant” OR “large scale” OR “scalable”) AND ((“object” OR “image”) AND (“search” OR “recognition” OR “retrieval” OR “matching”)) AND “vocabulary tree”)*

Para filtragem dos trabalhos e eliminação de falsos-positivos, foram utilizados os seguintes critérios de seleção:

[1] Disponível em <https://www.scopus.com/search/form.uri?display=basic>
[2] Disponível em <http://ieeexplore.ieee.org/Xplore/home.jsp>
[3] Disponível em <http://www.acm.org/publications/digital-library>
[4] Disponível em <http://www.sciencedirect.com/>
[5] Disponível em <http://www.springer.com/br/>

- **Critérios de inclusão**:
  - Trabalho que propõe busca de imagens a partir de imagens de exemplo;
  - Trabalho que propõe uma solução escalável de reconhecimento de objetos em imagens utilizando árvores de vocabulário como base;
  - Trabalho que utiliza características extraídas de imagens a partir de regiões, cantos e bordas;
  - Trabalho que propõe um reconhecimento de objetos robusto à rotação, ponto de vista, iluminação, escala e oclusão.
- **Critérios de exclusão**:
  - Trabalho que não suporta modificação ou aumento de imagens na base de dados;
  - Trabalho publicado anterior ao ano de 2007;
  - Trabalho que não está escrito em português ou inglês;
  - Trabalho com menos de quatro páginas;
  - Trabalho que não possua descrição de métodos e algoritmos utilizados;
  - Trabalho que não esteja disponível para download, de forma gratuita, através de bases de dados acessíveis pela Universidade Federal do Pampa (UNIPAMPA).

Cada trabalho teve seus títulos, palavras-chave e resumos analisados, de forma a classificá-los. Os trabalhos escolhidos foram lidos de forma mais aprofundada, buscando obter um maior entendimento sobre estes. Por fim, os trabalhos mais relevantes foram escolhidos.

3.2 Resultados da Pesquisa

Inicialmente, utilizando a chave de busca em forma de pesquisa avançada, foram retornados 83 trabalhos. Após a análise dos seus resumos, palavras-chave e títulos e a aplicação dos critérios de filtragem, foram selecionados 17 trabalhos. Após leitura destes 17 trabalhos e nova aplicação dos critérios de filtragem, por fim, 5 trabalhos foram selecionados para compor o mapeamento sistemático.

Em seu trabalho, Shiwei et al. (2017) apresentam um novo método para adição de novas imagens durante a fase de busca. À medida que novas imagens vão sendo adicionadas à base de dados, os centroides da VocabTree vão sofrendo distorção, o que implica na necessidade de reconstrução da VocabTree. O método proposto por eles permite adicionar

imagens à base de dados durante a fase de busca, com distorção reduzida dos centroides em cada nodo da VocabTree.

Wang et al. (2016) propõem um sistema para reconhecimento de veias do dedo, o qual, segundo os autores, apresenta vantagens ao método tradicional de sistema de digitais. Cada imagem é dividida em regiões de interesse (manchas), as quais são utilizadas como imagens de treinamento. A escolha da VocabTree como estrutura base deve-se à grande quantidade de imagens a serem tratadas, sendo proporcional à população.

Gomes et al. (2016) apresentam uma nova abordagem a qual baseia-se no paradigma *MapReduce* (DEAN; GHEMAWAT, 2004) para a construção da VocabTree. Programas escritos utilizando este paradigma são automaticamente paralelizados e executados em um conjunto de máquinas de produção.

Jiang et al. (2015) propõem um método escalável para busca e diagnóstico de massas mamográficas. A VocabTree é construída a partir de um conjunto de imagens as quais contém imagens de massas e de falsos-positivos. Ao fim da consulta, o diagnóstico é dado a partir das imagens retornadas como mais semelhantes. Caso imagens de massas sejam retornadas como semelhantes, o diagnóstico é dado como massa. Caso contrário, é um falso-positivo.

Por fim, Banlupholsakul, Ieamsaard e Muneesawang (2014) apresentam um novo método para pontuação em sistemas de reconhecimento de pontos de referência. Após a realização da consulta, é utilizado um sistema de seleção o qual analisa as imagens semelhantes retornadas, classificando os descritores entre importantes e não importantes. Após isso, as imagens são ranqueadas novamente, a partir dos descritores mais importantes.

Após leitura aprofundada dos trabalhos selecionados, as questões levantadas foram respondidas da seguinte forma (Tabela 1):

Tabela 1 – Trabalhos Relacionados.

Título do Trabalho	**Autores**	**Q1**	**Q2**	**Q3**
Online Real-time Image Retrieval Based on Large scale Vocabulary Tree	(SHIWEI et al., 2017)	SIFT	K-Means	Similaridade entre Histogramas
Binary Search Path of Vocabulary Tree Based Finger Vein Image Retrieval	(WANG et al., 2016)	SIFT	K-Means	Sobreposição de vetores binários
MapReduce Vocabulary Tree: An Approach for Large Scale Image Indexing and Search in The Cloud	(GOMES et al., 2016)	ORB	K-Majority	Distância de Bhattacharyya
Computer-Aided Diagnosis of Mammographic Masses Using Scalable Image Retrieval	(JIANG et al., 2015)	SIFT	K-Means	Similaridade média entre caminhos
Re-ranking Approach to Mobile Landmark Recognition	(BANLUPHOLSAKUL; IEAMSAARD; MUNEESAWANG, 2014)	SIFT	K-Means	Similaridade entre Histogramas

Fonte: Elaborado pelo autor.

3.2.1 Algoritmos utilizados para a extração de descritores

A grande maioria dos autores estudados optaram pela utilização do algoritmo de extração de descritores (subseção 2.2.2) SIFT (LOWE, 2004), o qual tornou-se referência em sua classe de algoritmos desde sua publicação. Shiwei et al. (2017), Wang et al. (2016), Jiang et al. (2015) e Banlupholsakul, Ieamsaard e Muneesawang (2014) o utilizam em seus métodos propostos, mesmo este sendo patenteado.

Gomes et al. (2016) optou por utilizar o algoritmo de extração de descritores ORB (RUBLEE et al., 2011), o qual é um algoritmo livre, rápido e robusto. Comparado ao SIFT, o ORB perde em qualidade, mas há um ganho substancial em velocidade e economia de memória, sendo ótimo para grandes quantidades de dados e sistemas embarcados.

3.2.2 Método utilizado para a classificação dos descritores

Após a definição de qual extrator de características utilizar, o próximo passo é definir qual algoritmo será utilizado para classificar essas características. Shiwei et al. (2017), Wang et al. (2016), Jiang et al. (2015) e Banlupholsakul, Ieamsaard e Muneesawang (2014) optaram pela utilização do algoritmo *K-Means* (seção 2.5), o qual é o mais clássico e comumente utilizado para agrupamento de dados.

Descritores gerados pelo ORB são inapropriados para o *K-Means*. O ORB gera descritores binários e o *K-Means* trabalha com dados sobre o espaço Euclidiano. Devido a isso, Gomes et al. (2016) optou pela utilização do algoritmo *K-Majority* (GRANA; BORGHESANI; CUCCHIARA, 2013), o qual utiliza uma abordagem semelhante ao *K-Means*, modificando a distância Euclidiana pela distância Hamming e um centroide selecionado pela maioria.

3.2.3 Estratégia adotada para o cálculo da pontuação

Após a construção da VocabTree, é preciso definir um sistema de pontuação entre as imagens presentes na base de dados e uma imagem de consulta. Shiwei et al. (2017) e Banlupholsakul, Ieamsaard e Muneesawang (2014) utilizam a mesma abordagem que Nister e Stewenius (2006), calculando a pontuação através da similaridade entre os histogramas de cada imagem, a partir do TF-IDF de cada nodo pertencente aos caminhos de descida das imagens (Equação 2.5).

Wang et al. (2016) utiliza uma abordagem baseada em uma VocabTree binária. Durante a descida dos descritores das imagens, cada nodo recebe um valor binário (0 ou 1). Caso o descritor de dada imagem passar pelo nodo, este é assinalado com o valor 1. Caso contrário, este recebe o valor 0. Assim é formado um vetor binário para cada descritor de cada imagem, os quais representam o caminho de descida de cada imagem pela VocabTree binária. O valor da pontuação é dado pelo valor total de sobreposições dos vetores binários de cada imagem com os vetores da imagem de consulta.

Gomes et al. (2016) utiliza um método semelhante a Shiwei et al. (2017) e Banlupholsakul, Ieamsaard e Muneesawang (2014), o qual utiliza a distância entre histogramas para calcular a pontuação. Após a descida dos descritores e a computação dos histogramas a partir dos TF-IDF dos nodos, a distância Bhattacharyya é utilizada para calcular a similaridade entre dois histogramas. A distância Bhattacharyya é bastante utilizada na área estatística para calcular a similaridade entre duas distribuições de probabilidade.

Jiang et al. (2015) utiliza uma abordagem na qual a similaridade total entre duas imagens é dada pela similaridade média entre os caminhos de descida entre duas imagens. O conjunto de descritores relativos à imagem de consulta é comparado aos conjuntos de descritores de cada imagem da base de dados. A similaridade entre os caminhos de dois descritores é definida como uma contagem ponderada entre seus nodos em comum. Cada descritor terá seu valor ponderado calculado, somado aos valores dos demais descritores da imagem e dividido pelo total de descritores comparados (descritores da imagem de consulta + descritores da imagem da base de dados), resultando na média de similaridade entre os descritores de duas imagens.

3.3 Considerações do Capítulo

Através do mapeamento realizado, pôde-se conhecer, em maior detalhe, as diferenças entre abordagens e a capacidade de adaptação que árvores de vocabulário aceitam. Modificações na extração de características, no algoritmo de classificação, no grau da árvore e no sistema de pontuação podem ser facilmente testados para a definição de abordagens mais robustas para cada objetivo. Pôde-se verificar que a maioria dos trabalhos estudados optam pelo algoritmo de extração de descritores SIFT (LOWE, 2004), o qual continua sendo o preferido dentre os autores. Ainda, é notável a baixa utilização de métodos e algoritmos livres, como ORB (RUBLEE et al., 2011), KAZE (ALCANTARILLA; BARTOLI; DAVISON, 2012) e BRISK (LEUTENEGGER; CHLI; SIEGWART, 2011) para a extração de características.

Os diferentes sistemas de pontuação vistos reforçam a ideia de adaptação que árvores de vocabulário possuem, sendo possível considerar inúmeras características. Entretanto, todos os métodos consideram a descida dos descritores pela VocabTree para o cálculo das pontuações.

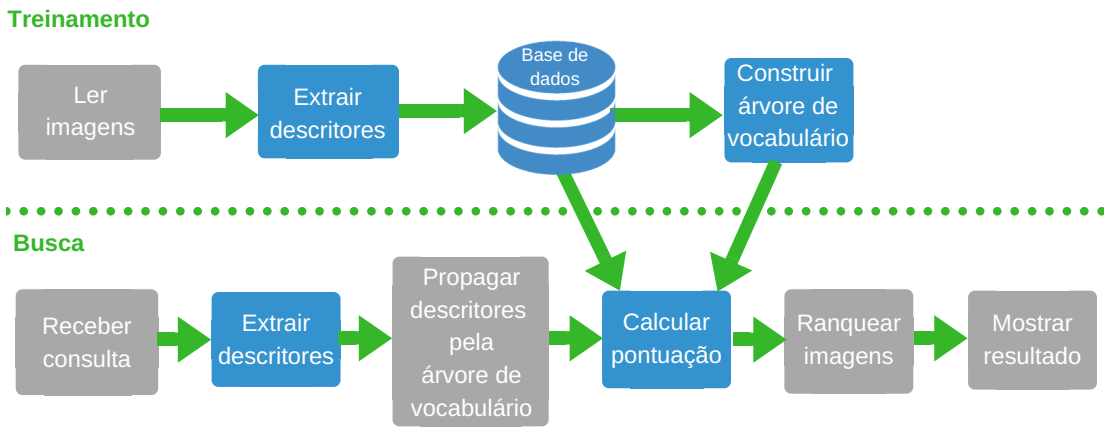
Durante o processo de busca por trabalhos, nenhum trabalho baseado em árvores de vocabulário de código aberto pôde ser encontrado. Entretanto, o trabalho proposto por Gomes et al. (2016) utiliza de métodos livres e foi de grande ajuda para alcançar os objetivos deste trabalho.

4 DESENVOLVIMENTO DO TRABALHO

Neste capítulo é apresentado o desenvolvimento de uma aplicação CBIR escalável de busca de imagens semelhantes utilizando apenas métodos e algoritmos livres, a qual foi dado o nome de OpenVT. A aplicação está disponível em <https://github.com/giullianopaz/OpenVT>, estando sob a licença *GNU GPL v3.0*[1].

Como base, é utilizado o método proposto por Nister e Stewenius (2006), o qual utiliza uma estrutura hierárquica denominada árvore de vocabulário (seção 2.7) para a indexação e quantização das informações extraídas de imagens. Como mostra a Figura 12, a aplicação funciona em duas principais fases: a fase de treinamento e a fase de busca. Em azul, estão destacados os pontos críticos que este trabalho concentrou esforços, estando diretamente relacionados à eficiência final da aplicação.

Figura 12 – Visão geral da aplicação.



Fonte: Elaborado pelo autor.

Durante a fase de treinamento, os descritores das imagens do conjunto de treinamento são extraídos e armazenados na base de dados. Armazenando os descritores na base de dados, poupa-se tempo e recursos para futuras execuções da OpenVT, não sendo necessário extrair os descritores novamente. Após o término da construção da VocabTree, a OpenVT passa a aceitar consultas. Na fase de busca, recebem-se imagens de consulta, extraindo seus descritores e propagando estes pela VocabTree, comparando-os com os centroides em cada nodo. A semelhança entre as imagens é dada a partir da semelhança entre as trajetórias de descida dos descritores de cada imagem pela VocabTree. Por fim, as imagens da base de dados são ranqueadas e retornadas ao usuário.

Após uma breve análise sobre ferramentas utilizadas em projetos de visão computacional, a biblioteca de visão computacional OpenCV (versão 3.1.0), junto à linguagem de programação Python (versão 3.5.2), foram escolhidas para o desenvolvimento deste

[1] Disponível em <https://opensource.org/licenses/GPL-3.0>

projeto. Devido à grande comunidade de desenvolvedores e entusiastas sobre visão computacional que optam por utilizar a biblioteca, OpenCV possui um vasto número de métodos e algoritmos eficientes já implementados, como foi visto na seção 2.3. A escolha da linguagem de programação Python se deve às suas características, as quais proveem um rápido desenvolvimento e uma grande quantidade de módulos. O MySQL[2] foi escolhido como sistema de gerenciamento de base de dados (SGBD), devido á sua ótima integração com a linguagem de programação Python e devido a ser o SGBD utilizado por Nister e Stewenius (2006) em seus testes.

Durante todo o desenvolvimento da aplicação, foram consideradas inúmeras tecnologias, métodos e algoritmos para as mais variadas finalidades. Por se tratar de uma aplicação de código aberto, a OpenVT foi pensada para ser a mais flexível possível. Há um arquivo de configuração (*settings.py*) onde todas as configurações relativas à aplicação estão sujeitas aos utilizadores. Dentre elas, algumas podem ser citadas, como: configurações relativas à base de dados, como nome da base de dados, *host*, senha e usuário; algoritmo de extração de descritores; métodos de cálculo entre histogramas; largura e altura da VocabTree; diretório com as imagens de treinamento e quantidade de imagens semelhantes a serem mostradas aos usuários durante buscas.

Este capítulo está disposto da seguinte forma: na seção 4.1 é apresentada a extração de descritores pela OpenVT e sua forma de armazená-los; na seção 4.2 é discorrido sobre os algoritmos de *clustering* utilizados neste trabalho, assim como suas implementações; na seção 4.3 é explicado sobre a construção da VocabTree pela OpenVT; na seção 4.4 é discorrido sobre o funcionamento das buscas por imagens semelhantes pela OpenVT; por fim, na seção 4.5 é apresentada uma pequena aplicação web para a execução da OpenVT.

4.1 Extração dos descritores das imagens

Buscando não ater-se a apenas um algoritmo de extração de descritores, a OpenVT aceita vários algoritmos presentes no OpenCV. Dentre os algoritmos aceitos, alguns são: KAZE (ALCANTARILLA; BARTOLI; DAVISON, 2012), o qual gera descritores de valores *float*, e ORB (RUBLEE et al., 2011), BRISK (LEUTENEGGER; CHLI; SIEGWART, 2011) e AKAZE (ALCANTARILLA; BARTOLI; DAVISON, 2013), os quais geram descritores de valores binários. Entretanto, a OpenVT foi escrita para aceitar os mais diversos algoritmos, aceitando descritores dos mais variados tamanhos (dimensões) e valores (binários e *float*). Caso um futuro utilizador deseje adicionar um novo algoritmo de extração de descritores, basta adicioná-lo ao método *extract* da classe *DescriptorExtractor* e à lista de descritores, no arquivo de configuração (*settings.py*). Algoritmos não listados no arquivo de configuração não são aceitos e farão a OpenVT exibir um erro e parar sua execução.

Os algoritmos de extração de descritores binários geram descritores como um vetor

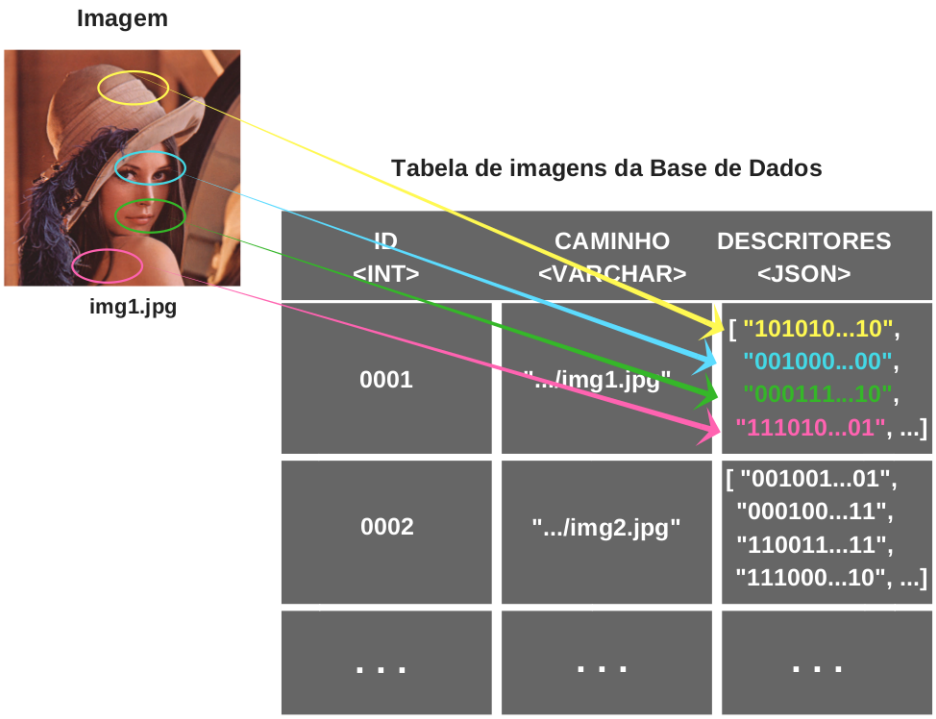
[2] Disponível em <https://www.mysql.com/>

de inteiros. Por exemplo, o descritor gerado pelo ORB (RUBLEE et al., 2011) é um vetor com 32 elementos, onde cada elemento é um valor inteiro entre 0 e 255. Para obter o real valor de um descritor ORB, é necessário transformar cada elemento do vetor para seu respectivo valor binário. Ou seja, o valor "42" representa a *bit-string* "00101010". Transformando cada um dos 32 elementos de um descritor ORB, obtém-se uma *bit-string* de 256 bits (32*8). Este é o real valor de um descritor ORB.

Para agilizar o processo, os métodos para fazer a conversão de inteiro para *bit-string* foram escritos em Cython. Para evitar processamentos desnecessários durante a construção da VocabTree e na fase de busca, os decritores binários são todos convertidos para sua versão binária antes de armazená-los na base de dados. Descritores que não são do tipo binário não necessitam de pré-tratamento.

Como ilustra a Figura 13, inicialmente as imagens do conjunto de treinamento (*dataset*) são lidas, uma por uma, e seus descritores são extraídos e armazenados em uma tabela da base de dados. Cada linha da tabela contém um identificador, a localização (diretório) da imagem e seus descritores. Cada imagem possui entre 700 e 1000 descritores, dependendo da imagem e do algoritmo de extração de descritores utilizado.

Figura 13 – Armazenamento das imagens e seus descritores.



Fonte: Elaborado pelo autor.

Além dos algoritmos de extração de descritores providos pelo OpenCV, foi consi-

derado utilizar os modelos pré-treinados[3] do Keras[4] para a extração de descritores. Entretanto, estes extraem apenas um descritor por imagem. Como a VocabTree utiliza estatísticas sobre as trajetórias dos descritores das imagens para mensurar suas semelhanças, não é possível utilizar a VocabTree com apenas um descritor por imagem. Mediante isto, iniciaram-se estudos alternativos, buscando formas de multiplicar os descritores extraídos pelos modelos pré-treinados do Keras. Inicialmente, foram utilizadas janelas deslizantes como tentativa de extrair descritores de regiões menores das imagens do conjunto de treinamento. Contudo, esta estratégia não surtiu bons resultados, onde a acurácia ficou abaixo dos 40%. A segunda estratégia pensada foi utilizar a classe *ImageDataGenerator*[5] do Keras para gerar um pequeno conjunto de imagens semelhantes, a partir de uma das imagens presentes no conjunto de treinamento. Ou seja, para cada imagem do conjunto de treinamento, seriam geradas várias imagens afim de extrair seus descritores. Com isso, cada imagem do conjunto de treinamento teria vários descritores. Entretanto, esta estratégia resultou em um desempenho muito abaixo do esperado. Então, os descritores extraídos pelos modelos pré-treinados do Keras foram desconsiderados deste trabalho.

4.2 Algoritmos de Clustering

O algoritmo de *clustering* é de suma importância para a eficiência final da OpenVT. Como a OpenVT aceita dois principais tipos de descritores (numéricos e binários), dois algoritmos de *clusteting* são necessários. Para descritores numéricos, o algoritmo escolhido foi o clássico *K-Means*. O *K-Means* do módulo Python Scikit Learn[6] foi utilizado e adaptado para retornar uma tupla contendo os centroides e os grupos. Para descritores do tipo binário, foi escolhido o *K-Majority* (GRANA; BORGHESANI; CUCCHIARA, 2013), o qual é um algoritmo para *clusterizar* dados que não pertencem ao espaço Euclidiano.

Inicialmente, o *K-Majority* foi implementado totalmente em Python. Devido à sua lenta execução, iniciaram-se implementações alternativas em Cython e C++. As versões em C++ mostram-se muito superiores às versões em Cython e, principalmente, em Python. Entretanto, módulos implementados em C++ apresentam certas desvantagens. Segundo Gorelick e Ozsvald (2014), o Python utiliza uma região privada da memória, onde toda e qualquer variável do Python só existe nesta região. Além do coletor de lixo do Python só ser capaz de desalocar variáveis presentes nesta região, não há métodos ou formas do desenvolvedor liberar memória manualmente. O Python não permite isto. Perante essas características, variáveis alocadas pelo C++ tornam-se inacessíveis para o coletor de lixo do Python, ocasionando em um enorme desperdício de memória. Como meio termo, a escolha, então, se deu por utilizar o Cython, ganhando em velocidade,

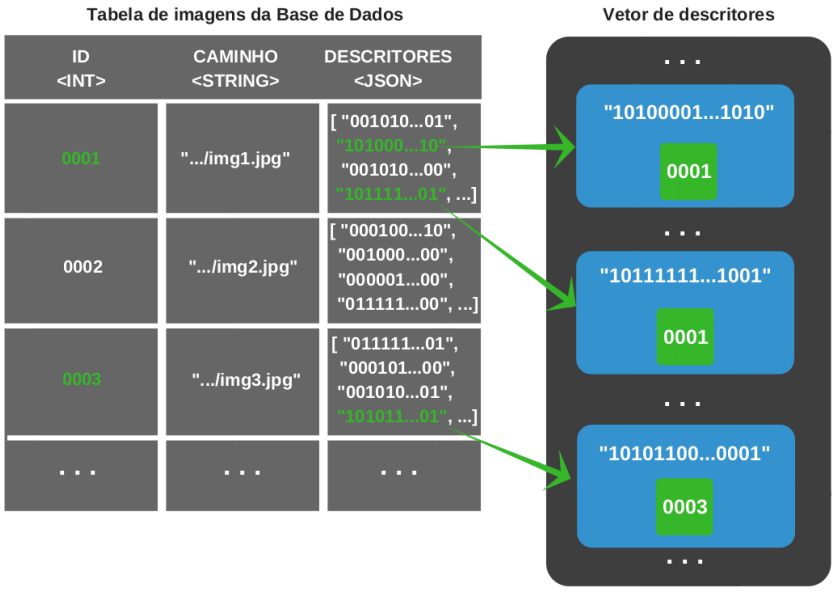
[3] Disponível em <https://keras.io/applications/>
[4] Disponível em <https://keras.io/>
[5] Disponível em <https://keras.io/preprocessing/image/>
[6] Disponível em <http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>

porém sem perder eficiência na gerência de memória. Embora o Cython não seja tão veloz quanto o C++, este trabalho optou por perder um pouco em tempo de execução, ao invés de perder em gerência de memória.

Durante o desenvolvimento em Cython, constatou-se problemas de dependências com certos módulos, como o Numpy. Segundo Smith (2015), Cython ainda apresenta incompatibilidade com a grande maioria de módulos Python. Devido a isso, foi decidido implementar em Cython apenas a parte mais crítica do *K-Majority*: o sistema de votos. Como a distância Hamming apenas recebe duas *string-bits* e retorna sua diferença, não há necessidade de alocação de memória. Então, a distância Hamming foi implementada em C++. Comparada com as primeiras versões escritas em Python puro, a versão atual do *K-Majority*, mesclada com Cython e C++, apresenta um ganho de até 15 vezes no tempo de execução.

O *K-Means* adaptado e o *K-Majority* foram implementados para serem o mais abstratos possíveis para a OpenVT, onde os dois algoritmos recebem um vetor de descritores e retornam os centroides e os grupos *clusterizados*, não havendo diferença em sua utilização. No arquivo de configuração (*settings.py*), há duas listas de algoritmos extratores de descritores. Uma com nomes de algoritmos que geram descritores binários e outra para os que geram descritores *float*. Mediante estas listas, a OpenVT decide qual dos algoritmos de *clustering* será instanciado.

Figura 14 – Vetor de descritores.



Fonte: Elaborado pelo autor.

4.3 Construção da árvore de vocabulário

Como foi explicado na seção 2.7, a VocabTree é uma estrutura de dados em árvore que indexa e quantiza hierarquicamente valores que refletem características de imagens (descritores). A classe *VocabularyTree* é constituída de dois atributos: um ponteiro para o primeiro nodo (raiz) da árvore e os arquivos invertidos, implementados em uma estrutura de dados do Python chamada "dicionários". Para a construção da VocabTree, é necessário agrupar os descritores de todas as imagens em um vetor comum a todos.

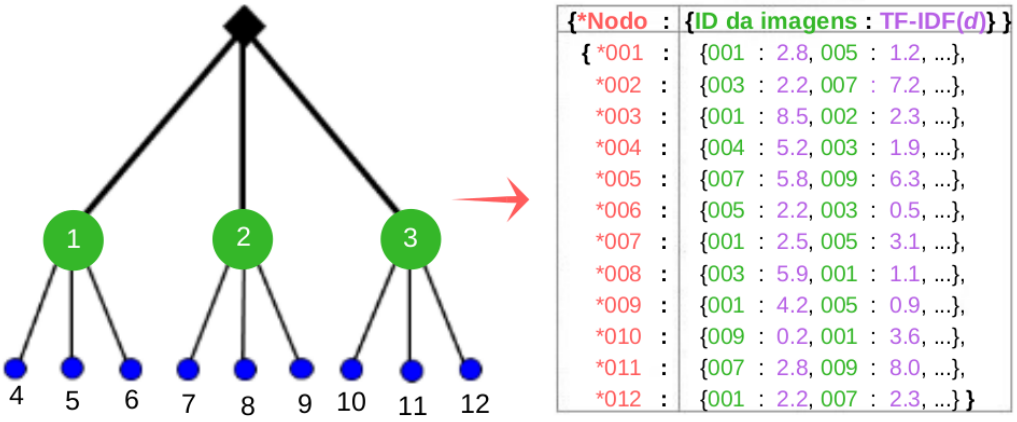
A Figura 14 ilustra o vetor de descritores, onde os descritores são lidos da base de dados e transformados em objetos (em azul), os quais são constituídos do seu valor (*bit-string* ou vetor numérico) e do identificador da imagem a qual pertence, evitando que descritores se desassociem de suas respectivas imagens de origem. Foram criadas duas subclasses para os dois diferentes tipos de descritores. *StrDescriptor* é uma subclasse da classe *Str* do Python, sendo utilizada para descritores binários. A classe *StrDescriptor* herda todos os métodos e atributos da sua classe mãe, sendo adicionado o atributo *IMAGE_ID*, que armazena o identificador da imagem a qual o descritor pertence. *ArrayDescriptor* é semelhante à *StrDescriptor*, porém é uma subclasse da classe *ndarray* do Numpy, servindo para descritores numéricos. Com isso, o vetor de descritores passado para a VocabTree é um vetor de objetos de uma destas duas subclasses, os quais carregam consigo, durante toda a execução da aplicação, como atributo, a que imagem pertencem. A escolha de qual subclasse utilizar é realizada pela OpenVT, dependendo do algoritmo de extração de descritores utilizado.

Após o vetor de descritores ser formado, este vetor é passado para a raiz da VocabTree. Dentro do nodo raiz da VocabTree, o vetor de descritores é *clusterizado* e separado em grupos, os quais são passados para seus nodos filhos. Este processo repete-se em cada nodo da VocabTree, até o fim de sua construção. A VocabTree é construída recursivamente, onde cada nodo da árvore é uma instância da classe *Node*, os quais têm, como atributo: nível que o nodo pertence; peso do nodo; lista de k centroides; e uma lista de k filhos, sendo, cada um, novas instâncias da classe *Node*.

Por economia de memória e por questões de eficiência de busca, apenas os últimos níveis da VocabTree são considerados, sendo apenas neles calculados os pesos dos nodos e os TF-IDF das imagens da base de dados. Como explicado na subseção 2.7.2, os nodos mais significativos estão próximos às folhas. Contudo, a quantidade de níveis considerados pode ser modificada no arquivo de configuração.

Os TF-IDF das imagens da base de dados são armazenados em arquivos invertidos, implementados como dicionários. Cada elemento do dicionário contém os dados relativos às imagens que passaram por determinado nodo, sendo o endereço de memória (ponteiro) deste nodo utilizado como chave. Com isso, durante a descida dos descritores das imagens de consulta, basta registrar os ponteiros dos nodos os quais passarem seus descritores, utilizando-os para recuperar os dados de cada nodo dos arquivos invertidos. A Figura 15

Figura 15 – Relação entre nodos e arquivos invertidos.



Fonte: Elaborado pelo autor.

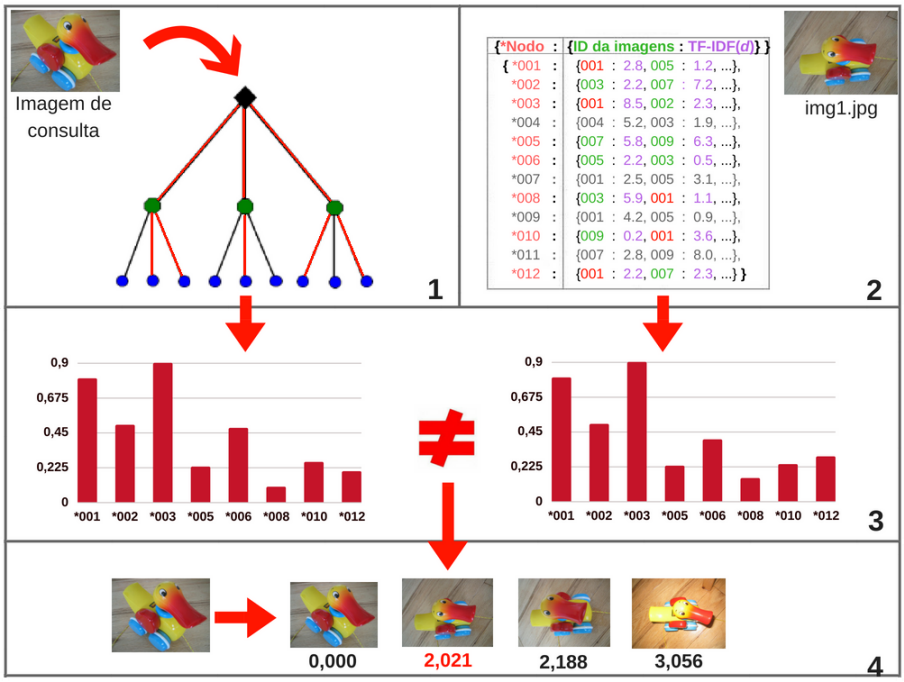
ilustra os nodos da VocabTree relacionados ao dicionários (arquivos invertidos).

4.4 Consulta com árvore de vocabulário

Como foi discutido na subseção 2.7.2, durante a fase de busca, são extraídos os descritores de cada imagem de consulta recebida, propagando seus descritores pela VocabTree e formando seu histograma. Dentro do método para explorar a VocabTree, dependendo do tipo dos descritores utilizados, a OpenVT pode utilizar a distância Hamming ou a distância Euclidiana para fazer a comparação com os centroides de cada nodo. Para cada nodo que os descritores da imagem de consulta passam, seu endereço de memória (ponteiro) é armazenado. A partir deles, os histogramas das imagens presentes na base de dados são formados, acessando seus TF-IDF armazenados nos arquivos invertidos. Como os dados relativos às imagens da base de dados estão espalhados pelo arquivo invertido, é necessário um pré-processamento para compilá-los em um vetor. Com o histograma da imagem de consulta e os histogramas das imagens da base de dados formados, estes são normalizados e comparados utilizando um dos métodos disponíveis no OpenCV, como a distância Chi-Square ou Bhattacharyya. Após todos os histogramas serem comparados, suas imagens são ordenadas por ordem crescente relativas às suas pontuações. Então, as imagens mais semelhantes são retornadas ao usuário.

A Figura 16 ilustra o passo a passo de uma consulta. No quadrante 1, os descritores da imagem de consulta são propagados pela VocabTree, formando seu histograma e registrando os nodos os quais seus descritores passaram. No quadrante 2, os TF-IDF de uma das imagens da base de dados – representada pelo identificador "*001*" – são recuperados dos arquivos invertidos, utilizando os nodos registrados no quadrante 1, formando seu histograma. Este passo é repetido para todas as imagens que tiveram pelo menos

Figura 16 – Consulta usando árvore de vocabulário.



Fonte: Elaborado pelo autor.

um descritor passando pelos nodos registrados no quadrante 1. No quadrante 3, os histogramas das imagens são normalizados e comparados. Então, no quadrante 4, as imagens semelhantes são retornadas em ordem crescente, relativas às suas pontuações.

4.5 Aplicação web para OpenVT

Foi desenvolvida uma pequena aplicação web para a execução da OpenVT. Para o desenvolvimento, foi utilizado o microframework web Flask[7], o qual é um módulo do Python próprio para a criação de pequenas aplicações web.

Foi criada apenas uma página web, a qual é utilizada para receber a imagem de consulta e mostrar a lista de imagens semelhantes. Para executar a aplicação, deve-se executar o arquivo "*extract.py*" para extrair os descritores do conjunto de imagens e armazená-los na base de dados. Então, basta executar "*app.py*", o qual faz toda a configuração necessária para o servidor local, constrói a VocabTree e fica aguardando requisições de consultas. A cada requisição, o método *image_search* é executado, o qual executa uma consulta à base de dados utilizando a VocabTree. Toda a configuração relativa à aplicação encontra-se no arquivo "*settings.py*".

Na Figura 17, é mostrada uma captura de tela da execução da OpenVT utilizando o ORB, com 50 mil folhas, para 400 imagens de rostos extraídas do conjunto de imagens

[7] Disponível em <http://flask.pocoo.org/>

Figura 17 – OpenVT para 400 imagens com 50 mil folhas.

Fonte: Elaborado pelo autor.

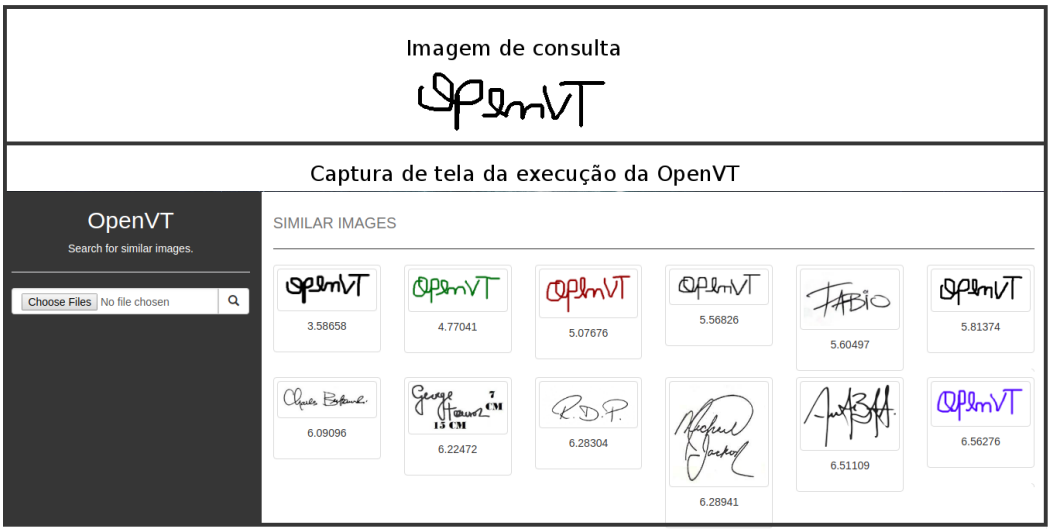
Caltech256[8]. Estas 400 imagens são divididas em 20 grupos, onde cada grupo contém 20 imagens semelhantes. A página é constituída de um botão para carregar a imagem de consulta e um botão para executar a busca. Após o botão de busca ser acionado, as imagens semelhantes são retornadas ordenadas pela sua pontuação, a qual é mostrada abaixo da imagem.

Para demonstrar o desempenho da OpenVT com imagens únicas que não foram utilizadas por outros trabalhos, foi criado um conjunto de imagens de assinaturas escritas à mão. Algumas imagens foram obtidas através do Google Imagens[9] e outras foram criadas pelo autor deste trabalho no software de edição de imagens Gimp[10].

A Figura 18 mostra a captura de tela da execução da OpenVT, utilizando o KAZE com 10 mil folhas, para 273 imagens de assinaturas escritas à mão, contendo 6 imagens que representam assinaturas fictícias da palavra "OpenVT". A imagem de consulta utilizada é outra assinatura fictícia da palavra "OpenVT", a qual não se encontra no conjunto de treinamento. Mais exemplos da execução da OpenVT são apresentados no APÊNDICE A, APÊNDICE B e APÊNDICE C.

[8] Disponível em <http://www.vision.caltech.edu/Image_Datasets/Caltech256/>
[9] Disponível em <https://www.google.com/imghp?hl=pt-pt>
[10] Disponível em <https://www.gimp.org/>

Figura 18 – OpenVT para 273 imagens com 10 mil folhas.



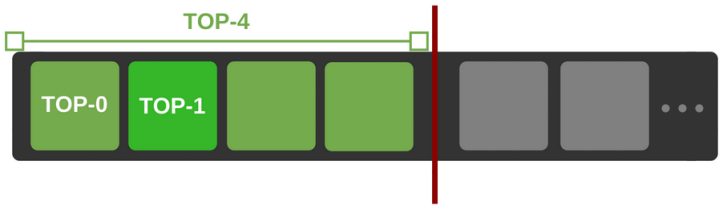
Fonte: Elaborado pelo autor.

5 RESULTADOS

Neste capítulo são apresentados os resultados obtidos utilizando a OpenVT, aplicação resultante deste trabalho. Foram avaliados: métodos de comparação entre histogramas; acurácia; algoritmos de extração de descritores; configuração e tempo de construção da VocabTree; uso de memória RAM; e a relação entre quantidade de descritores, folhas e a acurácia. Os algoritmos livres utilizados neste trabalho foram comparados com algoritmos patenteados, que utilizam modelos pré-treinados de redes neurais e com o método original apresentado por Nister e Stewenius (2006). Para a realização dos testes, foi utilizado o conjunto de imagens utilizado por Nister e Stewenius (2006), *UKBench*[1], o qual contém 10.200 imagens com objetos rotacionados, com diferentes pontos de vista e iluminação. O conjunto é dividido em 2.550 classes, onde cada classe possui 4 imagens semelhantes.

Para a realização dos testes, foi utilizado um computador com processador Core i3, 3.1 GHz, de 5ª geração e 6 GB de RAM. Para mensurar a acurácia da aplicação com diferentes quantidades de imagens, foram extraídas, do conjunto de imagens, três subconjuntos, com 100, 500 e 1.400 imagens, onde cada subconjunto contém imagens diferentes dos demais.

Figura 19 – Estratégia para avaliar a precisão da aplicação.



Fonte: Elaborado pelo autor.

A Figura 19 ilustra a estratégia utilizada para avaliar a acurácia da OpenVT. Cada classe contém 4 imagens semelhantes de um mesmo objeto, com diferentes condições de visualização. Comumente, aplicações CBIR são testadas com um conjunto de imagens teste diferente do conjunto de treinamento. Entretanto, este trabalho opta por utilizar imagens de teste contidas no conjunto de treinamento – semelhante a Nister e Stewenius (2006), com a finalidade de mensurar a acurácia também para consultas de imagens iguais. Cada imagem do conjunto de imagens é utilizada como imagem de teste, ou seja, para um conjunto de 100 imagens, cada uma das 100 imagens é utilizada como teste, resultando em 100 consultas. Em 100% das consultas, nos testes realizados, imagens iguais são retornadas no topo do escore, o qual foi chamado de *TOP-0*, significando que a aplicação não falha em retornar imagens iguais. O *TOP-1* representa a porcentagem da segunda

[1] Disponível em <https://archive.org/details/ukbench>

posição do escore, sendo a primeira imagem semelhante e não-igual à imagem de consulta. Este tem a finalidade de mensurar a capacidade da aplicação em retornar a imagem mais semelhante e não-igual à imagem de consulta. O *TOP-4*, utilizado por Nister e Stewenius (2006), representa a porcentagem de as quatro imagens semelhantes à imagem de consulta estarem entre as quadro primeiras posições do escore, buscando mensurar a capacidade da aplicação em retornar todas as imagens semelhantes à imagem de consulta. Abaixo segue um pseudocódigo do cálculo da acurácia da aplicação.

```
para cada imagem_semelhante faca
    se imagem_semelhante == escore[0] entao
        TOP_0 += 1
    fim-se
    se imagem_semelhante em escore[:2] entao
        TOP_1 += 1
    fim-se
    se imagem_semelhante em escore[:4] entao
        TOP_4 += 1
    fim-se
fim-para

TOP_0_PERCENT = TOP_0/quantidade_imagens*100
TOP_1_PERCENT = TOP_1/quantidade_imagens*100*2
TOP_4_PERCENT = TOP_4/quantidade_imagens*100*4
```

Na seção 5.1 é apresentada uma análise entre os métodos de comparação entre histogramas; na seção 5.2 são apresentados resultados relativos à comparação da utilização da OpenVT com algoritmos de extração de descritores livres e patenteados; na seção 5.3 são apresentados resultados relativos à comparação entre a OpenVT e Redes Neurais Convolucionais (CNNs); na seção 5.4 é apresentada a comparação entre os resultados obtidos pela OpenVT e os resultados obtidos pelo método original apresentado por Nister e Stewenius (2006); por fim, na seção 5.5 é discorrido sobre a utilização de memória pela OpenVT.

5.1 Métodos de comparação entre histogramas

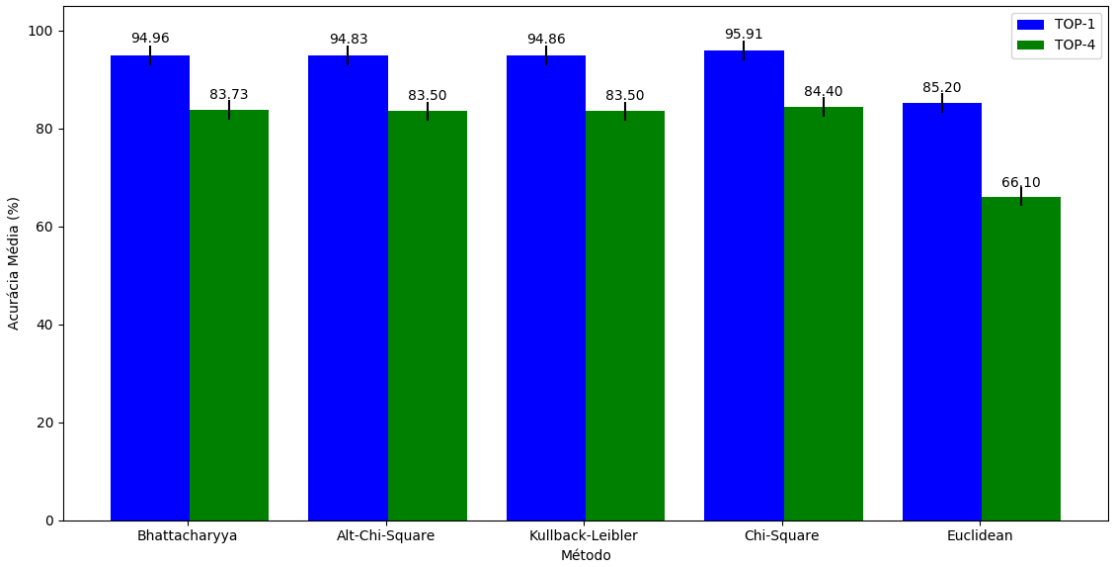
Diante inúmeros métodos para se calcular a similaridade entre histogramas, foram executados testes com a finalidade de definir o método principal utilizado pela aplicação proposta neste trabalho. Os métodos selecionados foram: distância Bhattacharyya; distância Chi-Square e sua versão alternativa; e distância Kullback-Leibler, estas providas pelo OpenCV[2]. Ainda, a distância Euclideana, provida pela biblioteca Scipy[3] do Python,

[2] Disponível em <https://docs.opencv.org/3.0-beta/doc/tutorials/imgproc/histograms/histogram_comparison/histogram_comparison.html>

[3] Disponível em <https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.spatial.distance.euclidean.html>

também foi testada. Para os testes, foi utilizado o algoritmo de extração de descritores ORB, para 100 imagens, com 10 mil folhas, 500 imagens, com 80 mil folhas, e 1.400 imagens, com 160 mil folhas, sendo as configurações que obtiveram os melhores resultados para cada conjunto de imagens.

Figura 20 – Comparação entre os métodos para cálculo de similaridade entre histogramas.



Fonte: Elaborado pelo autor.

A média das porcentagens *TOP-1* e *TOP-4* relativas aos resultados da execução da aplicação para 100, 500 e 1.400 imagens foram calculadas e plotadas em um gráfico, o qual é mostrado na Figura 20. O método de comparação de histogramas escolhido foi a distância Chi-Square, a qual obteve a maior porcentagem média. As distâncias Bhattacharyya, Chi-Square alternativa e Kullback-Leibler obtiveram resultados muito semelhantes, porém um pouco abaixo do método escolhido. A distância Euclidiana obteve os piores resultados. Embora todos os métodos ainda possam ser escolhidos por futuros utilizadores, aconselha-se utilizar o método principal escolhido.

5.2 OpenVT com algoritmos livres vs OpenVT com algoritmos patenteados

Para verificar o desempenho de métodos livres, foi feita uma comparação entre a utilização da OpenVT com algoritmos de extração de descritores livres e patenteados. Os testes foram executados para 1.400 imagens, testando cada algoritmo com 10 mil, 100 mil, 160 mil e 270 mil folhas. Com isso, foi possível avaliar qual a relação entre tempo de construção da VocabTree, acurácia, quantidade de folhas e quantidade de descritores.

A Tabela 2 mostra os resultados obtidos com a execução de seis algoritmos de extração de descritores, sendo dois destes patenteados. Ao todo, dezoito métodos foram

testados. Todos os métodos foram executados considerando 2 níveis (sem contar as folhas) para o cálculo da pontuação, o qual foi constatado como melhor estratégia em testes preliminares. A primeira coluna, **ME**, significa o nome dos métodos, com a finalidade de facilitar citações no texto. A coluna **ALG** refere-se ao nome do algoritmo extrator de descritores. Os algoritmos patenteados são assinalados com um "*****". A coluna **VT** refere-se à quantidade de níveis (L) e ao grau da árvore (k), resultando na quantidade de folhas da VocabTree, dada por k^L. A coluna **DF** reflete a proporção entre a quantidade de descritores e a quantidade de folhas da VocabTree. **TC** refere-se ao tempo de construção da VocabTree. Por fim, **TOP-1** e **TOP-4** referem-se à acurácia do método.

Tabela 2 – OpenVT com algoritmos livres vs OpenVT com algoritmos patenteados.

ME	**ALG**	**VT**	**DF**	**TC**	**TOP-1 (%)**	**TOP-4 (%)**
A	KAZE	$20^4 = 160K$	980K/160K=6,2	33min	95,4	81,2
B	*SIFT	$20^4 = 160K$	800K/160K=5	40min	94,9	81,4
C	KAZE	$23^4 = 270K$	980K/270K=3,6	37min	94,5	79,1
D	*SIFT	$23^4 = 270K$	800K/270K=2,9	40min	94,2	78,5
E	ORB	$20^4 = 160K$	1200K/160K=7,5	1h12min	93,0	77,8
F	ORB	$23^4 = 270K$	1200K/270K=4,4	1h06min	92,9	76,2
G	*SIFT	$10^5 = 100K$	800K/100K=8	30min	92,1	74,6
H	*SURF	$20^4 = 160K$	1000K/160K=6,2	32min	91,5	73,7
I	KAZE	$10^5 = 100K$	980K/100K=9,8	25min	91,0	72,1
J	KAZE	$10^4 = 10K$	980K/10K=98	17min	90,7	71,9
K	*SURF	$23^4 = 270K$	1000K/270K=3,7	43min	90,8	71,7
L	ORB	$10^5 = 100K$	1200K/100K=12	58min	90,7	71,7
M	*SIFT	$10^4 = 10K$	800K/10K=80	21min	90,2	71,1
N	ORB	$10^4 = 10K$	1200K/10K=120	1h17min	86,9	65,7
O	*SURF	$10^4 = 10K$	1000K/10K=100	18min	85,9	64,0
P	*SURF	$10^5 = 100K$	1000K/100K=10	26min	86,2	63,1
Q	AKAZE	$20^4 = 160K$	1000K/160K=6,2	1h42min	67,8	43,2
R	BRISK	$20^4 = 160K$	700K/160K=4,4	2h10min	56,1	31,8

Fonte: Elaborado pelo autor.

A ordem utilizada na Tabela 2 foi formada pela média simples entre o *TOP-1* e o *TOP-4*. O algoritmo de extração de descritores que se saiu melhor nos testes foi o KAZE, com 160 mil folhas (método **A**), seguido pelo algoritmo patenteado SIFT, também com 160 mil folhas (método **B**). Dos 3 melhores algoritmos, 2 destes são algoritmos livres (KAZE e ORB), sendo um do tipo *float* e outro do tipo binário. Um padrão se seguiu durante os testes, onde um *TOP-1* alto é seguido por um *TOP-4* alto também. Os algoritmos AKAZE e BRISK obtiveram resultados muito abaixo dos demais, sendo adicionados na Tabela 2 apenas para conhecimento.

DF é um valor estatístico que permite aproximar a quantidade ótima de folhas para uma dada quantidade de descritores, dado por

$$DF = \frac{nd}{k^L} \tag{5.1}$$

onde k é o grau da árvore, L a quantidade de níveis e nd a quantidade de descritores extraídos do conjunto de imagens de treinamento. Intuitivamente, espera-se que aumentar a quantidade de folhas, elevará a acurácia. Entretanto, os resultados mostram que, se a quantidade de folhas aproximar-se muito da quantidade de descritores, pode resultar em um sobreajuste (*overfitting*), afetando a acurácia da aplicação. Da mesma forma, se **DF** for um valor muito grande, pode não haver uma quantização suficiente. As melhores colocações de cada algoritmo, sendo o método **A** do KAZE, **B** do SIFT, **E** do ORB e **H** do SURF, mostram que a proporção entre quantidade de folhas e quantidade de descritores (**DF**) deve ficar entre 5 e 7,5. Com isso, é possível aproximar um valor ótimo, calculando a média dos melhores resultados, chegando a

$$\frac{nd}{k^L} \simeq 6 \tag{5.2}$$

Sabendo disso, não há a necessidade de despender tempo e esforço computacional para descobrir as melhores configurações para a OpenVT. Basta utilizar a fórmula da Equação 5.2. É importante frisar que a proporção **DF** só foi testada com a OpenVT, necessitando de mais estudos sobre sua veracidade para os demais trabalhos utilizando VocabTrees e outros conjuntos de imagens. Não estando no escopo deste trabalho, questões relativas às demais aplicações utilizando árvore de vocabulário não serão abordadas.

Na coluna **TC**, estão os tempos de construção da VocabTree para cada método. Quanto mais folhas, maior o tempo necessário para construir a VocabTree. Comparando o método **C** que utiliza o KAZE com 270 mil folhas com o método **J** que utiliza o KAZE com 10 mil folhas, fica clara a diferença de tempo necessário para a construção da VocabTree. Entretanto, comparando os algoritmos do tipo binário, que utilizam o *K-Majority* como algoritmo de *clustering*, e os algoritmos do tipo *float*, que utilizam o *K-Means*, é notável a diferença entre os seus **TC**. O *K-Majority* que, por definição, deveria ser mais rápido que o *K-Means*, está levando, em média, o dobro do tempo para *clusterizar* seus descritores. Isso se deve a sua implementação. Embora partes da implementação do *K-Majority* terem sido feitas em Cython e C++, ainda é necessário concentrar um pouco mais de esforços para igualá-lo às implementações mais baixo nível, como as dos módulos otimizados do Python, Numpy, Scipy e Scikit Learn.

5.3 OpenVT vs Redes Neurais Convolucionais

A melhor forma de avaliar o desempenho da OpenVT é colocá-la à prova contra o estado da arte em reconhecimento e classificação de imagens. O estado da arte em reconhecimento de imagens são as redes neurais convolucionais (CNN) (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). CNNs baseiam-se no córtex visual dos animais e utilizam matrizes de convolução para codificar mapas de características. Diferente da VocabTree, que quantiza características de imagens semelhantes, as CNNs classificam imagens em

classes predefinidas antes do seu treinamento. Ou seja, VocabTrees e CNNs são métodos diferentes, com aplicações diferentes e resultados diferentes. Entretanto, com certas modificações, é possível utilizar as CNNs para extrair descritores globais de imagens.

O Keras[4], biblioteca de aprendizagem profunda (*deep learning*) do Python, provê modelos pré-treinados[5] que são utilizados para predição e extração de descritores. Estes modelos são treinados com milhões de imagens[6], as quais estão divididas em milhares de classes. Como os modelos pré-treinados do Keras extraem apenas um descritor global por imagem, não é possível utilizar a VocabTree.

Em seu trabalho, Franky (2018) utiliza modelos pré-treinados do Keras para extrair descritores globais de imagens de gatos e cachorros. Então, utilizando o *K-Means*, os descritores são *clusterizados* em dois grupos, sendo um grupo com descritores das imagens de gatos e outro de cachorros. Com algumas modificações, foi possível expandir o método utilizado por Franky (2018) para *clusterizar* descritores de 1.400 imagens em 350 grupos (4 imagens semelhantes por grupo). Para agilizar o processo, os descritores foram *clusterizados* em um *K-Means* hierárquico, utilizando uma estrutura em árvore, onde cada folha da árvore representa um grupo de imagens semelhantes. Recebendo uma imagem de consulta, seu descritor é propagado pela árvore, comparando-o com os centroides de cada nodo, até chegar em uma das folhas que representam seu grupo. Após isso, os descritores das imagens desse grupo são comparados diretamente com o descritor da imagem de consulta, utilizando a distância Euclidiana. Por fim, as imagens são ordenadas por semelhança de seus descritores e retornadas.

A Tabela 3 mostra os resultados obtidos pelos dois melhores algoritmos livres da OpenVT (KAZE e ORB), apresentados na Tabela 2, comparando-os com a execução da aplicação feita utilizando os modelos pré-treinados do Keras. Os testes foram feitos para 1.400 imagens (mesmo conjunto de imagens utilizado na seção 5.2).

Ao todo, oito modelos pré-treinados do Keras foram utilizados (assinalados com "*") para os testes. Os métodos (coluna **ME**) foram ordenados pela média simples entre *TOP-1* e *TOP-4*. Os 3 melhores métodos são modelos pré-treinados do Keras, entretanto, considerando apenas o *TOP-1*, a OpenVT, utilizando o KAZE, ficou em segundo lugar. Além disso, a OpenVT mostrou-se melhor que 5 dos 8 modelos pré-treinados do Keras, constatando que a OpenVT está no mesmo nível que o estado da arte em reconhecimentos de imagens.

A utilização de CNNs para extração de descritores ainda é pouco estudada, sendo mais utilizada para predição de classes. Embora os modelos pré-treinados do Keras tenham se saído melhor que a OpenVT nos testes realizados, não há garantias de que estes modelos escalem eficientemente para grandes quantidades de imagens, a qual é uma das principais característica da VocabTree. Além disso, não há garantias de que as CNNs

[4] Disponível em <https://keras.io/>
[5] Disponível em <https://keras.io/applications/>
[6] Disponível em <http://www.image-net.org/>

Tabela 3 – OpenVT vs modelos pré-treinados do Keras.

ME	ALG	VT	DF	TOP-1 (%)	TOP-4 (%)
A	*DENSENET201	–	–	96,1	87,9
B	*DENSENET169	–	–	95,2	83,8
C	*MOBILENET	–	–	95,2	83,5
D	KAZE	$20^4 = 160K$	980K/160K=6,2	95,4	81,2
E	*RESNET50	–	–	93,9	81,9
F	*XCEPTION	–	–	93,6	80,6
G	KAZE	$23^4 = 270K$	980K/270K=3,6	94,5	79,1
H	*DENSENET121	–	–	92,0	79,1
I	ORB	$20^4 = 160K$	1200K/160K=7,5	93,0	77,8
J	*INCEPTIONV3	–	–	92,9	76,8
K	ORB	$23^4 = 270K$	1200K/270K=4,4	92,9	76,2
L	KAZE	$10^4 = 10K$	980K/10K=98	91,0	72,1
M	KAZE	$10^5 = 100K$	980K/100K=9,8	90,7	71,9
N	ORB	$10^5 = 100K$	1200K/100K=12	90,7	71,7
O	*INCEPTIONRESNETV2	–	–	89,3	71,5
P	ORB	$10^4 = 10K$	1200K/10K=120	86,9	65,7

Fonte: Elaborado pelo autor.

funcionarão bem para um conjunto de imagens muito diferentes das imagens utilizadas para o seu treinamento. Como CNNs apenas foram utilizadas como forma de comparação, não cabe a este trabalho analisar seus limites.

5.4 OpenVT vs método original

Para avaliar se a OpenVT manteve um desempenho satisfatório, comparada ao método original apresentado por Nister e Stewenius (2006), foram comparados os resultados de Nister e Stewenius (2006) para 1.400 imagens com os obtidos pela OpenVT. Os resultados utilizados da OpenVT são os mesmo apresentados na seção 5.2 e seção 5.3. Como não foram especificadas quais imagens do conjunto de imagens *UKBench*[7] foram utilizadas por Nister e Stewenius (2006), espera-se que os resultados não sejam totalmente precisos. Entretanto, os resultados dos dois métodos para a mesma quantidade de imagens já é um ótimo parâmetro de comparação.

A Tabela 4 apresenta os resultados dos três métodos pertencentes ao trabalho original, estando assinalados com "*". Como Nister e Stewenius (2006) utilizaram apenas o *TOP-4* em seus testes, apenas estes são comparados. Os métodos utilizados no trabalho original foram testados com inúmeras configurações, então, apenas os melhores resultados para suas respectivas configurações (grau e quantidade de níveis) são considerados. Os métodos considerados são: k igual a 10 e L igual a 6, resultando em 1 milhão de folhas (método ***A**); k igual a 10 mil e L igual a 1, resultando em 10 mil folhas (método ***B**); e k igual a 10 e L igual a 4, resultando em 10 mil folhas (método ***C**).

[7] Disponível em <https://archive.org/details/ukbench>

Tabela 4 – OpenVT vs método original.

ME	**ALG**	**VT**	**DF**	**TOP-4 (%)**
*A	SIFT	$10^6 = 1M$	1400K/1000K$\simeq 1,4$	90,6
*B	SIFT	10K[1] $= 10K$	1400K/10K$\simeq$ 140	86,0
*C	SIFT	$10^4 = 10K$	1400K/10K$\simeq$ 140	81,3
D	KAZE	$20^4 = 160K$	980K/160K= 6,2	81,2
E	KAZE	$23^4 = 270K$	980K/270K= 3,6	79,1
F	ORB	$20^4 = 160K$	1200K/160K= 7,5	77,8
G	ORB	$23^4 = 270K$	1200K/270K= 4,4	76,2

Fonte: Elaborado pelo autor.

Em seu trabalho, Nister e Stewenius (2006) utilizaram 1 mil descritores por imagem. Ou seja, aproximadamente, para 1.400 imagens, 1,4 milhão de descritores foram utilizados. O método ***A** mostrou-se o melhor, entretanto, utilizando a Equação 5.1, é possível verificar que sua proporção entre quantidade de folhas e quantidade de descritores (**DF**) é aproximadamente 1,4, não condizendo com a proporção **DF** ótima encontrada na seção 5.2. Além disso, os métodos ***B** e ***C** mesmo com uma menor quantidade de folhas, obtiveram melhores resultados que a OpenVT. Mediante estes fatos, constatou-se que a proporção **DF** ótima só é válida para a OpenVT, não sendo sempre verdade para outros trabalhos utilizando árvore de vocabulário.

Embora os resultados do trabalho original se mostrarem superiores aos da OpenVT, a proximidade dos resultados da OpenVT constatam sua eficiência. Seu *TOP-1* acima dos 95% e seu *TOP-4* acima dos 80% tornam a OpenVT uma ótima alternativa livre.

5.5 Análise da utilização de memória

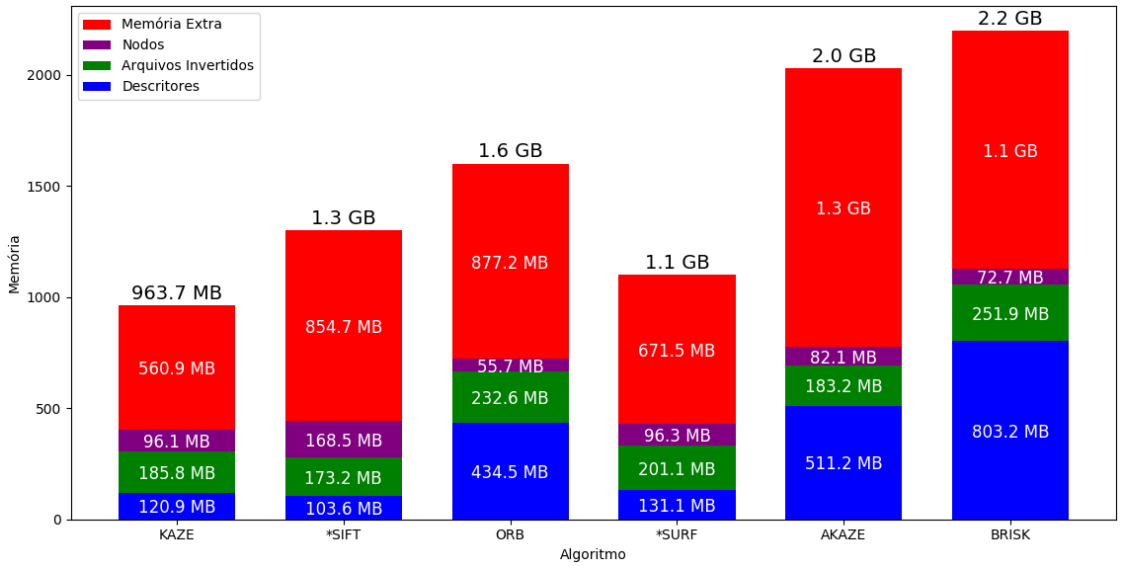
Desde o início do desenvolvimento deste trabalho, a memória RAM foi tratada como um recurso crítico. Um dos principais motivos pelos quais o Python foi escolhido como linguagem de programação para o desenvolvimento deste trabalho é sua abstração do gerenciamento de memória. Ou seja, o Python é o único responsável por alocar e desalocar memória, liberando completamente o desenvolvedor para preocupar-se apenas em resolver seu problema. Entretanto, isto retira completamente, do desenvolvedor, o controle sobre este recurso.

Durante o desenvolvimento do trabalho, a memória RAM foi analisada utilizando módulos do Python, como: *sys*, *psutils* e *os*, para acessar variáveis de sistema; e *memory_profiler*[8], o qual analisa a memória utilizada em cada linha do código.

Na Figura 21, é mostrado o gráfico de utilização de memória RAM para cada algoritmo de extração de descritores utilizado com a OpenVT neste trabalho. Os dados foram obtidos a partir de execuções para 1.400 imagens e 160 mil folhas. As cores do gráfico representam: em azul, a memória necessária para alocar o vetor de descritores; em verde,

[8] Disponível em <https://pypi.org/project/memory_profiler/>

Figura 21 – Uso da memória RAM para 1.400 imagens com 160 mil folhas.



Fonte: Elaborado pelo autor.

a memória necessária para alocar os arquivos invertidos; em roxo, a memória necessária para alocar os nodos da VocabTree; e, em vermelho, está representada a memória extra utilizada pelo Python.

Analisando o gráfico, é possível constatar três principais características: é necessário mais memória para armazenar o vetor de descritores para algoritmos de extração de descritores binários; é necessário menos memória para armazenar os nodos da VocabTree para algoritmos de extração de descritores binários; e mais da metade da memória utilizada pela OpenVT é lixo de memória ocasionado pelas limitações do coletor de lixo do Python.

O fato de ser necessária mais memória para descritores binários se deve à sua implementação. Descritores binários são implementados como *strings* (*StrDescriptor*), enquanto que descritores numéricos são implementados como vetores do Numpy (*ArrayDescriptor*), sendo o Python melhor em reutilizar memória para valores numéricos. Entretanto, a OpenVT necessita de menos memória para armazenar os nodos da VocabTree para descritores binários. Especula-se que é necessária mais memória para armazenar os centroides numéricos do que os centroides binários. Ou seja, uma lista pequena de descritores binários utiliza menos memória do que uma lista pequena de descritores numéricos, porém, quando estas listas tendem a tamanhos muito grandes, o Python reutiliza melhor a memória da lista de descritores numéricos. Como este tipo de análise não encontra-se nos objetivos deste trabalho, será feita em trabalhos futuros.

A memória extra, representada em vermelho na Figura 21, resulta de questões ainda não resolvidas no desenvolvimento do coletor de lixo do Python. Segundo Gorelick

e Ozsvald (2014), o coletor de lixo do Python é implementado como um grafo e utiliza contadores de referências (ponteiros). Com isso, o coletor de lixo desaloca endereços de memória que não estão sendo mais utilizados e estão com seu contador de referência zerado. Contudo, mesmo na versão 3.5.x do Python, os desenvolvedores da linguagem ainda não encontraram uma solução suficientemente boa para ciclos no grafo do coletor de lixo. Mediante isso, a filosofia do Python é: na dúvida, não desalocar. Ou seja, para evitar desalocar memória que pode ainda estar sendo utilizada, o coletor de lixo do Python prefere manter memória alocada. Além disso, o fato do OpenCV ser escrito em C++ pode contribuir para esta memória extra, como foi abordado na seção 4.2. Esta e outras questões que tangem o escopo deste trabalho serão estudas em trabalhos futuros.

6 CONSIDERAÇÕES FINAIS

Buscando fomentar novos trabalhos e a criação de novas aplicações CBIR, este trabalho teve como objetivo geral a construção de um motor de busca por imagens semelhantes escalável e de código aberto, sendo robusto às diferentes condições que as imagens podem se encontrar. Para alcançar este objetivo, o método proposto por Nister e Stewenius (2006) (árvore de vocabulário), junto a métodos e algoritmos livres, foram utilizados, resultando em uma aplicação de código livre e desempenho satisfatório.

A partir do mapeamento sistemático da literatura, contatou-se a notoriedade do trabalho apresentando por Nister e Stewenius (2006), sendo citado por mais de 2 mil outros trabalho, somente na base de dados *Scopus*[1] e que árvores de vocabulário são estruturas adaptáveis que proporcionam uma grande liberdade de escolha sobre algoritmos de extração de descritores e de *clustering*. A constatação da efetividade do método, somada a imensa preferência pelo algoritmo patenteado de extração de descritores SIFT, afirmaram a importância de um trabalho que busque utilizar algoritmos livres junto à árvore de vocabulário.

Durante o desenvolvimento da OpenVT, analisaram-se questões relativas à utilização de memória, tempo de execução e melhores tecnologias livres. Diante recursos computacionais moderados e limitações de tecnologias utilizadas, renúncias foram feitas. Para restringir a utilização de memória RAM, ganhos em tempo de execução foram sacrificados. Dos quatro objetivos específicos, apenas escalabilidade não pôde ser testada. Devido às limitações do coletor de lixo do Python e a impossibilidade do desenvolvedor desalocar memória manualmente, a OpenVT acaba por usar mais do que o dobro de memória RAM do que realmente deveria. Este desperdício de memória somado aos recursos computacionais limitados impossibilitaram a utilização da OpenVT para elevadas quantidades de imagens.

O uso eficiente de memória e a redução do tempo de execução foram os principais desafios encontrados. A escolha do Python se deu pelas suas características como linguagem de programação de alto nível, proporcionando uma maior produtividade ao desenvolvedor. Entretanto, o preço por esta abstração foi pago em memória. Embora o Python permita implementar módulos eficientes em C++ para agilizar tarefas, implementá-los torna-se uma tarefa árdua, sendo mais complexa do que codificar em C++ puro. Tarefas que necessitaram de um maior desempenho foram escritas em Cython, reduzindo o tempo de execução da OpenVT sem desperdiçar memória.

A OpenVT foi desenvolvida para ser o mais flexível possível, buscando simplificar testes realizados por futuros utilizadores. Todas as questões relativas à aplicação estão sujeitas ao utilizador em seu arquivo de configuração (*settings.py*). Aproveitando a tipagem dinâmica do Python, a aplicação funciona para diversos algoritmos de extração de descritores, onde ela escolhe qual algoritmo de *clustering* e qual método para o cálculo de

[1] Disponível em <https://www.scopus.com/search/form.uri?display=basic>

distância utilizar, baseando-se no tipo dos descritores (*StrDescriptor* ou *ArrayDescriptor*).

Para avaliar a qualidade da OpenVT, foram realizados testes, comparando-a com algoritmos de extração de descritores patenteados, CNNs e os resultados obtidos por Nister e Stewenius (2006). A OpenVT com algoritmos de extração de descritores livres mostrou-se melhor do que com algoritmos patenteados, constatando que métodos livres não só podem se igualar aos patenteados, como podem os superar. Embora tenha obtido resultados inferiores às CNNs, a OpenVT, utilizando o KAZE, obteve resultados superiores a cinco dos oito modelos pré-treinados do Keras. Comparada aos resultados apresentados por Nister e Stewenius (2006), a OpenVT apresentou uma diferença de quase 10% entre seus melhores resultados, o que implica que ainda há possibilidade de aprimoramento de sua acurácia. Obtendo mais de 95% em seu *TOP-1* e mais de 80% em seu *TOP-4*, a OpenVT provou estar no mesmo nível que outras aplicações semelhantes.

A partir dos resultados da seção 5.2, constatou-se uma relação entre a acurácia da aplicação e sua proporção entre quantidade de descritores e de folhas (**DF**). Embora ainda seja necessário uma validação com diferentes conjuntos de imagens, esta constatação auxilia futuros utilizadores a encontrar as melhores configurações para a VocabTree da OpenVT.

Dentre as principais contribuições desde trabalho, estão: verificação da possibilidade de utilização de apenas métodos e algoritmos livres, se igualando e, inclusive, superando algoritmos patenteados; aplicação flexível e de código aberto para auxiliar futuros desenvolvedores; aplicação web para busca de imagens semelhantes; e verificação da possibilidade de desenvolvimento de uma aplicação de alto desempenho escrita em uma linguagem de programação de alto nível e interpretada.

Durante o desenvolvimento deste trabalho, várias questões técnicas e arquiteturais surgiram. Para trabalhos futuros, pretende-se: verificar a validade da proporção **DF** para outros trabalhos utilizando VocabTree; verificar a escalabilidade da OpenVT; implementar adição de imagens durante a fase de busca; testar a adição de informações adicionais para cada imagem, como cores, classes e *tags*; estudar formas de otimizar a gerência de memória do Python; estudar formas de otimizar o tempo de execução do *K-Majority*; e estudar como codificar descritores binários para utilizar menos memória.

REFERÊNCIAS

ABBAS, O. A. Comparisons between data clustering algorithms. **The International Arab Journal of Information Technology**, v. 5, Julho 2008. Citado na página 38.

ALCANTARILLA, P. F.; BARTOLI, A.; DAVISON, A. J. Kaze features. **European Conference on Computer Vision**, p. 214–227, Outubro 2012. Citado 3 vezes nas páginas 34, 51 e 54.

ALCANTARILLA, P. F.; BARTOLI, A.; DAVISON, A. J. Fast explicit diffusion for accelerated features in nonlinear scale spaces. **British Machine Vision Conference**, Setembro 2013. Citado na página 54.

ANDERSON, B. J. et al. Adapting k-medians to generate normalized cluster centers. In: **SDM**. [S.l.: s.n.], 2006. p. 165–175. Citado na página 38.

BANLUPHOLSAKUL, K.; IEAMSAARD, J.; MUNEESAWANG, P. Re-ranking approach to mobile landmark recognition. **International Computer Science and Engineering Conference**, n. 6978203, p. 251–254, Julho 2014. Citado 3 vezes nas páginas 49, 50 e 51.

BAY, H.; TUYTELAARS, T.; GOOL, L. V. Surf: Speeded up robust features. In: **European Conference on Computer Vision**. [S.l.: s.n.], 2006. p. 404–417. Citado na página 34.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV: Computer Vision with the OpenCV Library**. 1a edição. ed. [S.l.]: O'Reilly, 2008. Citado na página 34.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library**. 1a edição. ed. [S.l.]: O'Reilly, 2016. Citado na página 36.

CALONDER, M. et al. Brief: Binary robust independent elementary features. **European Conference on Computer Vision**, v. 6314, p. 778–792, Setembro 2010. Citado na página 34.

CAO, D.; YANG, B. An improved k-medoids clustering algorithm. In: **Computer and Automation Engineering**. [S.l.: s.n.], 2010. Citado na página 38.

CATENA, M.; MACDONALD, C.; OUNIS, I. On inverted index compression for search engine efficiency. **European Conference on Information Retrieval**, v. 8416, p. 359–371, Abril 2014. Citado na página 39.

CHEN, C. **Handbook of Pattern Recognition and Computer Vision**. 5a edição. ed. [S.l.]: World Scientific, 2015. Citado na página 32.

CISCO. **Cisco Visual Networking Index: Forecast and Methodology, 2016–2021**. 2017. Disponível em: <https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf>. Acesso: 3 nov. 2017, 22:12:00. Citado na página 25.

DEAN, J.; GHEMAWAT, S. **MapReduce: Simplified Data Processing on Large Clusters**. 2004. Disponível em: <https://static.googleusercontent.com/media/research.google.com/pt-BR//archive/mapreduce-osdi04.pdf>. Acesso em: 2 nov. 2017, 19:52:00. Citado na página 49.

FRANKY. **Using Keras Pre-trained Models for Feature Extraction in Image Clustering**. 2018. Disponível em: <https://medium.com/@franky07724_57962/using-keras-pre-trained-models-for-feature-extraction-in-image-clustering-a142c6cdf5b1>. Acesso em: 07 jun. 2018, 12:50:00. Citado 2 vezes nas páginas 32 e 68.

GOMES, H. M. et al. Mapreduce vocabulary tree: An approach for large scale image indexing and search in the cloud. **IEEE International Conference on Multimedia Big Data**, n. 7545016, p. 170–173, Abril 2016. Citado 3 vezes nas páginas 49, 50 e 51.

GORELICK, M.; OZSVALD, I. **High Performance Python: Practical Performant Programming for Humans**. 1a edição. ed. [S.l.]: O'Reilly Media, 2014. Citado 3 vezes nas páginas 37, 56 e 72.

GRANA, C.; BORGHESANI, D.; CUCCHIARA, R. A fast approach for integrating orb descriptors in the bag of words model. **The International Society for Optical Engineering**, v. 8667, Fevereiro 2013. Citado 4 vezes nas páginas 38, 39, 50 e 56.

JIANG, M. et al. Computer-aided diagnosis of mammographic masses using scalable image retrieval. **IEEE Transactions on Biomedical Engineering**, v. 62, n. 6937176, p. 783–792, Fevereiro 2015. Citado 3 vezes nas páginas 49, 50 e 51.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. **Advances in Neural Information Processing Systems**, 2012. Citado na página 67.

LEUTENEGGER, S.; CHLI, M.; SIEGWART, R. Y. Brisk: Binary robust invariant scalable keypoints. **International Conference on Computer Vision**, n. 6126542, p. 2548–2555, Novembro 2011. Citado 3 vezes nas páginas 34, 51 e 54.

LISIN, D. A. et al. Combining local and global image features for object class recognition. **IEEE Computer Society Conference on Computer Vision and Pattern Recognition**, Setembro 2005. Citado na página 32.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. **International Journal of Computer Vision**, v. 60, p. 91–110, Novembro 2004. Citado 4 vezes nas páginas 33, 34, 50 e 51.

MACQUEEN, J. B. Some methods for classification and analysis of multivariate observations. **Berkeley Symposium on Mathematical Statistics and Probability**, p. 281–297, 1967. Citado na página 38.

MADUSHAN, D. **Introduction to K-means Clustering**. 2017. Disponível em: <https://medium.com/@dilekamadushan/introduction-to-k-means-clustering-7c0ebc997e00>. Acesso em: 21 mai. 2018, 11:48:00. Citado na página 38.

MAHAPATRA, A. K.; BISWAS, S. Inverted indexes: Types and techniques. **International Journal of Computer Science**, p. 292–384, 2011. Citado na página 40.

MALLICK, S. **Blob Detection Using OpenCV (Python, C++)**. 2015. Disponível em: <https://www.learnopencv.com/blob-detection-using-opencv-python-c/>. Acesso: 10 out. 2017, 10:06:00. Citado na página 32.

MARENGONI, M.; STRINGHINI, D. **Tutorial: Introdução à Visão Computacional usando OpenCV**. 2009. Disponível em: <https://ai2-s2-pdfs.s3.amazonaws.com/a0a6/301a239519f117ca6c0398d5230ff15c67ff.pdf>. Acesso em: 4 set. 2017, 14:33:00. Citado 2 vezes nas páginas 31 e 35.

MARS, D. E. **Multi-View Vocabulary Trees for Mobile 3D Visual Search**. Dissertação (Mestrado) — KTH Royal Institute of Technology, Outubro 2014. Disponível em: <https://riunet.upv.es/bitstream/handle/10251/52510/David%2BEbr%C3%AD%2BMars%2BKTH%2BMaster%2BThesis%2BFinal.pdf?sequence=1>. Citado na página 41.

MATAS, J. et al. Robust wide baseline stereo from maximally stable extremal regions. **Image and Vision Computing**, v. 22, p. 761–767, Setembro 2004. Citado na página 33.

MORDVINTSEV, A.; K., A. **Introduction to OpenCV-Python Tutorials**. 2013. Disponível em: <http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html>. Acesso: 25 out. 2017, 20:08:00. Citado na página 37.

NISTER, D.; STEWENIUS, H. Scalable recognition with a vocabulary tree. **IEEE Computer Society Conference on Computer Vision and Pattern Recognition**, v. 2, n. 1641018, p. 2161–2168, Junho 2006. Citado 19 vezes nas páginas 11, 13, 26, 27, 28, 29, 31, 33, 34, 40, 50, 53, 54, 63, 64, 69, 70, 73 e 74.

NIXON, M.; AGUADO, A. **Feature Extraction Image Processing for Computer Vision**. 3a edição. ed. [S.l.]: Elsevier, 2012. Citado na página 31.

PATTANIYIL, N.; ZANIBBI, R. Combining tf-idf text retrieval with an inverted index over symbol pairs in math expressions. In: **NTCIR**. [S.l.: s.n.], 2014. Citado na página 39.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: **International Conference on Evaluation and Assessment in Software Engineering**. [S.l.: s.n.], 2008. p. 68–77. Citado na página 47.

PI, Y. et al. **Theory of Cognitive Pattern Recognition**. 2008. Disponível em: <http://cdn.intechopen.com/pdfs/5795.pdf>. Acesso em: 15 out. 2017, 20:32:00. Citado na página 32.

RIEMENSCHNEIDER, H.; DONOSER, M.; BISCHOF, H. **Finding Stable Extremal Region Boundaries**. 2009. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.8085&rep=rep1&type=pdf>. Acesso em: 6 out. 2017, 15:22:00. Citado na página 32.

ROSTEN, E. et al. Faster and better: a machine learning approach to corner detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 32, n. 4674368, p. 105–119, 2010. Citado 2 vezes nas páginas 33 e 34.

RUBLEE, E. et al. Orb: an efficient alternative to sift or surf. **International Conference on Computer Vision**, n. 6126544, p. 2564–2571, Novembro 2011. Citado 5 vezes nas páginas 34, 50, 51, 54 e 55.

SHIWEI, H. et al. Online real-time image retrieval based on large scale vocabulary tree. **IEEE International Conference on Signal Processing**, n. 7877971, p. 953–956, Março 2017. Citado 4 vezes nas páginas 48, 49, 50 e 51.

SIVIC, J.; ZISSERMAN, A. Video google: A text retrieval approach to object matching in videos. **IEEE International Conference on Computer Vision**, v. 2, p. 1470–1477, Outubro 2003. Citado 3 vezes nas páginas 33, 34 e 40.

SMITH, K. **Cython: A Guide for Python Programmers**. 1a edição. ed. [S.l.]: O'Reilly Media, 2015. Citado 2 vezes nas páginas 37 e 57.

WANG, K. et al. Binary search path of vocabulary tree based finger vein image retrieval. **IAPR International Conference on Biometrics**, n. 7550056, Agosto 2016. Citado 2 vezes nas páginas 49 e 50.

YADAV, J.; SHARMA, M. A review of k-mean algorithm. **International Journal of Engineering Trends and Technology**, v. 4, Julho 2013. Citado na página 38.

ZHOU, W.; LI, H.; TIAN, Q. **Recent Advance in Content-based Image Retrieval: A Literature Survey**. 2017. Disponível em: <https://arxiv.org/pdf/1706.06064.pdf>. Acesso em: 23 out. 2017, 21:05:00. Citado 2 vezes nas páginas 25 e 26.

Apêndices

APÊNDICE A – BUSCA POR IMAGENS SEMELHANTES A UM VIOLÃO COM A OPENVT

Figura 22 – Busca por imagens semelhantes a um violão com a OpenVT.

Fonte: Elaborado pelo autor.

Captura de tela da execução da OpenVT, utilizando o KAZE com 10 mil folhas, para 100 imagens retiradas do conjunto de imagens *UKBench*[1].

[1] Disponível em <https://archive.org/details/ukbench>

APÊNDICE B – BUSCA POR IMAGENS SEMELHANTES À TORRE DE PISA COM A OPENVT

Figura 23 – Busca por imagens semelhantes à Torre de Pisa com a OpenVT.

Fonte: Elaborado pelo autor.

Captura de tela da execução da OpenVT, utilizando o ORB com 50 mil folhas, para 300 imagens retiradas do conjunto de imagens *Caltech256*[1].

[1] Disponível em <http://www.vision.caltech.edu/Image_Datasets/Caltech256/>

APÊNDICE C – BUSCA POR IMAGENS SEMELHANTES A UM GIRASSOL COM A OPENVT

Figura 24 – Busca por imagens semelhantes a um girassol com a OpenVT.

Fonte: Elaborado pelo autor.

Captura de tela da execução da OpenVT, utilizando o ORB com 50 mil folhas, para 300 imagens retiradas do conjunto de imagens *Caltech256*[1].

[1] Disponível em <http://www.vision.caltech.edu/Image_Datasets/Caltech256/>