UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENAÇÃO DE CIÊNCIA DA COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL BONIOLO

MÓDULO DE RECONHECIMENTO DE IMAGENS ANÔMALAS BASEADO EM MICROSSERVIÇOS

TRABALHO DE CONCLUSÃO DE CURSO

SANTA HELENA
2019

RAFAEL BONIOLO

MÓDULO DE RECONHECIMENTO DE IMAGENS ANÔMALAS BASEADO EM MICROSSERVIÇOS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Prof. Me. Anderson Brilhador
Universidade Tecnológica Federal do Paraná

SANTA HELENA
2019

TERMO DE APROVAÇÃO

"Módulo de Reconhecimento de Imagens Anômalas Baseado em Microsserviços "

por

"Rafael Boniolo"

Este Trabalho de Conclusão de Curso foi apresentado às 17h do dia 05 de Dezembro de 2019 na sala 28 como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação na Universidade Tecnológica Federal do Paraná - UTFPR - Câmpus Santa Helena. O(a) aluno(a) foi arguido pela Banca de Avaliação abaixo assinados. Após deliberação, a Banca de Avaliação considerou o trabalho APROVADO.

Prof. Me. Anderson Brilhador
(Presidente - UTFPR/Santa Helena)

Prof. Dr. Thiago França Naves
(Avaliador 1 - UTFPR/Santa Helena)

Prof. Me. Giuvane Conti
(Avaliador 2 - UTFPR/Santa Helena)

Dedico este trabalho aos meus familiares por todo apoio e carinho, e ao meu grande e eterno amigo Fernando, *In Memorian*.

AGRADECIMENTOS

A Deus acima de tudo, por me dar forças e bençãos para vencer os obstáculos e chegar até aqui.

Aos meus amigos por toda troca de conhecimento, experiência e risos.

A minha namorada e aos meus familiares por todo incentivo, apoio e carinho.

Aos meus professores por todo o conhecimento que recebi e agradeço em especial ao meu orientador e amigo Prof. Me. Anderson Brilhador.

Tudo posso naquele que me fortalece. (Filipenses 4:13).

RESUMO

BONIOLO, Rafael. Módulo de Reconhecimento de Imagens Anômalas Baseado em Microsserviços. 2019. 73 f. Trabalho de Conclusão de Curso – Curso de Bacharelado em Ciência da Computação, Universidade Tecnológica Federal do Paraná. Santa Helena, 2019.

Este trabalho apresenta o desenvolvimento de um módulo reconhecedor de imagens anômalas ao contexto da construção civil com base na arquitetura de microsserviços. Utilizam-se os classificadores KNN e OC-SVM juntamente com os descritores locais HOG, SIFT, SURF e ORB em diferentes dimensionalidades do espaço de característica utilizando o PCA. A base de dados foi construída por meio de *Web Crawler* o qual contempla imagens pertencentes a construção civil e também de outras classes anômalas. Para microsserviços, utilizou-se o *framework Spring* na construção dos serviços em níveis de *front-end*, *back-end* e gerenciamento. A melhor combinação entre descritor, classificador e PCA deu origem a um modelo de classificação a ser implantado em um serviço *web*. O descritor local HOG sem aplicação de PCA junto com o classificador KNN obtiveram o melhor desempenho, alcançando 96,2% de taxa na métrica precisão quando feito a tarefa de detecção de imagens anômalas, na qual o modelo de classificação resultante foi posteriormente embutido e disponibilizado para uso em uma aplicação web.

Palavras-chave: Visão computacional; Reconhecimento de padrões; Reconhecimento de imagens anômalas; Microsserviços.

ABSTRACT

BONIOLO, Rafael. Microservices Based Anomalous Image Recognition Module. 2019. 73 f. Trabalho de Conclusão de Curso – Curso de Bacharelado em Ciência da Computação, Universidade Tecnológica Federal do Paraná. Santa Helena, 2019.

This work presents the development of an anomalous image recognition module in the context of civil construction based on microservices architecture. The KNN and OC-SVM classifiers are used together with the local descriptors HOG, SIFT, SURF and ORB in different PCA dimensions. The dataset was built using Web Scrapping which includes images that belongs to construction and also from other anomalous classes. For microservices, the Spring Framework was used to build services in the frontend, backend and management levels. The best combination of descriptor, classifier and PCA originated a classification model to be deployed in a web service. The HOG local descriptor no use of PCA with the KNN classifier obtained the best performance, achieving 96.2 % metric precision rate when performing the anomalous image detection task, in which the resulting classification model was later embedded and made available for use in a web application.

Keywords: Computer vision; Pattern recognition; Anomalous image recognition; Microsservice.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
BRIEF	Binary Robust Independent Elementary Features
CBIC	Câmara Brasileira da Indústia da Construção
DBSCAM	Density-Based Spatial Clustering of Applications with Noise
FAST	Features from Accelerated Segment Test
FN	Falso Negativo
FP	Falso Positivo
GPU	Graphics Processing Unit
HOG	Histograms of Oriented Gradients
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDS	Sistema de Detecção de Intrusão
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	Json Web Token
KNN	K-Nearest Neighbors
LoG	Laplaciano da Gaussiana
MVC	Model-View-Controller
MS	Milissegundo
OC	One-Class
ORB	Oriented FAST and rotated BRIEF
PCA	Principal Component Analysis
PIB	Produto Interno Bruto
PNADC	Pesquisa Nacional por Amostra de Domicílios Continua

PNG	Portable Network Graphics
REST	Representational State Transfer
RGB	Red Green Blue
SIFT	Scale-Invariant Feature Transform
SSD	Solid-State Drive
SURF	Speeded Up Robust Features
SVM	Support Vector Machine
URI	Uniform Resource Identifier
VANT	Veículo Aéreo Não-Tripulado
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo

LISTA DE SÍMBOLOS

ϕ	Letra grega Fi
γ	Letra grega Gama
ρ	Letra grega Rô
τ	Letra grega Tau
θ	Letra grega Theta
$\sum$	Somatório

SUMÁRIO

1 INTRODUÇÃO

Segundo a Câmara Brasileira da Indústria da Construção CBIC (2018), a construção civil e o desenvolvimento econômico estão profundamente ligados, sendo notável a importância da Construção Civil por meio de seus números. O setor é responsável pela ocupação de 6,8 milhões de pessoas, conforme dados divulgados pela PNADC (Pesquisa Nacional por Amostra de Domicílios Contínua), e responsável por 22,4% do PIB (Produto Interno Bruto) da Indústria. O setor possui uma cadeia produtiva muito extensa, que gera e distribui emprego e renda na economia brasileira.

A indústria da construção civil pode ser considerada um setor prioritário na alocação de recursos escassos devido a seus efeitos econômicos e sociais positivos e seu papel é fundamental na sustentação do desenvolvimento econômico e na geração de empregos (TEIXEIRA; CARVALHO; SILVA, 2012).

A construção civil é munida de aplicações computacionais em suas diversas áreas, seu objetivo é facilitar o processo e maximizar a produtividade. Com a chegada da indústria 4.0, cada vez mais, construtoras, fornecedores e consumidores estão conectados por serviços digitais. De acordo com CAVALCANTI et al. (2018), esta é a principal tendência da atualidade, ponto obrigatório para os negócios que pretendem conquistar mais espaço no mercado, auxiliando no aumento da lucratividade e a diminuir a concorrência.

O desenvolvimento de aplicações computacionais que auxiliem no desempenho e na produtividade de tal área possuem extrema importância. O processo de automação na identificação de imagens pode ser utilizado em aplicações de grande escala, possibilitando que, de forma autônoma, a aplicação final consiga identificar se uma imagem pertence ou não ao contexto do construção civil. A construção de um serviço integrável que disponibilize a análise de contexto dessas imagens pode trazer maior confiabilidade e seriedade para outras aplicações.

Com o avanço do reconhecimento de imagens e padrões, e com o surgimento constante de novas tecnologias e metodologias, torna-se indispensável o uso de uma arquitetura escalável e incremental.

1.1 OBJETIVOS

Nesta sessão, serão apresentados os objetivos deste de Trabalho de Conclusão de Curso.

1.1.1 Objetivos Gerais

Propõe-se criar uma plataforma de classificação de imagens anômalas para o contexto da construção civil baseada na arquitetura de microsserviços.

1.1.2 Objetivos Específicos

- Construir uma base de dados com imagens relacionadas a produtos da construção civil;
- Desenvolver um método para o reconhecimento de imagens anômalas;
- Realizar um comparativo entre as arquiteturas monolíticas e de microsserviços;
- Implementar um módulo de reconhecimento de imagens anômalas utilizando a arquitetura de microsserviços com o *framework Spring*;
- Validar as implementações.

1.2 IMPACTOS FUTUROS

O desenvolvimento deste trabalho contribuirá com a comunidade acadêmica na área de reconhecimento de imagens anômalas, no contexto da construção civil, utilizando descritores locais, além de orientar implementações futuras de soluções em software baseada na aplicação de módulos com a arquitetura de microsserviços.

1.3 JUSTIFICATIVA

Diante da importância demonstrada pelo setor da construção civil, torna-se crucial o desenvolvimento de produtos e pesquisas relacionadas a tal área. Dado o avanço tecnológico atual, o aumento do uso da tecnologia na construção civil e diante de tais dificuldade, é de extrema importância o desenvolvimento de serviços que disponibilizem a classificação dessas imagens de forma autônoma. Com isso, este trabalho propõem o desenvolvimento de um módulo de classificação de imagens anômalas ao contexto da construção civil. Donde será possível classificar uma imagem em relação a anomalia ou não, através de um sistema baseado em microsserviços.

1.4 DELIMITAÇÕES DO TRABALHO

Para o desenvolvimento do módulo, foram estudadas duas arquiteturas de sistemas, limitando-se à implementação da arquitetura de microsserviços devido ao período de tempo disponível. Somente foram utilizadas imagens com as *tags* indicadas na seção 4.1.1. Utilizou-se quatro descritores de características e apenas uma combinação entre eles, resultante da soma de todas as características. Realizou-se experimentos utilizando PCA com somente duas e três dimensões. A fim de realizar a classificação, estudou-se apenas dois classificadores.

1.5 ORGANIZAÇÃO DO TRABALHO

Este trabalho está estruturado em seis capítulos: Introdução, Estado da Arte, Referencial Teórico, Metodologia, Resultados e Conclusão. No **Estado da Arte** é possível observar trabalhos correlatos a este. No **Referencial Teórico** é apresentado ao leitor o embasamento teórico utilizado para a construção deste trabalho. Na **Metodologia** são apresentadas as técnicas utilizadas para construir a solução proposta por este trabalho. Em **Resultados** são descritos os resultados obtidos pelos experimentos realizados neste trabalho. A **Conclusão** encerra este trabalho fortalecendo seus pontos principais e a realização dos objetivos esperados.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta literaturas que sustentam a base teórica da solução proposta por este trabalho. Seu propósito é apresentar as principais técnicas utilizadas para compor o método proposto, os conceitos aqui demonstrados são básicos para o entendimento deste trabalho.

2.1 FUNDAMENTOS DE IMAGENS DIGITAIS

Uma imagem é representada de forma computacional por um modelo matemático. O termo imagem refere-se a uma função de intensidade luminosa bidimensional, denotada por $f(x, y)$, em que o valor ou amplitude de f nas coordenadas espaciais (x, y) fornece a intensidade (brilho) da imagem naquele ponto. Como a luz é uma forma de energia, $f(x, y)$ deve ser positiva e finita (GONZALEZ; WOODS, 2000). A Figura 1 ilustra a convenção do sistema de coordenadas em uma imagem digital.

Figura 1 – Convenção do sistema de coordenadas para imagens digitais.

Um modelo físico para a intensidade de uma cena sob observação pode ser expressado em termo de produto entre dois componentes. A quantidade de luz incidente na cena e a quantidade de luz refletida pelos objetos presentes na cena. Esses componentes são chamados iluminância e reflectância, respectivamente, e são representados por $i(x, y)$ e $r(x, y)$. Assim a função $f(x, y)$ (Equação 1) pode ser representada como

$$f(x, y) = i(x, y)r(x, y) \tag{1}$$

para (Equação 2)

$$0 < i(x, y) < \infty e 0 < r(x, y) < 1 \tag{2}$$

respectivamente (PEDRINI; SCHWARTZ, 2008).

A equação $r(x, y)$ indica que a reflectância é limitada entre 0 (absorção total) e (reflectância total). A natureza de $i(x, y)$ é determinada pela fonte de luz, enquanto $r(x, y)$ é determinada pelas características dos objetos na cena (GONZALEZ; WOODS, 2000).

Os valores para os componentes $i(x, y)$ e $r(x, y)$ da Equação 1 são limites teóricos. A iluminância é medida em lúmem/m^2 ou lux, enquanto a refletância é medida em valores percentuais ou no intervalo entre 0 e 1 (PEDRINI; SCHWARTZ, 2008).

Os pontos indicados pelas coordenadas de x e y de uma imagem são chamados de *pixels*. A Figura 2 mostra a pigmentação dos *pixels* da imagem com tons de cinza, onde cada pixel possui um valor variante no intervalo de 0 à 255. Um modelo bastante utilizado para representação de cores em imagens é o padrão RGB (*Red, Green e Blue* ou em português, Vermelho, Verde e Azul).

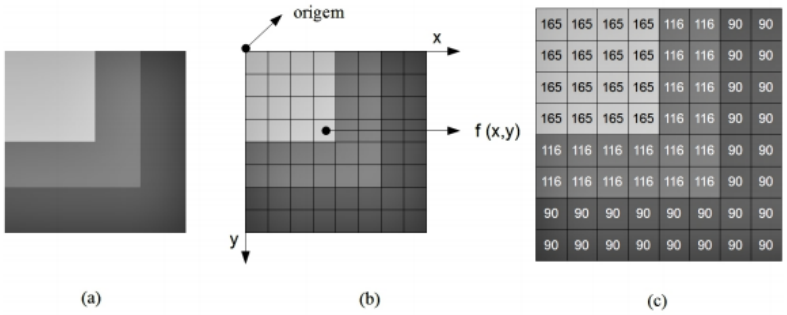

Figura 2 – Descrição dos valores dos *pixels* de uma imagem digital. (a) Imagem original, (b) Imagem discreta e (c) Intensidade dos *pixels*.

Fonte: Brilhador (2015).

As três cores primárias estão em três vértices de um cubo, acompanhados por outras três cores primárias complementares (ciano, magenta e amarelo), que estão distribuídas em outros três vértices, o vértice junto a origem é o preto e o mais afastado é correspondente ao branco (PEDRINI; SCHWARTZ, 2008).

Brilhador (2015) descreve que o número de bits utilizados para representar um pixel pode ser chamado de profundidade de pixel. No caso das imagens pertencentes ao padrão RGB, a profundidade de pixel é de 24 bits, ou seja, cada pixel possui um triplo valor [R,G,B] sendo que cada valor representa uma imagem de 8 bits. A junção das três imagens forma uma imagem no padrão RGB.

A transformação do padrão RGB para a escala de cinza pode ser feita utilizando a média da intensidade dos canais, como por exemplo, seja [0, 255, 51], sua transformação se dá pela Equação 3

$$C = \frac{R + B + G}{3}, \tag{3}$$

logo o valor de C aplicado ao padrão RGB ($[102, 102, 102]$) resulta em um tom da cor cinza.

Gonzalez e Woods (2010) mostram que, dado um pixel p nas coordenadas (x, y), p possui quatro vizinhos horizontais e verticais, cujas coordenadas são dadas pela Equação 4

$$(x+1, y),(x-1, y),(x, y+1),(x, y-1), \tag{4}$$

exceto se o pixel estiver na borda da imagem. Tal conjunto de pixel é conhecido como *vizinhança-de-4* (Figura 3 - (a)) e é representado por $N_4(p)$, onde cada vizinho está a uma unidade de distância. Os quatro vizinhos diagonais de p são representados pelas seguintes coordenadas (Equação 5)

$$(x+1, y+1),(x+1, y-1),(x-1, y+1),(x-1, y-1), \tag{5}$$

são denotados por $N_D(p)$ e chamados de *vizinhança-de-8* (Figura 3 - (b)).

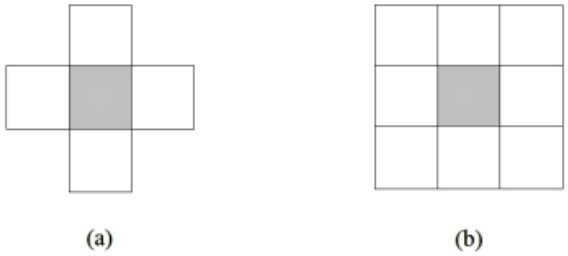

Figura 3 – Vizinhança de um pixel. (a) vizinhança-de-4. (b) vizinhança-de-8.

Fonte: adaptado de Pedrini e Schwartz (2008).

2.2 EXTRAÇÃO DE CARACTERÍSTICAS EM IMAGENS

Na detecção de anomalias as técnicas comumente são aplicadas sobre uma base de dados e, portanto, são sensíveis as formas de representar os objetos que compõem este conjunto de dados. Os objetos podem ser representados por um conjunto de um ou mais atributos, onde cada atributo pode pertencer as diferentes tipos de dados: binários, categóricos ou contínuos (COSTA, 2014).

Para que a detecção de anomalias seja realizada com sucesso, é necessário extrair padrões que sejam adequados a distinção dos objetos que são representadas pelo conjunto de atributos. Em análise de imagens, os padrões são extraídos por descritores. Os descritores, geralmente, representam informações de textura, cor e forma, além de dados globais e locais das imagens (GONZALEZ; WOODS, 2000).

O processo de atribuição de padrões realizados pelos descritores, também é chamado de extração de características. Este processo pode ser realizado de forma automática

ou semiautomática. É composto por um vetor de características, que armazena os padrões dos objetos pertencentes a imagem, e pode ser representado da seguinte forma (Equação 6):

$$x = [x_1,x_2,...,x_n]^T, \tag{6}$$

onde x_i representa o i-ésimo descritor, n é o número total de descritores relacionados ao padrão e T identifica que o vetor foi transposto (BRILHADOR, 2015).

Nas próximas seções serão apresentados os métodos de extração de descritores locais propostos para o trabalho.

2.3 DESCRITORES LOCAIS

Bueno (2011) mostr que os descritores locais de imagens, são computados ao redor de pontos de interesse, e são utilizados em aplicações de visão computacional e processamento de imagens de forma eficaz. Os pontos de interesse em sua maioria, são vértices de contornos ou regiões homogêneas na imagem, são encontrados por detectores e possuem normalmente as seguintes informações: uma coordenada na imagem 2D, uma orientação e uma escala. Para um detector ser considerado bom é necessário apresentar alta covariância a transformações geométricas (como rotação e mudança de escala) e invariância a transformações radiométricas (como ruído ou variação luminosa). Dessa forma, os mesmos pontos de interesse devem ser reconhecidos independente da distorção da imagem.

Nas próximas subseções serão apresentados quatro descritores locais que serão utilizados na realização dos experimentos, são eles: HOG, SIFT, SURF e ORB.

2.3.1 HOG - *Histograms of Oriented Gradients*

Segundo Dalal e Triggs (2005), o este método se baseia na avaliação de histogramas locais normalizados de orientações de gradiente de uma matriz densa. A ideia básica é que a aparência e a forma do objeto local geralmente podem ser bem caracterizadas pela distribuição de gradientes de intensidade locais ou direções das arestas, mesmo sem o conhecimento preciso do gradiente ou das posições das arestas correspondentes.

A Figura 4 representa o fluxo de execução do descritor HOG, o seguinte fluxo baseia-se no fluxo demonstrado por Said, Atri e Tourki (2011), tal procedimento é representado pelos seguintes passos:

1. - Ao iniciar, o algoritmo com a imagem já convertida para escalas de cinza, divide a imagem em janelas, onde cada janela possui tamanho p x p.
2. - O próximo passo é definir as células de tamanho q x q (*pixels*) pertencentes a janela selecionada.

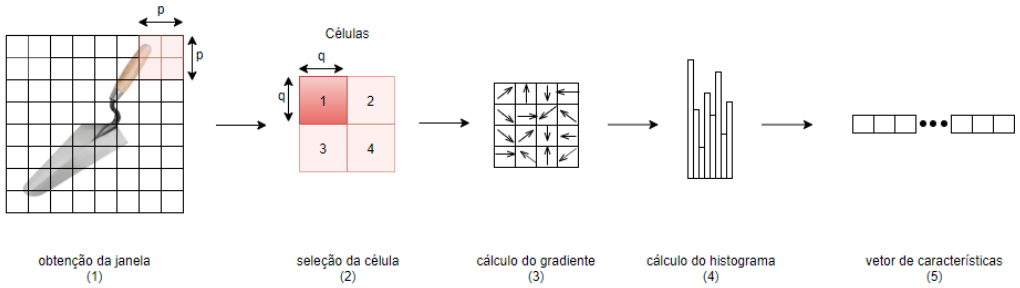

Figura 4 – Fluxo de execução da metodologia HOG.

3. - O cálculo do gradiente, executado em sequência, encontra a variação de intensidade dos *pixels* vizinhos ao *pixel* selecionado, representada pelas setas apresentadas neste passo na Figura 4. O cálculo do gradiente é um estágio crítico na formação dos descritores. A precisão das orientações calculadas e dos histogramas depende desse estágio e, portanto, os resultados estão intimamente relacionados ao método empregado para calcular o gradiente da imagem. Uma das forma de cálculo é por meio do algoritmo de Sobel, que é um dos operadores mais simples e que fornece bons resultados.
   Para uma determinada imagem I, o gradiente horizontal e o gradiente vertical de cada pixel (x, y), é calculado como visto nas equações 7 e 8.

$$G_x(x,y) = \left| \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \right| * I(x,y) \tag{7}$$

$$G_x(x,y) = \left| \begin{bmatrix} +1 & +2 & +1 \\ +0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \right| * I(x,y) \tag{8}$$

   Em cada pixel (x, y), as aproximações dos gradientes horizontais e verticais são combinadas como visto na equação 9.

$$G(x,y) = \sqrt{G_x(x,y)^2 + G_y(x,y)^2} \tag{9}$$

   A orientação do gradiente é calculada pela equação 10.

$$(x,y) = Arctan(\frac{G_y(x,y)}{G_x(x,y)}) \tag{10}$$

4. - O cálculo do histograma de gradiente é realizado acumulando votos em posições para cada orientação. Os votos são ponderados pela magnitude de cada gradiente. Por fim, o algoritmo realiza a união de todos os histogramas obtidos em um único vetor de características.

A Figura 5 mostra uma imagem contida na base de dados (*dataset*) da seção 4.1.1, na esquerda, a imagem em sua forma original de entrada, e na direta a imagem de saída, já foi processada pelo descritor *HOG*, e teve seu gradiente calculado, apontando os contornos de acordo com a intensidade da mudança de cores.

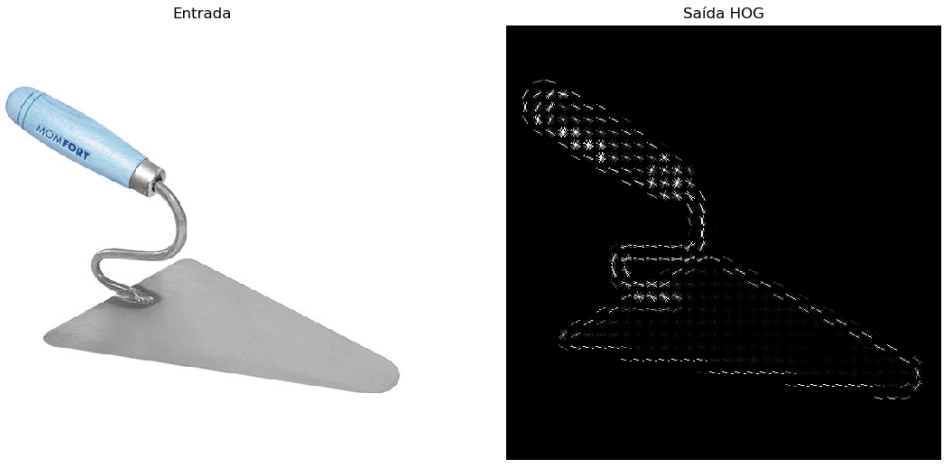

Figura 5 – Exemplo de uma imagem descrita pelo HOG.

2.3.2 SIFT - *Scale Invariant Feature Transform*

Os recursos utilizados no SIFT são invariantes para escalonamento de imagem, translação e rotação, e parcialmente invariantes para alterações de iluminação e projeção 3D. Tais características compartilham propriedades similares com neurônios no córtex temporal inferior que são utilizados no reconhecimento de objetos na visão dos primatas (LOWE, 1999).

Segundo Lindeberg (2012), o descritor SIFT pode ser visto como um histograma dependente da posição das direções do gradiente local em torno do ponto de interesse. Para obter invariância de escala do descritor, o tamanho dessa vizinhança local precisa ser normalizado de uma maneira invariante de escala. Para obter a invariância rotacional do descritor, uma orientação dominante nesta vizinhança é determinada a partir das orientações dos vetores de gradiente dessa vizinhança e é usada para orientar a grade sobre a qual o histograma dependente de posição é computado com relação a essa orientação dominante para alcançar invariância rotacional.

Os próximos parágrafos demonstram o funcionamento do SIFT e são baseados nas afirmações de Machado et al. (2008) e Piotto (2016). A Figura 6 representa as etapas da execução do algoritmo.

O algoritmo SIFT utiliza uma estrutura conhecida como espaço-escala, que consiste em realizar operações de redução da escala, a fim de obter uma pirâmide gaussiana da imagem original. Cópias da imagem original com diferentes níveis de desfoque são produzidas em diferentes escalas. De tal forma é possível notar pontos que são invariantes a escala (Figura 6 (1)).

O próximo passo do algoritmo é encontrar os *keypoint* no espaço-escala através da busca de máximo ou mínimo de uma função de diferença de Gaussianas, tal função é uma aproximação do Laplaciano da Gaussiana (LoG). Muitos pontos serão identificados, por isso é necessário descartar os pontos menos promissores (Figura 6 (2)).

Em seguida, todo *keypoint* selecionado recebe uma orientação e um vetor de características baseado no gradiente do *pixel* vizinho (Figura 6 (3) e (4)). O conjunto de *keypoints* juntos formam um vetor de feições capaz de descrever uma área local da imagem amostrada em relação aos eixos de coordenadas espaço-escala. Todo *keypoint* possui uma localização tanto nas coordenadas espaciais (x,y) quanto no eixo das escalas (σ) e o vetor de características descreve a vizinhança em torno desses pontos em relação a cada escala utilizada do espaço-escala. Com essa descrição em várias escalas obtém-se a invariância a escala.

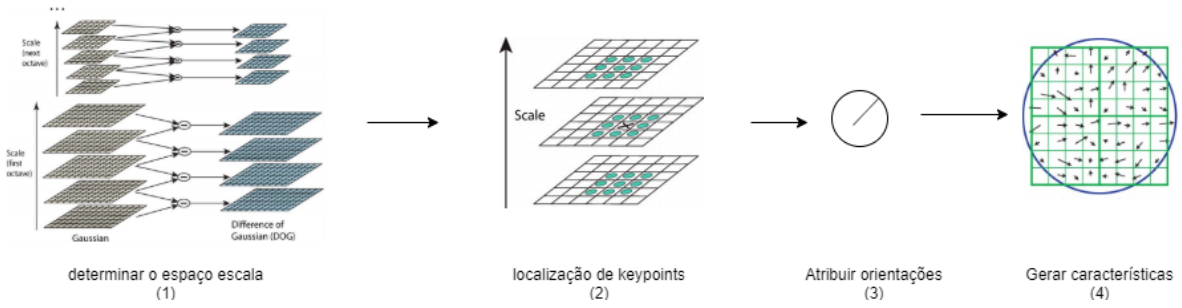

Figura 6 – Fluxo de execução da metodologia SIFT.

Fonte: adaptado de Lowe (2004).

A Figura 7 exibe uma imagem pertencente ao *dataset* descrito na seção 4.1.1 descrita pelo SIFT. Os círculos coloridos identificados na imagem, geralmente nas bordas do objeto, correspondem aos pontos chaves detectados pelo algoritmo, tais pontos irão dar origem ao conjunto de características desta imagem.

2.3.3 SURF - Speeded-Up Robust Features

De acordo com Santana et al. (2015) o algoritmo SURF é, na verdade, uma versão mais acelerada do SIFT. O processo de detecção de *keypoints* do SURF aproxima o LoG com um filtro *Box Filter*. Uma vantagem dessa aproximação é que a operação de convolução com o *Box Filter* pode ser facilmente calculada com a ajuda de imagens integral.

Figura 7 – Identificação de *keypoints* pelo SIFT.

No processo de identificação dos *keypoints* ocorre a convolução desses filtros com a imagem em diversas escalas e 4 orientações, xx, xy, yx e yy (Figura 8 (1)). O uso de filtros caixa sobre uma imagem integral (Figura 8 (2)) é feito em tempo linear no tamanho da imagem, independentemente do tamanho do filtro. No caso de filtros gaussianos, a complexidade de tempo é na ordem do tamanho da imagem multiplicada pelo tamanho do filtro. Em seguida, assim como o determinante da gaussiana, pontos de máximo local são selecionados, eliminando-se pontos em arestas e em regiões de pouco contraste da imagem (BUENO, 2011).

De acordo com Bay et al. (2008), para extração do descritor, o primeiro passo consiste em construir uma região quadrada com 16 sub-regiões e centrada em torno do *keypoint* (Figura 8 (3)). Para ser invariável à rotação da imagem, identifica-se uma orientação reproduzível para os pontos de interesse. Para esse propósito, primeiro calcula-se as respostas da *Haar wavelet* nas direções x e y dentro de uma vizinhança circular de raio $6s$ ao redor do ponto de interesse. A etapa de amostragem depende da escala e é escolhida para ser s. De acordo com o restante, também o tamanho das *wavelets* depende da escala e é definido como um comprimento lateral de $4s$. Portanto, podemos novamente usar imagens integrais para filtragem rápida. Os filtros usados são apresentados na Figura 8 (4). Apenas seis operações são necessárias para calcular a resposta na direção x ou y em qualquer escala.

Para cada região calcula-se a resposta de *Haar wavelet* (Figura 8 (5)) para 25 pontos de amostras regularmente espaçados. Para simplificar, é chamado de dx a resposta *Haar wavelet* na horizontal e dy a resposta *Haar wavelet* na vertical. As respostas dx e dy são somadas em cada sub-região e formam um primeiro conjunto de entradas no vetor de recursos. Para trazer informações sobre a polaridade das mudanças de intensidade é extraído também a soma dos valores absolutos das respostas, $|dx|$ e $|dy|$. De tal forma,

cada região possui um vetor de 4 dimensões descrito por $v = (\sum_{dx}, \sum_{dy}, \sum_{|dx|}, \sum_{|dy|})$. Sendo assim, o vetor de características do SURF é dotado de 64 dimensões.

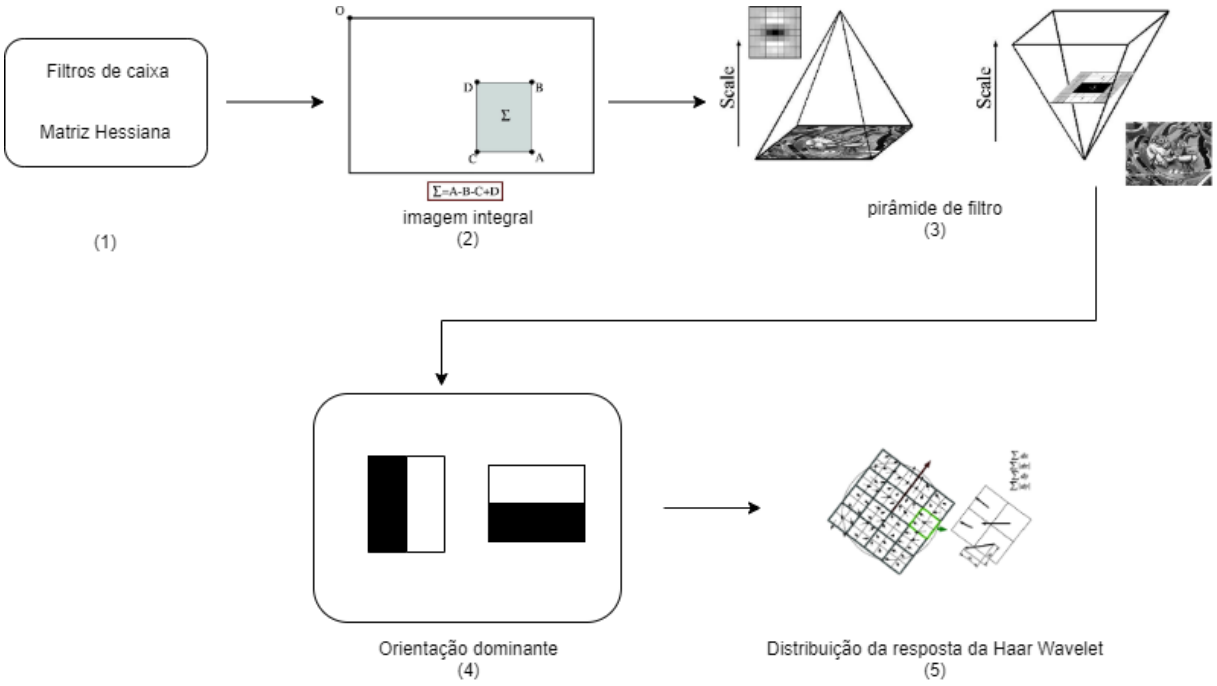

Figura 8 – Fluxo de execução da metodologia SURF.

Fonte: Adaptado de Bay et al. (2008).

A Figura 9 mostra uma imagem do *dataset*, exibido na seção 4.1.1, descrita pelo algoritmo SURF, onde, através dos círculos coloridos é possível localizar os *keypoints*.

Figura 9 – Identificação de *keypoints* pelo SURF.

2.3.4 ORB - *Oriented FAST and rotated BRIEF*

ORB é uma alternativa aos métodos SIFT e SURF, possuindo uma maior velocidade, menos danos ao ruído, e não necessitando de aceleração de GPU (*Graphics*

Processing Unit, ou Unidade de Processamento Gráfico). Devido a velocidade de processamento, o ORB pode ser usado para desempenho em tempo real e em dispositivos de baixa potência, como dispositivos móveis.

O ORB é, basicamente, uma fusão do detector FAST (*Features from Accelerated Segment Test*) com o descritor BRIEF (*Binary Robust Independent Elementary Features*) (SANTANA et al., 2015).

A Figura 10 descreve o fluxo de funcionamento do descritor ORB, porém, para compreender seu funcionamento, é necessário compreender sua composição: detector de *keypoints* (FAST) e descritor de *keypoints* (BRIEF). Esta etapa será baseada em Rublee et al. (2011).

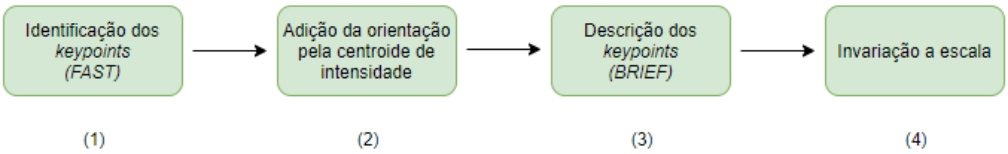

Figura 10 – Fluxo de execução da metodologia ORB.

O primeiro passo a ser feito é detectar os *keypoints* na imagem. O FAST leva consigo um parâmetro que é o limiar de intensidade entre o pixel central e os outros *pixels* contidos em um raio circular. O FAST-9 (raio circular de 9) é utilizado por possuir um bom desempenho.

Pelo fato do FAST não possuir uma boa medida de angularidade, é empregado uma medida de canto de *Harris*, para ordenar os *keypoints* do FAST. Para um valor n de *keypoints*, primeiro é definido um limite baixo o suficiente para obter mais do que N *keypoints*. O FAST não possui recurso em escala múltipla. Desta maneira uma pirâmide de escala da imagem é empregada, e os recursos FAST (filtrados por *Harris*) são produzidos em cada nível da pirâmide.

Para se obter a orientação de canto (Figura 10 (2)), é utilizada uma técnica chamada de centroide da intensidade (ROSIN, 1999). O centroide de intensidade assume que a intensidade de um canto é deslocada de seu centro e esse vetor pode ser usado para imputar uma orientação. Rosin (1999) define os momentos de uma região como (Equação 11):

$$m_pq = \sum_{x,y} x^p y^q I(x,y), \tag{11}$$

e com esses momentos é possível encontrar a centroide (Equação 12):

$$C = \begin{pmatrix} \frac{m_{10}}{m_{00}} & \frac{m_{01}}{m_{00}} \end{pmatrix} \tag{12}$$

e então é possível construir um vetor do centro dos campos, O, para o centroide, $\vec{OC}$. A orientação da região é simplesmente (Equação 13):

$$\theta = athan2(m_{01}, m_{10}), \tag{13}$$

onde athan2 denota um arco tangente.

Para obter invariância para à rotação de tais medidas, os momentos são calculados com um x e y permanecendo dentro da região circular de raio r. Empiricamente é definido que r seja o tamanho da correção, de modo que x e y variem de $[-r, r]$. Quando $\mod C$ se aproxima de 0, a medida se torna instável, porém com FAST isto raramente acontece.

O segundo passo consiste em descrever os *keypoints* anteriormente detectados, e para isso, o BRIEF realiza algumas operações (Figura 10 (3)).

O descritor BRIEF consiste em descrever uma cadeia de bits de uma região da imagem, através de um conjunto de testes binários de intensidade. Considerando uma região de uma imagem suavizada, p. Um teste binário τ é definido pela Equação 14:

$$\tau(\mathbf{p}; x, y) := \begin{cases} 1 & : n\ \mathbf{p}(\mathrm{x}) < \mathbf{p}(\mathrm{y}) \\ 0 & : n\ \mathbf{p}(\mathrm{x}) \geq \mathbf{p}(\mathrm{y}) \end{cases} \tag{14}$$

onde $p(x)$ é a intensidade de p em um ponto x. O recurso é definido como um vetor de n testes binários (Equação 15):

$$f_n(\mathbf{p}) := \sum_{1 \leq i \leq n} 2^{i-1} \tau(\mathbf{p}; x_i, y_i) \tag{15}$$

É considerado o valor de $n = 256$ para o ORB. É de extrema importância suavizar a imagem antes de executar os testes, a suavização pode ser obtida usando uma imagem integral, em que cada ponto de teste é uma sub janela de 5x5 de uma região de 31x31 *pixels*.

Para que o BRIEF seja invariante a rotação (Figura 10 (4)), é necessário que algumas alterações sejam feitas. Um método eficiente é orientar o BRIEF de acordo com a orientação dos *keypoints*. Para qualquer conjunto de recursos de n testes binários no local (x_i, y_i), defina a matriz 2 x n (Equação 16):

$$\mathbf{S} = \begin{pmatrix} x_1,\ldots,x_n \\ y_1,\ldots,y_n \end{pmatrix} \tag{16}$$

Usando a orientação da região θ e a matriz de rotação correspondente $\mathbf{R}_\theta$, é construída a versão orientada de $\mathbf{S}_\theta$ de $\mathbf{S}$ (Equação 17):

$$\mathbf{S}_\theta = \mathbf{R}_\theta \mathbf{S} \tag{17}$$

e então, o BRIEF torna-se orientado (Equação 18)

$$g_n(\mathbf{p}, \theta) := f_n(\mathbf{p}) | (x_i, y_i) \in \mathbf{S}_\theta \tag{18}$$

O ângulo foi discretizado em incrementos de 12 graus, juntamente uma tabela foi construída, com objetivo de prover consultas de padrões do BRIEF já pré-computados. Enquanto a orientação do *keypoint* for consistente, o conjunto correto de pontos S_θ será usado para calcular seu descritor.

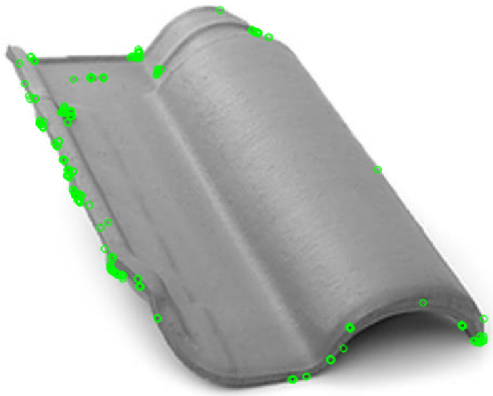

Figura 11 – Identificação de *keypoints* pelo ORB.

2.4 SELEÇÃO DE CARACTERÍSTICAS

A seleção de características tem como objetivo encontrar um subconjunto mínimo de recursos de acordo com alguns critérios razoáveis para que o novo modelo seja tenha o desempenho igual ou superior ao modelo original (LIU; MOTODA, 2012).

2.4.1 PCA - *Principal Component Analysis*

O PCA (Análise do Componente Principal) é uma técnica multivariada que analisa uma tabela de dados na qual as observações são descritas por várias variáveis dependentes, quantitativas e inter-correlacionadas. Seu objetivo é extrair as informações importantes da tabela, representá-las como um conjunto de novas variáveis ortogonais chamadas componentes principais e exibir o padrão de similaridade das observações e das variáveis como pontos nos mapas (ABDI; WILLIAMS, 2010).

Wold, Esbensen e Geladi (1987) definem o PCA como o processo de extração de padrões dominantes na matriz em termos de um conjunto complementar de pontuação e gráficos de carregamento.

O PCA reduz o número de variáveis no vetor de características, eliminando as que não possuem importância significativa, de tal maneira é possível obter um grupo de características menor e simplificado. Na Figura 12 é possível visualizar a representação em três dimensões dos dados presentes no *dataset*, apresentado na seção 4.1.1. Após aplicação do PCA e extração dos três componentes principais. Os pontos em azul representam as instâncias da base que são anomalias, e os pontos em preto representam as instâncias normais. Os 3 eixos representam as 3 principais características contidas no vetor, e mostram a separação das instâncias anômala .

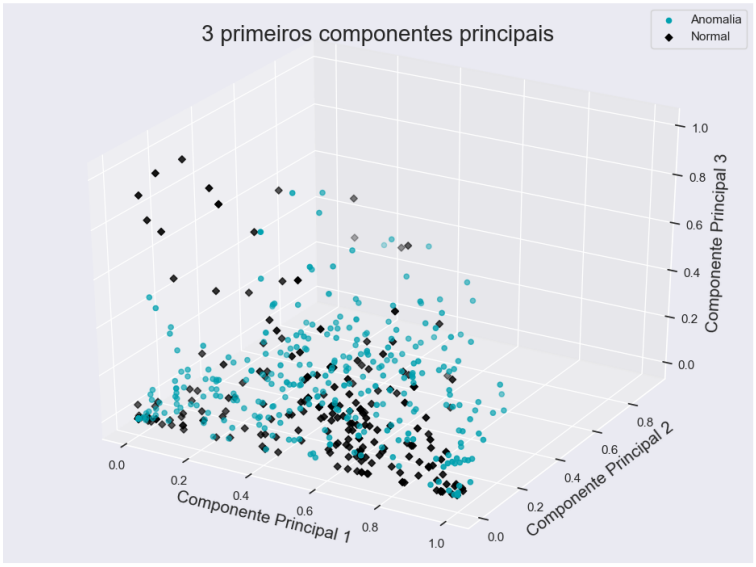

Figura 12 – Representação das 3 principais características do *dataset* (4.1.1) extraídas pelo descritor HOG.

2.5 CLASSIFICAÇÃO DE IMAGENS ANÔMALAS

Nesta seção serão abordados dois classificadores para identificar imagens consideradas anômalas. Toda a esfera de imagens consideradas normais está proposta na seção 4.1.1.

2.5.1 OC-SVM - *One-Class Support Vector Machine*

A abordagem OC (*One-Class*) utiliza somente dados de uma das classes para realizar o treinamento e seus principais objetivos são a distinção entre duas classes quando há um grande desbalanceamento na disponibilidade das mesmas e a detecção de *outliers* (valores atípicos)(COSTA, 2014). Ao utilizar um OC-SVM, a maioria dos dados é separada da origem por uma grande margem no espaço dimensional superior (LUKASHEVICH; NOWAK; DUNKER, 2009).

De acordo com Costa (2014), o OC-SVM busca em um espaço de alta dimensionalidade, encontrar um hiperplano que separe as amostras que forem consideradas anomalias e as que pertencem ao hiperplano encontrado. Neste caso, assume-se que a origem representa todos os pontos de baixa similaridade com a base de treinamento e busca-se maximizar a margem entre o hiperplano encontrado e um hiperplano paralelo a esse que passa pela origem. A Figura 13 ilustra um hiperplano *One-Class*, onde 1 representa imagem normal e -1 imagem anômala, de acordo com o modelo de classificação a ser gerado pelo conjunto de dados proposto na seção 4.1.1.

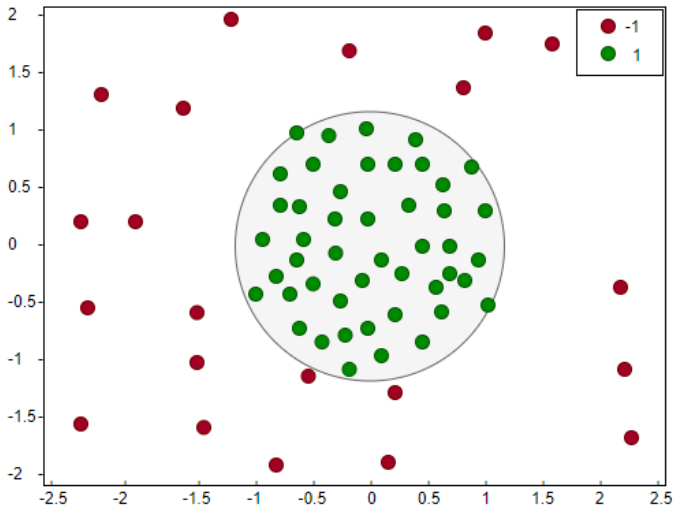

Figura 13 – Representação do hiperplano *One Class*.

Segundo Lukashevich, Nowak e Dunker (2009), dado os vetores de treinamento $\mathbf{x}_i \in R^n$, $i \in [l]$ o modelo é estimado da seguinte forma (Equação 19):

$$\frac{min}{\mathbf{w}, b, \epsilon, \rho} \frac{1}{2}\mathbf{w}^t\mathbf{w} - \rho + \frac{1}{vl}\sum_{i=1}^{l}\epsilon_i \tag{19}$$

sujeito a (Equação 20)

$$\mathbf{w}^t\phi(\mathbf{x}_i) \geq \rho - \epsilon_i, \epsilon_i \geq 0 \tag{20}$$

onde $p/|\mathbf{w}|$ especifica a distância do hiperplano de decisão até a origem, e ϵ_i são variáveis de folga introduzidas. O parâmetro *trade-off* $v \in [0,1]$ corresponde a uma fração esperada de *outliers* dentro dos vetores de característica. Uma solução do sistema acima permite o uso da função de decisão: sgn $(\sum_{i-1}^{l} \alpha_i K(\mathbf{x}_i, \mathbf{x}) - \rho)$, onde $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ é uma função *kernel*, equivalente a um produto escalar dos vetores de recurso mapeados no espaço dimensional maior, e α é um vetor com multiplicadores de Lagrange necessário para resolver ***eq1***.

Lukashevich, Nowak e Dunker (2009) propõem um método de otimização baseado na minimização das diferenças entre a fração de outlier f_{out}, a fração de suporte fSV e o parâmetro trade-off v. O mínimo γ ótimo $\sqrt{(v - fSV)^2 + (v - f_{out})^2}$.

2.5.2 KNN - *K-Nearest Neighbors*

Segundo Coppin (2015), o algoritmo do vizinho mais próximo (*nearest neighbor*s) é um exemplo de aprendizado baseado em instância, onde este armazena dados de treinamento e utiliza estes dados para determinar a classificação para cada novo fragmento de dados que for encontrado. Os dados de treinamento são armazenados, e quando uma nova instância for encontrada, o algoritmo compara os dados de treinamento para encontrar os K vizinhos mais próximos.

De acordo com Faceli et al. (2011), o algoritmo KNN se estende do algoritmo 1-NN, onde a classificação do objeto depende de seu vizinho mais próximo. Cada objeto representa um ponto do espaço definido por seu vetor de característica, adicionando uma métrica neste espaço, é possível calcular a distancia entre dois pontos. A métrica mais usual é a da distância Euclidiana, representada pela equação 21.

$$d(x_i, x_j) = \sqrt{\sum_{l=1}^{d} (x_i^l - x_i^l)^2} \tag{21}$$

Para calcular um exemplo não conhecido pelo conjunto de treinamento, o algoritmo calcula a distância do novo objeto com os objeto recebidos do conjunto de treinamento, e então classifica o objeto de entrada com a classe do objeto de menor distância.

O K-NN utiliza o mesmo processo, mas de forma que o número de vizinhos próximos seja variável e definido por K, quando o valor de K é maior que 1, para cada objeto de entrada são definidos K vizinhos mais próximos, cada vizinho vota em sua própria classe, a classe mais votada é atribuída ao objeto de entrada.

A variação do parâmetro K faz com que sejam calculados K vizinhos mais próximos da nova instância com base no conjunto de treinamento.

Como visto na Figura 14, a variação do parâmetro K, pode alterar o resultado da classificação. O circulo vermelho, representa o simbolo de entrada, e os círculos amarelo e roxo, as classes A e B. Se o valor de K for igual a 3, os 3 vizinhos mais próximos do objeto de entrada são 2 elementos da classe B e 1 da classe A, portanto o objeto é classificado

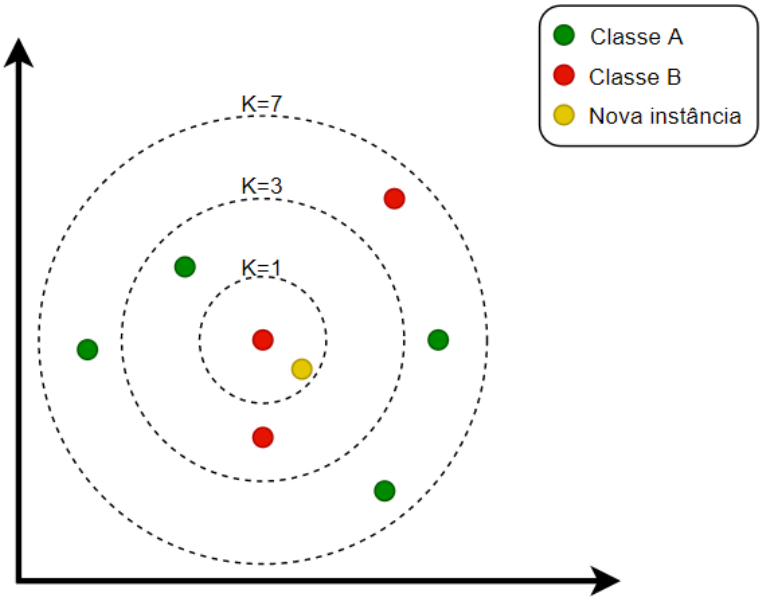

Figura 14 – Variação do parâmetro K no algoritmo KNN.

Fonte: Adaptado de José (2018).

como pertencente a classe B. Se o parâmetro K for igual a 6, existem 4 vizinhos mais próximos pertencentes a classe A e 2 pertencentes a classe B, portando o objeto será classificado com pertencente a classe A.

2.6 AVALIAÇÃO DE DESEMPENHO

Esta seção descreve alguns dos métodos de avaliação e validação aplicados aos classificadores, com finalidade de estimar o desempenho sobre determinada base de dados.

2.6.1 Matriz de Confusão

Pedrini e Schwartz (2008) descrevem a matriz de confusão como uma matriz quadrada C, tal que C apresenta um número de linhas equivalente ao número de classes contidas na classificação, em que a entrada $c_{i,j}$ indica o número de amostras atribuídas a classe ω_i dado que a classe correta é a classe ω_j. De tal forma, os elementos contidos na diagonal principal da matriz denotam o número de amostras corretamente classificadas, ou seja, onde $y_i = f(x_i)$.

De acordo com Pedrini e Schwartz (2008) é possível obter a taxa de erros a partir da razão entre o número de elementos presentes na diagonal principal, a Equação 22 demonstra o cálculo:

$$erro = 1 - \left(\frac{1}{n} \sum_{i=1}^{m} c_{i,i} \right), \tag{22}$$

tal que, m denota o número de classes contidas no experimento e n o número total de amostras.

Dado uma matriz C proveniente de uma classificação de 100 amostras

$$\mathbf{C} = \begin{pmatrix} 20 & 2 & 6 \\ 0 & 36 & 17 \\ 7 & 3 & 9 \end{pmatrix}$$

é obtido a seguinte equação (Equação 23):

$$erro = 1 - \left(\frac{1}{100} \sum_{i=1}^{3} c_{i,i} \right), \tag{23}$$

logo é possível obter

$$erro = 0{,}36.$$

Brilhador (2015) descreve a matriz de confusão como uma matriz de erros e acertos onde é possível analisar a capacidade do classificador em reconhecer diferentes classes e instâncias. Verdadeiro Positivo (VP) e Verdadeiro Negativo (VN) indicam o nível de acerto do classificador, enquanto Falso Positivo (FP) e Falso Negativo (FN) indicam o nível de erro do classificador.

Tabela 1 – Matriz de confusão.

Classes Reais	**Classes Preditas**	
Classe 1	VP	FN
Classe 2	FP	VN

A *precisão* pode ser descrita pela quantidade de itens corretos que realmente foram classificados como corretos, dada pela Equação 24:

$$preciso = \frac{VP}{VP + FP}. \tag{24}$$

O *revocação* é descrito pela frequência da ocorrência de exemplos de uma classe, ou seja, quando um item dito da classe X realmente pertence a classe X, descrito por (Equação 25):

$$revocao = \frac{VP}{VP + FN}. \tag{25}$$

A métrica *F1 Score* combina dados de *precisão* e *revocação*, sendo possível a construção de um indicativo único de qualidade do modelo, descrita como (Equação 26):

$$F1Score = \frac{2 * precisão * revocação}{precisão + revocação}. \tag{26}$$

Optou-se por não utilizar as métricas de acurácia e o erro, pelo fato da base ser desbalanceada, ou seja, possui números diferentes de elementos para cada classe. Quando uma base é desbalanceada, tais métricas não oferecem um bom resultado (CASTRO; BRAGA, 2011).

2.6.2 Validação Cruzada

A validação cruzada (tradução de *Cross-Validation*) divide o conjunto de treinamento original composto por n amostras em k conjuntos, tal que cada conjunto possua a mesma quantidade de elementos. Utiliza-se cada um dos k conjuntos como teste enquanto os outros conjuntos são considerados na etapa de treinamento. Se $k = n$, então a validação cruzada é denominada *leaving-one-out*, em que apenas um elemento é atribuído ao conjunto de teste de cada vez. Embora a maioria das amostras sejam utilizadas para determinar $f(x)$, o *leaving-one-out* apresenta um elevado custo computacional, dado que a quantia de execuções é proporcional ao número de amostras disponíveis (PEDRINI; SCHWARTZ, 2008).

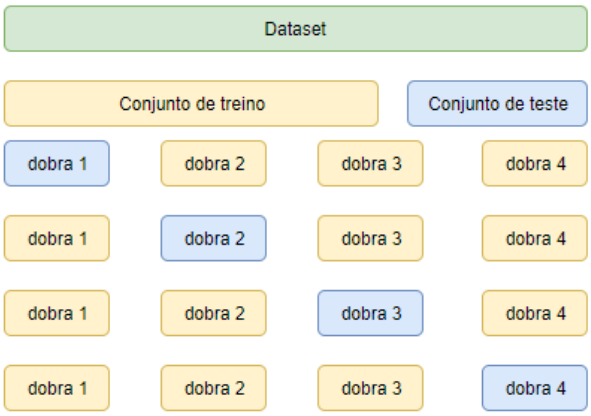

Figura 15 – Exemplo de funcionamento da validação cruzada com valor de k igual a 5.

Fonte: Adaptado de Scikit-learn (2019).

A Figura 15 mostra o funcionamento da validação cruzada, onde a base foi dividida em K conjuntos e cada conjunto possui K dobras. Para cada conjunto, a parte de treinamento é igual a K-1 dobras e o conjunto de teste é igual a 1 dobra.

2.7 ARQUITETURAS DE SISTEMAS

Para compreender e reconhecer a importância da arquitetura de microsserviços é necessário iniciar pela arquitetura monolítica.

2.7.1 Arquitetura Monolítica

As aplicações ou sistemas monolíticos são sistemas de *software* desenvolvidos com variados componentes e funcionalidades agrupados dentro de um grande sistema, fazendo desta uma aplicação projetada e construída em uma única unidade de *software* (ROSA, 2016). Neste esquema de aplicação, o código criado é separado em camadas variadas, mas ainda assim, todo o código pertence a um único ponto de execução, provocando o forte acoplamento.

Meloca (2017) define a arquitetura monolítica como o agrupamento de diversos módulos que residem na mesma aplicação e comunicam-se diretamente por meio chamadas de métodos. Na arquitetura monolítica nenhuma funcionalidade do sistema existe e opera por si só.

As aplicações monolíticas são caracterizadas por possuir um único ponto de falha, proveniente de um único ponto de execução, uma vez que todos os processos pertencem ao mesmo *software*. Todo o processamento requerido através do cliente, é realizado por um único programa.

A realização do processo de *deploy* em uma aplicação monolítica gera apenas um arquivo executável, partindo deste principio, qualquer alteração no *software*, por menor que seja, exigem a atualização completa do sistema.

2.7.2 Arquitetura de Microsserviços

A abordagem de microsserviços é um estilo de arquitetura inspirado na computação orientada a serviços que recentemente começou a ganhar popularidade (DRAGONI et al., 2017). Tal arquitetura pode ser definida como uma metodologia de desenvolvimento de softwares independentes no processo de *deploy*, pequenos e responsáveis por uma fração autônoma da lógica do negócio, que se comunicam através de um mecanismo leve (BRILHANTE et al., 2018). Microsserviço são serviços pequenos e autônomos que trabalham juntos (NEWMAN, 2015).

Um microsserviço é um pequeno aplicativo que pode ser implantado, dimensionado e testado de forma independente e que possui uma única responsabilidade. É uma responsabilidade única, no sentido original, que ela tenha uma única razão para mudar e/ou uma única razão para ser substituída (Thönes, 2015).

Um dos benefícios do uso de microsserviços é a capacidade de publicar uma aplicação grande como um conjunto de aplicações menores (microsserviços) que podem ser desenvolvidos, implementados, implantados, operados e monitorados independentemente (VILLAMIZAR et al., 2015).

Namiot e Sneps-Sneppe (2014) definem microsserviço como um serviço leve e independente que executa funções únicas e colabora com outros serviços similares usando uma interface bem definida

Os microsserviços permitem que as equipes gerenciem grandes aplicações usando uma metodologia mais prática, na qual melhorias incrementais são executadas por pequenas equipes em repositores de código e implantações independentes (VILLAMIZAR et al., 2015).

2.7.3 Comparativo Entre Arquiteturas

A Figura 16 exibe o comparativo do funcionamento entre a arquitetura monolítica e arquitetura de microsserviços, onde na arquitetura monolítica todos os módulos estão presentes em uma única aplicação com um único ponto de falha. Na arquitetura de microsserviços é possível notar que existem várias aplicações com um único propósito (uma aplicação maior), nessa arquitetura existem vários pontos de falha.

Os múltiplos pontos de falha proporcionam maior estabilidade para aplicação, de tal maneira, a falha ou o desligamento de um dos serviços não inviabiliza todo o uso da aplicação. Tomando como exemplo uma aplicação comercial, a falha em um serviço de cadastro não permitiria o acesso de novos usuários, mas permitiria que os usuários existentes realizassem suas tarefas normalmente.

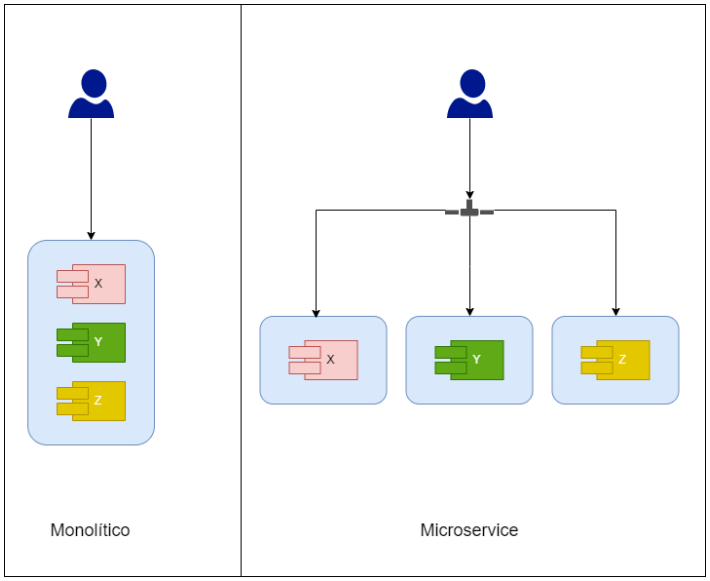

Figura 16 – Arquitetura Monolítica x Arquitetura de Microsserviços.

A Tabela 2 mostra o comparativo de um conjunto de propriedades e suas características perante as arquiteturas monolítica (2.7.1) e de microsserviços (2.7.2).

Na arquitetura de microsserviços o processo de desenvolvimento pode ser realizado por times dispersos e desacoplados, visto que, o desenvolvimento está separado por módulos (ou serviços) e cada equipe pode ser responsável por seu módulo específico e ter seu conhecimento focado neste único módulo. Na arquitetura monolítica geralmente os times são acoplados, todo o sistema está atrelado a um único ponto de execução e a um único ponto de falha, de tal maneira, todo time responsável pela aplicação deve

Tabela 2 – Comparativo entre arquitetura monolítica e arquitetura de microsserviços. .

Categoria	**Monolítica**	**Microsserviços**
Desenvolvimento	Times acoplados	Times dispersos
Conhecimento do sistema	Geral	Por módulo
Linguagens	Única	Múltiplas
Deploy	Único	Múltiplos
Implantação	Geral	Por módulo
Gerenciamento	Geral	Por módulo

conhecer todo o sistema (Villamizar et al., 2016). Ao utilizar os microsserviços é possível que as equipes de desenvolvimento possam trabalhar focadas em áreas específicas, como por exemplo: *Machine Learning*, *Deep Learning* etc.

Ao trabalhar com microsserviços, é possível que cada time desenvolva seu módulo em uma linguagem de programação específica, necessitando apenas unificar a forma de comunicação (exempo: REST). Da mesma maneira, processo de *deploy* e implantação de cada serviço pode ser individual, facilitando a integração contínua da aplicação e minimizando o impacto de um nova atualização (SANTOS, 2017).

Com relação ao gerenciamento, o uso da arquitetura monolítica permite que a aplicação possa ser gerenciada como um todo e não de forma individual, o que não acontece no uso de microsserviços, onde cada módulo pode ser gerenciado individualmente, permitindo a extração de métricas que podem auxiliar na melhora do desempenho do microsserviços (Hasselbring; Steinacker, 2017).

O uso da arquitetura de microsserviços permite vantagens no desenvolvimento e manutenção da aplicação, bem como no entendimento de suas falhas e deficiências. A escalabilidade, ponto chave para a evolução de qualquer aplicação, pode ser facilmente utilizada com a adição de novos serviços.

3 ESTADO DA ARTE

Neste capitulo será apresentado o Estado da Arte do presente trabalho, a explicação fez-se através de duas seções, será tratado sobre: **Módulo Detector de Imagens Anômalas** e **Microsserviços**.

3.1 MÓDULO DETECTOR DE IMAGENS ANÔMALAS

A detecção de conteúdos inadequados ou anômalos vem sendo abordada por diversos pesquisadores, com intuito de atribuir maior segurança e confiabilidade ao meio utilizador. Nesta subseção serão apresentados alguns trabalhos que utilizam a técnica de detecção de anomalias para a resolução de problemas.

Moreira et al. (2016) e Wehrmann et al. (2018) propuseram detectores para conteúdos pornográficos em vídeos, visando manter a tranquilidade dos pais enquanto seus filhos navegam livremente pela *web*, sem a necessidade de uma análise humana.

Tian et al. (2014) descrevem a implementação de um reconhecedor de imagens pornográficas baseado na fusão de informação de cores e formas para representação de palavras. Uma análise em diferentes espaços de cores foi proposta por Balamurali e Chandrasekar (2018) com a finalidade de identificar *pixels* de pele humana, removendo a imagem nítida e identificando apenas a taxa de *pixels* de pele, com o objetivo de identificar imagens obscenas.

Theiler e Cai (2003) abordam um método para classificar imagens incomuns sobre um conjunto de dados com base em uma reamostragem de atributos de dados. Segundo os autores, a reamostragem possui uma "classe de fundo" e, em seguida utilizam uma classificação binária para distinguir dados do conjunto de treinamento original dos dados da "classe de fundo". Assim como neste trabalho, Theiler e Cai (2003) utilizam uma base de dados com dados considerados dentro do padrão, e usaram características variantes ao padrão definido para identificar anomalias.

Cavalcanti (2016) propôs um método para análise de elementos antrópicos na floresta amazônica. O autor diz que as infrações geralmente ocorrem em locais remotos e de difícil acesso, e por este motivo são utilizados VANTs (Veículos Aéreos Não-Tripulados) para cobrir grandes áreas de floresta em um curto espaço de tempo. O problema gerado por essa abordagem é a grande quantidade de dados uma vez que a análise humana é cansativa e propensa a erros. Partindo desse princípio o autor desenvolveu uma avaliação de técnicas de segmentação de imagens, inspeção de características a serem extraídas, seguido da classificação supervisionada destes segmentos para detecção de elementos antrópicos em imagens aéreas da floresta amazônica. Cavalcanti (2016) obteve uma precisão superior a 94% utilizando classificadores unários, um destes classificadores foi o *One-Class SVM*

(seção 2.5.1).

Eitelvein (2015) apresenta um classificador *One-Class SVM* para a detecção de intrusão na rede baseado na detecção de anomalias no tráfego de rede. A base de dados utilizada possui dados de tráfego normal da rede, portando, comportamentos fora do comum são considerados anomalias.

Perdisci et al. (2006) desenvolveram uma técnica para extrair recursos de *Payload*. Utilizaram um algoritmo de recursos de *clusters* que originalmente foi proposto para problemas de classificação de texto e criaram um Sistema de Detecção de Intrusão (*IDS*) baseado em anomalias, utilizando um conjunto de classificadores *One-Class SVM* que trabalham em diferentes espaços de recursos.

Ramteke e Monali (2012) propuseram um método de classificação de imagens médicas utilizando as classes "Normal"e "Anormal". O sistema proposto consiste em quatro fases: pré-processamento, extração de características, classificação e pós-processamento. Os autores utilizaram o classificador KNN, seu desempenho em comparação com o classificador SVM foi superior. Se imagem foi classificada como anormal, o passo de pós-processamento é aplicado sobre a imagem e a região anormal é realçada na imagem.

Um método de detecção de spam em emails foi proposto por (Harisinghaney et al., 2014), utilizou-se os algoritmo: KNN, Naïve Bayes e DBSCAN reverso. Os autores realizaram o comparativo entre os 3 algoritmos, utilizaram os dados com pré-processamento e sem pré-processamento, o KNN obteve o segundo melhor desempenho.

Este trabalho construiu uma base de imagens considerados dentro de um padrão estabelecido, ou seja, imagens relacionadas as mais diversas áreas da construção civil. Posteriormente, extrair características que representem os pontos relevantes das imagens, e utilizando os classificadores *One-Class SVM* e *KNN*, identificar as imagens que não pertencem ao padrão de dados definido.

3.2 MICROSSERVIÇOS

A utilização e estudo da arquitetura de microsserviços tomou grandes proporções devido a seus benefícios. Esta subseção a apresenta alguns trabalhos relacionados a utilização de microsserviços na solução de problemas.

Krylovskiy, Jahn e Patti (2015) propuseram um método para a construção de um sistema IoT para cidades inteligentes com o uso da arquitetura de microsserviços, de acordo com os autores, tal arquitetura permite implementar a flexibilidade e a escalabilidade necessária para acompanhar tal evolução da área. Os autores afirmam que o desacoplamento da arquitetura permite que os desenvolvedores com conhecimentos distintos possam desenvolver diferentes regras de negócio simultaneamente. Afirmam ainda que, o gerenciamento descentralizado permite que cada parceiro concentre-se somente no seu trabalho, sendo este, livre para implementar qualquer tecnologia.

Ribeiro (2018) propôs o uso da arquitetura de microsserviços para utilizar um

método de classificação de dados. De acordo com o autor a *Deep Learning* está em um momento de ascensão, necessitando de um ambiente escalável e modularizado. O autor realizou a implementação de um sistema de recomendações de ponto turísticos e afirma que foio possível modularizar o sistema e apesar das limitações de hardware, possível obter uma diminuição no tempo das requisições.

Gadea et al. (2016) propuseram um sistema de edição de textos compartilhados utilizando a arquitetura de microsserviços, integrando distintas linguagens de programação e bancos de dados, adicionando o reconhecimento facial nas imagens inseridas no texto, assim como a proposta deste trabalho, o autor utilizou microsserviços com *API REST* para realizar o reconhecimento de imagens.

Este trabalho propõe a construção de um conjunto de microsserviços que disponibilizem um serviço integrável de detecção de anomalias.

4 METODOLOGIA

Neste capítulo descreve-se as metodologias utilizadas para o desenvolvimento do módulo de reconhecimento de imagens anômalas baseado em microsserviços.

4.1 IMPLEMENTAÇÃO DO MÓDULO DE RECONHECIMENTO DE IMAGENS ANÔMALAS

A sequência de passos determinante para a identificação de imagens anômalas está contida nesta seção, a Figura 17 mostra o fluxo de passos necessários para a geração de um resultado. Os passos estão descritos nas próximas subseções.

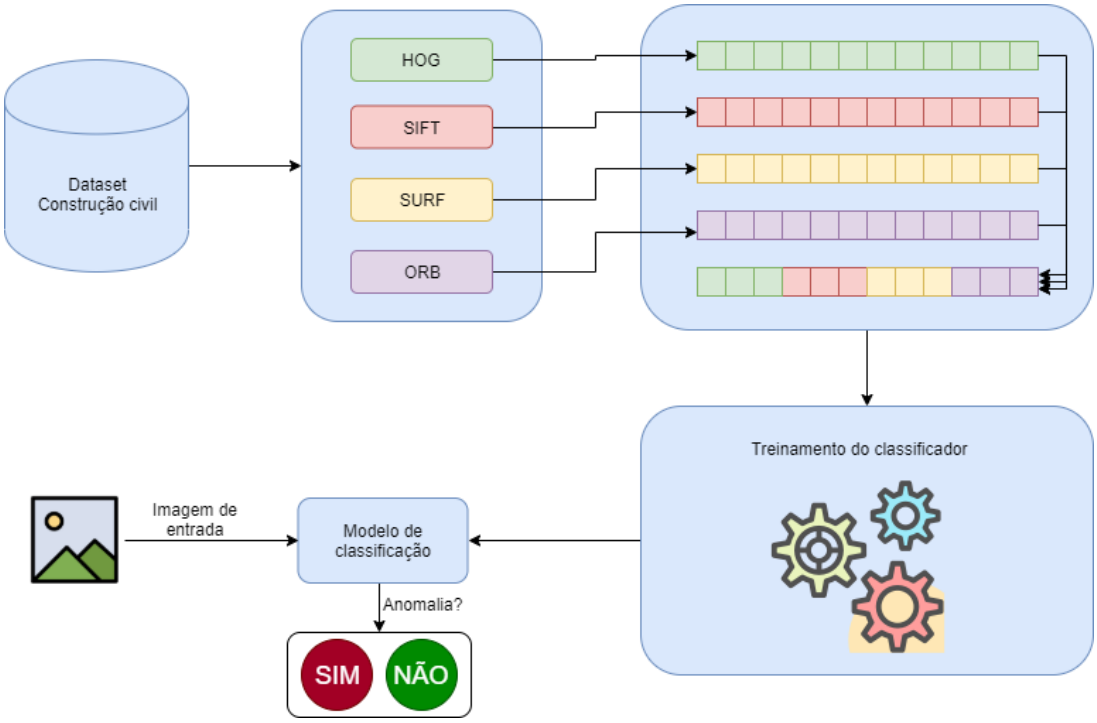

Figura 17 – Fluxo de atividades para o processo de reconhecimento de imagens anômalas.

4.1.1 Construção do *Dataset*

A construção do *dataset* foi realizada por meio do processo chamado de *Web Scrapping*, que consiste em realizar a manipulação de páginas *html* e varrer o arquivo buscando o conteúdo desejado. O processo foi dividido em 3 etapas:

1. Buscar as imagens: foram buscadas imagens disponibilizadas no *Google Images*[1], com conteúdos relacionados a construção civil.

[1] *https://www.google.com/imghp?hl=pt-BR*

2. Coletar a *URL* para download da imagens: utilizou-se um *script* na linguagem *JavaScrit* no navegador (*browser*) que consiste em coletar a *URL* de todas as imagens exibidas no *html* e agrupa-las em um arquivo de texto.
3. Fazer download das imagens: foi criado outro *script* em *JavaScript* e executado sobre o *NodeJs*[2] para fazer o download das imagens com base no arquivo de texto do passo anterior.

Após a finalização do processo de obtenção das imagens, elas passaram por uma análise humana, visando o enquadramento das imagens em suas devidas classes e a máxima remoção do ruído.

As classes de imagens que irão compor o *dataset* são as seguintes: martelo, serra, torneira, parafuso, betoneira, telha, colher, pá, inchada, trena e carriola. Cada classe de imagens será composta por 100 arquivos no formato *png*. Exemplos dessas imagens são apresentas na Figura 18.

Figura 18 – Exemplo de imagens que irão compor o dataset.

O processo de reconhecimento de imagens anômalas foi realizado também com base na classificação binária, sendo necessário a obtenção de imagens que não pertencem ao conjunto padrão de dados. As classes de imagens que não compõem o *dataset* e serão utilizadas nos experimentos são as seguintes: carro, moto, notebook, árvore e cachorro, optou-se por utilizar tais categorias pois todas são anômalas ao contexto da construção civil. Exemplos dessas imagens são apresentadas na Figura 19.

4.1.2 Classificação de Imagens Anômalas

O primeiro passo para identificação das imagens anômalas é extrair as características das imagens pertencentes ao conjunto de imagens, para isto foi utilizado os descritores HOG, SIFT, SURF e ORB, apresentados na seção 2.3. Para realizar a extração de características utilizou-se a biblioteca *Opencv-Python*[3] (BRADSKI, 2000) que é uma biblioteca

[2]https://nodejs.org
[3]https://opencv.org

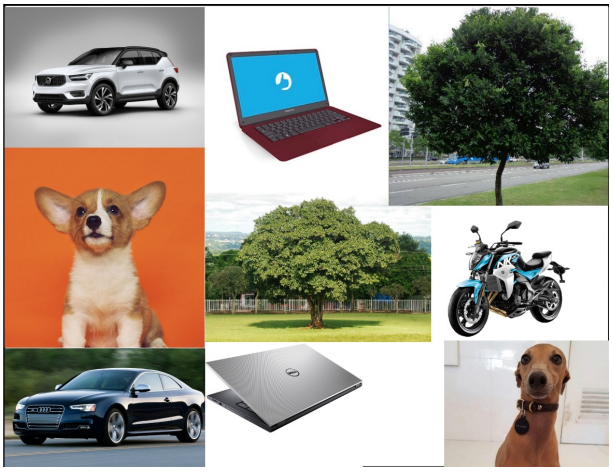

Figura 19 – Exemplo de imagens que anômalas.

OpenSource utilizada para o processamento de imagem. A versão utilizada funciona sobre a versão 3.7.4 da linguagem *Python*.

A aplicação dos descritores citados sobre o *dataset* (seção 4.1.1) deu como resultado quatro vetores de características distintos, mas com tamanhos iguais, para cada imagem pertencente ao *dataset*. Após o processo de extração de características, foi possível combinar os quatro vetores de características formando um único vetor contendo as características de todos os descritores, para uma determinada imagem. A junção de todos os vetores de características formam o conjunto de dados utilizado no treinamento do classificador.

Para implementação dos classificadores utilizou-se a biblioteca *scikit learn*[4] (PEDREGOSA et al., 2011). Para o treinamento dos classificadores utilizou-se a validação cruzada com *k=10*, um valor bastante utilizado em trabalhos relacionados a classificação de imagens, separando os dados em duas classes: normais e anômalas.

Realizou-se um conjunto de experimentos para conjunto de características:

HOG: A implementação deste descritor realizou-se meio da biblioteca *skimage*[5] (WALT et al., 2014), precisamente no módulo *skimage.feature*. Além dos parâmetros padrão que a biblioteca oferece, alterou-se somente a quantidade de *pixels* por célula, informado no parâmetro *pixels_per_cell* com o valor de *(16,16)*. O vetor de características obtido pelo uso do HOG possui 1574 posições (quantidade de imagens) por 2304 dimensões, equivalentes ao número de características extraídas.

ORB: Utilizou-se o algoritmo disponibilizado pela biblioteca *openvc-python* através da importação do módulo *cv2*, não foi alterado nenhum parâmetro no descritor. Como o número de *keypoints* pode variar de acordo com cada imagem, utilizou-se somente os 32 melhores, caso a imagem não possuísse a quantidade mínima de *keypoints*, acrescentou-se zeros ao fim do vetor para manter a mesma dimensionalidade para

[4]https://scikit-learn.org
[5]https://scikit-image.org

todas as imagens, que no caso foi 2048.

SIFT e SURF: Ambos algoritmos estão disponibilizados de forma gratuita na biblioteca *opencv-contrib-python*, utilizou-se apenas os parâmetros fornecidos por padrão para os algoritmos. Assim como no ORB, também selecionou-se os 32 melhores *keypoints*, acrescentando zeros a imagem que possui menos que 32. Obteve-se também, 2048 dimensões no vetor de características.

COMBINADO: Posterior a execução dos 4 descritores acima citados, realizou-se a união dos mesmos por meio da biblioteca *numpy*, onde a função *concatenate* permitiu unir os 4 vetores de características citados em apenas um, respeitando as dimensões, obteve-se um vetor com 8448 dimensões.

A seleção de *keypoints* acima citada para os descritores SIFT, SURF e ORB, consiste em alterar o processo padrão de execução destes algoritmos. Ao detectar os *keypoins* na imagem calcula-se os que retornam maior valor, e seleciona-se os 32 primeiros.

Após o estabelecimento dos vetores de característica, aplicou-se o PCA, apresentado na seção 2.4.1, com o intuito de reduzir a dimensionalidade e aumentar o desempenho dos classificadores. Os valores de PCA foram escolhidos de forma empírica, visando reduzir ao máximo a dimensionalidade dos vetores de características, porém sabe-se que existem outras metodologias para realizar determinada tarefa baseando-se na mudança de desempenho. Portanto, a próxima etapa é classificar as imagens. Para classificação das imagens foi utilizado o classificador KNN e o classificador OC-SVM, descritos nas seções 2.5.2 e 2.5.1, respectivamente.

Para melhorar os resultados dos experimentos, os parâmetros dos classificadores foram ajustados por meio do *Grid Search* conforme os vetores de características relatados nesta seção, e sobre as características selecionados pelo algoritmo de PCA de dimensionalidade igual a 2 e 3.

O desempenho dos classificadores treinados foi avaliado pelas métricas apresentadas na seção 2.6. O modelo com maior desempenho entre as métricas foi incorporado no módulo de reconhecimento de imagens anômalas, proposto na seção 4.2.4. O modelo é exportado e importado por meio da biblioteca *joblib* (JOBLIB, 2019)[6]. Necessitando ainda realizar a implementação dos servidores e comunicação entre eles, realizou-se os passos descritos na seção 4.2, permitindo que a aplicação desenvolvida possa identificar imagens que não pertencem ao contexto da construção civil.

Os experimentos foram desenvolvidos em linguagem *python*, as bibliotecas utilizadas foram *scikit-learn*, *scikit-image*, *numpy* [7] (WALT; COLBERT; VAROQUAUX, 2011), *pandas*[8] (MCKINNEY, 2010), *matplot*[9] (HUNTER, 2007), *opencv-python* (BRADSKI,

[6]https://joblib.readthedocs.io
[7]https://numpy.org
[8]https://pandas.pydata.org
[9]https://matplotlib.org/

2000), *opencv-contrib-python*, *tqdm*[10] (COSTA-LUIS, 2019), além dos módulos nativos do *python*, *os*, *sys* e *argparse*. O *Dataset* usado nos experimentos foi descrito na seção 4.1.1. O *hardware* utilizado para o desenvolvimento dos experimentos está descrito na Tabela 3.

Tabela 3 – Configuração do *hardware* utilizado nos experimentos.

Componente	**Descrição**
Placa mãe	ASUS M5A97
Placa de vídeo	AMD Radeon R7 360 Series
Memória	HiperX 1600 MHz 3x4GB
Processador	AMD FX-8350 4.0 GHz
SSD	Kingston SUV400S37 120GB

4.1.3 Ajustes de Parâmetros do Classificador

A seleção de parâmetros para o classificador pode ser feita através da técnica de *Grid Search*. Este é o processo de executar o ajuste de parâmetros para determinar os valores ideais para um determinado modelo, isso é significativo, pois o desempenho de todo o modelo é baseado nos valores de parâmetros especificados (KRISHNI, 2019).

O *Grid Search* é originalmente uma pesquisa exaustiva com base no subconjunto definido do espaço dos parâmetros (SYARIF; PRUGEL-BENNETT; WILLS, 2016). Dado um grupo de configurações, o algoritmo insere uma a uma no classificador, avalia suas métricas de retorno e seleciona a melhor. Ao fim do processo, obtêm-se o melhor conjunto de parâmetros para tal instância do problema.

Para realizar o ajuste de parâmetros de ambos os classificadores, foram realizados testes com todos os algoritmos descritores e com a combinação deles. Utilizando o técnica de *Grid Search* contida na biblioteca *scikit learn*, foi possível testar de forma automatizada cada combinação e encontrar o melhor parâmetro de cada classificador, aumentando a eficiência de cada algoritmo perante ao descritor.

Utilizou-se variações de valores do *PCA* para efetuar o ajuste de parâmetros, para os 4 descritores, foi utilizado o PCA com valores 2 e 3, e também uma versão sem o uso do PCA, para definir qual o melhor conjunto de parâmetros dentre os testados para cada descritor, o algoritmo utiliza a métrica de *precisão*.

Dentre os possível parâmetros para o classificador KNN, utilizou-se os seguintes:

k : Este valor indica a quantidade de elementos vizinhos a serem considerados para realizar a classificação.

weights : Ao utilizar "*distance*", realiza-se uma votação, considerando a influência dos vizinhos mais próximo ao elemento, logo, sua distância influi de forma inversamente proporcional. A opção *"uniform"* não considera a proporção da distância e sim somente o número de vizinhos.

[10] https://github.com/tqdm/tqdm

metric : Indica qual a formula a ser utilizado para calcular a distância entre dois pontos.

Assim como no classificador KNN, utilizou-se um conjunto de parâmetros a serem testados no OC-SVM. Os parâmetros representam definições utilizadas na execução do algoritmo, os possíveis valores são:

kernel : Função que mapeia os dados originais para outra disposição no espaço, a técnica é utilizada para que os dados sejam separado por uma reta, e não por curvas.

gamma : Coeficiente de *kernel*, auxilia no ajuste do conjunto de dados, influindo diretamente na generalização, pode resolver problemas de super ajuste.

nu : Um limite superior na fração de erros de treinamento e um limite inferior da fração de vetores de suporte.

As Tabelas 4 e 5 apresentam os parâmetros ajustados no *Grid Search* e seus possíveis valores para os dois classificadores, a descrição dos parâmetros está contida na seção 4.1.3.

Tabela 4 – Parâmetros aplicados ao *Grid Search* para o classificador KNN.

Nome	**Valores**
k	3, 5, 7, 11, 13, 15, 19
weights	distance, uniform
metric	euclidean, manhattan

Tabela 5 – Parâmetros aplicados ao *Grid Search* para o classificador OC-SVM.

Nome	**Valores**
kernel	linear, poly, rbf, sigmoid
gamma	0.1, 0.01, 0.001, 0.0001
nu	0.1, 0.3, 0.5, 0.7, 0.9

4.2 IMPLEMENTAÇÃO DOS MÓDULOS E SERVIDORES

A Figura 20 ilustra o conjunto de arquiteturas que compuseram a estrutura dos servidores do *back-end*. Tal ecossistema é constituído por quatro servidores, em níveis de gerenciamento, *back-end* e *front-end*.

A arquitetura de microsserviços é responsável por agrupar os servidores alocados para compor o conjunto de funcionalidades do sistema como um todo. Tal arquitetura possibilita uma maior consistência e disponibilidade da aplicação e permite que os servidores operem de forma independente. Dessa forma, todos os servidores puderam ser executados em uma mesma máquina física e compartilhar da mesma *URI*, mudando apenas a porta de conexão, facilitando o desenvolvimento.

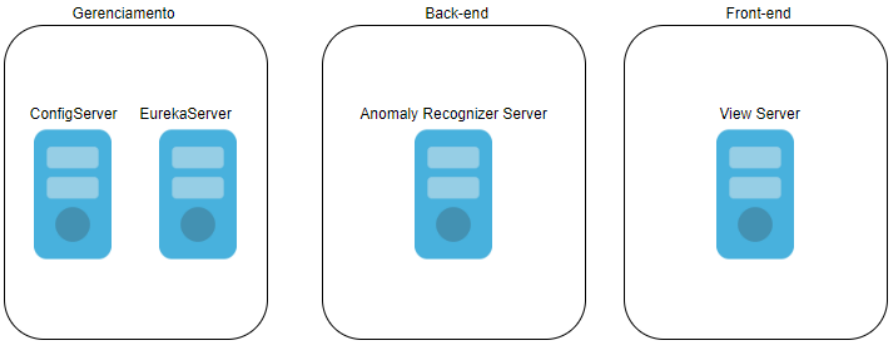

Figura 20 – Organização dos microsserviços.

A arquitetura de microsserviços trouxe para aplicação características positivas, como: heterogeneidade tecnológica, resiliência, escalabilidade e facilidade de implantação (MOREIRA; BEDER, 2016).

Para realizar a implementação dos servidores exibidos na Figura 20 utilizou-se um conjunto de tecnologias e *frameworks*. Todos servidores serão construídos na linguagem Java, versão 8, e farão uso do *framework Spring* [11]. O *Spring* é um *framework* voltado para o desenvolvimento de aplicações corporativas para a plataforma Java, baseado nos conceitos de inversão de controle e injeção de dependência (WEISSMANN, 2014).

Para os servidor de gerenciamento, utilizou-se os módulo *Config Server* e *Eureka Server* do *Spring Cloud*, e para o servidor de *front-end* (*View Server*) o módulo *Spring MVC* e *Spring Boot*. O servidor do *back-end* (*Anomaly Recognition Server*) utilizou o *framework Spring Boot* e *Spring Web*.

No total, a aplicação utilizou quatro servidores, com finalidade de melhorar a escalabilidade da aplicação, cada servidor tem sua função determinada. São eles: *Config Server*, *Eureka Server*, *Anomaly Recognizer Server* e *View Server*.

4.2.1 *Config Server*

Este é um servidor que consume o módulo *Config Server* implementado e distribuído pelo *framework Spring Cloud*. Seu único objetivo é mapear as configurações para os outros servidores.

Através um repositório remoto, o servidor de configuração busca dados de configurações (Figura 21), e distribui para os outros servidores pertencentes ao microsserviços. Funcionando como um maestro, o *Config Server* deve coordenar e distribuir as configurações e evitar conflitos entre portas.

[11] https://spring.io

```yaml
server:
  port: 9093

eureka:
  instance:
    hostname: localhost
    port: 9091
  client:
    registerWithEureka: true
    fetchRegistry: false
    serviceUrl:
      defaultZone: http://${eureka.instance.hostname}:${eureka.instance.port}/eureka/
  server:
    wait-time-in-ms-when-sync-empty: 3000
```

Figura 21 – Arquivo em um repositório remoto utilizado pelo *Config Server* para auto configurar o *Eureka Server*.

4.2.2 *Eureka Server*

Este é um servidor que consome o módulo *Eureka Server*, implementado e distribuído pelo *framework Spring Cloud*. Sua função é monitorar os servidores pertencentes ao microsserviços.

O *Eureka* é uma solução de *Service Discovery open source* desenvolvida pela plataforma *Netflix*, donde é possível registrar um servidor *Eureka* e seus clientes como visto na Figura 22.

Com o *Eureka Server* foi possível traçar métricas sobre a máquina hospedeira e disponibilizar status sobre os servidores (Figura 22). Dessa maneira, foi possível identificar com maior facilidade algumas falhas do sistema.

4.2.3 *View Server*

Dentro do *View Server* utilizou-se a arquitetura MVC (*Model-View-Controller*), de forma simples, sua implementação apenas será utilizada para enviar o conjunto de imagens ao *Anomaly Recognizer Server*, e receber o relatório da classificação. A comunicação deste servidor com o *Anomaly Recognizer Server* se dará por meio de requisições via protocolo *HTTP* por meio do padrão REST.

Os dados transitados entre o *back-end* e o *front-end* em parte estão em formato *JSON*, necessitando assim de um conversor de objeto Java para *JSON*, e para isso será utilizado o *Jackson*, implicitamente fornecido pelo *framework Spring*. A Figura 23 demonstra a conversão de um objeto Java para um *JSON*.

4.2.4 *Anomaly Recognizer Server*

Este servidor é responsável pelo reconhecimento de imagens consideradas anômalas, após o envio de imagens, este servidor é encarregado de transferir a imagem recebida para um módulo *python* (seção 4.1.2) e retornar o resultado. A Figura 24 ilustra o funcionamento deste servidor.

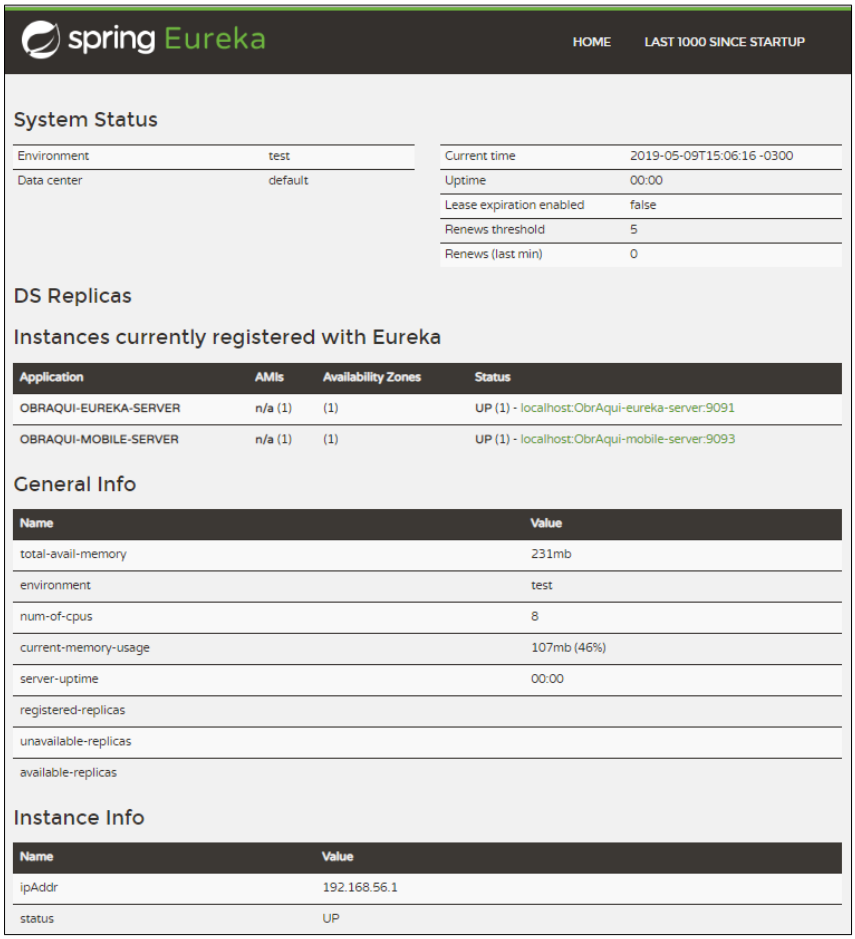

Figura 22 – Execução do *Eureka Server* em um ambiente de microsserviços.

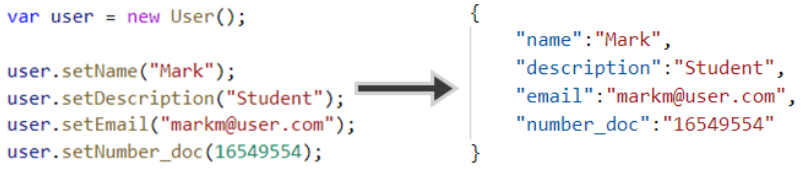

Figura 23 – Transformação de um objeto Java para um *JSON*.

Para realizar a implementação deste servidor, criou-se um conjunto de *builders* e *facades* (GAMMA, 2009), que em tempo de execução, são responsáveis por identificar o ambiente de execução (sistema operacional) e gerar o comando responsável por executar o módulo reconhecedor de anomalias.

Ao enviar uma imagem via *HTTP* para a devida identificação, o servidor a armazena em um pasta temporária e lhe atribui um código temporário, que garante que não existam arquivos duplicados. Quando uma imagem é recebida, o servidor executa um método que gera o *script* de execução, baseado no nome da imagem e no caminho relativo

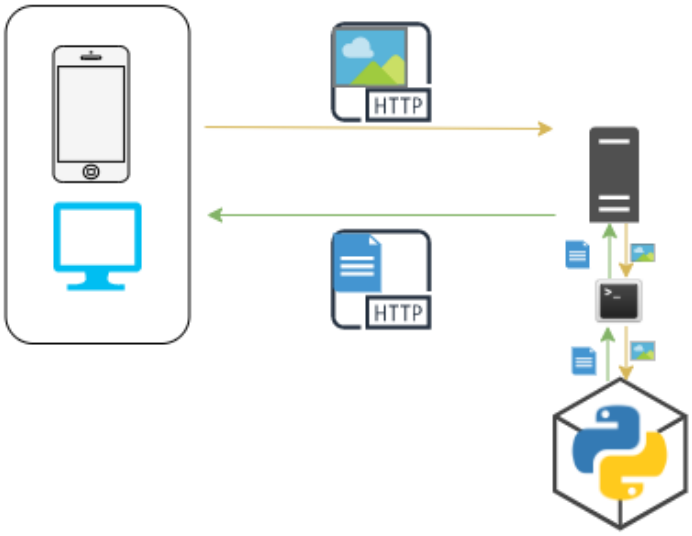

Figura 24 – Processo de reconhecimento de anomalias.

dos dados da máquina hospedeira.

Para enviar os arquivos ao módulo reconhecedor, utilizou-se uma técnica de passagem de parâmetros via argumentos. Essa técnica consiste em anexar parâmetros junto ao comando de execução da aplicação, desta forma, ao iniciar a execução o *script* identifica os argumentos e adiciona a lista *argv* (*Argument Value*), possibilitando a leitura posterior. Os argumentos recebidos indicam o caminho absoluto das imagens no sistema operacional e seu respectivo nome.

Com o *script* de comando pronto, é necessário criar a execução, e para isso utilizou-se um *Process* no *Java*. O *Process* é uma instância do *prompt* de comando do sistema operacional utilizado, porém não possui entrada e saída de dados por forma própria, mas utiliza a entrada e saída do processo pai. Desta maneira, torna-se possível criar um *Process* pela aplicação *Java* e executar o *script Python*. Ao fim do processamento, o *Process* recebe o relatório de resultado, que é transformado em *JSON*, e retornado para o cliente *HTTP* requerente. Ao fim do processo, as imagens temporárias são removidas do servidor.

4.2.5 Forma de Comunicação

O tráfego de dados entre o cliente (*front-end*) e o servidor foi realizado por meio do padrão REST (*Representational State Transfer*), utilizando a comunicação via protocolo HTTP (*Hypertext Transfer Protoco*). Tal implementação exigiu a utilização da arquitetura *Cliente-Servidor*, que resume-se em separar o cliente do servidor e uni-los por um padrão de comunicação. A Figura 25 descreve o uso da arquitetura *Cliente-Servidor* com o padrão

REST para o *back-end* referenciado na Figura 20.

Figura 25 – Arquitetura Cliente-Servidor com uso do padrão REST.

Os dados que irão trafegar entre o *front-end* e o *back-end*, como visto na imagem 25, serão transportados por mensagens *HTTP*. Tais mensagens são compostas por três partes:

- requisição: composto por uma URI *(Uniform Resource Locator)*, pelo método *HTTP* e pela versão do protocolo *HTTP*.
- cabeçalho da mensagem: conhecido como *header*, é responsável por transmitir informações adicionais entre o cliente e o servidor.
- corpo da mensagem: é utilizado para armazenar a mensagem a ser transferida (*JSON*, multipart/form-data, text, e etc.)

O cliente fará chamadas *HTTP* para o serviço do *back-end*, as chamadas estão configuradas com as *URI*'s e métodos *HTTP* do *back-end*. O *header* será utilizado para carregar o informações do formato do corpo da mensagem e o corpo será composto pelos formatos JSON e *multipart/form-data* para transportar o relatório e imagens, respectivamente.

5 RESULTADOS E DISCUSSÕES

Os resultados desse trabalho foram divididos em três partes. Primeira, avaliação de tempo para extração das características pelos descritores locais utilizados. Segunda, análise dos classificadores para reconhecimento de imagens anômalas em aplicações relacionadas ao contexto da construção civil. Terceira, implementação da aplicação com base no melhor modelo de classificação gerado.

5.1 AVALIAÇÃO DE TEMPO

Neste primeiro experimento foi avaliado o tempo de extração das características dos descritores locais, apresentados na seção 2.3. A avaliação foi feita de forma empírica, ou seja, a análise do tempo leva em consideração o tempo de execução do código. O tempo de execução, foi contabilizado pela diferença de tempo entre a hora de inicio e fim da execução do algoritmo com base na média de 5 execuções.

A Tabela 6 apresenta o tempo médio da extração de características sobre uma única imagem. A média foi calculada sobre as 1574 imagens com 256 x 256 de altura e largura, que compõem o *dataset* descrito na seção 4.1.1. O descritor com menor tempo para extração das características foi o ORB que obteve o tempo médio de 17.57ms, sendo duas vezes mais rápido que o descritor SIRF que possui arquitetura semelhante para extração de características. O experimento demonstrou que o ORB é algoritmo mais indicado para aplicações que levam em consideração o tempo de respostas das requisições, sendo o caso deste trabalho, que apresenta um estudo de implementação do módulo de reconhecimento de imagens anômalas para ambiente de desenvolvimento *web*.

Tabela 6 – Tempo médio em milissegundo de extração de características por imagem.

Descritor	**Tempo**
HOG	42.99ms
SIFT	39.60ms
SURF	24.35ms
ORB	17.57ms
COMBINADO	124.52ms

Os descritores SIFT e SURF são invariantes a rotação e a escala, para alcançar tal característica, ambos os descritores aplicam filtros sobre a imagens de entrada, obtendo "cópia" dessa imagens em diferentes escalas. Esse processo, é extremamente custoso e está diretamente vinculado a resolução das imagens. Devido, as limitações de *hardware*, foi realizado um pré-processamento nas imagens, padronizando as resoluções das imagens de entrada além de manter bons índice de tempo na extração das características. O tempo

médio por imagem consumido de vetor de características combinado é composto pela soma dos tempos médios individuais de cada descritor.

5.2 RECONHECIMENTO DE IMAGENS ANOMÂLAS

Neste segundo experimento foram utilizadas as características extraídas pelos descritores apresentados na seção 2.3.

5.2.1 KNN

Esta seção apresenta os experimentos realizados com o classificador KNN conforme as características utilizadas: os vetores de características descritos na seção anterior, e características selecionadas pelo PCA de 2 e 3 dimensionalidades.

Na Tabela 7 apresentam-se os conjuntos de parâmetros encontrados com os descritores sem o uso de PCA. Os parâmetros foram selecionados levando em consideração os resultados da classificação que apresentam melhores índices de *precisão* no reconhecimento de imagens anômalas.

Tabela 7 – Seleção de parâmetros do classificador KNN com todos os descritores sem PCA.

Descritor	***k***	***weigths***	***metric***
HOG	3	distance	manhattan
SIFT	5	distance	euclidean
SURF	19	distance	manhattan
ORB	7	distance	euclidean
COMBINADO	19	distance	euclidean

Os resultados apresentados na Figura 26, foram obtidos utilizando os parâmetros apresentados na Tabela 7. De forma geral, o descritor HOG obteve os melhores índices em todas as métricas avaliadas no experimento, alcançando a marca de 90,9% com a média *f1score* para a detecção de anomalias e 96,2% para a detecção de imagens consideradas normais, tal métrica representa a média harmônica da *precisão* e *revocação*, que alcançaram a marca de 96,2% e 86,4% para anomalias e 94,1% e 98,4% para imagens normais, respectivamente.

Esses resultados demonstram, que considerando apenas os dados brutos do vetor de características, as técnicas nas quais realizam a identificação dos *keypoints* para extração das características, como ORB, SIFT e SURF, se mostraram inferiores a técnicas que utilizam toda informação disponível na imagem, no caso, o descritor HOG. Provavelmente, esses resultados ocorreram devido a grande variação de forma, cores, iluminação, posicionamento, etc. que existem nos objetos de uma mesma classe, conforme apresentado na Figura 27. Os métodos baseados em *keypoints* necessitam que os pontos de extração

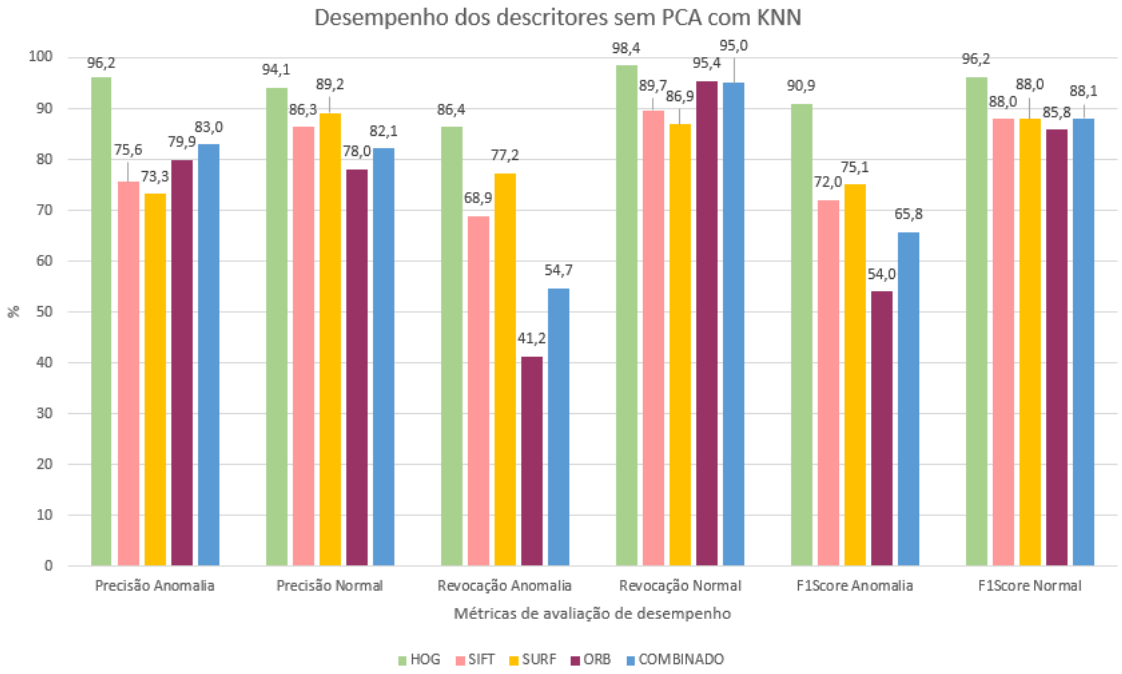

Figura 26 – Gráfico de desempenho dos descritores sem PCA com o classificador KNN.

sejam localizados em partes semelhantes em relação aos objetos, e por consequência, as informações próximas ao ponto que são utilizadas para descrever a imagem devem ser parecidas com as outras imagens da mesma classe. Enquanto, o descritor HOG concatena os histogramas locais normalizados extraídos de todas as divisões da imagem, e devido a essa concatenação ele apresenta uma quantidade maior de informações globais pois considera todas as partes da imagem.

Figura 27 – Mosaico de diferentes imagens da categoria “martelo”.

Extraindo o PCA de duas dimensionalidades dos vetores de características apresentados anteriormente, obteve-se os resultados apresentados na Figura 28. Os parâmetros utilizados no experimento foram apresentados na Tabela 8.

Tabela 8 – Seleção de parâmetros do classificador KNN com todos os descritores com PCA de duas dimensionalidades.

Descritor	***k***	***weigths***	***metric***
HOG	13	uniform	manhattan
SIFT	19	uniform	euclidean
SURF	19	uniform	euclidean
ORB	19	uniform	manhattan
COMBINADO	19	uniform	euclidean

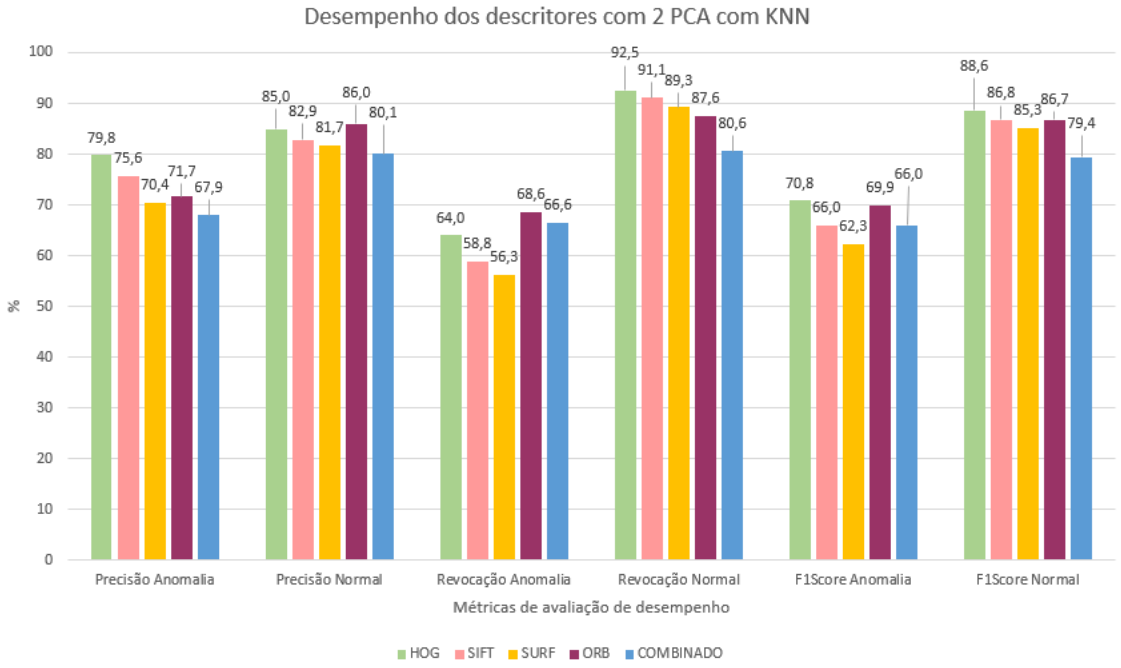

Figura 28 – Gráfico de desempenho dos descritores com PCA de 2 dimensionalidades com o classificador KNN.

Neste experimento o descritor com melhor desempenho obtido foi o HOG, ele se demonstrou superior em quase todas as métricas extraídas. Para a detecção do conjunto de anomalias, o HOG obteve uma *precisão* de 79,8% e um *revocação* de 64%, logo sua *f1score* foi de 70,8%. Ao avaliar a capacidade do experimento na detecção de imagens normais, é possível notar uma grande diferença, uma vez que a *f1score* ficou em 88,6%, mostrando que tal experimento tem maior potencial na detecção de imagens normais. Analisando somente a média harmônica entre as duas possibilidades de *precisão* e *revocação*, percebe-se uma leve similaridade nos valores obtidos, com diferenças que chegam a 1,1% e 2,1% com relação a classes de anomalias e normais respectivamente. De forma geral, os melhores índices de classificação foram obtidos pelo descritor HOG, porém as taxas de *precisão*, *revocação* e *f1score* demonstraram valores inferiores ao experimento anterior sem aplicação do PCA.

A Tabela 9 mostra o conjunto de parâmetros obtidos pela busca exaustiva para o conjunto de descritores com o uso de 3 dimensionalidades no PCA. Na Figura 29 é apresentado o gráfico resultante da classificação com três componentes principais (3 PCA) sobre o conjunto de parâmetros da Tabela 9.

Tabela 9 – Seleção de parâmetros do classificador KNN com todos os descritores com PCA de três dimensionalidades.

Descritor	***k***	***weigths***	***metric***
HOG	19	uniform	euclidean
SIFT	19	distance	euclidean
SURF	13	distance	euclidean
ORB	19	uniform	euclidean
COMBINADO	19	uniform	euclidean

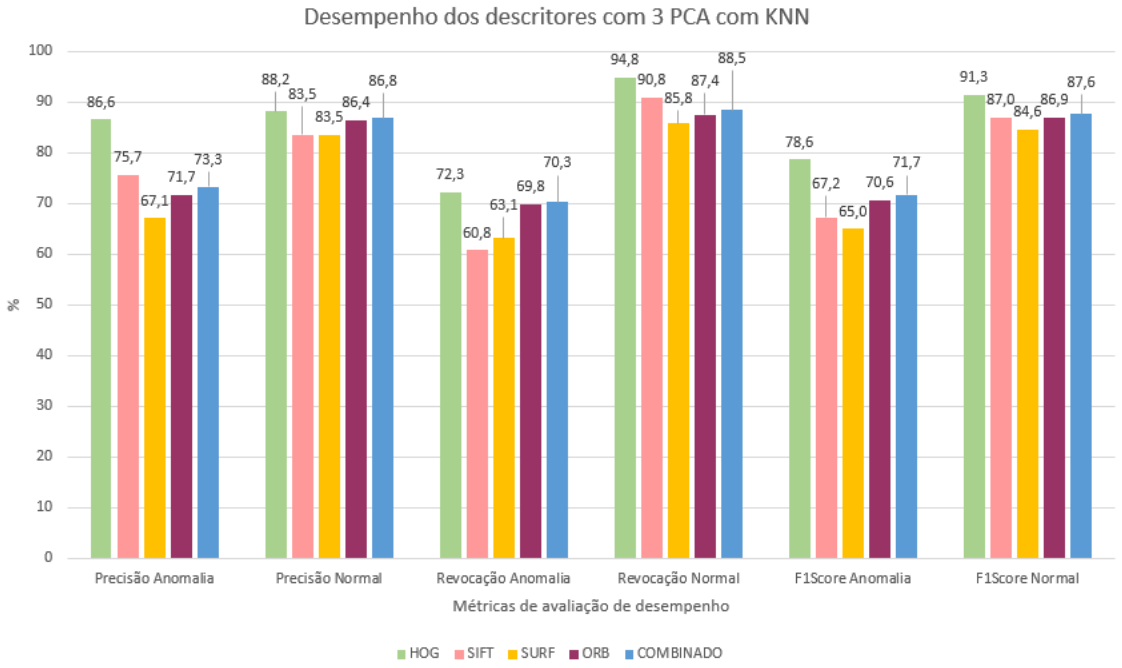

Figura 29 – Gráfico de desempenho dos descritores com PCA de 3 dimensionalidades com o classificador KNN.

Neste experimento, nota-se uma leve similaridade no desempenho dos descritores SIFT, SURF, ORB e do vetor combinado, porém o melhor desempenho novamente está relacionado com o descritor HOG. Avaliando as métricas de *precisão* e *revocação* para as imagens anômalas e normais, percebe-se superioridade no uso do HOG, sua *f1score* obteve valores de 78,6% e 91,3% com relação a imagens anômalas e normais.

Com o classificador KNN, os melhores resultados obtidos foram utilizando o descritor HOG, que trabalha com a descrição completa da imagem, ao invés de detectar os *keypoints* como os outros descritores. As Tabelas 10 e 11 apresentam os descritores que

demonstraram os melhores resultados em relação aos vetores de características utilizados, os dados foram divididos em duas tabelas para melhorar a visualização.

Tabela 10 – Melhores resultados para cada PCA utilizando o classificador KNN - Anomalia.

PCA	**Descritor**	**Precisão Anomalia**	**Revocação Anomalia**	**F1 Anomalia**
Sem	HOG	96,2%	86,4%	90,9%
2	HOG	79,8%	64%	70,8%
3	HOG	86,6%	72,3%	78,6%

Tabela 11 – Melhores resultados para cada PCA utilizando o classificador KNN - Normal.

PCA	**Descritor**	**Precisão Normal**	**Revocação Normal**	**F1 Normal**
Sem	HOG	94,1%	98,4%	96,2%
2	HOG	85%	92,5%	88,6%
3	HOG	88,2%	94,8%	91,3%

Como visto nas Tabelas 10 e 11, o descritor HOG obteve resultados superiores aos demais descritores em todas as métricas avaliadas. A versão mais eficiente do descritor HOG ocorreu sem o uso do PCA. O PCA demonstrou seu melhor desempenho com 3 dimensionalidade, com diferença de 2,7% na *f1score* para o experimento com PCA de dimensionalidade 2.

O experimento com melhor desempenho utilizando o classificador KNN foi o HOG sem PCA, este experimento obteve uma alta taxa de *precisão* nas detecções de anomalia e normal, ou seja, de todas as imagem que o classificador indicou ser anomalia, ele acertou 96,2% e de todas que indicou ser normal, acertou 94,1%. Avaliando o *revocação* é possível observar que quando um dado elemento é realmente de uma classe quanto o classificador acerta, ou seja, para todos os elementos da classe anomalia, o classificador acertou 86,4% e para todos os elementos normais, o classificador conseguiu identificar corretamente 98,4%.

5.2.2 OC-SVM

Esta seção apresenta os experimentos realizados com o classificador OC-SVM separados por diferentes valores de PCA de dimensionalidades 2 e 3, e aplicados aos 4 descritores, bem como ao vetor de características combinado, visto na seção 4.1. Essa etapa de experimentos não contempla a busca de parâmetros com a opção de *kernel poly* nos casos do descritor SIFT e do vetor combinado, devido a limitações de hardware, sendo inviável a execução do algoritmo em tempo hábil. A Tabela 12 apresenta o conjunto de parâmetros resultantes do algoritmo *Grid Search* para o classificador OC-SVM aplicada a cada descritor sem o uso do PCA.

Tabela 12 – Seleção de parâmetros do classificador OC-SVM com todos os descritores sem PCA.

Descritor	***nu***	***gamma***	***kernel***
HOG	0.9	0.01	rbf
SIFT	0.9	0.0001	linear
SURF	0.9	0.1	sigmoid
ORB	0.9	0.0001	linear
COMBINADO	0.9	0.0001	linear

A Figura 30, apresenta o gráfico resultante da execução do classificador OC-SVM com os conjuntos de parâmetros exibidos na Tabela 12. Observando a Figura 30, nota-se que a maioria das métricas avaliadas apresentam índices abaixo de 50%. Duas hipóteses são consideradas para geração de tais resultados: (1) um ajuste incorreto dos parâmetros, mesmo utilizando ferramentas que auxiliam esse processo como a busca exaustiva; (2) a distribuição dos características extraídas não proporcionou uma divisão bem definida das classes de imagens anômalas e normais, impedindo um ajuste preciso da função *kernel* do classificador OC-SVM. Apesar, dos baixos índices, o descritor HOG possui o maior desempenho sobre todas as métricas avaliadas para ambas as classes, mostrando ser a classificação mais eficiente deste experimento.

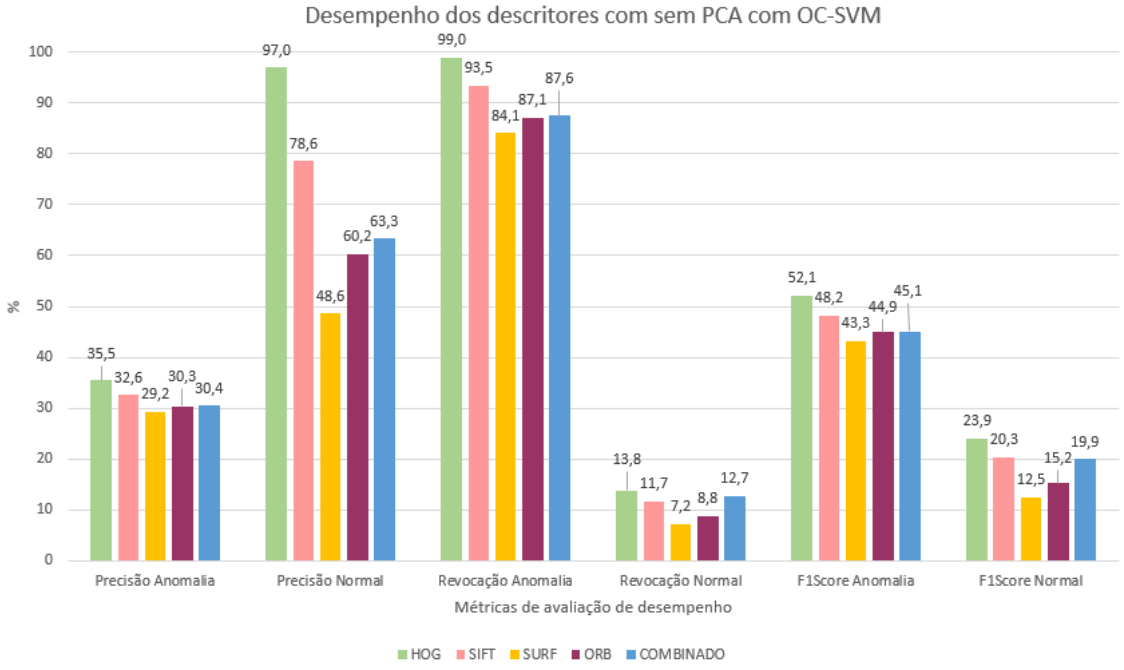

Figura 30 – Gráfico de desempenho dos descritores sem OC-SVM com o classificador OC-SVM.

Conforme os resultados da Figura 31, percebe-se que mesmo após a aplicação do PCA, os resultados em sua maioria se mantiveram com uma taxa inferior a 50%. Os parâmetros utilizados na classificação são apresentados na Tabela 13.

Percebe-se que os níveis de *precisão* obtidos para as classes normais são superiores as classes de anomalias, ao comparar o maior valor de cada classe, percebe-se uma diferença de 49,1%. O mesmo fato ocorre de forma inversa ao realizar a mesma comparação em relação ao *revocação*, desta vez, a diferença em função dos maiores valores chega a 47,5%. Ao avaliar os níveis de *f1score* obtidos, percebe-se que o descritor SURF possui maior desempenho com relação à detecção de anomalias, e o descritor HOG possui vantagem na classificação de classes normais. Como o objetivo deste trabalho é a detecção de anomalias, considera-se que o SURF possui o melhor desempenho deste experimento.

De forma geral, percebe-se que o classificar possui um grande desempenho em detectar anomalias, porém, muitas de suas classificações com relação a anomalia estão incorretas, no caso do HOG, apenas 35,5% das vezes em que o classificador indicou anomalia realmente era anomalia. Na detecção de imagens normais, o classificador possui um alto desempenho para acertos, ou seja, quase todas as imagens consideradas normais, realmente eram normais, porém percebe-se que o classificador quase nunca indica uma imagem como sendo normal.

Tabela 13 – Seleção de parâmetros do classificador OC-SVM com todos os descritores com PCA de duas dimensionalidades.

Descritor	***nu***	***gamma***	***kernel***
HOG	0.9	0.01	rbf
SIFT	0.9	0.0001	sigmoid
SURF	0.9	0.01	sigmoid
ORB	0.1	0.1	poly
COMBINADO	0.9	0.01	sigmoid

A Tabela 14 apresenta os parâmetros utilizados no experimento com PCA de dimensionalidade 3. De acordo com o gráfico da Figura 32, observa-se que os descritores possuem uma grande disparidade de desempenho, avaliando a classe de anomalias percebe-se que o HOG possui o maior desempenho (40,4%), porém, ao avaliar a métrica de *revocação*, é possível observar a superioridade do vetor combinado, 92,7%, se comparado ao HOG, a diferença chega a 30,9%, de tal maneira a métrica de *f1score* é fundamental para definir o melhor descritor para tal classe. A *f1score* do HOG para a detecção de anomalias possui valores de 48,6%, demonstrando-se o mais adequado. A detecção de classes normais, possui melhor valor de *precisão* e *revocação* com o descritor HOG, 77,9% e 58,6% respectivamente. Portanto, para este experimento, o HOG se demonstrou superior para a classificação das duas classes.

De acordo com as Tabelas 15 e 16, o descritor HOG se demonstrou mais eficiente que os demais descritores no experimento sem o uso do PCA. O descritor SURF, obteve o seu melhor desempenho com a aplicação do PCA de duas dimensionalidades.

Em detecção de anomalias o HOG sem PCA demonstra-se superior aos demais

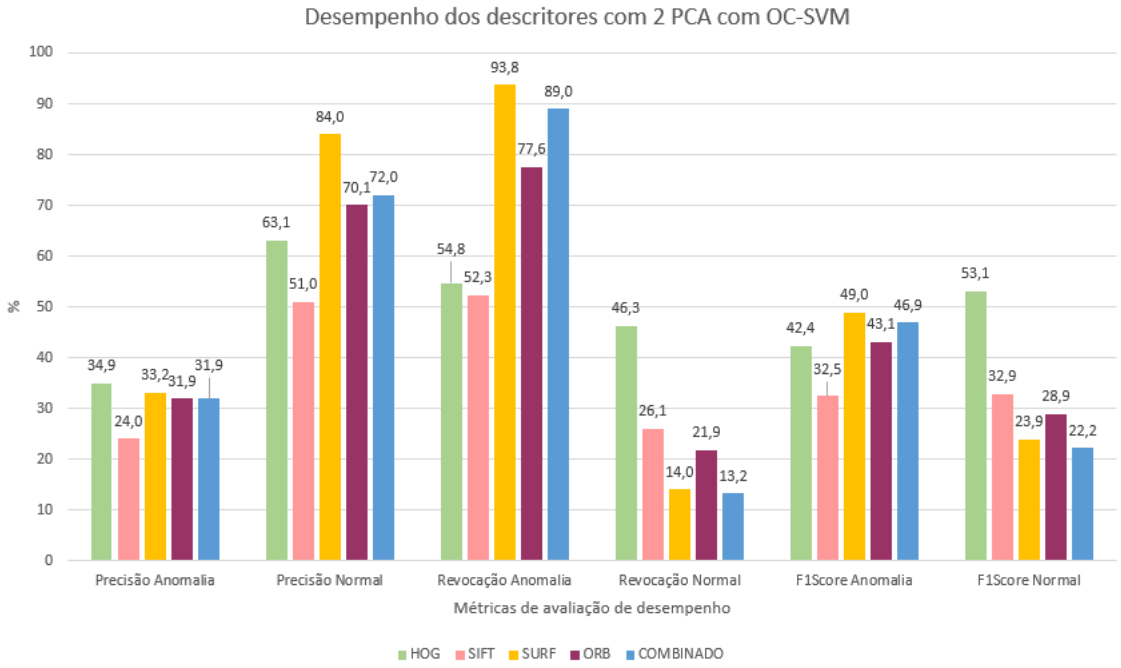

Figura 31 – Gráfico de desempenho dos descritores com 2 PCA com o classificador OC-SVM.

Tabela 14 – Seleção de parâmetros do classificador OC-SVM com todos os descritores com PCA de três dimensionalidades.

Descritor	***nu***	***gamma***	***kernel***
HOG	0.9	0.01	sigmoid
SIFT	0.9	0.1	sigmoid
SURF	0.9	0.0001	linear
ORB	0.9	0.1	sigmoid
COMBINADO	0.9	0.1	sigmoid

experimentos, porém, ao avaliar os resultados para a classificação de classes consideradas normais, o HOG com PCA de dimensionalidade 3 possui melhor desempenho. Considerando o objetivo deste trabalho, considera-se o descritor HOG sem o uso do PCA como sendo o descritor mais recomendado ao se utilizar o classificador OC-SVM.

Tabela 15 – Melhores resultados para cada PCA utilizando o classificador OC-SVM - Anomalia.

PCA	**Descritor**	**Precisão Anomalia**	**Revocação Anomalia**	**F1 Anomalia**
Sem	HOG	35.5%	99%	52.1%
2	SURF	33,2%	93,8%	49%
3	HOG	40,4%	61,8%	48,6%

Avaliando de forma geral os melhores resultados por PCA para a detecção de anomalias, observa-se valores mais altos de *revocação* em relação a *precisão*. Isto indica

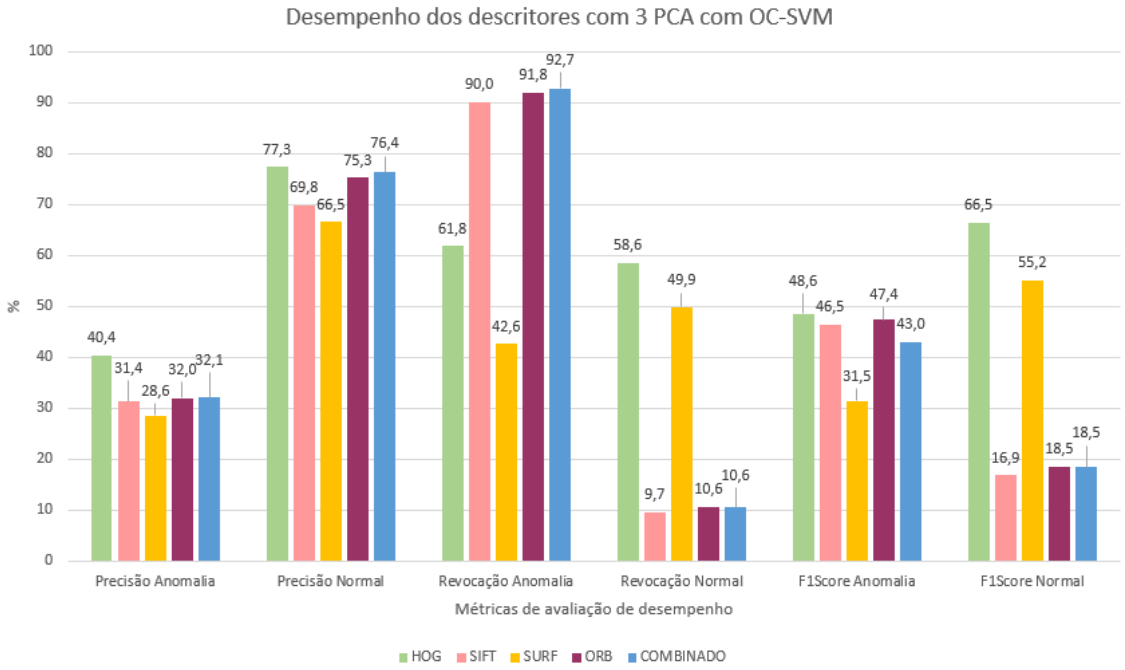

Figura 32 – Gráfico de desempenho dos descritores com 3 PCA com o classificador OC-SVM.

Tabela 16 – Melhores resultados para cada PCA utilizando o classificador OC-SVM - Normal.

PCA	Descritor	Precisão Normal	Revocação Normal	F1 Normal
Sem	HOG	97%	13,8%	29,9%
2	SURF	33,2%	93,8%	49%
3	HOG	77,3%	58,6%	66,5%

que o algoritmo classifica, na maioria das vezes, todas as instâncias na mesma classe. De forma geral, o algoritmo indicou que a maioria das imagens são anomalias, acertando bastante essa classificação (até 99%), porém a *precisão* é baixa, pois grande parte da imagens ditas anômalas não são. Ao analisar as métricas de classificação para a detecção de imagens normais, percebe-se que sem o uso do PCA o HOG possui um bom nível de *precisão*, porém um baixo *revocação*, demonstrando que o algoritmo indica poucas vezes uma imagem como normal, e na maioria das vezes ele acerta.

5.2.3 IMPLEMENTAÇÕES

Ao criar o servidor *Config Server* e um repositório *online* de configurações, foi possível administrar de forma simples e dinâmica a configuração de execução dos servidores. Com a implementação do *Eureka Server* foi possível analisar dados específicos de cada servidor, podendo assim, reconhecer falhas e consumos dos mesmos.

Por meio do modelo de classificação devidamente salvo, o passo seguinte foi criar

um *script python* que fosse capaz de importar o modelo de classificação, extrair as características e classificar a imagem de entrada.

Após a criação do *script* foi possível criar o *Anomaly Recognize Server*. A cada requisição *HTTP* na rota */classifier/single* em modo *POST* portando uma imagem, o servidor gera um comando específico e insere em um *Process*, iniciando o então uma chamada ao *script*, passando como parâmetro o caminho da imagem, ao fim da classificação, essa mesma requisição HTTP retorna como resposta o resultado da classificação. O mesmo serve para a classificação de várias imagem, sendo necessário apenas modificar a rota para */classifier/multiple*. O *front-end* implementado permite apenas o envio de uma imagem por vez.

O desenvolvimento do *View Server* e sua conexão com o *Anomaly Recognize Server* possibilitou a visualização do resultado da classificação e promoveu a facilidade no envio de novas imagens a serem classificadas.

Como visto na Figura 33, com todo o conjunto sendo executado, ao acessar a URL é possível ver a tela de classificação, onde ao clicar no nome da imagem, é possível adicionar uma nova imagem e posteriormente, clicando em "classificar", é possível realizar a classificação da imagem.

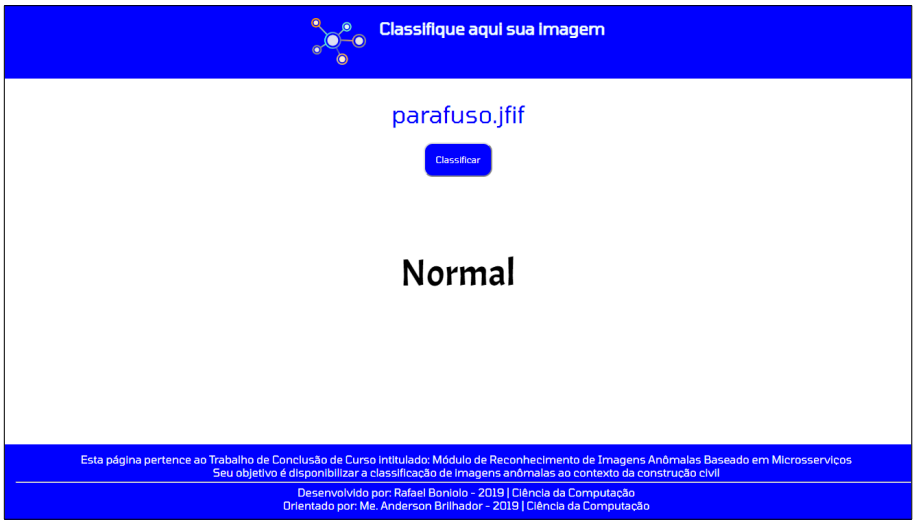

Figura 33 – Tela de envio de imagens para classificação - Normal.

No exemplo de classificação visto na Figura 33, utilizou-se uma imagem de um parafuso que é exibido na Figura 34, esta imagem não pertence ao *dataset* (seção 4.1.1). Na Figura 33, pode ser visualizado o resultado da classificação, que neste caso, retornou apontando que a imagem pertence a classe normal, acertando a classificação.

Figura 34 – Imagem de parafuso para teste da aplicação.

Aplicou-se também o teste da aplicação com imagens que são anômalas ao contexto da construção civil, utilizou-se como teste, uma imagem de de grupo de ciclistas, vista na Figura 35. Como resultado, obteve-se a detecção da anomalia, que pode ser vista na Figura 36, a aplicação obteve sucesso nesta classificação, apresentando o texto de "Anomalia" na tela.

Figura 35 – Imagem de ciclistas para teste da aplicação.

Todas as implementações neste trabalho citadas estão disponível no *GitHub*. As as informações individuais e configurações de cada repositório estão disponíveis no *Readme.md*, os *liks* seguintes referem-se ao conjunto de microsserviços utilizados e ao repositório de configurações:

View Server : *https://github.com/rafaelboniolo/TCC-ViewServer.*
Eureka Server : *https://github.com/rafaelboniolo/TCC-EurekaServer.*
Config Server : *https://github.com/rafaelboniolo/TCC-ConfigServer.*
Anomaly Recognizer Server : *https://github.com/rafaelboniolo/TCC-AnomalyRecognizerServer.*
Configurações : *https://github.com/rafaelboniolo/TCC-MicroservicesConfig.*

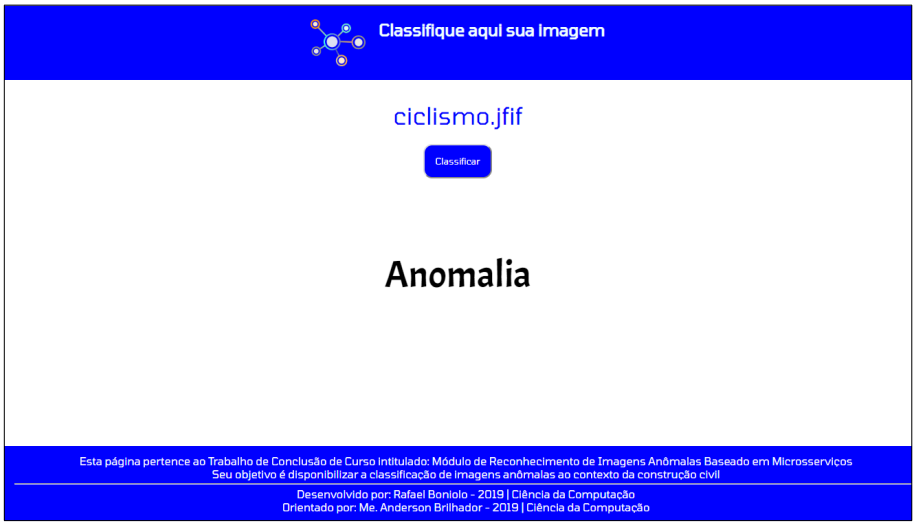

Figura 36 – Tela de envio de imagens para classificação - Anomalia.

Os próximos repositórios disponibilizam as implementações das aplicações: *Classifier* e *Web Scrapper*. O *Classifier* possui um conjunto de *scripts* utilizados para gerar todos os testes realizados neste trabalho, esta implementação contempla o conjunto de descritores, PCA, *Cross Validation*, classificadores, etc. O *Web Scrapper* trás os *scripts* utilizados para realizar o *download* das imagens. São eles:

Classifier : *https://github.com/rafaelboniolo/TCC-Classifier.*
Web Scrapper : *https://github.com/rafaelboniolo/TCC-Scrapper.*

6 CONCLUSÃO

De acordo com os experimentos realizados e exibidos nas seções 5.2.1 e 5.2.2, o descritor HOG demonstrou ser melhor opção para ambos os classificadores avaliados neste trabalho (KNN e OC-SVM).

As variações de PCA, com dimensionalidades 2 e 3, não obtiveram resultado positivo se comparado com os experimentos que não fizeram uso do PCA, visto que estes possuíram melhores desempenhos.

Dentre os de experimentos realizados com o conjunto de possíveis parâmetros visto nas Tabelas 4 e 5, o algoritmo de classificação KNN demonstrou-se superior ao classificador OC-SVM. Portanto, para tal base de dados e para o conjunto de experimentos realizados, recomenda-se o uso do classificador KNN com o descritor HOG sem o uso do PCA. A combinação desse métodos alcançou a marca de 90,9% e 92,6%, de *f1score* para classe de imagens anômala e normal, respectivamente.

Com base nos dados obtidos provenientes dos experimentos realizados nas seções 5.2.1 e 5.2.2, o módulo reconhecedor de imagens anômalas foi implementado com o modelo de classificação gerado do treinamento do KNN com o HOG. O uso da arquitetura de microsserviços permite que o modelo de classificação possa ser substituído ou incrementado sem grandes impactos negativos na aplicação.

Com o uso da arquitetura de microsserviços, foi possível isolar cada tarefa em seu determinado servidor e assim detectar e gerenciar com maior facilidade os problemas encontrados durante o desenvolvimento.

6.1 TRABALHOS FUTUROS

Propõe-se como futuros experimentos, a realização do ajuste de outros parâmetros dos classificadores, bem como o teste de mais valores para cada parâmetro. O ajuste de parâmetro para os descritores é de extrema importância para otimizar os resultados da classificação, bem como a variação da quantidade mínima de características obtidas no descritores ORB, SIFT, SURF. Propõe-se investigar e comparar novos métodos de reconhecimento de anomalias, como por exemplo, métodos baseados em *Deep learning* e aplicar algoritmos que auxiliem na variação das dimensões do PCA.

6.2 CONSIDERAÇÕES FINAIS

Este trabalho apresentou a criação de um *dataset* de imagens relacionadas a construção civil e um módulo de detecção de anomalias em imagens com variações de descritores locais e de classificadores bem como suas tecnologias. A implementação do módulo é baseada na arquitetura de microsserviços e suas tecnologias e formas de implementação.

A variação de descritores, PCA, classificadores e seus parâmetros foram essenciais para encontrar o melhor modelo de classificação para o *dataset* criado e garantir um bom resultado de classificação.

A arquitetura de software neste trabalho proposta garantiu o gerenciamento, a escalabilidade e o desenvolvimento incremental do módulo, possibilitando futuras variações nos métodos da detecção de anomalias sem impactos no conjunto final. As ferramentas disponibilizadas pelo *framework Spring* foram fundamentais na realização do gerenciamento e das configurações dinâmicas.

O método neste trabalho proposto provou-se ser eficaz na detecção de anomalias, com a inclusão de mais *tags* ao *dataset*, o método torna-se totalmente viável para ser utilizado em maior escala.

Referências

ABDI, H.; WILLIAMS, L. J. Principal component analysis. **Wiley interdisciplinary reviews: computational statistics**, Wiley Online Library, v. 2, n. 4, p. 433–459, 2010.

BALAMURALI, R.; CHANDRASEKAR, A. Multiple parameter algorithm approach for adult image identification. **Cluster Computing**, EUA, p. 1–9, 2018.

BAY, H. et al. Speeded-up robust features (surf). **Computer vision and image understanding**, EUA, v. 110, n. 3, p. 346–359, 2008.

BRADSKI, G. The OpenCV Library. **Dr. Dobb's Journal of Software Tools**, 2000.

BRILHADOR, A. **Análise semi-automática do arranjo espacial de plantas de milho utilizando visão computacional**. Dissertação (Mestrado em Informática) — Universidade Tecnológica Federal do Paraná, Cornélio Procópio, PR, 2015.

BRILHANTE, J. L. G. A. P. et al. Mestrado em Informática, **Uma abordagem para construção de microsserviços reativos baseadas em filas assíncronas**. [S.l.]: Universidade Federal da Paraíba, 2018.

BUENO, L. M. **Análise dos descritores locais de imagens no contexto de detecção de semi-réplicas**. Dissertação (Mestrado em Ciência da Computação) — Universidade Estadual de Campinas, Instituto de Computação, Campinas, SP, 2011.

CASTRO, C. L. d.; BRAGA, A. A. P. A. Aprendizado supervisionado com conjuntos de dados desbalanceados. **Sba: Controle & Automação Sociedade Brasileira de Automática**, scielo, v. 22, p. 441 – 466, 10 2011. ISSN 0103-1759.

CAVALCANTI, L. C. A. M. **Detecção de elementos antrópicos em imagens aéreas da floresta amazônica**. Dissertação (Mestrado em Informática) — Universidade Federal do Amazonas, Instituto de Computação, Manaus, AM, 2016.

CAVALCANTI, V. Y. S. de L. et al. Indústria 4.0: desafios e perspectivas na construção civil. **Revista Campo do Saber**, Cabedelo, PB, v. 4, n. 4, 2018.

CBIC. **A Construção Civil pode dar um novo ânimo à economia**. 2018. Disponível em: <https://cbic.org.br/a-construcao-civil-pode-dar-um-novo-animo-a-economia-2>. Acesso em: 3 mar. 2019.

COPPIN, B. **Inteligência artificial**. [S.l.]: Grupo Gen-LTC, 2015.

COSTA, G. d. B. P. d. **Detecção de anomalias utilizando métodos paramétricos e múltiplos classificadores**. Dissertação (Mestrado em Ciências - Ciências da Computação e Matemática Computacional) — Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, São Carlos, SP, 2014.

COSTA-LUIS, C. O. D. tqdm: A fast, extensible progress meter for python and cli. **Journal of Open Source Software**, v. 4, n. 37, p. 1277, 2019.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: SCHMID, C.; SOATTO, S.; TOMASI, C. (Ed.). **International Conference on Computer Vision & Pattern Recognition (CVPR '05)**. San Diego, United States: IEEE Computer Society, 2005. v. 1, p. 886–893.

DRAGONI, N. et al. Microservices: Yesterday, today, and tomorrow. In: _____. **Present and Ulterior Software Engineering**. Cham: Springer International Publishing, 2017. p. 195–216. ISBN 978-3-319-67425-4. Disponível em: <https://doi.org/10.1007/978-3-319-67425-4_12>.

EITELVEIN, L. **Implementação e avaliação de um mecanismo de detecção de anomalias em uma ferramenta smart grid**. Monografia (Bacharelado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul, Instituto de Informática, Porto Alegre, RS, 2015.

FACELI, K. et al. **Inteligência artificial: uma abordagem de aprendizado de máquina**. [S.l.]: LTC, 2011.

Gadea, C. et al. A microservices architecture for collaborative document editing enhanced with face recognition. In: **2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)**. [S.l.: s.n.], 2016. p. 441–446. ISSN null.

GAMMA, E. **Padrões de projetos: soluções reutilizáveis**. São Paulo, SP: Bookman editora, 2009.

GONZALEZ, R.; WOODS, R. **Processamento de imagens digitais**. 1. ed. São Paulo, SP: Edgard Blucher, 2000.

GONZALEZ, R. C.; WOODS, R. C. **Processamento digital de imagens.** 3. ed. São Paulo, SP: Pearson, 2010.

Harisinghaney, A. et al. Text and image based spam email classification using knn, naïve bayes and reverse dbscan algorithm. In: **2014 International Conference on Reliability Optimization and Information Technology (ICROIT)**. [S.l.: s.n.], 2014. p. 153–155.

Hasselbring, W.; Steinacker, G. Microservice architectures for scalability, agility and reliability in e-commerce. In: **2017 IEEE International Conference on Software Architecture Workshops (ICSAW)**. [S.l.: s.n.], 2017. p. 243–246. ISSN null.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing in Science & Engineering**, v. 9, n. 3, p. 90–95, 2007.

JOBLIB. **Joblib: running Python functions as pipeline jobs**. 2019. Disponível em: <https://joblib.readthedocs.io/en/latest/index.html>. Acesso em: 18 nov. 2019.

JOSÉ, I. **KNN (K-Nearest Neighbors) #1**. 2018. Disponível em:<https://medium.com/brasil-ai/knn-k-nearest-neighbors-1-e140c82e9c4e>. Acesso em: 25 out. 2019.

KRISHNI. **An introduction to Grid Search**. 2019. Disponível em:<https://medium.com/datadriveninvestor/an-introduction-to-grid-search-ff57adcc0998>. Acesso em: 21 nov. 2019.

Krylovskiy, A.; Jahn, M.; Patti, E. Designing a smart city internet of things platform with microservice architecture. In: **2015 3rd International Conference on Future Internet of Things and Cloud**. [S.l.: s.n.], 2015. p. 25–30. ISSN null.

LINDEBERG, T. Scale invariant feature transform. **Scholarpedia**, v. 7, n. 5, p. 10491, 2012.

LIU, H.; MOTODA, H. **Feature selection for knowledge discovery and data mining**. [S.l.]: Springer Science & Business Media, 2012. v. 454.

LOWE, D. G. Object recognition from local scale-invariant features. **International Conference on Computer Vision**, Kerkyra, Grécia, v. 99, p. 1150–1157, 1999.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. **International Journal of Computer Vision**, v. 60, n. 2, p. 91–110, Nov 2004. ISSN 1573-1405.

LUKASHEVICH, H.; NOWAK, S.; DUNKER, P. Using one-class svm outliers detection for verification of collaboratively tagged image training sets. **2009 IEEE International Conference on Multimedia and Expo**, New York, NY, p. 682 – 685, 2009.

MACHADO, W. R. S. et al. Estudo e proposta de adaptação do algoritmo sift em relação ao problema de iluminação em imagens. **Anais do WVC'2008, IV workshop de visão computacional**, Bauru, SP, 2008.

MCKINNEY, W. Data structures for statistical computing in python. In: WALT, S. van der; MILLMAN, J. (Ed.). **Proceedings of the 9th Python in Science Conference**. [S.l.: s.n.], 2010. p. 51 – 56.

MELOCA, R. M. **Um comparativo entre frameworks para microsserviços**. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2017.

MOREIRA, D. et al. Pornography classification: The hidden clues in video space–time. **Forensic science international**, v. 268, p. 46–61, 2016.

MOREIRA, P. F. M.; BEDER, D. M. Desenvolvimento de aplicações e micro serviços: Um estudo de caso. **Revista TIS**, São Carlos, SP, v. 4, n. 3, p. 209–215, 2016. Disponível em: <http://www.revistatis.dc.ufscar.br/index.php/revista/article/view/364/127>. Acesso em: 20 abr. 2019.

NAMIOT, D.; SNEPS-SNEPPE, M. On micro-services architecture. **International Journal of Open Information Technologies**, v. 2, n. 9, p. 24–27, 2014.

NEWMAN, S. **Building microservices: designing fine-grained systems**. Sebastopol, CA: O'Reilly Media, 2015. Disponível em: <http://cds.cern.ch/record/2010622>.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PEDRINI, H.; SCHWARTZ, W. **Análise de imagens digitais: princípios, algoritmos e aplicações**. São Paulo, SP: Thomson Learning, 2008.

PERDISCI, R. et al. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. **Sixth International Conference on Data Mining**, Hong Kong, p. 488–498, 2006.

PIOTTO, J. G. d. S. **Reconhecimento facial usando descritores locais e redes complexas**. Dissertação (Mestrado em Informática) — Universidade Tecnológica Federal do Paraná, Cornélio Procópio, PR, 2016.

RAMTEKE, R.; MONALI, Y. K. Automatic medical image classification and abnormality detection using k-nearest neighbour. **International Journal of Advanced Computer Research**, International Journal of Advanced Computer Research(IJACR), v. 2, n. 4, p. 190–196, 2012.

RIBEIRO, J. L. S. **Arquitetura baseada em microsserviços para classificação de dados**. Monografia (Bacharelado em Engenharia de Software) — Universidade Federal do Rio Grande do Norte, 2018.

ROSA, T. P. **UM MÉTODO PARA O DESENVOLVIMENTO DE SOFTWARE BASEADO EM MICROSSERVIÇOS.** Monografia (Bacharelado em Engenharia de Software) — Universidade Federal do Ceará, Quixadá, CE, 2016.

ROSIN, P. L. Measuring corner properties. **Computer Vision and Image Understanding**, v. 73, n. 2, p. 291–307, 1999.

RUBLEE, E. et al. Orb: An efficient alternative to sift or surf. **International Conference on Computer Vision**, Barcelona, v. 11, n. 1, p. 2, 2011.

Said, Y.; Atri, M.; Tourki, R. Human detection based on integral histograms of oriented gradients and svm. In: **2011 International Conference on Communications, Computing and Control Applications (CCCA)**. [S.l.: s.n.], 2011. p. 1–5.

SANTANA, B. A. S. et al. Análise de desempenho de algoritmos detectores de keypoints para um sistema de navegação visual de robôs baseados em smartphones. **Conference on Graphics, Patterns and Images, 28. (SIBGRAPI)**, Porto Alegre, RS, 2015.

SANTOS, L. **Microserviços: dos grandes monólitos às pequenas rotas**. 2017. Disponível em:<https://medium.com/trainingcenter/microservi%C3%A7os-dos-grandes-mon%C3%B3litos-%C3%A0s-pequenas-rotas-adb70303b6a3>. Acesso em: 21 nov. 2019.

SCIKIT-LEARN. **Cross-validation: evaluating estimator performance**. 2019. Disponível em: <https://scikit-learn.org/stable/modules/cross_validation.html>. Acesso em: 14 out. 2019.

SYARIF, I.; PRUGEL-BENNETT, A.; WILLS, G. Svm parameter optimization using grid search and genetic algorithm to improve classification performance. **TELKOMNIKA (Telecommunication Computing Electronics and Control)**, v. 14, p. 1502, 12 2016.

TEIXEIRA, L. P.; CARVALHO, F. A.; SILVA, J. M. A. Desempenho da construção brasileira no período 1990-2008. **RDE - Revista de Desenvolvimento Econômico**, Salvador, BA, v. 14, n. 25, 2012.

THEILER, J. P.; CAI, D. M. Resampling approach for anomaly detection in multispectral images. **Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery IX**, Orlando, Florida, v. 5093, p. 230–241, 2003.

Thönes, J. Microservices. **IEEE Software**, v. 32, n. 1, p. 116–116, Jan 2015.

TIAN, C. et al. Pornographic image classification based on top down color-saliency based bow representation. **Proceedings 2014 IEEE International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)**, Wuhan, Hubei, p. 273–278, 2014.

VILLAMIZAR, M. et al. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In: . [S.l.: s.n.], 2015.

Villamizar, M. et al. Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures. In: **2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)**. [S.l.: s.n.], 2016. p. 179–182.

WALT, S. v. d.; COLBERT, S. C.; VAROQUAUX, G. The numpy array: A structure for efficient numerical computation. **Computing in Science & Engineering**, v. 13, n. 2, p. 22–30, 2011.

WALT, S. van der et al. scikit-image: image processing in python. **PeerJ**, v. 2, p. e453, jun. 2014. ISSN 2167-8359.

WEHRMANN, J. et al. Adult content detection in videos with convolutional and recurrent neural networks. **Neurocomputing**, v. 272, p. 432–438, 2018.

WEISSMANN, H. **Vire o jogo com Spring Framework**. São Paulo, SP: Casa do Código, 2014.

WOLD, S.; ESBENSEN, K.; GELADI, P. Principal component analysis. **Chemometrics and intelligent laboratory systems**, Elsevier, v. 2, n. 1-3, p. 37–52, 1987.