UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE LORENA

MARIANA OLEONE MARINHO DE CASTRO

Estudo de visão computacional e criação de um *Haar Cascade* utilizando a biblioteca *OpenCV* em *Python* para detecção de objetos

Lorena
2021

MARIANA OLEONE MARINHO DE CASTRO

Estudo de visão computacional e criação de um *Haar Cascade* utilizando a biblioteca *OpenCV* em *Python* para detecção de objetos

Trabalho de Graduação apresentado à Escola de Engenharia de Lorena da Universidade de São Paulo como requisito parcial para conclusão da Graduação do curso de Engenharia Física.

Orientadora: Profª. Drª. Mariana Pereira de Melo

Lorena
2021

Ficha catalográfica elaborada pelo Sistema Automatizado
da Escola de Engenharia de Lorena,
com os dados fornecidos pelo(a) autor(a):

De Castro, Mariana Oleone Marinho
Estudo de Visão computacional e criação de um *Haar Cascade* utilizando a biblioteca *OpenCV* em *Python* para detecção de objetos
Mariana Oleone Marinho de Castro; orientadora Profª. Drª. Mariana Pereira de Melo. - Lorena, 2021.

Nº p. 44 Monografia apresentada como requisito parcial para a conclusão de Graduação do Curso de Engenharia Física - Escola de Engenharia de Lorena da Universidade de São Paulo. 2021

1. Aprendizado de máquina. 2. Visão computacional. 3. *Python*. 4. *OpenCV*. 5. *Haar Cascade*. I. Estudo de Visão computacional e criação de um *Haar Cascade* utilizando a biblioteca *OpenCV* em *Python* para detecção de objetos. II. De Melo, Mariana Pereira, orient.

Dedico este trabalho aos meus pais, Rafael Marinho de Castro e Márcia Oleone de Oliveira, e minha irmã, Cecília Oleone Marinho de Castro, por serem as pessoas que me motivam a lutar e ser forte todos os dias, e por serem as pessoas mais importantes da minha vida.

Agradecimentos

Os agradecimentos deste trabalho vão primeiramente para minha família, que me ensinaram todos os valores que hoje acredito e sigo, por sempre estarem ao meu lado me apoiando em todas as minhas decisões e serem a melhor família que eu poderia ter.

Especialmente aos meus pais, Rafael e Márcia, por me darem apoio para estar hoje concluindo meu curso, se não fosse por eles, jamais chegaria onde estou. À minha irmã, meu melhor presente e porto seguro, por ter me acompanhado em todas as fases da minha vida e nunca ter soltado minha mão. Cecília, faço e farei de tudo por você, pra sempre.

Não poderia deixar de agradecer ao meu avô Antônio e minha bisavó Wilma, que hoje são estrelas lindas neste céu, e pelas minhas duas avós queridas, Ana Maria e Marlene, e minha tia Marlúcia. Eles foram os meus familiares mais próximos e com os quais convivi todos os momentos da minha vida, sempre me apoiando com muito amor.

Agradeço também às minhas amigas Gabriela Medeiros e Carolina Rocha, por serem minhas irmãs que a vida me deu. Com elas aprendi o verdadeiro significado de amizade e companheirismo.

Um grande e especial agradecimento aos meus dois companheiros do dia a dia: meus cachorros Ozzy e Joey. Com eles, não existe um dia de solidão, obrigada por serem os melhores cachorros que eu poderia ter.

E por fim, não poderia deixar de agradecer ao Murilo Bigoto, um companheiro muito especial de vida, que esteve ao meu lado, de tantas formas diferentes, em praticamente toda minha jornada na faculdade e fora dela. Obrigada por tudo.

À todos, eu os amo muito, agradecimentos não são suficientes para verbalizar a gratidão e amor que sinto.

Resumo

Os estudos e as aplicações de visão computacional estão cada vez mais presentes em nosso cotidiano. Seus avanços trazem contribuições para problemáticas em áreas de Engenharia, no controle de qualidade, na área de segurança, detecção de objetos e até mesmo na identificação de pessoas em fotos do celular, dentre outras. Este trabalho teve como objetivo a criação de um detector de objetos do tipo *Haar Cascade* com a biblioteca *OpenCV* utilizando o *Python*. Foram criados três diferentes tipos de classificadores para a identificação de canecas. Foi possível, a partir de um banco de dados contendo 3200 imagens do tipo negativas - imagens que não contém o objeto-alvo canecas - criar uma amostra de imagens positivas - imagens que contém o objeto-alvo canecas. Essas imagens positivas serviram de insumo para a criação de um ou mais vetores que contém as informações de coordenadas das canecas nas imagens. Para o classificador 1 foi criado um vetor com um tipo de caneca nas imagens positivas, já para os classificadores 2 e 3, foram criados dez vetores com dez tipos de canecas nas imagens positivas. A diferença entre o classificador 2 e 3 é que este último possui seus parâmetros de treino ajustados de modo a melhorar a performance dos resultados. O classificador 3, de fato, foi o melhor classificador construído e, por isso, foi utilizado para testes com imagens em tempo real, com o uso da *webcam*, mostrando resultados bastante satisfatórios nesta última etapa.

Palavras-chave: Aprendizado de máquina, Visão computacional, *Python*, *OpenCV*, *Haar Cascade*.

Abstract

The studies and applications of computer vision are increasingly present in our daily lives. Their advances bring many contributions to problems in Engineering areas, factory processes, security, objects and people detection in cell phone pictures, among others. In this work, we aimed to create a *Haar Cascade* object detector with the *OpenCV* library using *Python*. Three different types of classifiers were created for mug identification. It was possible, from a database containing 3200 negative images - images that do not contain the target object mugs - to create a sample of positive images - images that contain the target object mugs. These positive images served as input for the creation of one or more vectors that contain the coordinate information of the mugs in the images. For classifier 1 a vector with one type of mug in the positive images was created, while for classifiers 2 and 3, ten vectors with ten types of mugs in the positive images were created. The difference between classifier 2 and 3 is that classifier 3 has its training parameters adjusted in order to improve the performance of the results. Classifier 3, in fact, was the best classifier built based on results obtained and could be used for testing with real-time images using the *webcam*.

Keywords: Machine learning, Computer vision, *Python*, *OpenCV*, *Haar Cascade*.

Lista de Ilustrações

Lista de Tabelas

Sumário

1 Introdução

As pesquisas e estudos sobre aprendizado de máquina se iniciaram há aproximadamente 70 anos atrás, e seu objetivo é ensinar o computador a partir da análise de dados e informações para que, de forma rápida, ele simule decisões humanas. Em seu início, não existiam computadores capazes de realizar estes estudos por exigirem um processamento muito grande. (CHAVES, 2012)

Com o avanço da tecnologia, atualmente já é possível realizar diversos tipos de estudos com o aprendizado de máquina, gerando assim, diversas aplicações utilizadas em nosso cotidiano, como assistentes virtuais, detecção de rostos em fotos no celular, sistemas de recomendação de vídeos nas redes sociais, dentre outras.

Existem três tipos de aprendizados de máquina, o aprendizado supervisionado, o não supervisionado e o aprendizado por reforço. Estes aprendizados possuem diferentes métodos de aplicação. No aprendizado supervisionado o computador conhece os dados e as decisões a serem tomadas, e armazena estas informações para aplicações em dados não conhecidos. No aprendizado não supervisionado, não se tem conhecimento sobre a classificação alvo, sendo necessário o computador aprender o comportamento dos dados ao longo do tempo. Por fim, o aprendizado por reforço compreendem metodologias onde o computador aprende o critério de classificação por tentativa e erro, com base em informações a serem fornecidas ao longo do tempo. (MONARD; BARANAUSKAS, 2003)

Restringindo um pouco o vasto conceito sobre aprendizado de máquina, temos a visão computacional. A visão computacional é uma área do aprendizado de máquina que visa ensinar o computador a identificar padrões em imagens e vídeos, tanto estático como em movimento. Estes padrões podem ser utilizados para a identificação de faces, classificação de objetos, ou até mesmo na área de controle de qualidade. (MILANO; HONORATO, 2014)

O presente trabalho fez uso do aprendizado supervisionado para uma aplicação de visão computacional, com o objetivo da criação de um *Haar Cascade*, um método de classificador de imagens e vídeos, para a identificação de um objeto-alvo. O objeto escolhido foi uma caneca e foram criados três tipos diferentes de classificadores com o auxílio da bibilioteca para visão computacional *OpenCV*, e linguagem computacional em *Python*.

O primeiro classificador foi construído utilizando uma imagem de caneca como referência para o computador aprender o que é o objeto e como identificá-lo em imagens. Os outros dois classificadores construídos utilizaram dez tipos de canecas diferentes para aumentar a detecção de canecas variadas, sendo que o último classificador foi criado com a otimização de parâmetros de treinamento do segundo classificador.

Com isso, foi possível realizar alguns testes, em imagens contendo ou não canecas, e

assim, comparar o desempenho dos três classificadores construídos. Para treinos mais robustos e de maior precisão, são necessários ajustes de parâmetros cujo processamento pode durar meses. Para o escopo deste trabalho, o classificador de melhor desempenho demorou cerca de 21 horas para ser treinado em um computador de uso pessoal.

Além disso, utilizou-se o classificador que obteve melhores resultados na etapa anterior para testes em tempo real, com o uso da *webcam*. Foi testado um caso em que existia uma caneca, e um caso em que continha um copo e, outro, uma taça. O classificador obteve resultados esperados, ou seja, identificou a caneca e não identificou incorretamente o copo e a taça como uma caneca.

Uma revisão teórica dos conceitos utilizados é apresentada na Seção 2, seguida pela metodologia apresentada na Seção 3, enquanto os resultados e as discussões são apresentados na Seção 4, finalizando com a conclusão na Seção 5.

2 Fundamentação Teórica

2.1 Aprendizado de máquina

Atualmente, é possível capturar dados e informações a todo momento e em diversos tipos de aparelhos eletrônicos. Pode-se, por exemplo, detectar padrões de sono de acordo com o uso de um relógio inteligente, agendar lembretes e compromissos com assistentes virtuais, ou até mesmo detectar o padrão de rostos na galeria de fotos do celular. Tudo está cada vez mais conectado e a inteligência articial segue ganhando cada vez mais espaço em nosso cotidiano.

O aprendizado de máquina é uma área de estudos dentro da inteligência artificial que consiste em ensinar a máquina a aprender de forma inteligente e automática (MONARD; BARANAUSKAS, 2003). O estudo acadêmico do aprendizado de máquina teve seu início por volta de 1950, propondo avanços significativos na tecnologia da época, como a proposta de que uma máquina poderia pensar de forma semelhante ao ser humano, com treinos e testes (CHAVES, 2012).

Entretanto, essa proposta não pôde ser aplicada quando criada, pois os computadores disponíveis naquela época não tinham capacidade para suportar esse tipo de processamento. Porém, atualmente, já é possível realizar os treinamentos que foram propostos a cerca de 70 anos atrás, e os avanços do aprendizado de máquina são encontrados em diversas situações do nosso dia a dia.

Em suma, o aprendizado de máquina faz uso de dois conjuntos de dados, de treino e de teste. A partir da análise do conjunto de dados de treino, é possível realizar o ajuste de parâmetros e a detecção de padrões. Essa etapa fornece ao computador informações capazes de serem aplicadas a outro conjunto de dados, os dados de teste. Assim, o conjunto de dados de teste não é utilizado para ajuste de parâmetros e são desconhecidos pelo computador, sendo utilizados para testar os modelos feitos com os dados de treino, validando a qualidade do modelo proposto e obtendo os resultados e análises para tais modelos (ALPAYDIN, 2020).

Este processo permite que o computador aprenda a partir de informações recebidas, reajustando parâmetros e atualizando o modelo constantemente, se aprimorando e se tornando cada vez mais capaz de detectar as soluções dos problemas propostos. Isso se caracteriza um aprendizado de máquina.

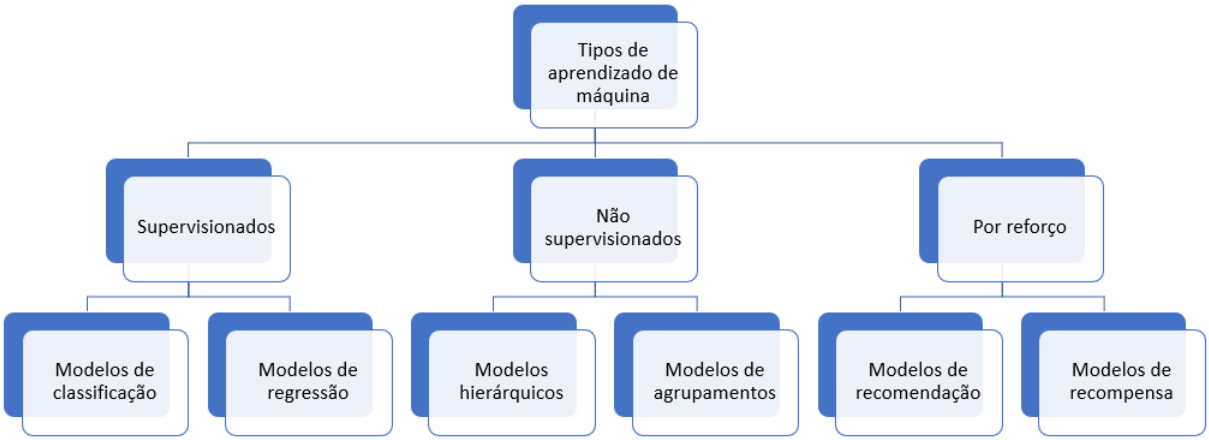
O aprendizado de máquina requer cuidado ao ajustar os parâmetros. Há casos em que os ajustes dos parâmetros realizados são muitos ao ponto do computador memorizar as respostas desejadas, o que é negativo, visto que não se leva em conta as variações existentes dos dados. Esse caso é chamado de *overfitting*. Um cenário de *overfitting* ocorre quando o modelo tem um excelente desempenho aplicado ao conjunto de dados de treino, mas se mostra ineficaz para

prever resultados quando aplicado ao conjunto de testes.

Por outro lado, há casos em que o modelo não é capaz de ajustar parâmetros, o que também provoca um resultado negativo, visto que não é possível detectar padrões nos dados, esse caso é chamado de *underfitting*. No caso de *underfitting*, o desempenho do modelo já é ruim no próprio treinamento, sendo incapaz de encontrar relações entre as variáveis. Em ambos os casos, o modelo não terá utilidade. (GÉRON, 2019)

Há três tipos de aprendizagem de máquina, como ilustrado na Figura 2.1.

Figura 2.1 – Tipos de aprendizado de máquinas e alguns exemplos de modelos mais utilizados.



Fonte: Autora.

Para um aprendizado supervisionado, é necessário fornecer uma amostra inicial para que a máquina aprenda com essa amostra e seja possível replicar em um banco de dados a ser estudado. Esta amostra inicial, que são os dados de treino, possui resultados pré definidos, que referenciam como os dados de saída devem se comportar. Os dados de saída armazenam o comportamento do modelo e o replica para os dados de teste. Estes dados de teste, desconhecidos pelo computador, serão utilizados para realizar estudos com as mesmas definições dos dados de treino.

Dentre os modelos de aprendizado supervisionado, os mais utilizados são os modelos de classificação e de regressão. O modelo de classificação utiliza padrões dos dados de treino para categorizar variáveis da base de dados. Neste caso, as categorizações são não numéricas como, por exemplo, classificações de gênero ou raça, faixa salarial com base em atividades financeiras, dentre outras classificações possíveis. Já o modelo de regressão utiliza variáveis numéricas, como saídas binárias (0 ou 1). Essas variáveis classificam a ocorrência ou não de algum evento de interesse na base de dados. Se o evento de interesse ocorreu atribui-se o valor 1 e, se o evento não ocorreu, atribui-se o valor 0. Essas padronizações nas bases de treino para estes modelos

geram insumos para que o modelo aprenda e replique este conhecimento na base de teste.

No aprendizado não supervisionado a amostra inicial não é avaliada para construção do modelo. A máquina identifica padrões e aprende ao longo do tempo. Neste caso, os eventos não são bem definidos como nos casos de aprendizados supervisionados. Para isso, o computador aprende com os padrões de forma exploratória e classifica gradativamente o comportamento da base de dados.

Os modelos de aprendizado não supervisionado mais utilizados atualmente são os modelos hierárquicos e agrupamentos. O modelo hierárquico é um tipo de modelo que cria uma árvore de decisões, onde o conjunto de dados é dividido em diversos subgrupos. Esse método analisa as variações de comportamento da base de dados até que estes comportamentos sejam padronizados.

No caso do agrupamento, o computador classifica a base de dados em *clusters*, ou centros, que são medidos com base na média das distâncias de comportamento de cada grupo identificado. Faz-se necessário escolher a quantidade de centros desejados, e a máquina classifica e agrupa as unidades do conjunto de dados que estão mais próximas uma das outras, até que o número de grupos formados seja a quantidade definida inicialmente.

Por fim, o aprendizado por reforço é aquele em que a máquina aprende com os erros indicados e melhora o modelo ao longo do aprendizado. Este aprendizado é adotado por métodos em que é necessário tomar seguidas decisões sobre os dados, até que essas decisões sejam próximas ao desejado ao final do treinamento, por tentativa e erro.

Os tipos de aprendizado por reforço mais utilizados são os métodos de recomendação e recompensa. Para os métodos de recomendação, o computador aprende com base na sugestão de uma decisão para o usuário, e armazena o aprendizado quando o usuário responde à ação. Por exemplo, no caso da recomendação de filmes e séries em plataformas de *streamings*, caso o usuário assista a recomendação, o computador entende que foi uma boa recomendação, caso contrário, aprende que esta recomendação não foi boa. Já no aprendizado de recompensa, o aprendizado é baseado em atitudes do computador em busca do melhor resultado. Por exemplo, ao ensinar o computador a jogar um jogo, a recompensa é a vitória, e o computador vai armazenando as melhores jogadas até conseguir entender o funcionamento completo do jogo e seja capaz de vencer sempre.

O tipo de aprendizagem de máquina utilizado neste trabalho é o aprendizado de máquina supervisionado. Foi necessário informar ao programa quais eram as imagens positivas e negativas - definições explicadas posteriormente -, com isso, o programa pôde aprender com as informações fornecidas e replicar os resultados em amostras de teste, neste caso, imagens.

2.2 Visão computacional

A visão computacional, uma subárea da aprendizagem de máquina, consiste na forma como o computador interpreta fotografias e vídeos e, a partir destes resultados, é capaz de encontrar padrões. Estes padrões permitem a extração de informações que servem para a manipulação de dados em estudos de imagens (MILANO; HONORATO, 2014).

O controle de padrões feito pela visão computacional pode ser até mais seguro do que a visão humana, por armazenarem os padrões e realizarem classificações de forma fácil e rápida para as problemáticas do dia a dia. Visto isso, ela pode ser utilizada para o reconhecimento facial, classificações de emoções, para segurança com a detecção de presença ou reconhecimento de pessoas, além do controle de qualidade em linhas de montagem. Uma aplicação usual e bastante conhecida é o reconhecimento e classificação de pessoas ou objetos em fotos tiradas com o celular pelo *Google*.

Diante do cenário tecnológico atual, busca-se cada vez mais que máquinas realizem operações repetitivas e padronizadas, de modo a tornar mais rápida a realização e conclusão das tarefas. Com o uso da visão computacional, é possível automatizar inúmeras atividades que até então eram feitas apenas por humanos, como o reconhecimento de faces, de assinaturas, de impressões digitais e também de objetos. (RUDEK; COELHO; JR, 2001)

O reconhecimento de padrões em imagens e vídeos, armazena informações que possibitam que a máquina consiga, além de distinguir diferentes imagens, também encontrar as semelhanças entre elas. O fluxo para um reconhecimento de padrão é representado pela Figura 2.2 abaixo.

Figura 2.2 – Fluxo para a obtenção e reconhecimento de padrões em uma imagem.

Fonte: Adaptado de Rudek, Coelho e Jr (2001)

De acordo com Rudek, Coelho e Jr (2001), este fluxo resume um processo genérico para um estudo de visão computacional. Para a etapa de aquisição de imagem, faz-se necessário o uso de equipamentos, como câmeras, para o registro da imagem ou vídeo, ou de programas de manipulação de imagens. Além disso, um software é necessário para realizar a leitura e interpretação dessa imagem para a máquina.

Atualmente, a forma mais difundida para leitura e interpretação de imagens é através do *Python*, com o uso da biblioteca *OpenCV*, ambos utilizados neste trabalho.

O pré-processamento da imagem deve-se ao fato de manipular certos parâmetros para que o modelo entenda e performe da melhor forma possível. Nesta etapa é feita, por exemplo, a

alteração de formato e/ou tamanho e/ou cor da imagem.

No segundo passo, realiza-se a segmentação das imagens, que é um método de separação de regiões de relevância para o modelo, fazendo com que o tempo e o custo operacional sejam reduzidos. Um dos métodos mais utilizados nesta etapa é o *método de vetores*, que também será utilizado neste trabalho.

A identificação do alvo desejado, isto é, do objeto de interesse, é dada pelo armazenamento do padrão das regiões de segmentação. Estas imagens são imagens de treino, ou imagens chamadas *positivas*. Uma vez armazenadas pela máquina e ajustados seus parâmetros, o modelo poderá realizar testes para avaliação de desempenho.

Os testes utilizando imagens diversas, que podem conter ou não o alvo desejado, é importante para realizar a avaliação do modelo criado que, posteriormente, pode ser aplicado à problemáticas, caso alcance desempenho desejável.

2.3 Interface e linguagem de programação

2.3.1 A linguagem *Python*

Python é uma linguagem de programação muito utilizada pela sua sintaxe bem definida e clara aos usuários. Consiste em uma programação orientada a objetos, que permite a leitura em seu código-fonte. É possível fazer o uso de bibliotecas, que são módulos pré-programados com funções prontas e disponíveis para serem utilizadas. Além disso, é composto de estruturas de alto nível, como dicionários e listas. (BORGES, 2014)

O começo do desenvolvimento da linguagem *Python* foi iniciada por Guido van Rossum, matemático programador holandês, por volta de 1980, e foi inspirada nas linguagens C e C++. Também vale ressaltar que *Python* é considerado um software livre, que pode ser instalado facilmente em diversos tipos de plataformas e possibilita também realizar uma portabilidade de plataformas de maneira fácil, de tipagem automática e forte. Além disso, o código em si é legível de forma simples, o que o torna mais didático seu aprendizado. (COELHO, 2007)

De acordo com a pesquisa realizada em 2019 pela Spectrum (2019), o *Python* é a linguagem mais utilizada atualmente no mundo, e possui uma gigante comunidade de programadores contribuindo para a criação de códigos e bibliotecas.

2.3.2 *Jupyter Notebook*

O *Jupyter Noteboook* é um aplicativo *web* muito utilizado para o desenvolvimento de projetos de código aberto em computação, na qual interage com diversas linguagens de programação, dentre elas, o *Python*. Possui fácil instalação e pode ser utilizado para desenvolver equações,

visualizar textos e fotos, transformar e tratar informações, construir gráficos e modelos a partir de um conjunto de dados, dentre outras aplicações. (JUPYTER, 2021)

O *Jupyter Notebook* foi utilizado neste trabalho para a interpretação dos códigos em *Python* e a visualização dos resultados obtidos neste estudo.

2.3.3 Biblioteca *Matplotlib*

A biblioteca *Matplotlib* faz uso do método de matrizes para criação de gráficos bidimensionais. Ela foi criada para ser simples e prática ao programador, de modo a possibilitar a criação ou visualização das informações com poucas ou até mesmo uma linha de código, preservando uma boa qualidade nas cores e textos inclusos. (HUNTER, 2021)

Esta biblioteca foi utilizada neste trabalho com a finalidade de visualizar, de forma lúdica e prática, os resultados obtidos com o tratamento das imagens e vídeos pela biblioteca *OpenCV*.

2.3.4 Biblioteca *OpenCV* e o método *Haar Cascade*

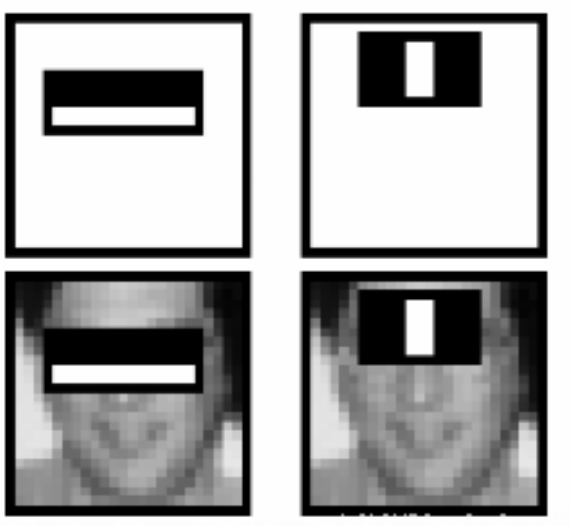
OpenCV é uma biblioteca voltada para visão computacional e suas aplicações, que possui código-fonte aberto, ou seja, é possível ver todo o passo-a-passo da criação das funções que constituem esta biblioteca. Ela possui uma estrutura em módulos de visão computacional que são capazes de tratar e analisar vídeos e fotos, detectar objetos, dentre outras aplicações. (OPENCV, 2021c)

O *Haar Cascade* é um método de treino para detecção de objetos que pertence a biblioteca *OpenCV*. Este método, criado pelos pesquisadores de visão computacional Paul Viola e Michael Jones, é conhecido como classificador em cascata, que consiste em realizar treinos e testes com aprendizado de máquina para a detecção de padrões em imagens negativas (imagens que não contém o objeto a ser detectado) e positivas (imagens que contém o objeto a ser detectado), este método utilizado é classificado como um aprendizado de máquina supervisionado. (OPENCV, 2021b)

De acordo com OpenCV (2021b), neste método são utilizados retângulos brancos e pretos em uma imagem na escala cinza, de forma a preenchê-la totalmente. Após este passo, o algoritmo consegue padronizar a soma desses retângulos de modo que seja possível distinguir diferentes imagens, ou até mesmo detectar regiões de somas padronizadas (objetos semelhantes) em imagens diversas. Essa soma é feita pela tonalidade de imagens em cada região de detecção de padrão e essas distribuições de retângulos são realizadas diversas vezes, de forma a encontrar o melhor limiar de classificação geral. Este tipo de método é chamado *Adaboost* e um exemplo pode ser visto na Figura 2.3.

As imagens do lado esquerdo da Figura 2.3 representam um padrão encontrado, em que o classificador detectou que a região dos olhos e da sobrancelha é mais escura que a região das bochechas e do nariz. Ademais, as imagens do lado direito evidenciam que a região dos olhos é

Figura 2.3 – Tipos de classificações de padrões com o método *Adaboost*.



Fonte: Adaptado de OpenCV (2021b)

mais escura que a região entre as sobrancelhas. Esses classificadores, somados, são os recursos utilizados pelo método para padronizar uma imagem e identificar padrões gerais de rostos.

Com estes classificadores, são identificados os padrões das imagens positivas e negativas dos rostos pois, tão importante quanto encontrar padrões da região desejada, é identificar regiões que não contém o padrão procurado. Dessa forma, o modelo poderá desempenhar melhor a detecção e, a partir da soma ponderada desses classificadores, criar um classificador eficiente, que é chamado de *classificador forte*. (OPENCV, 2021b)

De acordo com OpenCV (2021b), para este método ser mais eficiente, é necessário detectar filtros de classificação primária, para que o tempo de processamento seja reduzido. Por isso, foi criado os classificadores de cascata, que fazem o processo em etapas - ou cascatas - facilitando o processamento do modelo.

2.4 Métricas de desempenho dos classificadores

As métricas de verificação de desempenho dos classificadores são de suma importância para auxiliar, a partir do embasamento matemático, a identificar seu desempenho. Existem diversos tipos de métricas de desempenho, dentre as mais conhecidas, podemos destacar a precisão, revocação e a medida F.

Neste trabalho, V_P é o valor encontrado nos testes do classificador corretamente positivo, ou seja, quando o classificador identificou a caneca na região correta, enquanto F_P é o valor encontrado nos testes do classificador incorretamente positivo, ou seja, a quantidade de vezes

que o classificador identificou uma região na qual não existiam canecas. Por fim, F_N é o valor encontrado nos testes do classificador incorretamente negativo, ou seja, a quantidade de vezes que o classificador não identificou uma região que continham canecas.

A precisão, denotada como P, apresenta o valor de observações positivas corretamente classificadas pelo modelo de classificação. O cálculo da precisão é realizado pela equação 2.1 dada abaixo. (SOKOLOVA; JAPKOWICZ; SZPAKOWICZ, 2006)

$$P = \frac{V_P}{V_P + F_P} \tag{2.1}$$

A revocação, denotada como R, apresenta o cálculo da quantidade de observações positivas do modelo classificadas de forma correta, levando em conta todas as observações realizadas de forma positiva. Seu cálculo é realizado com a equação 2.2 abaixo. (SOKOLOVA; JAPKOWICZ; SZPAKOWICZ, 2006)

$$R = \frac{V_P}{V_P + F_N} \tag{2.2}$$

A medida F, denotada como M_F, é obtida a partir do cálculo das medidas de precisão e da revocação, e tem como objetivo fornecer uma melhor avaliação destes resultados. Seu cálculo é realizado com a equação 2.3 abaixo. (SOKOLOVA; JAPKOWICZ; SZPAKOWICZ, 2006)

$$M_f = \frac{2 \times P \times R}{P + R} \tag{2.3}$$

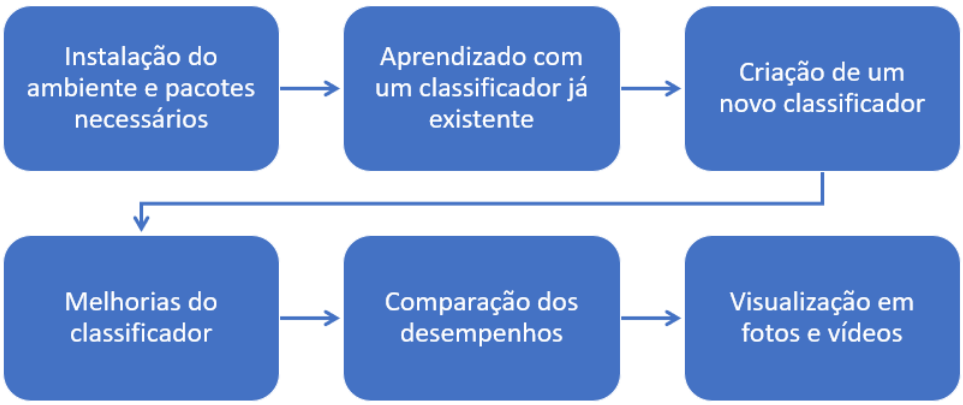
Nas três métricas de desempenho dos classificadores, os resultados dos cálculos das equações 2.1, 2.2 e 2.3 variam de 0 a 1, sendo que quanto mais próximo de 1, melhor será a performance do classificador avaliado.

3 Metodologia

3.1 Visão geral

Na Figura 3.1 é apresentado o fluxo geral do trabalho desenvolvido. Após a instalação do ambiente de programação e dos pacotes a serem utilizados, foi necessário adquirir os conhecimentos para a utilização da biblioteca do *Python* dentro do *Jupyter Notebook* bem como sobre a manipulação das imagens em classificadores já existentes no *OpenCV*, para que assim fosse possível criar um novo classificador.

Figura 3.1 – Fluxo geral do projeto desenvolvido.



Fonte: Autora.

Os classificadores necessitam de ajustes para melhor detecção dos alvos desejados, por conta disso, foram utilizados três tipos diferentes de ajustes neste trabalho, gerando assim, três classificadores diferentes. Estes classificadores foram usados em vídeos e fotos para a detecção de canecas - objeto selecionado para treinar o modelo.

Vale ressaltar que é possível criar um classificador com qualquer tipo de objeto ou alvo, entretanto, suas variações podem alterar o desempenho. Na construção de classificadores de carros, por exemplo, modelos e/ou posições diferentes das fotos de carros utilizadas para construção do modelo podem afetar o desempenho do classificador.

Bons classificadores, isto é, com performances mais assertivas, podem demorar muito tempo para serem criados. O classificador com melhor desempenho deste trabalho demorou cerca de 21 horas para ser treinado, e pode ser considerado um modelo simples quando comparado aos modelos disponíveis pela *Haar Cascade*, que podem demorar meses para serem construídos.

Neste trabalho foi utilizado um computador da marca *Acer*, com processador *Intel Core i5 10210U Quad core*, sistema operacional *Windows 10* de *64-bit*, velocidade do processador de *4,2 GHz*, memória interna de *8GB* e memória externa de *512GB* do tipo *SSD*.

3.2 Instalação do ambiente e pacotes necessários

O *Jupyter Notebook* é um ambiente de programação com instalação rápida e gratuita através do *site* Instaladores Anaconda (<https://www.anaconda.com/products/individual#Downloads>). Algumas bibiliotecas do *Python* já são instaladas automaticamente, enquanto outras precisam ser instaladas manualmente. As versões utilizadas das bibliotecas *OpenCV* e *Matplotlib* foram 3.4.8 e 3.2.2, respectivamente. Vale ressaltar que a instalação de versões específicas podem ser encontradas nos sites oficiais das respectivas bibliotecas.

3.3 Aprendizado com um classificador já existente

Para o estudo dos classificadores em cascata foi necessário explorar a biblioteca e os classificadores já existentes. Todos os classificadores já produzidos podem ser encontrados neste GitHub ou no *site* <https://github.com/opencv/opencv/tree/master/data/haarcascades>.

A seguir, é apresentado um exemplo já existente e disponível no GitHub que foi utilizado para aprendizado neste trabalho. Este classificador tem como objetivo identificar rostos em posição frontal:

```
#importando as bibliotecas
import cv2
import matplotlib.pyplot as plt

#importando a imagem a ser utilizada no modelo
imagem = cv2.imread('exemplo.jpg')

#importando o classificador a ser utilizado
classificador_rostos = 'haarcascade_frontalface_default.xml'
rastrear_rostos = cv2.CascadeClassifier(classificador_rostos)

#transformando a imagem em escala cinza
imagem2 = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)

#encontrando os rostos
rostos = rastrear_rostos.detectMultiScale(imagem2, 1.3, 10)

#coordenadas dos pixels e a quantidade de rostos encontrados
print(rostos)
print(len(rostos))

```

```
#identificando os rostos encontrados
identificador_rostos = imagem.copy()
for (x, y, w, h) in rostos:
    cv2.rectangle(identificador_rostos, (x, y), (x + w, y + h), (255,
    255, 255), 2)

#convertendo novamente a imagem para as cores RGB originais
identificador_rostos = cv2.cvtColor(identificador_rostos, cv2.
    COLOR_BGR2RGB)

#visualizando o resultado
plt.figure(figsize=(20,20))
plt.imshow(identificador_rostos)
```

O resultado do código de estudo acima é uma imagem com a identificação do rosto encontrado e suas coordenadas. Neste exemplo, em que foi utilizada uma fotografia da autora do trabalho, o classificador identificou um rosto com coordenadas [785, 377, 709, 709]. O resultado pode ser visualizado na Figura 3.2 abaixo.

Figura 3.2 – Resultado da aplicação do código de estudo no classificador de rostos frontais.

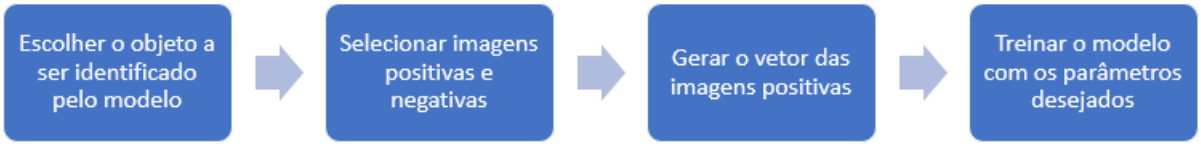
Fonte: Autora.

Há diversos classificadores existentes e cada um possui um objetivo único. Dentre eles, podemos citar a identificação de olhos, sorrisos, corpos, rostos de perfil, olhos com óculos, carros e placas de carro, dentre outras aplicações já existentes e disponíveis pela biblioteca.

3.4 Criação de um novo classificador

Para a criação de um novo classificador é necessário seguir, de forma geral, alguns passos, que são listados na Figura 3.3. Com a obtenção das imagens positivas e negativas é possível treinar o modelo com a criação de um vetor, que mapeia as coordenadas do objeto presente nas imagens. Note que a escolha de objetos-alvo que não apresentam muita variação de tamanho, forma e/ou cor tende a produzir melhores resultados.

Figura 3.3 – Fluxo para a criação de um *Haar Cascade*.



Fonte: Autora.

3.4.1 Obtenção das imagens positivas e negativas

O primeiro passo é selecionar diversas imagens negativas, ou seja, imagens que não contenham o objeto desejado. Ademais, serão necessárias uma ou mais imagens positivas, isto é, que contenham o objeto a ser identificado.

As imagens negativas foram obtidas no banco de dados da ImageNet (<https://www.image-net.org/>). No total foram utilizadas 3200 imagens, todas na escala cinza e com dimensão 100 x 100 *pixels*, salvas com os nomes padronizados em números de 1 a 3200.

Tem-se a seguir o código utilizado para realizar a padronização das imagens retiradas do *site ImageNet*.

```
1  #importando as bibliotecas
2  import urllib.request
3  import cv2
4  import os
5
6  #fornecendo o link com as imagens
7
8  link = 'http://image-net.org/api/text/imagenet.synset.geturls?wnid=
       n00483313'
9
10 #requisitos da biblioteca
11 headers = {}
12 headers['User-Agent'] = "Mozilla/5.0 (X11; Linux i686) AppleWebKit
       /537.17 (KHTML, like Gecko) Chrome/24.0.1312.27 Safari/537.17"
13
```

```
#gerando a chamada da API
requisicao = urllib.request.Request(link, headers = headers)
resp = urllib.request.urlopen(requisicao)

#criando as imagens e salvando na pasta "negativas"

retorno = resp.read().decode()
n = 1

if not os.path.exists('negativas'):
    os.makedirs('negativas')

for i in retorno.split('\n'):
    print(i)
    urllib.request.urlretrieve(i, "negativas/" + str(n) + ".jpg")
    img = cv2.imread("negativas/" + str(n) + ".jpg", cv2.
    IMREAD_GRAYSCALE)
    img2 = cv2.resize(img, (100, 100))
    cv2.imwrite("negativas/" + str(n) + ".jpg", img2)
    n += 1
```

Após a aquisição das imagens negativas, é necessário listar as imagens para serem utilizadas na construção do modelo. Essa listagem foi feita utilizando o comando ***dir /b *.jpg > lista.txt*** em um arquivo de extensão tipo *bat*. O resultado obtido é a criação de um arquivo de extensão tipo *txt* chamado "lista", que contém uma lista de todas imagens, começando com "1.jpg" e finalizando com "3200.jpg".

As imagens positivas são o conjunto de imagens negativas que contém, em regiões aleatórias, a presença da imagem do objeto alvo. Neste trabalho optou-se pela criação de um classificador que fosse capaz de reconhecer a presença de canecas, objeto este com poucas variações de formato e que não tem a cor como requisito para sua definição.

De forma lúdica, é apresentado abaixo o comando utilizado para a criação das imagens positivas a partir das negativas obtidas no banco de dados *ImageNet*:

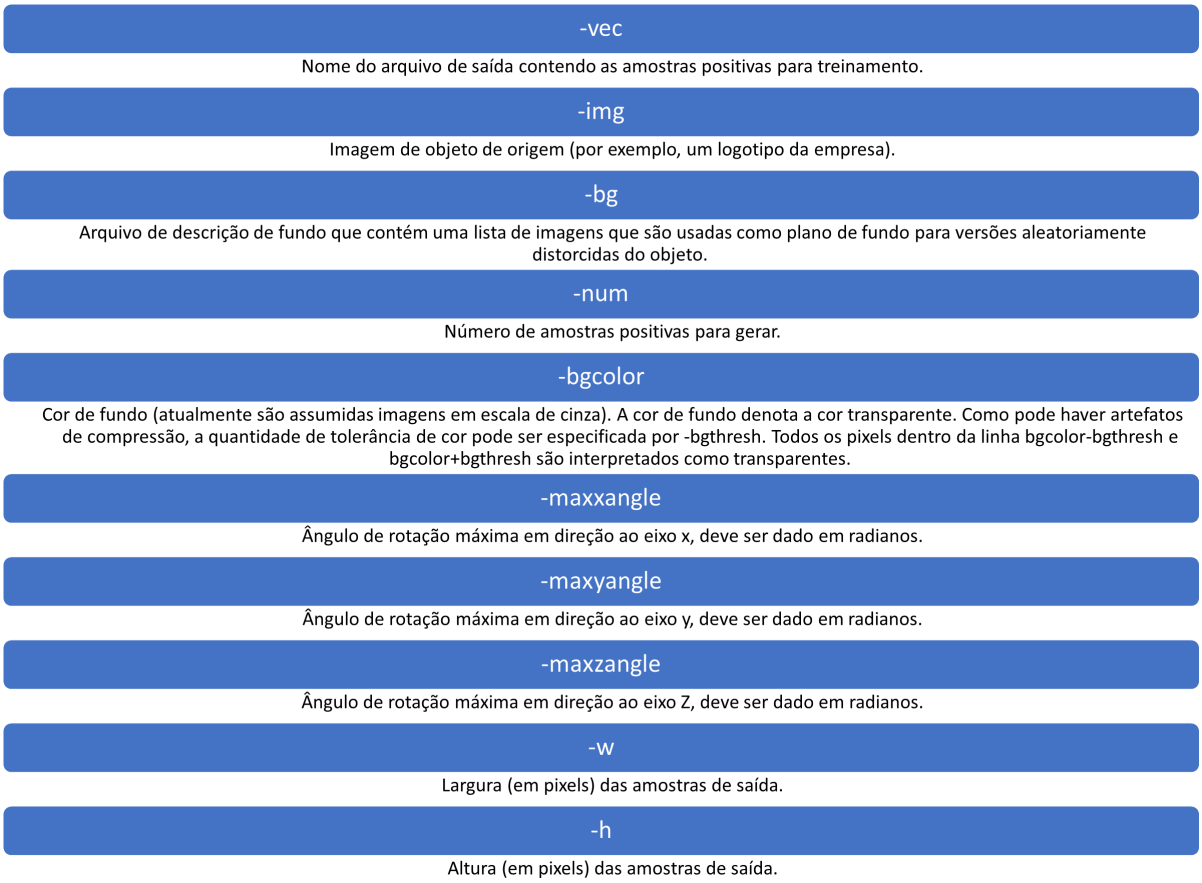
```
opencv_createsamples -img caneca.jpg -bg negativas/lista.txt -info
   positivas/positivas.lst -maxxangle 0.5 -maxyangle 0.5 -maxzangle 0.5
   -num 1800 -bgcolor 255
```

Vale ressaltar que para o modelo utilizado neste trabalho, todos os passos para o treinamento das imagens positivas e a escolha de parâmetros que podem ser utilizados para melhorar a performance do modelo foram encontrados no site da biblioteca *OpenCV* de referência OpenCV (2021a).

A Figura 3.4 lista os parâmetros que foram utilizados para a criação das imagens positivas neste trabalho. Tais parâmetros ajudam a melhorar a performance e o desempenho do modelo. O arquivo *positivas.lst* gerado contém as informações das imagens criadas e as respectivas

coordenadas da região em que o objeto - neste caso a caneca - está presente.

Figura 3.4 – Parâmetros utilizados para o método *Haar Cascade*.



Fonte: Adaptado de OpenCV (2021a)

3.4.2 Treinamento do modelo

Para o treinamento do modelo é necessário criar um vetor a partir das imagens positivas que servirá como orientação para que o modelo comece a compreender a figura de uma caneca, dadas as coordenadas da localização da caneca em cada imagem positiva. Este processo permitirá que o computador comece a entender os padrões presentes em todas as imagens e distingua uma caneca do restante da imagem.

A criação do vetor e o treinamento do modelo estão representados pelo código abaixo, respectivamente.

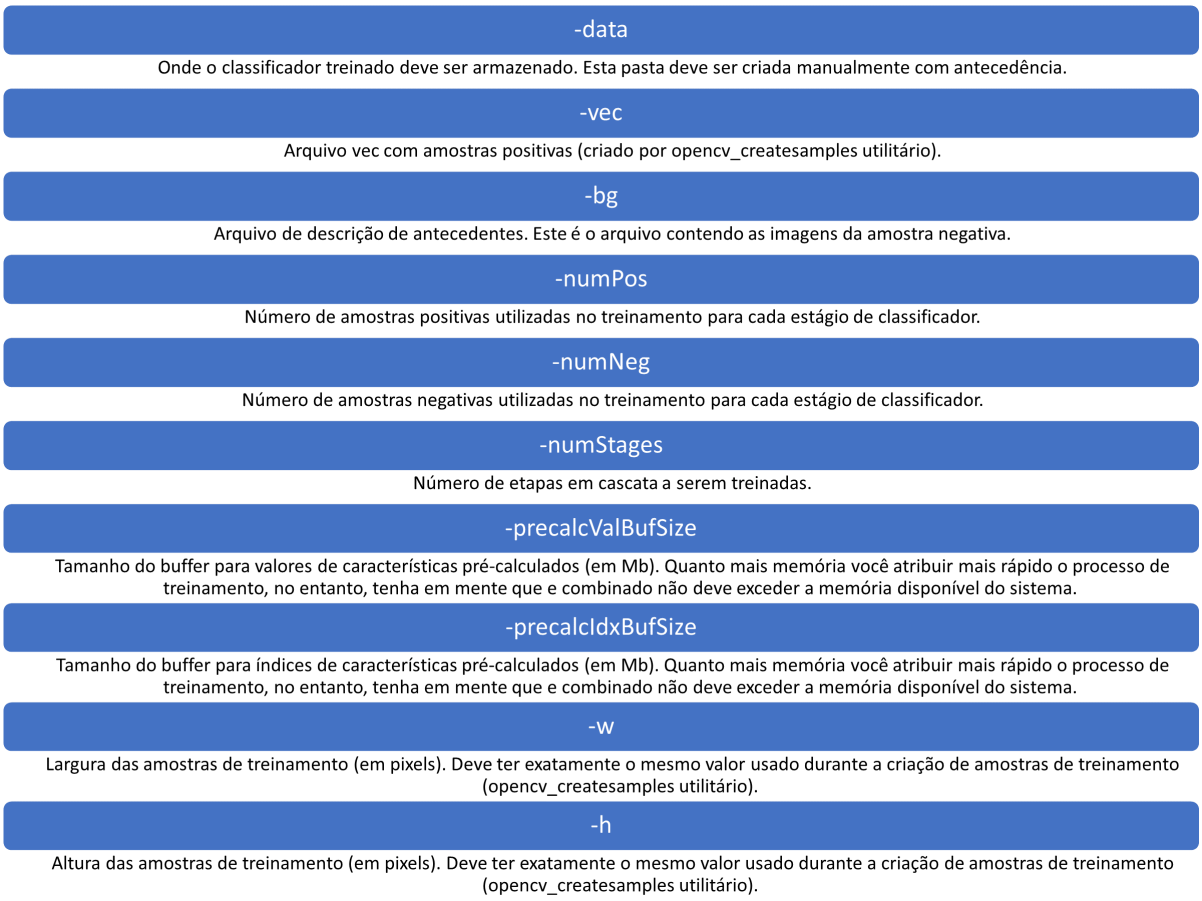
```
#criando o vetor
opencv_createsamples -info positivas/positivas.lst -num 1800 -w 18 -h 18
    -vec positivas.vec

#treinando o modelo
```

```
opencv_traincascade -data classificador -vec positivas.vec -bg lista.txt
    -numPos 1600 -numNeg 800 -numStages 10 -w 18 -h 18 -precalcBufSize
   1024 -precalcIdxBufSize 1024
```

Os parâmetros utilizados para as criações do vetor e do modelo estão representados na Figura 3.5.

Figura 3.5 – Parâmetros utilizados com o método *Haar Cascade*.



Fonte: Adaptado de OpenCV (2021a)

Como resultado é gerado um arquivo chamado *cascade.xml*. Este arquivo é o modelo pós-treinamento que pode ser utilizado nos códigos para a detecção das canecas. Abaixo observa-se a utilização do modelo criado:

```
#importando as bibliotecas
import matplotlib.pyplot as plt
import cv2

#importando o classificador
classificador_1 = 'cascade_1.xml'
rastreador_1 = cv2.CascadeClassifier(classificador_1)

```

```
#importando a imagem a ser utilizada no modelo
imagem = cv2.imread('teste.jpg')

#transformando a imagem em escala cinza
imagem2 = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)

#encontrando os objetos (neste caso, canecas)
objetos = rastreador_1.detectMultiScale(imagem2)

#coordenadas dos pixels e a quantidade de canecas encontradas
print(objetos)
print(len(objetos))

#identificando as canecas encontradas
imagem3 = imagem.copy()
for (x, y, w, h) in objetos:
    cv2.rectangle(imagem3, (x, y), (x + w, y + h), (0, 0, 0), 2)
#convertendo novamente a imagem para as cores RGB originais
imagem3 = cv2.cvtColor(imagem3, cv2.COLOR_BGR2RGB)

#visualizando o resultado
plt.figure(figsize=(10,10))
plt.imshow(imagem3)
```

3.4.3 Aprimoramento do classificador

Em busca de um classificador que apresente um melhor desempenho, dois procedimentos foram feitos: o aumento da variedade de canecas nas imagens positivas e a alteração de parâmetros do modelo, detalhados nas subseções seguintes.

3.4.3.1 Aumento da variedade de canecas nas imagens positivas

Para melhorar o desempenho do classificador construído é possível treinar o modelo com um conjunto de imagens de canecas, contemplando diferentes formas e tamanhos de canecas. Esse método de melhoria faz uso da união de vários vetores gerados pelas imagens das diferentes canecas para a criação de um vetor único, utilizado para o treinamento do modelo.

Portanto, para a melhoria do desempenho, foi necessário criar imagens positivas para diversos tipo de caneca. Foram utilizadas dez imagens de canecas de diferentes formas e tamanhos e, com isso, foi possível a geração de dez diferentes arquivos do tipo *positivas.lst*. Segue abaixo o código utilizado para a imagem da primeira caneca e a geração do primeiro arquivo de extensão *lst*. O procedimento foi replicado para a geração dos outros nove arquivos e suas respectivas imagens positivas.

```
opencv_createsamples -img caneca1.jpg -bg negativas/lista.txt -info
   positivas1/positivas1.lst -maxxangle 0.5 -maxyangle 0.5 -maxzangle
```

```
   0.5 -w 48 -h 48 -num 300 -bgcolor 255 -bgthresh 8
```

Após este passo realizado para as dez canecas, foi necessário gerar os vetores para cada uma destas canecas. O código abaixo foi utilizado para a geração do primeiro vetor, e o procedimento foi replicado para a geração dos outros nove vetores.

```
opencv_createsamples -info positivas1/positivas1.lst -num 200 -w 20 -h
   20 -vec vetor1.vec
```

De maneira subsequente, foi necessário unir os dez vetores criados para o treino do modelo. Visto isso, foi necessária a utilização de um código de união de vetores. O código utilizado, apresentado abaixo, foi uma adaptação de Seo (2007) criado por Blake Wulfe e encontrado em Wulfe (2014):

```
import sys
import glob
import struct
import argparse
import traceback

def exception_response(e):

  exc_type, exc_value, exc_traceback = sys.exc_info()
  lines = traceback.format_exception(exc_type, exc_value, exc_traceback)

  for line in lines:
    print(line)

def get_args():

  parser = argparse.ArgumentParser()
  parser.add_argument('-v', dest='vec_directory')
  parser.add_argument('-o', dest='output_filename')
  args = parser.parse_args()

  return (args.vec_directory, args.output_filename)

def merge_vec_files(vec_directory, output_vec_file):

  if vec_directory.endswith('/'):
    vec_directory = vec_directory[:-1]
  files = glob.glob('{0}/*.vec'.format(vec_directory))

  if len(files) <= 0:
    print('Vec files to be mereged could not be found from directory:
   {0}'.format(vec_directory))
    sys.exit(1)

```

```
34  if len(files) == 1:
35    print('Only 1 vec file was found in directory: {0}. Cannot merge a
     single file.'.format(vec_directory))
36    sys.exit(1)
37  prev_image_size = 0
38
39  try:
40    with open(files[0], 'rb') as vecfile:
41      content = b''.join((line) for line in vecfile.readlines())
42      val = struct.unpack('<iihh', content[:12])
43      prev_image_size = val[1]
44
45  except IOError as e:
46    print('An IO error occured while processing the file: {0}'.format(f)
     )
47    exception_response(e)
48
49  total_num_images = 0
50
51  for f in files:
52    try:
53      with open(f, 'rb') as vecfile:
54        content = b''.join((line) for line in vecfile.readlines())
55        val = struct.unpack('<iihh', content[:12])
56        num_images = val[0]
57        image_size = val[1]
58        if image_size != prev_image_size:
59          err_msg = """The image sizes in the .vec files differ. These
     values must be the same. \n The image size of file {0}: {1}\n
60            The image size of previous files: {0}""".format(f,
     image_size, prev_image_size)
61          sys.exit(err_msg)
62
63        total_num_images += num_images
64
65    except IOError as e:
66      print('An IO error occured while processing the file: {0}'.format(
     f))
67      exception_response(e)
68
69  header = struct.pack('<iihh', total_num_images, image_size, 0, 0)
70
71  try:
72    with open(output_vec_file, 'wb') as outputfile:
73      outputfile.write(header)
74
75      for f in files:
```

```
        with open(f, 'rb') as vecfile:
          content = b''.join((line) for line in vecfile.readlines())
          outputfile.write(bytearray(content[12:]))
  except Exception as e:
    exception_response(e)

if __name__ == '__main__':

  vec_directory, output_filename = get_args()
  if not vec_directory:
    sys.exit('mergvec requires a directory of vec files. Call mergevec.
    py with -v /your_vec_directory')

  if not output_filename:
    sys.exit('mergevec requires an output filename. Call mergevec.py
    with -o your_output_filename')

  merge_vec_files(vec_directory, output_filename)
```

Após a união dos dez vetores criados, foi possível realizar o treinamento do modelo. Os comandos abaixo representam a criação de um vetor único chamado *vetor.vec* e sua utilização no treino do modelo de identificação de canecas:

```
#criando o vetor
python mergevec.py -v vec/ -o vetor.vec

#treinando o novo modelo
opencv_traincascade -data classificador -vec vetor.vec -bg lista.txt -
   numPos 1800 -numNeg 1200 -numStages 10 -w 20 -h 20 -precalcBufSize
   1024 -precalcIdxBufSize 1024
```

Com isso, foi possível realizar novos testes, com as mesmas imagens utilizadas no classificador criado anteriormente para comparar os resultados obtidos. O código utilizado para realizar estes novos testes encontra-se a seguir:

```
#importando as bibliotecas
import matplotlib.pyplot as plt
import cv2

#importando o classificador
classificador_2 = 'cascade_2.xml'
rastreador_2 = cv2.CascadeClassifier(classificador_2)

#importando a imagem a ser utilizada no modelo
imagem = cv2.imread('teste.jpg')

#transformando a imagem em escala cinza
imagem2 = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
```

```
14
15 #encontrando os objetos (neste caso, canecas)
16 objetos = rastreador_2.detectMultiScale(imagem2)
17
18 #coordenadas dos pixels e a quantidade de canecas encontradas
19 print(objetos)
20 print(len(objetos))
21
22 #identificando as canecas encontradas
23 imagem3 = imagem.copy()
24 for (x, y, w, h) in objetos:
25     cv2.rectangle(imagem3, (x, y), (x + w, y + h), (0, 0, 0), 2)
26 #convertendo novamente a imagem para as cores RGB originais
27 imagem3 = cv2.cvtColor(imagem3, cv2.COLOR_BGR2RGB)
28
29 #visualizando o resultado
30 plt.figure(figsize=(10,10))
31 plt.imshow(imagem3)
```

3.4.3.2 Melhoria nos parâmetros do classificador

Para melhorar ainda mais o modelo desenvolvido, é possível ajustar alguns parâmetros de treino para aumentar sua eficiência na detecção do objeto-alvo, no caso, canecas. Foi realizado o aumento da altura e da largura das imagens utilizadas para a criação de dez vetores para este terceiro modelo, um para cada tipo diferente de caneca (modelos utilizados também no classificador anterior), afim de aumentar as chances do classificador identificar variações. Esses ajustes podem ser observado no código abaixo, que representa a criação do primeiro vetor para a primeira imagem das canecas utilizadas.

```
1 opencv_createsamples -info positivas1/positivas1.lst -num 200 -w 24 -h
     24 -vec vetor1.vec
```

Este mesmo código foi utilizado para a criação de mais nove vetores com os parâmetros alterados, totalizando dez vetores, um para cada imagem de caneca utilizada, e consequentemente, foi utilizado o código de união de vetores para o treino do modelo com melhores parâmetros. Os passos seguem os mesmos que utilizados anteriormente.

Para o treino, também foram ajustados parâmetros para melhoria de desempenho do modelo, que podem ser observadas no código abaixo.

```
1 opencv_traincascade -data classificador -vec vetor.vec -bg lista.txt -
     numPos 2000 -numNeg 3200 -numStages 15 -w 24 -h 24 -precalcBufSize
     1024 -precalcIdxBufSize 1024
```

O aumento do tamanho das imagens positivas para o treinamento aumenta consideravelmente o tempo de treino dos classificadores e, em contrapartida, garante que a qualidade da

detecção das imagens seja melhor pois as imagens positivas serão geradas com canecas com melhor definição. A qualidade das imagens do classificador 2 para o classificador 3 foi alterada de 20 *x* 20 *pixels* para 24 *x* 24 *pixels*, melhorando consideravelmente o desempenho do classificador, além de aumentar seu tempo de processamento.

Além disso, outro parâmetro ajustado foi a quantidade de imagens positivas e negativas no treino. No classificador 2 foram utilizadas 1800 imagens positivas e 1200 imagens negativas. Já para o classificador 3 foram utilizadas 2000 imagens positivas e 3200 imagens negativas.

Estes parâmetros alterados foram implementados e, quanto mais a máquina recebe informações iniciais dos padrões desejados nos resultados, imagens positivas, melhor ela aprende. O aumento da quantidade das imagens negativas também contribui para que o computador consiga diferenciar de forma mais assertiva o que não é uma caneca, ou seja, em teoria, ajuda a diminuir os resultados falsos positivos.

Por fim, o número de estágios de processamento foi aumentado de 10 para 15 para melhorar a performance do treinamento, pois como seu tempo aumenta, serão necessários mais estágios de armazenamento para diminuir as chances do treino encerrar por memória antes da sua finalização.

Os testes após os ajustes dos parâmetros foram feitos seguindo a lógica do código abaixo.

```
#importando as bibliotecas
import matplotlib.pyplot as plt
import cv2

#importando o classificador
classificador_3 = 'cascade_3.xml'
rastreador_3 = cv2.CascadeClassifier(classificador_3)

#importando a imagem a ser utilizada no modelo
imagem = cv2.imread('teste.jpg')

#transformando a imagem em escala cinza
imagem2 = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)

#encontrando os objetos (neste caso, canecas)
objetos = rastreador_2.detectMultiScale(imagem2)

#coordenadas dos pixels e a quantidade de canecas encontradas
print(objetos)
print(len(objetos))

#identificando as canecas encontradas
imagem3 = imagem.copy()
for (x, y, w, h) in objetos:
    cv2.rectangle(imagem3, (x, y), (x + w, y + h), (0, 0, 0), 2)
```

```
#convertendo novamente a imagem para as cores RGB originais
imagem3 = cv2.cvtColor(imagem3, cv2.COLOR_BGR2RGB)

#visualizando o resultado
plt.figure(figsize=(10,10))
plt.imshow(imagem3)
```

3.5 Detecção de objetos utilizando a webcam

A detecção de objetos pode ser feita em tempo real com o uso de uma webcam e a aplicação do modelo desenvolvido. Neste trabalho, foi testada a utilização da webcam para identificar uma caneca em tempo real, com a utilização do código apresentando a seguir:

```
#importando as bibliotecas
import cv2

#utilizando a webcam para detectar canecas
while True:

    conectado, frame = cv2.VideoCapture(0).read()
    imagemCinza = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    deteccoes = rastreador_3.detectMultiScale(imagemCinza, scaleFactor
   =1.3, minNeighbors=9, minSize=(60, 60))

    for (x, y, l, a) in deteccoes:
        cv2.rectangle(frame, (x, y), (x + l, y + a), (0, 0, 255), 2)
    cv2.imshow('Detector de canecas via webcam', frame)

    if cv2.waitKey(1) == ord('q'):
        break

video.release()
cv2.destroyAllWindows()
```

4 Resultados e Discussão

Os resultados obtidos foram gerados a partir dos treinamentos dos três classificadores diferentes neste presente trabalho, todos com seus devidos ajustes e detalhes apresentados no capítulo anterior. Para o primeiro modelo de classificador, foi utilizado um único modelo de caneca para a criação de um vetor, que foi utilizado para inserir a caneca em diversas coordenadas e posições diferentes em imagens que não continham inicialmente uma caneca presente. Com isso, o treinamento ensinou a máquina a identificar a presença de uma caneca a partir destas imagens chamadas "positivas". Além disso, também foi necessário ensinar ao computador, com imagens chamadas "negativas", quando não existia a presença de canecas.

Para a criação do segundo e terceiro classificadores, foram utilizadas, em vez de um modelo de caneca, dez modelos diferentes de canecas, para que o computador conseguisse interpretar as variações que o objeto caneca poderia ter, como tamanho, forma e/ou tonalidade de cor. Vale ressaltar que os treinos são feitos com imagens padronizadas em tons de cinza, portanto, a cor não interfere na decisão final do computador, mas as variações de tonalidades de cinza podem ajudar o computador a entender que canecas podem ter cores diferentes.

Além disso, para o terceiro classificador, foram ajustados alguns parâmetros, como altura das imagens positivas utilizadas para a criação de vetores de classificação, além do aumento dos estágios de processamento e aumento da quantidade de imagens positivas e negativas no treino do modelo. Como será apresentado a seguir, estes ajustes fizeram com que o terceiro classificador obtivesse melhor desempenho com relação ao segundo classificador, apesar dos dois utilizarem as mesmas dez imagens de modelos de canecas para a geração das imagens positivas.

Este capítulo apresenta os resultados obtidos com a aplicação dos três classificadores para imagens de testes não vistas anteriormente pelo computador, isto é, imagens não utilizadas para treinamento do modelo/classificador. A seguir são apresentados e discutidos os respectivos desempenhos, comparando medidas de precisão, revogação, medida F e tempo de processamento dos três classificadores.

4.1 Apresentação dos resultados obtidos

4.1.1 Testes com imagens contendo canecas

Após o treinamento dos classificadores, ou modelos, criados neste trabalho, foi possível aplicá-los e verificar seus respectivos desempenhos. Para cada um dos três modelos construídos, testes com oito imagens, sendo cinco delas imagens contendo canecas diferentes, foram feitos para comparar o desempenho de cada modelo.

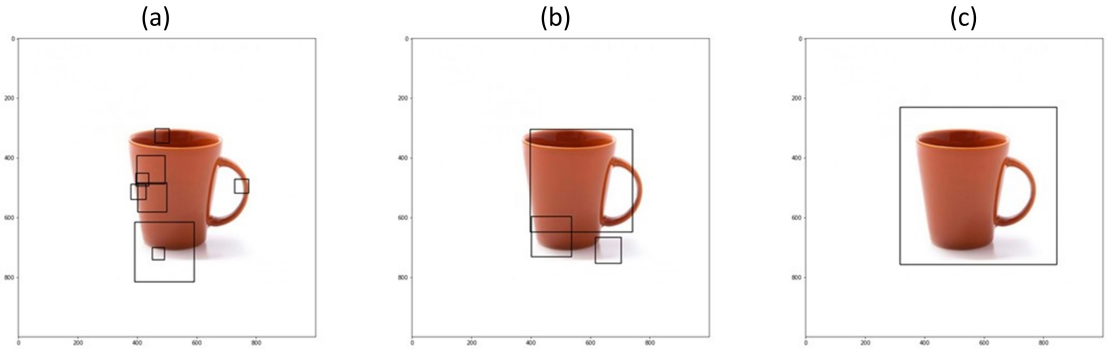
A Figura 4.1 representa uma caneca simples e sem elementos ao fundo e foi utilizada para o treinamento dos classificadores. Os classificadores irão identificar uma ou mais canecas na imagem e serão desenhados um ou mais retângulos nas regiões identificadas pelo modelo.

Figura 4.1 – Imagem 1 utilizada para os testes.

Fonte: <https://bit.ly/3dG0JmA>

A Figura 4.2 representa a aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.1.

Figura 4.2 – Resultado da aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.1. A figura (a) representa a aplicação do classificador 1, já a figura (b) representa a aplicação do classificador 2 e, por último, a figura (c) representa a aplicação do classificador 3.



Fonte: Autora

Pode-se observar a presença de retângulos nas figuras após a aplicação dos modelos, sendo que esses retângulos representam a quantidade de canecas encontradas pelo modelo utilizado. As métricas de classificação encontradas para cada modelo foram listadas na Tabela 4.1 abaixo.

Tabela 4.1 – Métricas de classificação para cada classificador aplicados à Figura 4.1.

Classificadores	F_P	F_N	V_P
Classificador 1	8	1	0
Classificador 2	2	0	1
Classificador 3	0	0	1

Fonte: Autora

Pode-se observar que a Figura 4.1 contém somente uma caneca, e de acordo com a Tabela 4.1, para os classificadores 1 e 2, foram encontradas mais de uma região que o computador interpretou que continham canecas. Estes resultados refletem que o desempenho destes modelos não está sendo desejável. Já o classificador 3 identifica a quantidade certa de canecas presente na imagem, obtendo o desempenho desejado.

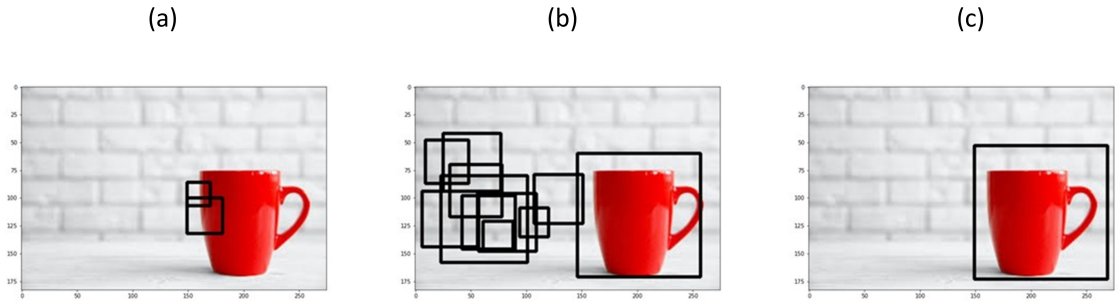
Outras imagens foram escolhidas para testar os classificadores construídos (Figuras 4.3, 4.5, 4.7 e 4.9). A Figura 4.3 foi escolhida por conter uma caneca e fundo com elementos presentes, neste caso, uma parede de tilojos. A Figura 4.5 foi escolhida por conter uma caneca, fundo com elementos presentes e outros objetos - que não são canecas. A Figura 4.7 foi escolhida por conter mais de uma caneca e fundo sem elementos. Por fim, a Figura 4.9 foi escolhida por conter mais de uma caneca e fundo com elementos presentes.

As Figuras 4.4, 4.6, 4.8 e 4.10 representam os resultados obtidos após a aplicação dos três modelos construídos para as Figuras 4.3, 4.5, 4.7 e 4.9, respectivamente. Nas Tabelas 4.2, 4.3, 4.4 e 4.5 é apresentada a quantidade de canecas encontradas em cada caso.

Figura 4.3 – Imagem 2 utilizada para os testes.

Fonte: <https://bit.ly/3hwZDe3>

Figura 4.4 – Resultado da aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.3. A figura (a) representa a aplicação do classificador 1, já a figura (b) representa a aplicação do classificador 2 e, por último, a figura (c) representa a aplicação do classificador 3.



Fonte: Autora

A Figura 4.4 representa a aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.3.

Pode-se observar que a Figura 4.3 também só contém uma caneca, para os classificadores 1 e 2, novamente foram encontradas mais de uma região que o computador interpretou que continham canecas. Novamente, pode-se perceber que os resultados dos classificadores 1 e 2 não são tão precisos quanto o classificador 3. A Tabela 4.2 abaixo contém as métricas de classificação encontradas para este caso.

Tabela 4.2 – Métricas de classificação para cada classificador aplicados à Figura 4.3.

Classificadores	F_P	F_N	V_P
Classificador 1	2	1	0
Classificador 2	10	0	1
Classificador 3	0	0	1

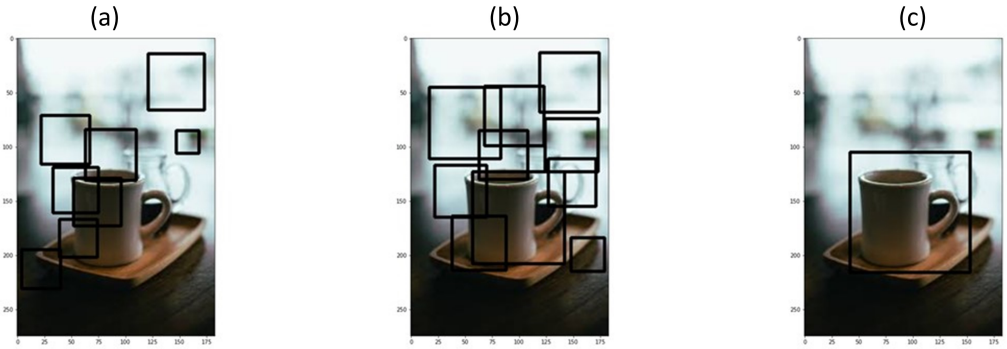
Fonte: Autora

Figura 4.5 – Imagem 3 utilizada para os testes.

Fonte: <https://bit.ly/3hdb1N7>

A Figura 4.6 representa a aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.5.

Figura 4.6 – Resultado da aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.5. A figura (a) representa a aplicação do classificador 1, já a figura (b) representa a aplicação do classificador 2 e, por último, a figura (c) representa a aplicação do classificador 3.



Fonte: Autora

Para a Figura 4.5, pode-se observar a presença de uma caneca, e os classificadores 1 e 2 encontraram novamente falsos negativos. Com isso, observa-se o classificador 3 novamente obtendo o desempenho esperado. A Tabela 4.3 abaixo contém as métricas obtidas para este caso.

Tabela 4.3 – Métricas de classificação para cada classificador aplicados à Figura 4.5.

Classificadores	F_P	F_N	V_P
Classificador 1	8	1	0
Classificador 2	9	0	1
Classificador 3	0	0	1

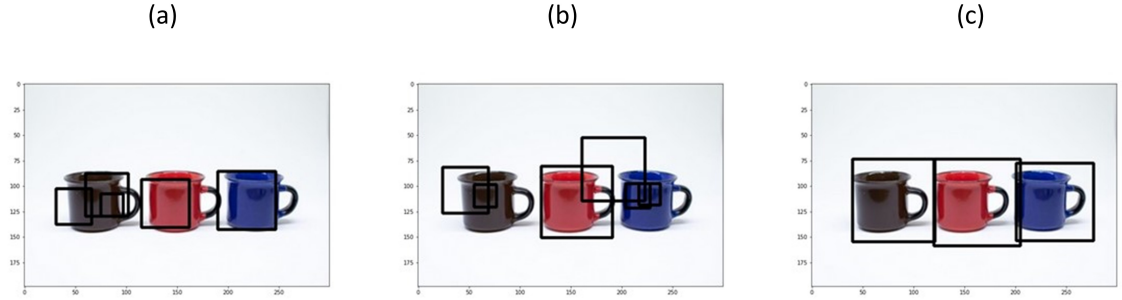
Fonte: Autora

Figura 4.7 – Imagem 4 utilizada para os testes.

Fonte: <https://bit.ly/3AsGZMY>

A Figura 4.8 representa a aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.7.

Figura 4.8 – Resultado da aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.7. A figura (a) representa a aplicação do classificador 1, já a figura (b) representa a aplicação do classificador 2 e, por último, a figura (c) representa a aplicação do classificador 3.



Fonte: Autora

Neste caso, a Figura 4.7 contém 3 canecas, e foi escolhida para testar o modelo com a presença de mais canecas na imagem. Novamente, para os casos dos classificadores 1 e 2 foram encontradas quantidades diferentes das desejadas, e para o classificador 3 foi possível identificar as 3 canecas, resultados das métricas de classificação listados na Tabela 4.4.

Tabela 4.4 – Métricas de classificação para cada classificador aplicados à Figura 4.7.

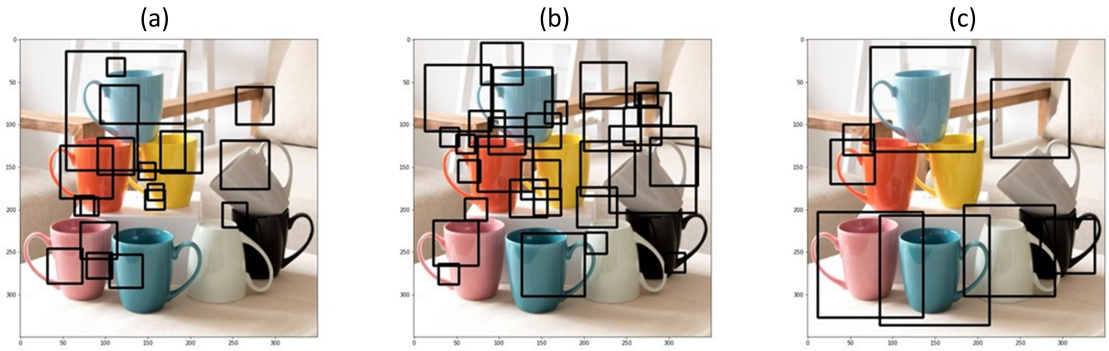
Classificadores	F_P	F_N	V_P
Classificador 1	3	1	2
Classificador 2	5	2	1
Classificador 3	0	0	3

Fonte: Autora

Figura 4.9 – Imagem 5 utilizada para os testes.

Fonte: <https://bit.ly/3qMbZ69>

Figura 4.10 – Resultado da aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.9. A figura (a) representa a aplicação do classificador 1, já a figura (b) representa a aplicação do classificador 2 e, por último, a figura (c) representa a aplicação do classificador 3.



Fonte: Autora

A Figura 4.10 representa a aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.9.

Para a imagem da Figura 4.9, temos o total de oito canecas presentes na foto, e foi possível observar um desempenho diferente para o classificador 3, que anteriormente havia acertado as quantidades e regiões onde as canecas estavam presentes. Nesta figura, apesar do classificador 3 identificar a quantidade certa de canecas, ele as identificou em regiões incorretas. Este resultado mostra falhas no classificador quando há uma grande quantidade de canecas muito unidas, pois há confusão em relação aos formatos das canecas, retornando falsos positivos, ou seja, identificando regiões onde não existem canecas. Os resultados das métricas de verificação foram listados na Tabela 4.5.

Tabela 4.5 – Métricas de classificação para cada classificador aplicados à Figura 4.9.

Classificadores	F_P	F_N	V_P
Classificador 1	18	7	1
Classificador 2	26	5	3
Classificador 3	4	4	4

Fonte: Autora

4.1.2 Testes com imagens não contendo canecas

Após a realização de testes dos classificadores construídos com imagens que continham pelo menos uma caneca, foram feitos testes em imagens na qual não existiam canecas, ou que

continham objetos próximos a canecas, como copos, para entender como os classificadores se comportariam nestas situações.

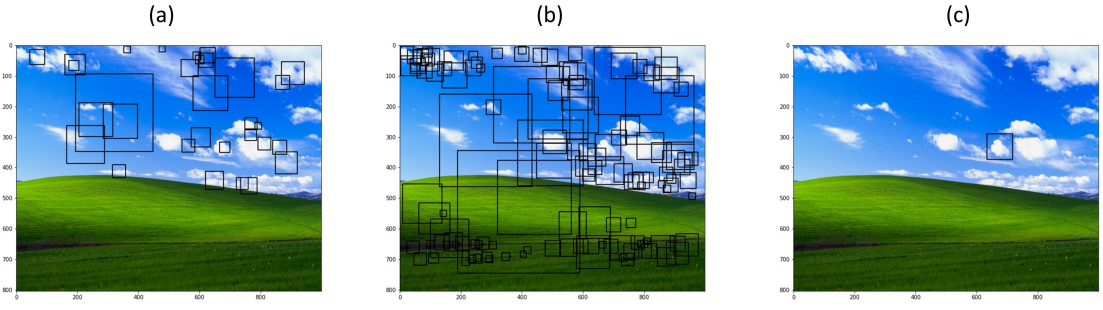
Para o primeiro teste sem canecas, foi escolhida a famosa imagem da tela do *Windows XP*, uma fotografia de paisagem, sem a presença de canecas. Esta imagem pode ser vista na Figura 4.11 abaixo.

Figura 4.11 – Imagem 6 utilizada para os testes.

Fonte: <https://bityli.com/6MqSn>

Para essa imagem, foram realizados testes com os três classificadores, sendo desejado que tais classificadores não identifiquem nenhuma caneca durante os testes. A Figura 4.12 representa a aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.11.

Figura 4.12 – Resultado da aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.11. A figura (a) representa a aplicação do classificador 1, já a figura (b) representa a aplicação do classificador 2 e, por último, a figura (c) representa a aplicação do classificador 3.



Fonte: Autora

Pode-se observar que os três classificadores apresentaram resultados não desejados, de acordo com a Tabela 4.6. Os classificadores 1 e 2 obtiveram desempenhos ruins, principalmente o classificador 2, por identificar muitos falsos positivos. Já o classificador 3, em comparação aos outros, teve um melhor resultado, apesar de identificar uma região de falso positivo.

Tabela 4.6 – Quantidade de canecas encontradas para cada classificador aplicado à Figura 4.11

Classificadores usados	Quantidade de canecas encontradas
Classificador 1	31
Classificador 2	131
Classificador 3	1
Resultado correto	0

Fonte: Autora

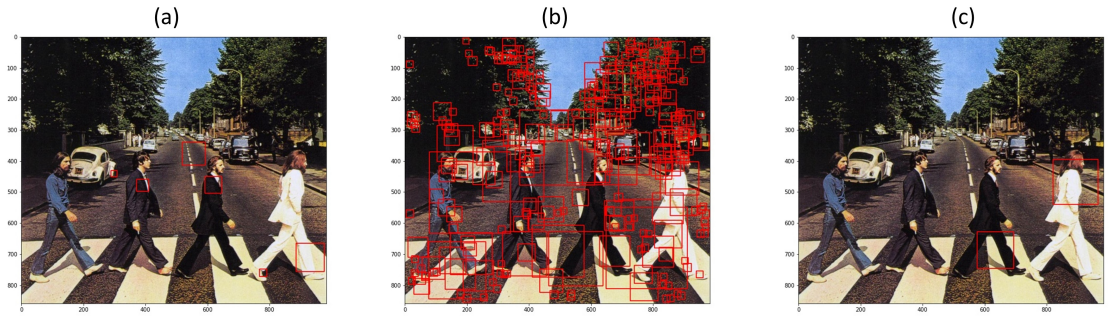
Também foram testados os classificadores em uma imagem com pessoas presentes, porém sem canecas. A Figura 4.13 é uma fotografia famosa da capa de um álbum musical dos *Beatles*.

Figura 4.13 – Imagem 7 utilizada para os testes.

Fonte: <https://bityli.com/IYXvy>

A Figura 4.14 representa a aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.13.

Figura 4.14 – Resultado da aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.13. A figura (a) representa a aplicação do classificador 1, já a figura (b) representa a aplicação do classificador 2 e, por último, a figura (c) representa a aplicação do classificador 3.



Fonte: Autora

De acordo com os resultados apresentados na Tabela 4.14, pode-se observar novamente que o desempenho dos classificadores 1 e 2 foram piores que o classificador 3, reforçando a alta taxa de falsos positivos encontrada pelo classificador 2 comparado aos demais classificadores.

Tabela 4.7 – Quantidade de canecas encontradas para cada classificador aplicado à figura 4.13.

Classificadores usados	Quantidade de canecas encontradas
Classificador 1	6
Classificador 2	287
Classificador 3	2
Resultado correto	0

Fonte: Autora

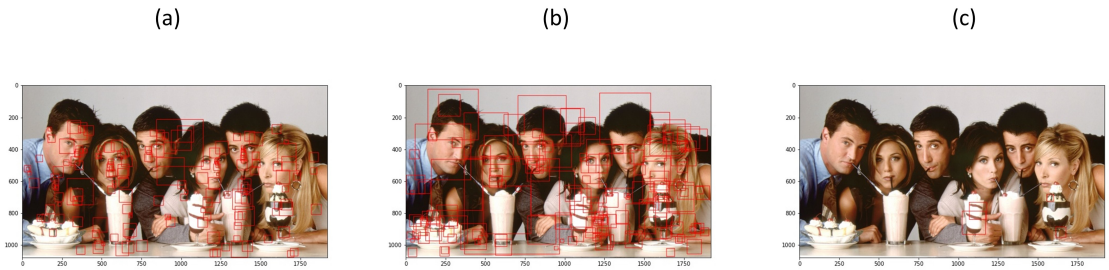
Por fim, para avaliação da qualidade dos classificadores construídos neste trabalho, estes foram testados em uma imagem com copos, que são objetos parecidos com canecas, para observar o desempenho dos classificadores. Para este teste, foi utilizada uma fotografia da série *Friends*, representada na Figura 4.15 abaixo.

Figura 4.15 – Imagem 8 utilizada para os testes.

Fonte: <https://bityli.com/TNsc1>

A Figura 4.16 representa a aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.15.

Figura 4.16 – Resultado da aplicação dos classificadores 1, 2 e 3 para a imagem da Figura 4.15. A figura (a) representa a aplicação do classificador 1, já a figura (b) representa a aplicação do classificador 2 e, por último, a figura (c) representa a aplicação do classificador 3.



Fonte: Autora

Podemos observar que os classificadores 1 e 2 obtiveram resultados próximos e ambos de mal desempenho, com uma grande quantidade de falsos positivos. Já o classificador 3 identificou quatro regiões incorretas (falsos positivos) porém, comparativamente, apresentou o melhor

resultado. Vale ressaltar também que a imagem contém muitas outras informações, como rostos e comidas e a mudança de cores no copo central confundiram o classificador 3, que acabou identificando a bebida como canecas empilhadas. Os resultados obtidos podem ser listados na Tabela 4.8 abaixo.

Tabela 4.8 – Quantidade de canecas encontradas para cada classificador aplicado à Figura 4.15.

Classificadores usados	Quantidade de canecas encontradas
Classificador 1	166
Classificador 2	155
Classificador 3	4
Resultado correto	0

Fonte: Autora

4.2 Discussão dos resultados

Neste trabalho três diferentes classificadores foram criados e testados, permitindo assim a comparação de tais modelos e a discussão dos resultados obtidos. Foram calculados três medidas para avaliação do desempenho dos classificadores, a precisão, a revocação e a medida *F*. As tabelas 4.9, 4.10, 4.11, 4.12 e 4.13 representam tais resultados, calculados com base nas equações 2.1, 2.2 e 2.3.

Tabela 4.9 – Medidas de precisão (P), revocação (R) e medida *F* (M_F) para a Figura 4.1.

Classificadores usados	P	R	M_F
Classificador 1	0,00	0,00	-
Classificador 2	0,33	1,00	0,50
Classificador 3	1,00	1,00	1,00

Fonte: Autora

Tabela 4.10 – Medidas de precisão (P), revocação (R) e medida *F* (M_F) para a Figura 4.3.

Classificadores usados	P	R	M_F
Classificador 1	0,00	0,00	-
Classificador 2	0,09	1,00	0,17
Classificador 3	1,00	1,00	1,00

Fonte: Autora

Tabela 4.11 – Medidas de precisão (P), revocação (R) e medida F (M_F) para a Figura 4.5.

Classificadores usados	P	R	M_F
Classificador 1	0,00	0,00	-
Classificador 2	0,10	1,00	0,18
Classificador 3	1,00	1,00	1,00

Fonte: Autora

Tabela 4.12 – Medidas de precisão (P), revocação (R) e medida F (M_F) para a Figura 4.7.

Classificadores usados	P	R	M_F
Classificador 1	0,40	0,67	0,50
Classificador 2	0,17	0,33	0,22
Classificador 3	1,00	1,00	1,00

Fonte: Autora

Tabela 4.13 – Medidas de precisão (P), revocação (R) e medida F (M_F) para a Figura 4.9.

Classificadores usados	P	R	M_F
Classificador 1	0,05	0,13	0,07
Classificador 2	0,10	0,38	0,16
Classificador 3	0,50	0,50	0,50

Fonte: Autora

Com esses resultados, pode-se realizar algumas observações relevantes. Tem-se também um resultado muito satisfatório para o classificador 3, sendo este o classificador que obteve melhores resultados. Deve-se observar que, para a Figura 4.9, este classificador não identificou todas as regiões corretamente, concluindo que, apesar do classificador 3 responder bem aos testes, mudar a posição tradicional frontal com a alça lateral das canecas faz com que o modelo não identifique bem essas variações. Além disso, a Figura 4.9 apresenta oito canecas muito unidas na imagem, e isto também pode ter afetado o resultados dos testes para este classificador.

O classificador 2 consiste num treino com imagens positivas com maior variedade de canecas. Visto isso, esperava-se que o classificador 2, comparativamente ao classificador 1, fosse melhor, como de fato ocorreu, conforme evidenciado pelas medidas de precisão e revocação apresentadas acima. Porém, deve-se observar a grande quantidade de regiões que não continham canecas (falsos-positivos) encontradas pelo classificador 2.

Para as imagens 4.11, 4.13 e 4.15 testadas sem a presença de canecas, não é possível determinar as medidas de precisão, revogação e medida F pois não existem canecas nas imagens, mas podemos observar o mesmo comportamento: o classificador 1 com resultados muito distantes do desejado, o classificador 2 identificando uma grande quantidade de falsos positivos para a maioria das imagens testadas, e por fim, o classificador 3 mantendo o melhor desempenho.

Vale ressaltar que o treinamento nos classificadores foi realizado com base na presença de canecas nas fotografias, para mais imagens sem a presença de canecas, deve-se aumentar a quantidade de imagens negativas em diferentes cenários, para que assim seja possível minimizar os falsos positivos dos classificadores. Ademais, o treino com mais canecas além dos dez modelos utilizados pode melhorar o resultado e fazer com que o classificador 3, por exemplo, não identifique mais regiões sem a presença de canecas. Estes treinos são demorados e requerem tempo e processamentos sofisticados.

Por fim, tem-se que, com base nos resultados obtidos, o melhor desempenho foi obtido pelo classificador 3, apresentando as melhores medidas de precisão, revogação e medida F para todos os casos apresentados. Vale ressaltar que, para obter um modelo robusto e com medidas de classificação mais altas, é necessário aumentar o tempo de processamento, o número de imagens positivas e a quantidade de imagens com variações de posição, tamanho e formato. Estes processamentos podem durar meses dependendo da máquina utilizada para tal treinamento.

O tempo de processamento de cada modelo utilizado neste trabalho está listado na Tabela 4.14.

Tabela 4.14 – Tempo de processamento para os três modelos utilizados neste trabalho.

Modelos	Tempo de treinamento
Modelo 1	43 minutos
Modelo 2	6 horas e 29 minutos
Modelo 3	21 horas e 4 minutos

Fonte: Autora

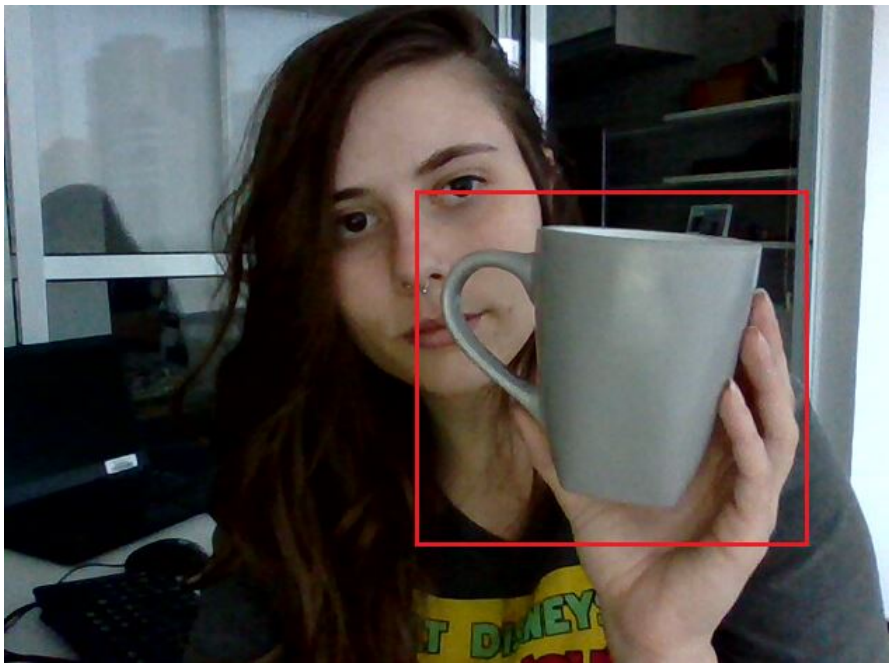
Vale ressaltar também que este trabalho foi realizado com um computador de configurações tradicionais para um computador de uso pessoal. Visto isso, o tempo de processamento é maior do que um treinamento realizado em computadores com configurações próprias para treinamentos de visão computacional ou qualquer outro tipo de treinamento de inteligência artificial. Portanto, o resultado obtido com o classificador 3 pode ser considerado um resultado satisfatório visando o escopo do projeto.

4.3 Aplicação do melhor modelo para a detecção de objetos em tempo real

Os modelos de identificação de objetos, aqui chamados de classificadores, podem ser utilizados em imagens e vídeos previamente prontos (estáticos) ou para identificação em tempo real. Os modelos já contruídos e disponibilizados pela biblioteca *OpenCV* podem ser utilizados para aplicações em imagens pela *webcam*. Para este trabalho, também foi testada essa possibilidade de aplicação.

O classificador 3 foi utilizado nesta aplicação devido aos resultados satisfatórios obtidos nos testes anteriores. O resultado do uso da *webcam* pode ser visto na Figura 4.17 abaixo.

Figura 4.17 – Identificação de uma caneca em tempo real com o uso da *webcam*.



Fonte: Autora.

Também foram realizados testes com dois objetos diferentes, um copo e uma taça. Em ambos os testes, o classificador não identificou canecas, o que foi um resultado esperado satisfatório. Estes testes podem ser observados nas Figuras 4.18 e 4.19 abaixo.

Figura 4.18 – Teste do classificador 3 com um copo na *webcam*.

Fonte: Autora.

Figura 4.19 – Teste do classificador 3 com uma taça na *webcam*.

Fonte: Autora.

5 Considerações Finais

5.1 Conclusão

Os estudos de visão computacional atualmente são muito importantes devido a sua vasta quantidade de aplicações, em uma área que pode ser considerada nova no ramo da computação. Suas criações com métricas de classificação confiáveis são demoradas e devem ser realizadas com parâmetros devidamente ajustados, além disso, deve-se contemplar as diversas variações do alvo que deseja ser identificado.

O estudo de visão computacional e a criação de modelos do tipo *Haar Cascade* podem agregar em diversas problemáticas no ramo da engenharia. Dentre elas, podemos citar a identificação de padrões para o controle de estoque de uma fábrica com a chegada de novas peças ou materiais e, também, para o controle de qualidade de uma linha de montagem, como a identificação das peças corretas para cada tipo de demanda a ser executada.

Para estudos voltados à Engenharia Física, podem ser criados modelos para a identificação de padrões em ligas ou amostras desconhecidas de diferentes materiais. Também pode ser utilizado para detecção, de forma mais rápida, de componentes presentes em placas de circuitos em aplicações para robótica, além de outras aplicações.

Vale ressaltar que, por ser uma problemática de classificações de aprendizados de máquina do tipo supervisionado, será necessário fornecer ao computador uma amostra inicial contendo o comportamento que é desejado que ele reproduza. Portanto, quanto mais amostras, melhor será o resultado dos classificadores.

Os resultados obtidos pelo classificador 1 desenvolvido neste trabalho não foram muito promissores. Para imagens com a presença de canecas, ele não desempenhava bem em identificar por completo a região em que a caneca se encontrava.

Já para o classificador 2 desenvolvido, onde foi aumentada a quantidade de modelos de canecas para seu treinamento, sua eficiência foi comprometida pela grande quantidade de falsos positivos. Todavia, a identificação da região onde a caneca se encontrava era de melhor performance que o classificador 1.

Por fim, para o classificador 3, modelo com os melhores resultados, apesar de ainda identificar regiões em que não continham canecas em testes realizados, no geral, apresentou um bom desempenho e pôde ser utilizado para identificação das canecas em tempo real com o uso da *webcam*. Além disso, para imagens que não continham canecas, ele foi o que identificou a menor quantidade de falsos positivos entre os classificadores desenvolvidos.

Para resultados mais satisfatórios, são necessários treinamentos com muitas imagens

positivas de diferentes formas e tamanhos para os objetos a serem identificados, ademais, muitas imagens negativas para ajudar a reduzir falsos positivos são necessárias, visto que alguns objetos podem ser semelhantes. Além disso, pode-se ajustar os parâmetros de treino, conforme realizado neste trabalho.

Os ajustes dos parâmetros podem melhorar muito o desempenho dos classificadores, entretanto, quanto mais ajustes são feitos visando melhorar a qualidade do classificador, mais demorado será o treinamento. Consequentemente, o treino exigirá mais do processamento da máquina utilizada. Para treinos em computadores domésticos, os parâmetros escolhidos foram os suportados pelo computador utilizado.

Para o objetivo do trabalho proposto, que foi o Estudo de Visão Computacional e a criação de um *Haar Cascade* utilizando a biblioteca *OpenCV* em *Python* para detecção de objetos, pode-se concluir satisfatórios os resultados obtidos.

Referências

ALPAYDIN, E. *Introduction to machine learning*. [S.l.]: MIT press, 2020.

BORGES, L. E. *Python para desenvolvedores: aborda Python 3.3*. [S.l.]: Novatec Editora, 2014.

CHAVES, B. B. *Estudo do algoritmo AdaBoost de aprendizagem de máquina aplicado a sensores e sistemas embarcados.* Tese (Doutorado) — Universidade de São Paulo, 2012.

COELHO, F. C. *Computação Científica com Python*. [S.l.]: Lulu. com, 2007.

GÉRON, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. [S.l.]: O'Reilly Media, 2019.

HUNTER, J. D. *History*. 2021. Disponível em: <https://matplotlib.org/stable/users/history.html>. Acesso em: 28 junho 2021.

JUPYTER. *Jupyter Notebook*. 2021. Disponível em: <https://jupyter.org/>. Acesso em: 23 junho 2021.

MILANO, D. de; HONORATO, L. B. *Visao computacional*. 2014.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, Manole, v. 1, n. 1, p. 32, 2003.

OPENCV. *Cascade Classifier Training*. 2021. Disponível em: <https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html>. Acesso em: 29 junho 2021.

OPENCV. *Haar Cascade*. 2021. Disponível em: <https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html>. Acesso em: 24 junho 2021.

OPENCV. *Introduction to OpenCV*. 2021. Disponível em: <https://docs.opencv.org/master/d1/dfb/intro.html>. Acesso em: 24 junho 2021.

RUDEK, M.; COELHO, L. d. S.; JR, O. C. Visão computacional aplicada a sistemas produtivos: Fundamentos e estudo de caso. *XXI Encontro Nacional de Engenharia de Produção-2001, Salvador*, 2001.

SEO, N. *Merge vec files created by createsamples*. 2007. Disponível em: <http://note.sonots.com/SciSoftware/haartraining/mergevec.cpp.html>. Acesso em: 1 julho 2021.

SOKOLOVA, M.; JAPKOWICZ, N.; SZPAKOWICZ, S. Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. In: SPRINGER. *Australasian joint conference on artificial intelligence*. [S.l.], 2006. p. 1015–1021.

SPECTRUM, I. *Interactive: The Top Programming Languages*. 2019. Disponível em: <https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2019>. Acesso em: 24 junho 2021.

WULFE, B. *Mergevec*. 2014. Disponível em: <https://github.com/wulfebw/mergevec>. Acesso em: 1 julho 2021.