ADILSON TORRES GREGÓRIO DE SOUZA
JHONATA ANTUNES

Monitoramento inteligente de ambientes com foco no bem-estar

São Paulo
2018

ADILSON TORRES GREGÓRIO DE SOUZA
JHONATA ANTUNES

Monitoramento inteligente de ambientes com foco no bem-estar

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro de Computação.

São Paulo
2018

ADILSON TORRES GREGÓRIO DE SOUZA
JHONATA ANTUNES

Monitoramento inteligente de ambientes com foco no bem-estar

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro de Computação.

Área de Concentração:

Engenharia de Computação

Orientador:

Prof. Dr. Reginaldo Arakaki

Co-orientador:

Prof. Dr. Jorge Luis Risco Becerra

São Paulo
2018

Nome: SOUZA, Adilson Torres Gregório de

Nome: ANTUNES, Jhonata

Título: Monitoramento inteligente de ambientes com foco no bem-estar

Monografia (Trabalho de Conclusão de Curso) apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do Título de Bacharel em Engenharia na área de Engenharia de Computação.

Trabalho entregue para o departamento em:

Responsáveis

Prof. Dr. Reginaldo Arakaki (Orientador)

Prof. Dr. Jorge Luis Risco Becerra
(Co-orientador)

Adilson Torres Gregório de Souza
(Orientado)

Jhonata Antunes (Orientado)

AGRADECIMENTOS

A Deus, que iluminou o caminho durante esta caminhada.

A esta universidade, seu corpo docente, direção e administração que proporcionaram um ambiente adequado para o aprendizado.

Ao professor Reginaldo Arakaki, pela orientação e pelo constante estimulo transmitido durante todo o trabalho.

Aos familiares, pelo carinho, amor e incentivo.

Aos amigos, companheiros de trabalho que fizeram parte desta formação e continuarão presentes por toda a vida.

E a todos que colaboraram direta ou indiretamente, na execução deste trabalho.

"Na vida, não existem soluções. Existem forças em marcha: é preciso criá-las e, então, a elas seguem-se as soluções."

-- Antoine de Saint-Exupéry

RESUMO

Com o avanço e barateamento da tecnologia das câmeras digitais, além de questões sociais, como segurança, levaram a um acentuado aumento na quantidade de câmeras distribuídas por todo o mundo. Na área de monitoramento, elas são utilizadas, na maior parte, como um recurso auxiliador, de forma a subutilizar o potencial de informação contida nas imagens. Atualmente, não existem muitas opções de ferramentas inteligentes de monitoramento capazes de automatizar o processo, dificultando tomadas de ações preventivas ou corretivas em tempo real em sistemas de larga escala. O objetivo do projeto foi desenvolver a arquitetura de um sistema de monitoramento inteligente de ambientes que, através da analise de vídeo, seja capaz de detectar eventos e realizar um conjunto de ações específico para cada evento. Os eventos estão relacionados ao bem-estar das pessoas presentes no ambiente, portanto, podem ser acidente de trânsito, assalto à mão armada, desmaio, incêndio, agressão física, entre outros. As ações dependem diretamente do evento, e podem ser acionar a polícia, corpo de bombeiros e ambulância. Para o projeto, foram implementados todos os módulos descritos pela arquitetura, além de um módulo de interface para gerenciar o sistema. Quanto a detecção de eventos, foi desenvolvido um algoritmo capaz de detectar desmaios. No qual a ação para o evento de desmaio é o envio de um e-mail contendo informações a respeito do tipo de evento, data, endereço e uma imagem. Portanto, se uma pessoa desmaiar em um ambiente monitorado pelo sistema, será gerado um e-mail de alerta.

Palavras-Chave – Arquitetura. Detecção de objetos. Detecção de desmaio. Eventos. Ações.

ABSTRACT

With the advancement and cheapness of the technology of digital cameras, in addition to social issues, such as safety, have led to a sharp increase in the number of cameras distributed around the world. In the monitoring area, they are used, for the most part, as a helper resource, in order to underutilize the information potential contained in the images. Currently, there are not many options for intelligent monitoring tools capable of automating the process, making it difficult to take preventive or corrective actions in real time on large scale systems. The objective of the project was to develop the architecture of a system of intelligent monitoring of environments that, through video analysis, is able to detect events and perform a set of actions specific to each event. The events are related to the well-being of the people present in the environment, therefore, they can be traffic accident, armed robbery, fainting, fire, physical aggression, among others. The actions depend directly on the event, and may be triggering the police, fire brigade and ambulance. For the project, all modules described by the architecture were implemented, as well as an interface module to manage the system. As for event detection, an algorithm capable of detecting fainting was developed. In which the action for the event of fainting is sending an e-mail containing information regarding the type of event, date, address and an image. Therefore, if a person faints in an environment monitored by the system, an alert e-mail will be generated.

Keywords – Architecture. Object detection. Fainting detection. Events. Actions.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API *Application Program Interface*. 28, 29, 54, 64, 69, 71

AWS Amazon Web Services. 56, 57, 69

CD *Continuous Delivery*. 30

CI *Continuous Integration*. 30

CNN *Convolutional Neural Network*. 35, 36

COCO *Common Objects in Context*. 35

CUDA *Compute Unified Device Architecture*. 39, 62, 63

DNN *Deep Neural Network*. 42, 63

DNS *Domain Name System*. 40

FPN *Feature Pyramid Network*. 35

FPS *Frames Per Second*. 58, 62, 72

FRCN *Faster R-CNN*. 35

GPU *Graphics Processing Unit*. 39, 63

HTTP *HyperText Transfer Protocol*. 53, 54, 55, 65

IaaS *Infrastructure as a Service*. 32

IoT *Internet of Things*. 33, 44

IoU *Intersection over Union*. 25, 26, 39

IP *Internet Protocol*. 16, 53, 55

IPSec *Internet Protocol Security*. 40

MQTT *Message Queuing Telemetry Transport*. 54, 55, 65, 71

NAT *Network Address Translation*. 41

NIST *National Institute of Standards and Technology*. 31

NMS *Non-Maximum Suppression*. 26, 39

OpenCV *Open Computer Vision*. 39, 40, 45, 55, 56, 70

PaaS *Platform as a Service*. 32

REST Representational State Transfer. 64

RFID *Radio frequency identification*. 33

RM-ODP *Reference Model of Open Distributed Processing*. 18, 26, 27, 44, 45, 47

SaaS *Software as a Service*. 32

SDN *Software-Defined Networking*. 40

SES Simple Email Service. 69

SHAVE *Streaming Hybrid Architecture Vector Engine*. 41

SMS *Short Message Service*. 47

SPARC *Scalable Processor Architecture*. 41

SSD *Single Shot Detector*. 35

UDP *User Datagram Protocol*. 53, 55

UML *Unified Modeling Language*. 28

VPN *Virtual Private Network*. 53

VPU *Vision Processing Unit*. 41, 42

WSN *Wireless sensor networks*. 33

YOLO *You Only Look Once*. 35, 39, 45, 62, 63

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

Com o avanço da tecnologia, as antigas câmeras analógicas estão sendo substituídas pelas câmeras IP, que são melhores em termos de resolução, visão noturna, entre outras características. O barateamento da tecnologia e problemas sociais, como segurança, levaram a um acentuado aumento na quantidade de câmeras distribuídas por todo o mundo. Na área de monitoramento, elas são utilizadas, na maior parte, como um recurso auxiliador. “Elas não substituem o guarda, simplesmente são uma ferramenta muito importante de apoio"(BERMÚDEZ, 2014).

O desenvolvimento de estudos e tecnologias na área de inteligência artificial possibilitou que alguns sistemas aplicassem inteligência ao monitoramento, como o ShotSpotter. Esse sistema é capaz de detectar precisamente a posição de um disparo de arma de fogo. Primeiro, softwares especializados analisam sinais de áudio para determinar potenciais disparos de arma de fogo; depois, o software determina a localização da fonte do áudio e analisa as características do som para determinar se é parecido com um disparo; em caso positivo, envia alertas para a polícia e equipes de emergência (ShotSpotter, 2018).

Na área de monitoramento inteligente por análise de vídeos, existe uma patente registrada na Oficina de Patentes Europeia, de número AU2018100039, que propõem soluções para o problema de segurança. O sistema identifica e rastreia pessoas que causam ameaça a segurança pública, incluindo determinação de rotas de fuga para longe da ameaça (EPO, 2018).

Os sistemas de monitoramento comuns, por vídeos de vigilância, são amplamente empregados, servindo como ferramenta auxiliar, e, pelo fato de ser analisado por um operador humano, possui múltiplas funções, porém, é pouco eficiente, pois requer muito trabalho manual. Os sistemas inteligentes, por sua vez, são eficientes e escaláveis, porém, ainda estão pouco desenvolvidos. Não há nenhum sistema específico que use monitoramento inteligente baseado em vídeos para detectar eventos, focados no bem estar, e prestar

assistência, quando necessário.

1.2 Objetivo

O objetivo deste trabalho é desenvolver a arquitetura de um sistema de Monitoramento Inteligente de Ambientes que seja capaz de processar imagens recebidas de câmeras para detectar eventos, relacionados ao bem estar das pessoas presentes no ambiente, e realizar um conjunto de ações específicas para cada evento. O sistema deve receber as imagens, utilizar técnicas de inteligência artificial para reconhecer objetos, aplicar algoritmos detectores de eventos e, quando detectado um evento, tomar um conjunto de ações. Os eventos estão relacionados ao bem estar das pessoas, portanto, podem ser acidente de trânsito, assalto à mão armada, desmaio, incêndio, agressão física, entre outros. As ações dependem diretamente do evento, e podem ser acionar polícia, corpo de bombeiros, ambulância e, dependendo do nível de integração disponível, acionar semáforos para isolar áreas ou, até mesmo, acionar robôs ou dispositivos de prestação de ajuda para atuarem no local do evento.

1.3 Justificativa

O rápido desenvolvimento da tecnologia das câmeras as tornou o principal meio de realizar o monitoramento de ambientes, porém, o mesmo não ocorreu com as ferramentas de monitoramento, de forma que as imagens são utilizadas como apoio, em vez de serem o ponto central, não aproveitando o grande potencial de informação que elas possuem e dificultando tomadas de decisão em tempo real em sistemas de larga escala.

Sistemas como o ShotSpotter iniciam a caminhada rumo a ferramentas de monitoramento inteligente, porém, esse sistema, em específico, além de detectar apenas um tipo de evento, depende da instalação de sensores de áudio, de forma a reduzir a aplicabilidade do sistema. Para contornar esse problema, é preciso desenvolver um sistema cuja principal fonte de dados já esteja difundida em diversos ambientes. Essa fonte de dados é o vídeo das câmeras, onde seu uso vem aumentando constantemente.

Alguns eventos, como acidentes, requerem que as vítimas sejam atendidas o mais rápido possível, contudo, em locais pouco movimentados, pode ser difícil que alguma pessoa detecte o evento, determinar o local do evento ou acionar a ajuda. Portanto, é necessário um sistema que seja capaz de detectar eventos em tempo real, para realizar

as ações necessárias para prestar assistência o quanto antes. Para aumentar a área de atuação do sistema, é preciso utilizar um mecanismo de observação que já esteja bem difundido nos diversos tipos de ambientes, como vídeos de vigilância.

1.4 Organização do trabalho

A seção 2 aborda os aspectos conceituais necessários para o desenvolvimento do trabalho, tendo para cada conceito e técnica utilizado uma breve explicação, tais como processamento de imagens, *machine learning* e RM-ODP. Na seção 3 é descrito com mais detalhes as principais tecnologias utilizadas. Na seção 4 é apresentado a metodologia utilizada no trabalho, como gerenciamento de tarefas, versionamento git e pesquisas realizadas. Na seção 5 é detalhado a especificação do projeto. Na seção 6, são detalhados os detalhes de implementação de cada serviço e módulo. Na seção 7, é documentado os testes e avaliações dos resultados obtidos. Na seção 8, é apresentado as considerações finais do trabalho.

2 ASPECTOS CONCEITUAIS

2.1 Processamento de imagens

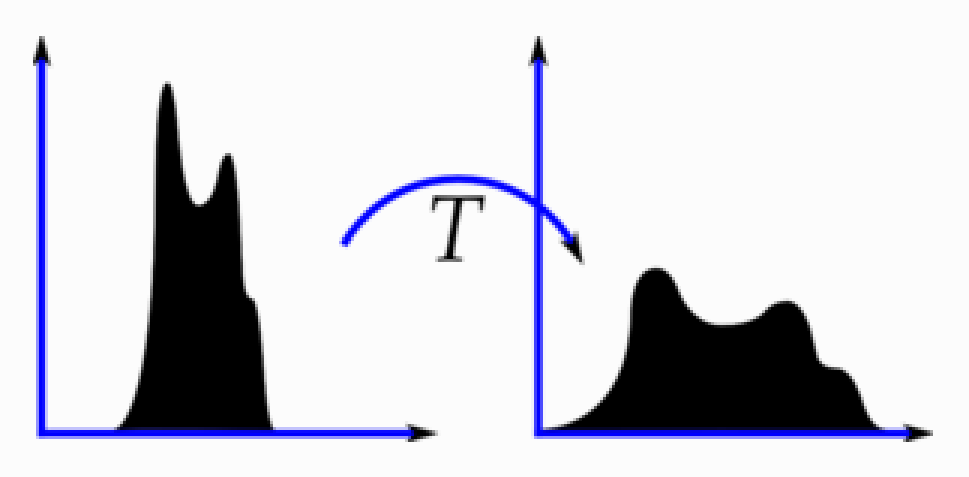
2.1.1 Equalização de histogramas

Histograma é a distribuição de frequências de um certo fenômeno ou característica, que pode ser representado visualmente por gráficos de colunas ou barras. Assim sendo, o histograma de uma imagem representa a distribuição de frequência dos componentes de cor. Para cores representadas com 8 bits, a abscissa varia de 0 a 255, representando a intensidade de um canal de cor, e a coordenada representa a quantidade de pixels de uma certa intensidade de cor.

Quando um certo ambiente não possui iluminação equilibrada, a imagem capturada pode ter uma alta frequência de pixels com intensidade de cor próximas, como no primeiro histograma da Figura 1. Isto é característico de imagens com pouco contraste de cores. A equalização de histogramas é uma operação que distribui as frequências para ambas as extremidades do gráfico. Imagens com as intensidades de cores melhor distribuídas apresentam um bom contraste.

O contraste é uma característica que evidencia os limites e bordas dos objetos contidos nas imagens, pois há uma variação brusca de cores. Essa característica é muito positiva para algoritmos detectores de objetos que se baseiam no contorno dos objetos.

Figura 1: Exemplo do resultado da operação de equalização de histogramas



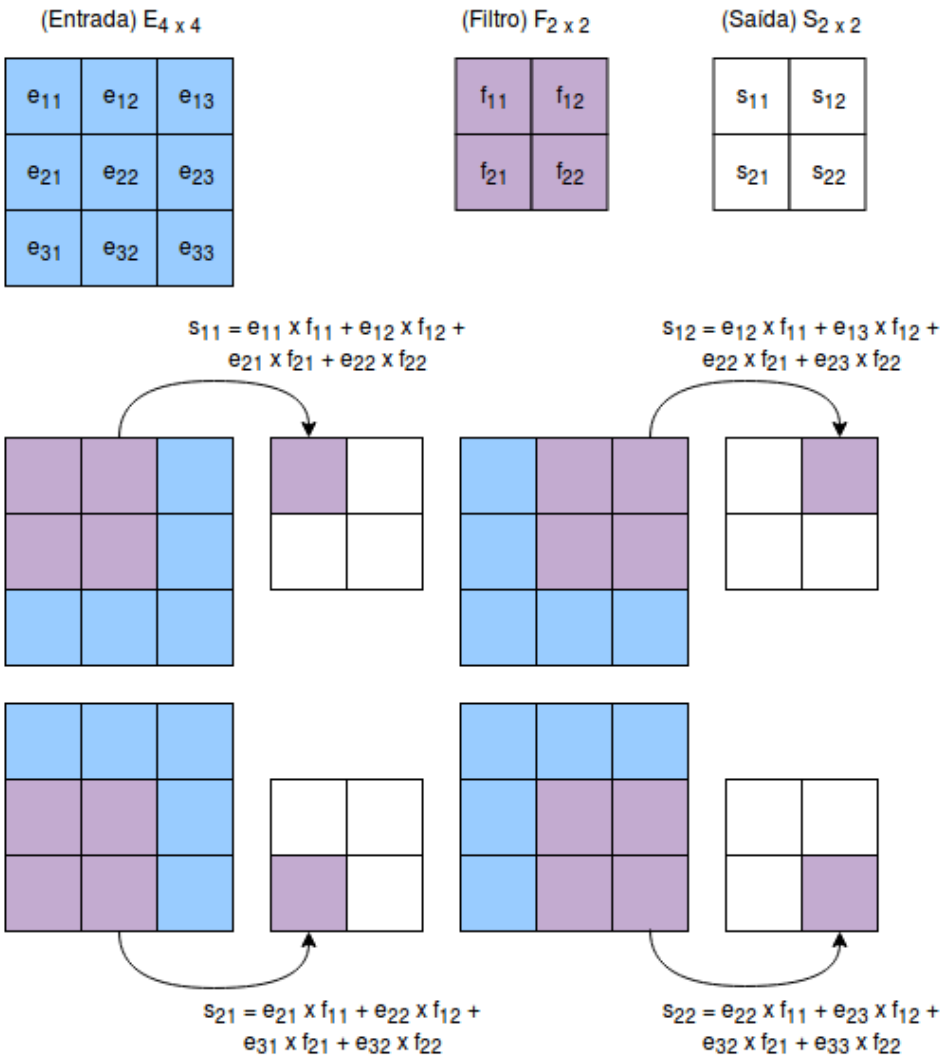
Fonte: (OpenCV-Python Tutorials, 2018)

2.1.2 Convolução matricial

No contexto de redes neurais, a convolução de matrizes, também referenciada apenas como convolução, é uma operação que empenha um importante papel no funcionamento de redes neurais convolucionais. Sua importância se dá pelo fato de que tal operação permite extrair características das imagens, como bordas, relevo, entre outras. Essa extração de características é fundamental para as redes neurais que detectam objetos.

O processo de convolução consiste em uma matriz $E_{n\times m}$ e um filtro $F_{f\times f}$, onde o filtro se sobrepõe a matriz de entrada, calcula um valor de saída, desliza para a posição seguinte e repete os passos seguintes até que o filtro tenha se sobreposto a todas as regiões possíveis (ver Figura 2). O cálculo do valor de saída de cada sobreposição é dado pela soma dos produtos dos valores sobrepostos da matriz de entrada e do filtro, como exemplificado na Figura 2. As dimensões da matriz de saída é dado por $(n-f+1)\times(m-f+1)$ (Andrew Ng., 2017).

Figura 2: Exemplo simples de convolução



Fonte: Autores

2.1.2.1 Padding

A convolução normal leva em consideração muito fracamente os valores das bordas da imagem, enquanto que os centrais, são utilizados mais de uma vez no cálculo de um valor do *feature map*. Para contornar esse problema, dando maior expressividade aos pixeis da borda, é possível adicionar um *padding*, ou seja, envolver a imagem com pixeis de valor nulo. Vale notar que os filtros geralmente (é possível observar na literatura) tem tamanho ímpar. Seja uma imagem de tamanho $n \times m$, *padding* p e um filtro $f \times f$, então, a matriz resultante (*feature map*) terá tamanho $(n + 2p - f + 1) \times (m + 2p - f + 1)$ (Andrew Ng., 2017).

2.1.2.2 Strided convolution

Strided convolution é uma convolução com um passo maior que um, ou, uma convolução normal é aquela onde $stride = 1$ (passo igual a um). O passo se refere a quantos pixels deve se deslocar (horizontal e verticalmente) o filtro sobre a imagem. Dessa forma, realizando um salto maior com o filtro sobre a imagem, os pixels serão considerados em menos valores da feature map resultante. Isso diminui a precisão da extração de características, porém, reduz o tamanho da saída (*feature map*), diminuindo o processamento e consumo de memória da rede neural, que pode utilizar diversos filtros ao mesmo tempo. A convolução de uma imagem de tamanho $n \times m$ e um filtro $f \times f$, com *padding* p e *stride* s resulta em uma matriz de tamanho $(\frac{n+2p-f}{s} + 1) \times (\frac{m+2p-f}{s} + 1)$ (Andrew Ng., 2017).

2.1.2.3 Pooling

Pooling é um tipo de técnica amplamente utilizada nas camadas das redes neurais convolucionais. Ela consiste em escolher um tamanho de janela de tamanho $f \times f$, um passo s, e deslizar a janela sobre a imagem. O processo é similar ao de convolução, onde a diferença está na computação do resultado. Os parâmetros f e s não participam do processo de aprendizagem, e são chamados de hiper-parâmetros, pois não existe uma regra para a definição de seus valores (Andrew Ng., 2017). A seguir, os dois tipos de *pooling* mais utilizados:

- *Max pooling*: consiste em escolher o máximo valor dentro da janela;
- *Average pooling*: consiste em calcular o valor médio dos valores da janela.

2.2 Machine Learning

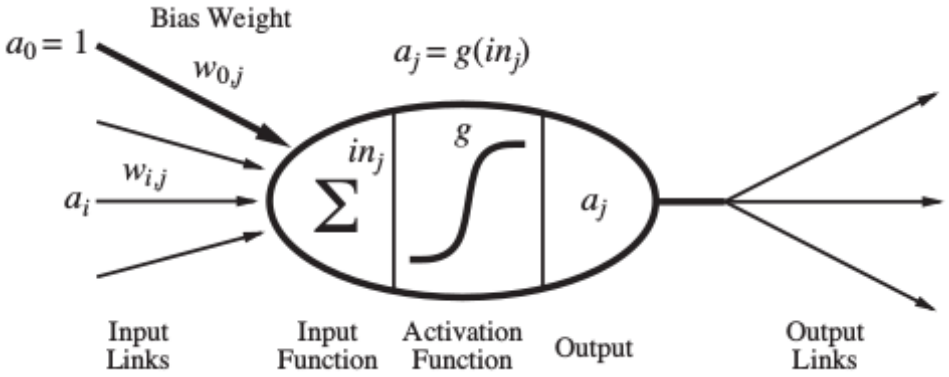
2.2.1 Redes Neurais

Os resultados dos estudos da neurociência assumem como hipótese que as atividades mentais consistem, primariamente, em atividades eletroquímicas nas redes de células do cérebro, chamadas neurônios (Russell e Norvig, 2009). Inspirados nesses princípios, foram desenvolvidos modelos matemáticos do cérebro humano, de forma a gerar grandes contribuições à área de inteligência artificial, através das redes neurais artificiais.

2.2.1.1 Neurônio artificial

O neurônio artificial é um modelo matemático do neurônio cerebral. Ele consiste em múltiplas ligações de entrada, uma função de ativação e uma ligação de saída, como ilustrado na Figura 3.

Figura 3: Modelo matemático de um neurônio simples. A saída da unidade é a_j, onde a_i é a saída da unidade i e w_{ij} é o peso das ligações da unidade i para esta unidade.

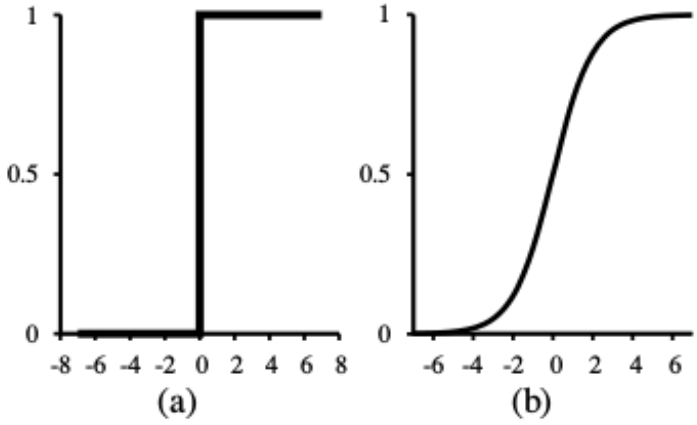


Fonte: (Russell e Norvig, 2009)

As ligações de entrada tem a importante característica de possuir pesos. Cada ligação possui seu valor numérico de peso, de forma que cada uma é percebida com uma certa importância.

A função de ativação é, tipicamente, do tipo limite rígido (*hard threshold*) ou função logística (*logistic function*). No primeiro caso, a unidade é chamada de *perceptron* e no segundo, *sigmoid perceptron*. Essas duas funções garantem a importante propriedade de que a rede, como um todo, pode representar uma função não linear. A vantagem de usar a função logística é que ela é diferenciável em todo seu domínio (Russell e Norvig, 2009).

Figura 4: Gráfico das funções (a) *hard threshold function* e (b)*logistic function*



Fonte: (Russell e Norvig, 2009)

2.2.1.2 Estrutura das redes neurais

Uma vez definidos os modelos matemáticos do neurônio, é preciso conectá-los, para formar uma rede. Há duas formas fundamentais de realizar esta tarefa, onde a primeira é conhecida como *feed-forward network* e a segunda, *recurrent network*.

As *feed-forward network* possuem conexão em apenas uma direção, de modo a formar um grafo não cíclico. Todo nó recebe como entrada dados dos nós anteriores, e passam dados para os nós posteriores, sem formar laços (Russell e Norvig, 2009). Este tipo de rede é uma função apenas dos valores de entrada, onde não há estados internos.

As *recurrent network* possuem conexões em ambas as direções, de modo que a saída de nós posteriores podem ser entrada de nós anteriores (Russell e Norvig, 2009). Este tipo de rede é uma função dos valores de entrada e dos estados internos. Dessa forma, uma característica importante deste tipo de rede é que ela passou memória de curto prazo, sendo um modelo mais próximo do cérebro humano. Uma de suas desvantagens é que a alta complexidade e o fato de que o modelo pode alcançar estados caóticos, apresentar oscilações ou comportamento caótico.

2.2.1.3 Rede neural convolucional

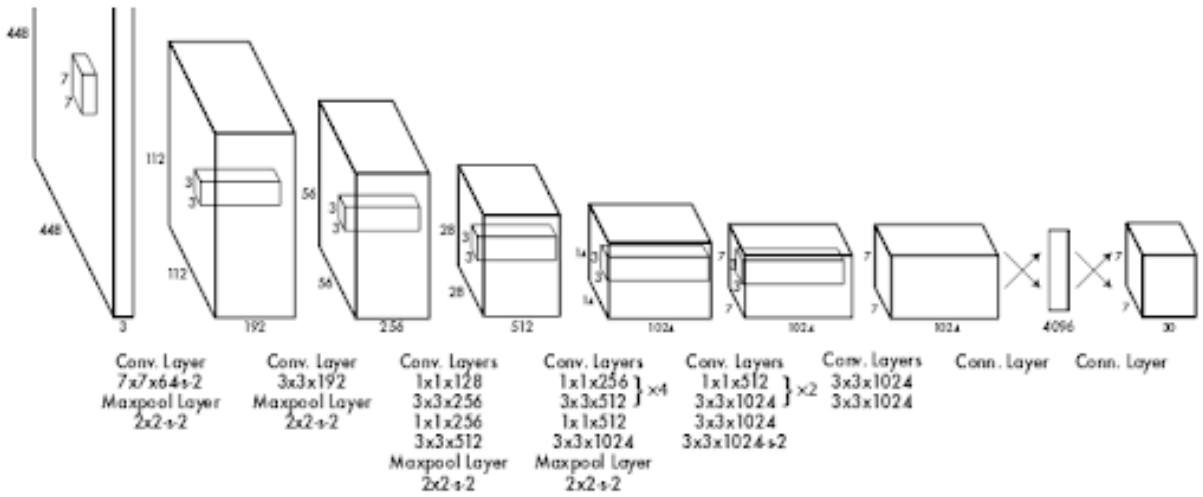
Rede neural convolucional é um tipo de rede neural artificial que é composta por camadas de convolução (*convolucional layers*) e redes completamente conectadas (*fully connected layers*). Ela é uma rede do tipo *feed-forward network* e pode conter *skip connections*, que consiste em ligar a saída de uma camada à entrada de outra camada cuja

distância é maior que um. Sua maior aplicação está na área de processamento de imagens, devido ao fato de possuir grande eficiência em extração de características (*feature extraction*) de imagens, fornecida pelas camadas convolucionais.

As *convolucional layers* são, geralmente, as primeiras da rede, seguidas das *fully connected layers*. Elas são compostas por convolução do tipo normal, *strided convolution*, *max pooling* e *average pooling*. Uma característica importante das *convolucional layers* é que diversos filtros podem ser aplicados em uma mesma camada, de forma que a dimensão dos dados diminui ao longo das camadas, enquanto que a profundidade aumenta.

As *fully connected layers* são camadas de rede neural formadas por neurônios completamente conectados, ou seja, a saída de um nó é uma das entradas de todos os nós da camada posterior. Essas camadas são utilizadas geralmente para codificar os dados de saída.

Figura 5: Exemplo de CNN do YOLO v1, que contém 24 *convolucional layers* e 2 *fully connected layers*



Fonte: (Redmon et al., 2015)

2.2.2 IoU

Intersection over Union (IoU) ou também conhecido como índice de Jaccard é uma medida de similaridade em conjuntos. Na área de *machine learning* é utilizado como uma medida de acurácia de um detector de objetos em um dado *dataset*. O cálculo é feito através de duas áreas, a primeira área é considerado um retângulo sobre o objeto a ser detectado e a segunda área o retângulo predito pelo detector de objetos. Com essas duas áreas o cálculo do IoU se torna a área de intersecção entre as áreas dos retângulos sobre

a área de união dos retângulos, valores abaixo de 0.5 são considerados de baixa qualidade enquanto que valores próximos de 1 são considerados adequados (Kosub, 2016).

2.2.3 NMS (Non-Maximum Suppresion)

Non-Maximum Suppression (NMS) é um algoritmo de pós processamento de imagem responsável por agrupar as detecções feitas sobre um mesmo objeto. Um detector de objetos possui essencialmente 3 passos, o primeiro é propor um espaço de buscas em janelas, o segundo é pontuar e refinar cada janela com um classificador ou regressor, e o último passo é juntar as janelas quando estas pertencem a um mesmo objeto, este último passo é feito através do NMS. Ele funciona selecionando altas pontuações detectadas e remove os vizinhos próximos com baixa confiança pois tem alta probabilidade de serem do mesmo objeto, assim ele realiza uma busca gulosa por máximos locais descartando vizinhos próximos. Uma de suas limitações é a seleção em imagens altamente populadas por objetos, onde a supressão de vizinhos próximos se torna mais difícil de ser realizada e requer um algoritmo mais robusto (Hosang, Benenson e Schiele, 2017).

2.2.4 K-means Clustering e caixas delimitadoras

K-means é um algoritmo de agrupamento, seu objetivo é obter k centroides que são usados para definir grupos, esse processo é chamado de *clustering* (Piech, 2013). Seu funcionamento consiste em inicialmente escolher aleatoriamente os centroides de cada grupo, para cada grupo ele seleciona pontos pertencentes a este grupo próximos ao centroide corrente, após feito isso segue alternando entre definir novos centroides baseados nos pontos selecionados para o grupo e a partir dele escolher novos pontos ao grupo.

Uma das formas de calcular caixas delimitadoras para os objetos detectados é através de *templates* chamados anchor boxes (caixas âncoras), essas caixas evitam que caixas delimitadoras com grandes dimensões sejam favorecidas (Yadav, 2017). Para obter as anchor boxes é utilizado o algoritmo K-means e IoU como métrica de distância para minimizar erros em caixas pequenas, assim cada anchor boxes define uma caixa com largura e alturas definidas no processo de treino do detector de objetos e são usadas na detecção como base para calcular as caixas delimitadoras de cada objeto.

2.3 Arquitetura RM-ODP

O *Reference Model of Open Distributed Processing* (RM-ODP) é um modelo de referência para arquitetura de sistemas distribuídos. A arquitetura fornece mecanismos para arquitetar softwares com processamento e informação distribuídas, suportando integração e interoperação das aplicações de uma forma confiável e consistente (Putman, 2001). O RM-ODP contém conceitos precisos e regras de estruturação que são usadas para a especificação do sistema, onde descreve como capturar as necessidades dos *stakeholders*, como capturar as semânticas do processamento de informação, como especificar os componentes suas interações e restrições, como selecionar produtos e tecnologias úteis para implementar o sistema. Cada área de foco do sistema é descrita de uma maneira consistente pelo modelo, de modo que cada decisão feita em uma área é refletida nas demais (Putman, 2001).

Existem 5 categorias de regras no RM-ODP, que junto com os conceitos servem como a fundação base para os pontos de vistas. As regras básicas são usadas por toda a especificação, nela é considerado a informação, os dados, processamento distribuído e o que constitui um sistema de processamento distribuído aberto (ODP). As regras de modelagem dos objetos são usadas para construir a arquitetura, considerando os objetos, estados, interfaces, entre outros elementos de modo que cada especificação da arquitetura é construída em termos de um modelo baseado em objetos. As regras estruturais são usadas como definição dos "como" do projeto, como descrever a comunicação entre interfaces, o que deve conter em um contrato, o que constitui uma política, quais são os grupos de objetos, entre outros. As regras de especificação são usadas para prover consistência entre os pontos de vistas, definindo o que é uma composição e um componente e como são relacionados, como especificar um comportamento, uma assinatura de uma interface, entre outros. As regras de conformidade tem a função de testar as conformidades dos processos e pontos onde esses testes podem ocorrer (Putman, 2001).

Os pontos de vistas do RM-ODP separa as áreas de preocupação do sistema em partes gerenciáveis para especificação da arquitetura. São 5 pontos de vistas: Empresa; Informação; Computação; Engenharia; e Tecnologia.

O ponto de vista empresa tem como perspectiva o modelo da empresa, ele deve ser facilmente entendido pelos *stakeholders* de negócio, garantindo que as necessidades de negócio estão satisfeitas pela arquitetura e provê uma especificação que permite a validação dessas necessidades com os usuários finais. Esse ponto de vista também permite que uma solicitação de um subsistema por parte de um cliente seja solicitada para ser arquitetada

e implementada seja como um produto ou parte de um sistema já existente, as partes que definem uma ferramenta, suas interações e as políticas que são aplicadas a elas também são fatores importantes a serem considerados nesse ponto de vista (Putman, 2001).

O ponto de vista informação define o conjunto de informação do sistema de duas formas, a primeira é o conteúdo do sistema, a segunda é a informação sobre o processamento do sistema, ou seja, seu comportamento. Na perspectiva de conteúdo a informação é vista como um modelo de banco de dados, sendo uma representação lógica dos dados no sistema distribuído. Na perspectiva do comportamento, é considerado as regras a serem seguidas no sistema, as políticas definidas pelos *stakeholders* sobre o sistema inteiro, sendo definido as restrições de todos os aspectos do sistema, restrições estas definidas pelas políticas, regras de possíveis mudanças no estado, invariantes no sistema, atributos de qualidades e a informação em si (Putman, 2001).

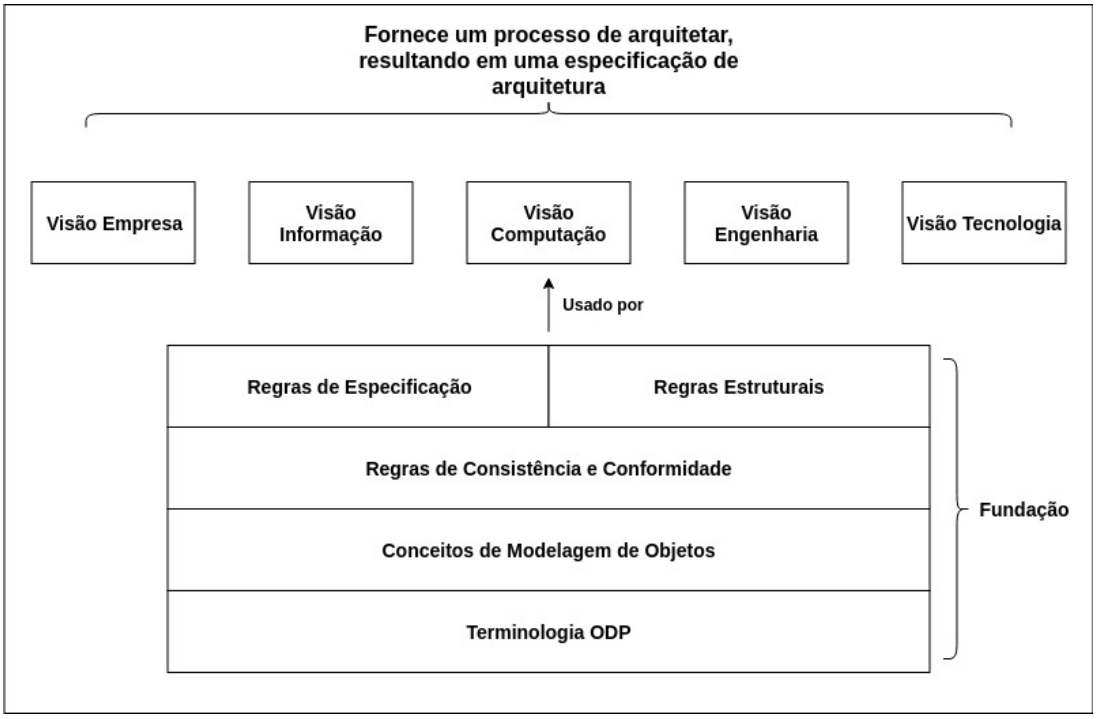
O ponto de vista computação divide o sistema em módulos funcionais que são capazes de serem distribuídos, esse ponto de vista toma como visão a perspectiva de um arquiteto de componentes da aplicação e interfaces do programa. Assim como modelos similares de arquitetura, como o modelo lógico do UML, a visão computação caracteriza os detalhes dos componentes e interfaces desconsiderando a forma de distribuição e aspectos de implementação, que são tarefas da visão engenharia. São definidos aspectos relacionados às interfaces (APIs), assim como a estrutura que irá garantir as qualidades do sistema, isto é, escalabilidade, interoperabilidade, segurança, portabilidade, entre outros aspectos que satisfazem as necessidades de negócio (Putman, 2001).

O ponto de vista engenharia expõe a natureza distribuída do sistema e disponibiliza definições que permitem uma abstração das restrições do sistema. É definido as características da infraestrutura distribuída, chamadas remota de processamento, comunicação cliente-servidor, interfaces assíncronas para sinalização, nível de mobilidade do software e interfaces, serviços com garantias de tolerância a falhas, entre outros. Essa visão se assemelha a funções de engenheiros de sistemas operacionais e engenheiros de redes, que devem considerar clientes de diversos tamanhos, protocolos de comunicação, servidores, tipo de armazenamento, barramentos, entre outros (Putman, 2001).

O ponto de vista tecnologia define o mapeamento entre os objetos arquitetados e interfaces com padrões específicos, tecnologias, produtos selecionados e código a ser desenvolvido. Descreve onde será aplicado os produtos escolhidos, tecnologias e também permite o teste de conformidade do sistema implementado, em relação a especificação arquitetural. É definido os critérios de seleção para as escolhas e também pontos de re-

ferência paras os testes de conformidade, esses pontos de referência são o programático, perceptivo, inter-funcionamento e intercomunicação. Cada ponto de referência define a informação necessária a ser observada durante os testes e como essa informação é refletida na especificação da arquitetura (Putman, 2001).

Figura 6: Visões do RM-ODP, regras e conceitos usados como base para as visões.



Fonte: (Putman, 2001)

2.4 Microsserviços

Microsserviços é uma abordagem para desenvolvimento de uma aplicação como um conjunto de pequenos serviços, onde cada serviço possui seu próprio processamento e possui comunicação com os demais e são construídos em torno das necessidades de negócios do sistema. Os serviços possuem um gerenciamento centralizado enxuto, servindo para poucas tarefas como identificar falhas, rebalanceamento dos serviços e instanciação dos mesmos. Entre as arquiteturas de microsserviços existem características comuns que destacam este modelo de um modelo monolítico, considerando as vantagens e desvantagens (Fowler e Lewis, 2014).

A divisão de componentes através de serviços é uma das principais características de microsserviços, eles servem como componentes do sistema, cada componente é uma unidade de software independente que pode ser trocada ou melhorada, onde procura mi-

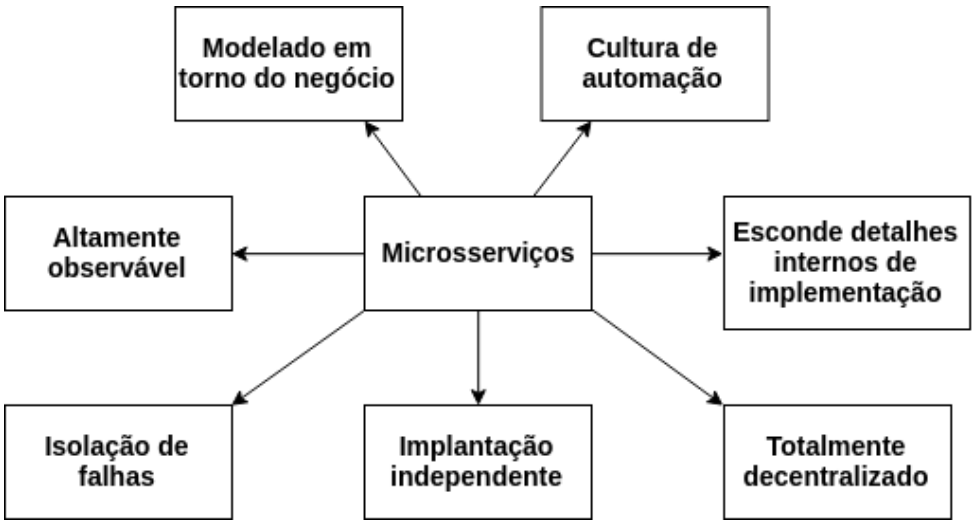
nimizar o uso de bibliotecas compartilhadas entre os serviços. Uma consequência de uso de serviços como componentes de um sistema é a necessidade de interfaces mais explícitas para cada serviço e com baixo acoplamento entre os serviços, um dos efeitos negativos dessa abordagem é que as chamadas remotas entre processos são mais custosas e assim devem possuir uma granularidade mais grossa nas APIs remotas, evitando congestionamento na rede entre troca de dados constantes entre serviços (Fowler e Lewis, 2014).

Por possuir uma base de construção dos serviços em torno das necessidades de negócios, cada serviço possui um conjunto maior de implementação da área de negócio, incluindo uma interface com usuário, banco de dados, comunicação externa com outros produtos para cada serviço implementado. Uma consideração nessa abordagem é o cuidado com a quantidade de tecnologias e ferramentas aplicadas em cada serviço, um equilíbrio deve ser considerado a fim de evitar dificuldade de aplicar manutenção sobre os serviços, isso é obtido através de padrões entre projetos e serviços entre as equipes que desenvolvem os serviços (Fowler e Lewis, 2014).

Seu foco de desenvolvimento é em relação a produtos e não a projetos, sendo que o objetivo de cada equipe esteja direcionado a um produto durante todo seu ciclo de vida, desde o desenvolvimento até sua manutenção, aumentando o contato entre os desenvolvedores e usuários finais. Se destacam também o uso de automatização na infraestrutura de desenvolvimento, tornando possível para as equipes fornecer entregas contínuas (*Continuous Delivery* (CD)) e integração contínua (*Continuous Integration* (CI)) (Fowler e Lewis, 2014).

Um dos fatores importante em determinar o quão pequeno deve ser um serviço é considerar o quanto o serviço está alinhado com a estrutura da equipe, se um serviço possui um conjunto de código grande demais para que uma equipe consiga gerenciar, pode ser necessário separá-lo em partes menores, considerando o objetivo de maximizar os benefícios que a arquitetura em microsserviços oferece, de modo que o uso heterogêneo de tecnologias entre os serviços não aumente a complexidade de gerenciamento, a resiliência (tolerância a falhas) esteja em um nível aceitável e sejam pequenos suficientes para que consiga ter escalabilidade quando necessário para cada serviço (Newman, 2015).

Figura 7: Princípios do Microsserviços.



Fonte: (Newman, 2015)

2.5 Computação em nuvem

Uma das transformações na maneira de fazer computação é utilizá-la como um modelo de serviços que são entregues de uma maneira similar a serviços tradicionais como eletricidade, água e gás. Nesse modelo, usuários acessam os serviços baseado em suas necessidades sem considerar onde os serviços são armazenados e como são entregues. Entre os paradigmas que abordaram essa visão de entregar computação como um serviço está incluso computação distribuída (*Grid Computing*), computação em *cluster (Cluster Computing*) e mais recente computação em nuvem *(Cloud Computing*) (Buyya et al., 2009). Computação em nuvem, caracteriza pela infraestrutura como uma "nuvem" na qual negócios e usuários são capazes de acessar a partir de qualquer lugar do mundo e além disso sob demanda, a partir disso se obteve a transformação de desenvolver software para milhões de usuários consumirem como um serviço mais do que para usuários executar em seus próprios computadores (Buyya et al., 2009).

Como uma definição mais formal, dada pelo instituto de padrões NIST, computação em nuvem é um modelo para possibilitar acessar a uma rede de computadores de recursos de computação compartilhadas de uma maneira ubíqua, conveniente e sob demanda, eles devem ser capazes de rapidamente prover e entregar estes recursos com o mínimo de esforço de gerência e interação com os provedores. O NIST considera para o modelo em nuvem 5 características essenciais, 3 modelos de serviços e 4 modelos de distribuição (Mell e Grance, 2011).

As características essenciais são: Serviços sob demanda; Acesso abrangente a rede; Agrupamento de recursos; Rápida elasticidade; e Serviços com métricas. Os serviços devem prover os recursos computacionais conforme a necessidade, sob demanda, sem depender de interações humanas com os fornecedores. Esses recursos devem está disponível pela rede e poderem ser acessados pelos mecanismos padrões de uso e acesso, tais como celulares, notebooks, tablets, garantindo um uso heterogêneo. Os recursos computacionais são agrupados para serem utilizados por múltiplos consumidores, onde componentes físicos e recursos virtuais são dinamicamente atribuídos conforme a demanda do consumidor. Os recursos e serviços devem ser capazes de ser provisionados e desalocados de uma maneira automática, com garantias de rápida escalabilidade. Tais recursos devem possuir métricas que, considerando o tipo de serviço e recurso, possam ser monitorados, controlados e fornecem uma transparência tanto para o fornecedor do serviço quanto para o consumidor que utiliza (Mell e Grance, 2011).

Para os modelos de serviços, é considerado software, plataforma e infraestrutura como serviços a serem oferecidos. No modelo *Software as a Service* (SaaS), o serviço a ser oferecido ao usuário é o uso de aplicações executadas na infraestrutura da nuvem, com fácil acesso e não necessita gerenciar ou controlar níveis abaixo da aplicação como redes, sistema operacional e servidores. No modelo *Platform as a Service* (PaaS), o serviço a ser oferecido ao usuário é a implantação na nuvem de uma infraestrutura criada pelo usuário com o uso de linguagens de programação, bibliotecas e ferramentas, de modo que também não gerencie os níveis abaixo. No modelo *Infrastructure as a Service* (IaaS), o serviço a ser oferecido são os recursos de processamento, armazenamento, rede e outros fundamentais para que o usuário seja capaz de rodar qualquer aplicação, incluindo sistemas operacionais personalizados, uso de *firewalls*, entre outros (Mell e Grance, 2011).

Os modelos de distribuição consiste em definir o nível de acesso a nuvem, desde um modelo mais restrito que é uma nuvem privada até o modelo mais permissivo como a nuvem pública. No modelo de nuvem privada, a infraestrutura da nuvem é fornecida exclusivamente a uma única organização, tendo seu gerenciamento pela própria organização ou uma terceira. Em um modelo de nuvem comunitária, seu uso é exclusivo de uma comunidade de consumidores de diversas organizações que tenham em comum os mesmos objetivos, políticas. Na nuvem pública, a infraestrutura é oferecida ao público em geral, seu uso pode ser gerenciado por uma empresa, centros acadêmicos, governos, ou combinações diversas. Além dos citados, existe o modelo híbrido que busca a combinação dos outros modelos com o objetivo de permanecer a usuários distintos mas que utilizem as mesmas tecnologias proprietárias ou padronizadas que permitam a portabilidade de dados

e aplicações, tais como balanceamento de carga entre nuvens (Mell e Grance, 2011).

2.6 IoT (Internet of Things)

Internet of Things (IoT) é um paradigma que pode ser definido como uma interconexão de diversos dispositivos, ou coisas, sobre a Internet, com diversas funcionalidades que permitem medições sobre ambiente, comunicação e execução de uma determinada ação. Existem diversas aplicações para IoT, podendo ser utilizado em um ambiente residencial como uma casa inteligente (Kelly, Suryadevara e Mukhopadhyay, 2013), fornecendo reduções de consumo e automatizações de ações como ligar uma lâmpada, bem como pode também ser utilizado em cidades inteligentes, com diversos dispositivos coletando dados, como câmeras, e fornecendo informações úteis para monitoração, administração dos recursos públicos, entre outros.

IoT pode ser definido em 3 categorias, sendo uma internet de pessoa para pessoa, pessoa para máquina ou máquina para máquina. Sendo que a visão do IoT, como conceito e paradigma, considera a presença pervasiva desses dispositivos capazes de interagir entre si e cooperar de forma a criar novos serviços e aplicações que tenham um objetivo em comum. Onde seja possível ter um mundo onde o real, digital e virtual convirjam para criar ambientes cada vez mais inteligentes (Patel, Patel et al., 2016).

Existe um conjunto de características fundamentais que se destacam no IoT. Interconectividade entre os dispositivos, qualquer coisa pode ser interconectada independente do contexto ou localização. Serviços relacionados a coisas, que consideram as restrições dos dispositivos e impactos relacionados, como a proteção da privacidade, consistência semântica entre entre coisas virtuais e físicas, entre outros. Heterogeneidade, cada dispositivo se baseia em tecnologias e hardwares distintos, e são capazes de comunicar entre si através de diferentes plataformas e redes. Mudanças dinâmicas e constantes, cada dispositivo pode mudar seu estado (conectado, desconectado, modo de suspensão, etc.) bem como sua localização, velocidade de conexão e quantidade de dispositivos envolvidos. Gerenciamento em grande escala, dado a quantidade de dispositivos conectados e interagindo, existe um grande conjunto de dados sendo trafegado pela Internet. Segurança (Safety), um fator essencial a ser considerado nos dispositivos e em seu processo de comunicação, principalmente aqueles relacionados ao bem-estar (Patel, Patel et al., 2016).

Existem atualmente 5 tecnologias que são amplamente utilizadas para desenvolvimento e comunicação de produtos e serviços baseados em IoT. Identificação por radio-

frequência (*Radio frequency identification* (RFID)), é uma tecnologia que permite identificação e captura de dados automática através de ondas de rádio, as tags utilizadas no RFID podem ser passivas, quando o leitor provê a energia necessária para transferir o dado, podem ser ativas, quando a tag possui uma bateria integrada e inicia as comunicações com o leitor, ou finalmente, podem ser do tipo semi-passiva, possuindo bateria para o chip interno mas a comunicação é através do leitor. Redes de sensores wireless (*Wireless sensor networks* (WSN)), é uma tecnologia que consiste em dispositivos autônomos distribuídos e equipados com sensores que permitem o monitoramento de condições físicas ou do ambiente, permitindo topologias diferentes de redes, com baixo consumo de energia, baixo custo e comunicação *wireless*. *Middleware*, é uma camada de software entre aplicações com o objetivo de facilitar ao desenvolvedores a realização da comunicação e lidar com entradas e saídas das aplicações, seu foco está em esconder os detalhes de diferentes tecnologias. Computação em nuvem que, como comentado na seção 2.5, consiste de um modelo de acesso sob-demanda a recursos compartilhados de computadores, redes, servidores, armazenamento, entre outros que são necessários para aplicações IoT que necessitam tomar decisões em tempo real. Aplicações IoT, que permitem interações entre dispositivos e humano para dispositivo de uma maneira confiável e robusta, onde se tem a necessidade de garantir que os dados sejam transmitidos e recebidos em um tempo aceitável.(Lee e Lee, 2015).

3 TECNOLOGIAS UTILIZADAS

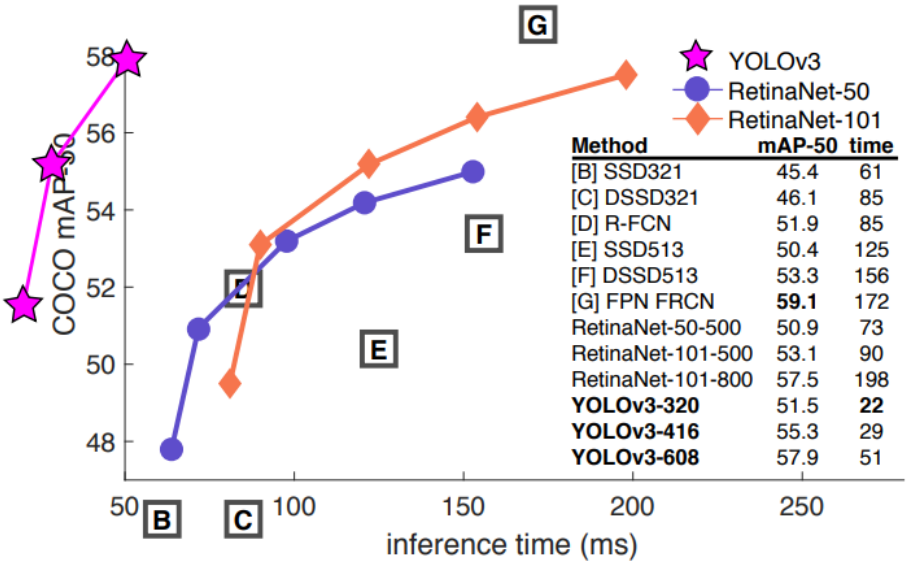
3.1 YOLO

You Only Look Once (YOLO), é um sistema de detecção de objetos direcionado para processamento em tempo real. Ele possui uma arquitetura unificada que é extremamente rápida (Redmon et al., 2015). O sistema possui três versões, YOLO, YOLO9000 e YOLOv3, e vem sendo melhorado incrementalmente ao longo do tempo. Este projeto utiliza a terceira versão do YOLO com um modelo pré-treinado na base de dados COCO *dataset*, sendo capaz de detectar 80 objetos diferentes.

Alguns sistemas de detecção de objetos usam a técnicas como *sliding window* e *region proposal*, onde é preciso rodar diferentes componentes mais de uma vez na imagem ou em partes dela e, adicionar pós processamento para refinar as detecções. Esses pipelines complexos são lentos e difíceis de otimizar, porque cada componente individual deve ser treinado separadamente (Redmon et al., 2015). O YOLO reformulou a detecção de objetos para um único problema de regressão, onde agora só é preciso olhar uma única vez em uma imagem. Simplificadamente, o processo consiste em redimensionar a imagem de entrada, executar uma única CNN e limitar as detecções com base na confiança (probabilidade de a detecção estar correta) de cada detecção.

Se comparado com outros modelos, o YOLO tem uma performance melhor, seu tempo de resposta chega a ser 3 vezes mais rápido que o SSD (*Single Shot Detector*). Uma de suas variantes (YOLOv3-608) possui uma acurácia alta comparável ao estado da arte em acurácia (modelo FPN FRCN) e ainda consegue ter um tempo de resposta baixo (50ms), na Figura 8, é mostrado uma comparação das variantes do YOLOv3 em relação aos outros modelos.

Figura 8: Comparação entre os modelos de detecção de objetos utilizando o dataset COCO como benchmark



Fonte: (Redmon e Farhadi, 2018)

O YOLOv3 utilizada um CNN de 53 camadas (ver Figura 9) para realizar a extração das características das imagens. Esta arquitetura é uma abordagem baseada no YOLOv2 e outras redes. Ela é significantemente maior que a última versão, porém ainda é eficiente.

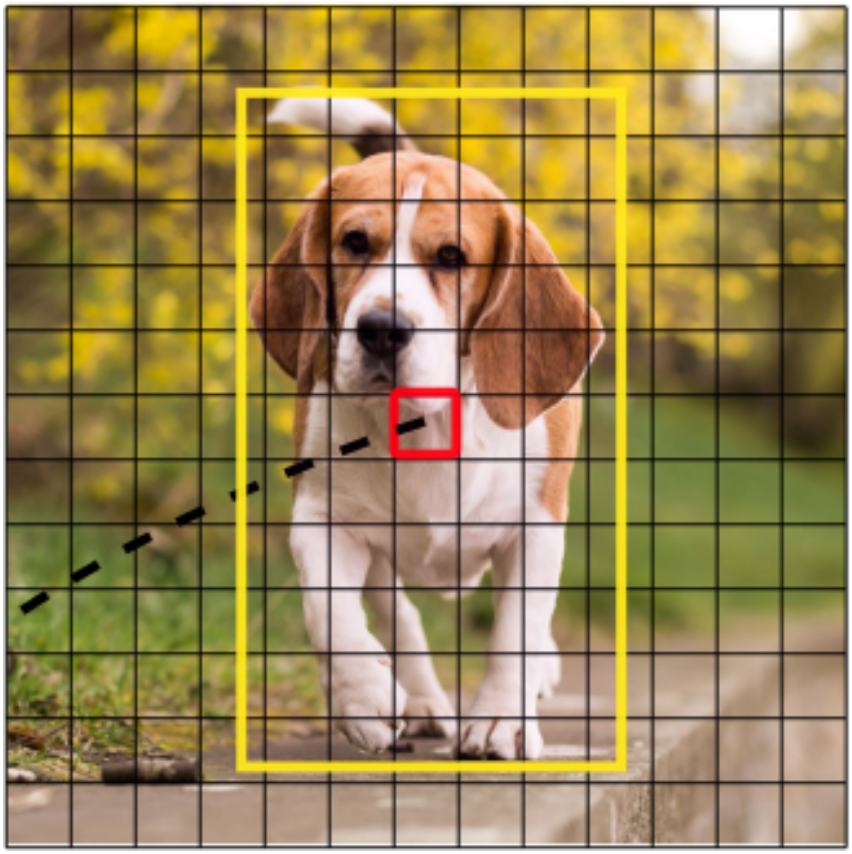
Figura 9: Arquitetura da rede neural do YOLOv3

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fonte: (Redmon e Farhadi, 2018)

A rede neural recebe uma imagem, cujo tamanho é redimensionado para 416×416 pixels. Essa imagem é dividida em uma grade $N \times N$. Cada célula é responsável por prever apenas um objeto (ver Figura 10), e fornece como saída B caixas delimitadoras. Cada caixa delimitadora contém cinco componentes, sendo eles, quatro que representam as coordenadas da caixa e um que representa a probabilidade de haver um objeto cujo centro está célula. Além disso, cada caixa fornece a probabilidade condicional de cada classe C, ou seja, a probabilidade de o objeto pertencer a uma certa classe. Assim, as predições são codificadas como $N \times N \times [B(5 + C)]$.

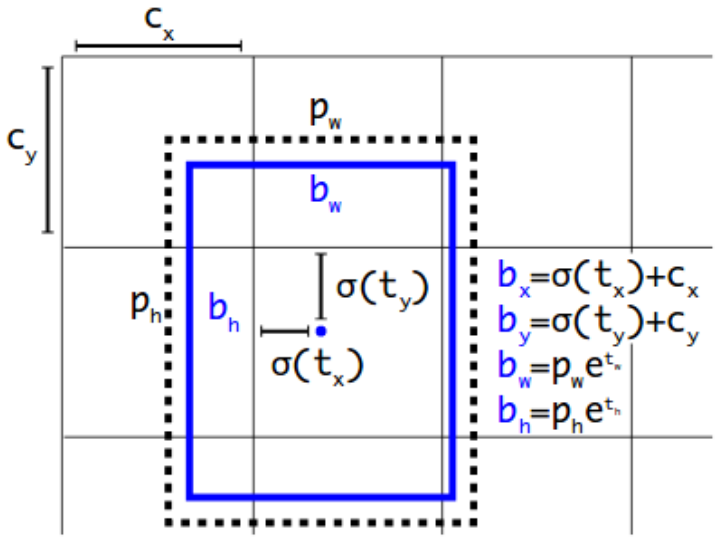
Figura 10: A célula que contém o centro do objeto é responsável por sua detecção



Fonte: (Kathuria, 2018)

Cada célula fornece como saída B caixas delimitadoras, sendo que as fórmulas apresentadas na Figura 11 são utilizadas para transformar a saída da rede em caixas delimitadoras, onde b_x, b_y, b_w e b_h são as coordenadas centrais x y, e largura e altura da caixa, respectivamente, t_x, t_y, $t_w t_h$ são as saídas da rede, c_x e c_y são as coordenadas do topo esquerdo da célula e p_x e p_x são dimensões de âncoras para a caixa.

Figura 11: A célula que contém o centro do objeto é responsável por sua detecção



Fonte: (Redmon et al., 2015)

O YOLOv3 faz predições em 3 escalas diferentes, para deixar a rede menos sensível a escala, sendo elas 13×13, 26×26 e 52×52. Além disso, cada célula faz previsões de três caixas, ou seja, $B = 3$, e a base de dados de treinamento contém 80 objetos, ou seja, $C = 80$. Portanto, a rede faz previsões de $[(13 \times 13) + (26 \times 26) + (52 \times 52)] \times 3 = 10647$ caixas delimitadoras para cada imagem.

Nem todas as 10647 caixas preditas pela rede necessariamente contém um objeto no centro. O primeiro passo para eliminar os resultados irrelevantes é ignorar as caixas cuja probabilidade de haver um objeto sejam menores que um limiar. Para eliminar detecções do mesmo objeto por células adjacentes, usa-se o NMS.

O YOLO impõe grandes restrições espaciais, devido ao fato de cada célula prever um número fixo de objetos. Logo, essa restrição limita a quantidade de objetos próximos na imagem. O treinamento da rede é baseado em uma função de redução de erros, de forma que essa função trata os erros em caixas grandes e pequenas da mesma forma. Um pequeno erro em uma caixa grande geralmente é inofensivo, mas um erro pequeno em uma caixa pequena gera um grande efeito na IoU. A principal fonte de erros são as localizações incorretas (Redmon et al., 2015).

3.2 OpenCV

Open Computer Vision (OpenCV) é uma biblioteca aberta de visão computacional. Ela foi iniciada na Intel, em 1999, e foi lançada em 2000. Ela é implementada na linguagem C, possui interface para linguagens como C++, Python e Java e está disponível para diferentes plataformas, como Windows, Linux, OS X, Android e iOS. Além disso, as interfaces baseadas em CUDA e OpenCL estão em desenvolvimento ativo para operações de GPU de alta velocidade (OpenCV-Python Tutorials, 2018).

Para auxiliar nas tarefas de processamento de imagens, o *Open Computer Vision* (OpenCV) vem com um conjunto de ferramentas para manipular imagens e vídeos, como ler, gravar, exibir na tela, comprimir, entre outros. Além disso, existem algumas funções para capturar posição do mouse, adicionar textos e formas simples (linhas, retângulos, elipses, círculos, etc) aos quadros.

Na área de processamento de imagens, existem alguns agrupamentos de técnicas, onde as principais estão descritas a seguir:

- Processamento de imagem: destacam-se os algoritmos de transformações geométricas, extração de gradientes, detecção de bordas, equalização de histogramas e segmentação de imagens
- Extração de características: destacam-se os algoritmos de detecção de cantos (e suas variações) e casamento de características
- Aprendizado de máquina: destacam-se os algoritmos kNN (*k Nearest Neighbour*), SVM (*Support Vector Machine*) e K-Means *Clustering*

3.3 ZeroTier

O ZeroTier é um hypervisor de rede distribuído construído sobre uma rede global *peer-to-peer* criptograficamente segura. Suas principais características consistem em possuir uma virtualização de rede avançada, gerenciamento de recursos como um *switch Software-Defined Networking* (SDN) e técnicas similares ao *Internet Protocol Security* (IPSec), servindo como redes locais ou globais conectando diversos tipos de dispositivos, como celulares, computadores e equipamentos de rede (ZeroTier, 2018).

Sua arquitetura consiste virtualização das camadas Ethernet, similar ao modelo VXLAN, porém em uma rede totalmente criptografada. Possui descoberta de vizinhança

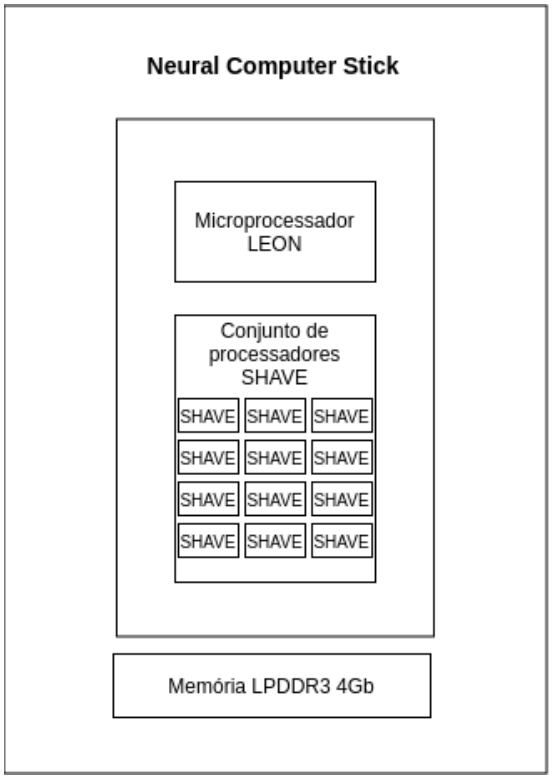
similar a descoberta de rede da Internet com o *Domain Name System* (DNS), utiliza o conceito de planetas (*planet*) e luas (*moons*) para serem os servidores responsáveis por identificar e gerenciar as mudanças de IPs na rede (ZeroTier, 2018).

Assim, é possível criar redes privadas com o ZeroTier conectando diversos dispositivos de uma maneira simples, sem preocupar em como irão se conectar, se estão por trás de um *Network Address Translation* (NAT), como está a qualidade do link entre os nós ou qual caminho realizar a comunicação de uma maneira segura(ZeroTier, 2018).

3.4 Intel Movidius Neural Computer Stick (NCS)

O *Intel Movidius Neural Computer Stick* (NCS) é um dispositivo dedicado para processamento na área de visão computacional. Este dispositivo conta com uma *Vision Processing Unit*, com Intel Movidius Myriad 2, que possui em sua arquitetura (mostrada na Figura 12) uma memória DRAM LPDDR3 de 4Gbits, um microprocessador *Scalable Processor Architecture* (SPARC) LEON e 12 processadores *Streaming Hybrid Architecture Vector Engine* (SHAVE) que é arquitetura híbrida contendo múltiplas unidades funcionais para alto paralelismo e vazão, tendo em seu design um baixo consumo de energia e maximização de performance (Intel, 2018).

Figura 12: Arquitetura da Intel Movidius NCS



Fonte: (Intel, 2018)

Seu funcionamento consiste em após o dispositivo NCS está conectado, o arquivo com o modelo da rede neural é carregado na memória, o processador LEON executa um *firmware* que é responsável por coordenar o recebimento do arquivo com o modelo da rede neural e imagens para inferência, além de cuidar do monitoramento da temperatura. O conjunto de processadores SHAVE recebem os dados processados pelo processador LEON, e realiza as operações de paralelismo, como multiplicação de matrizes. O resultado da rede neural e estatísticas associadas é mandado de volta a máquina que está conectada ao NCS. A comunicação entre o dispositivo NCS e a máquina é feita através da biblioteca NCAPI (Neural Compute API), que cuida da inicialização do dispositivo e carrega o *firmware* necessário para inicializar o recebimento dos arquivos a serem processados (Intel, 2018).

Sua aplicação está principalmente na rápida prototipação, validação e desenvolvimento de redes neurais profundas (DNN) em aplicações instaladas em dispositivos. Com sua VPU de baixo consumo, permite que aplicações de inteligência artificial não dependam de conexões com a nuvem, como detecção de objetos a partir de uma câmera. Suporta *frameworks* de desenvolvimento TensorFlow e Caffe, e sistemas operacionais de 64 bits como o Raspberry Pi 3 ou Ubuntu 16.04.

4 METODOLOGIA DO TRABALHO

4.1 Gerenciamento de tarefas

Para o desenvolvimento do projeto, foi utilizado a metodologia ágil Scrum, com o objetivo de otimizar o processo de pesquisa e desenvolvimento do trabalho, onde através de Sprints foram definido os ciclos de pesquisa, desenvolvimento e testes do sistema.

Para o controle das entregas, foi utilizado a metodologia Kanbam, através da utilização da ferramenta de gerenciamento online Trello.

4.2 Gerenciamento de código

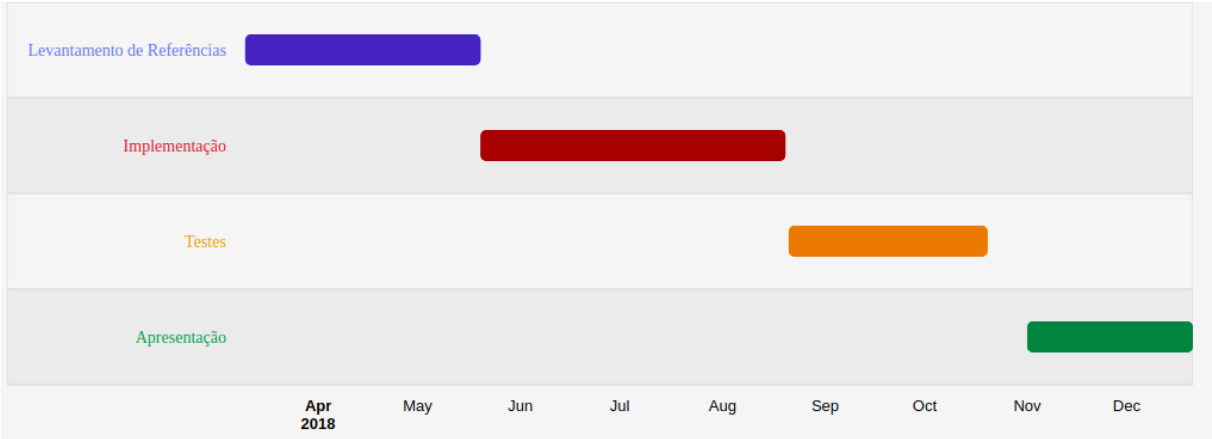
Para o gerenciamento de código, foi utilizado o sistema de versionamento Git, através da ferramenta online GitHub, para hospedar e versionar os códigos utilizados e desenvolvidos pela equipe.

Cada serviço do sistema foi hospedado em um repositório distinto com instruções de instalação e execução.

4.3 Cronograma

Foi realizado um cronograma inicial, considerando o tempo necessário em cada etapa do trabalho, como é mostrado na Figura 13.

Figura 13: Cronograma inicial



Fonte: Autores

4.4 Fases do trabalho

Para o desenvolvimento do trabalho, inicialmente, foi realizado pesquisas em *papers* e estudos sobre o estado da arte em monitoramento de ambientes, sobre algoritmos de reconhecimento de objetos e aspectos conceituais necessário para o desenvolvimento da monografia. A partir disso, foi elaborado uma arquitetura, afim de atender aos objetivos propostos, na qual maquetes para análise de viabilidade foram feitas para verificar as limitações em relação ao tempo de desenvolvimento e testes da solução a ser proposta. Após essas fases, foi realizado a definição de uma solução técnica com o escopo adequado ao trabalho, e implementado a solução. Com a solução implementada, testes foram realizados para medição dos resultados.

4.4.1 Pesquisa

Entre o conjunto de pesquisas realizadas, destacam-se pesquisas de publicações científicas e industriais na área de monitoramento inteligente, onde patentes foram identificadas e produtos que propõem soluções próximas ao nosso objetivo. Foi pesquisado cenários atuais para entender como as câmeras de vigilância estão sendo utilizadas atualmente e quais informações estão sendo extraídas.

Entre os aspectos conceituais importantes para o desenvolvimento da arquitetura, foi pesquisado sobre técnicas de processamento de imagem, *machine learning*, mais especificamente, redes neurais. Para a arquitetura, foi pesquisado e definido o uso do modelo RM-ODP, assim como a abordagem de microsserviços para o desenvolvimento dos mó-

dulos do sistema. Entre as tecnologias utilizadas, foi pesquisado sobre computação em nuvem para garantia dos requisitos de escalabilidade do sistema e IoT como forma de coleta da informação necessária do ambiente de atuação.

Para o algoritmo de detecção de objetos, foram pesquisados técnicas para criar um modelo adequado de detecção de objetos, onde através de ferramentas de *machine learning* como TensorFlow, ser desenvolvido um modelo e treiná-lo para o reconhecimento de objetos, principalmente pessoas em diferentes cenários, atendendo ao requisito de ser um sistema em tempo real. Através de pesquisas, foi identificado a inviabilidade de treinar um modelo do zero no qual atendesse aos requisitos de precisão aceitável para o sistema e que respondesse em um tempo adequado. Nessas pesquisas foi encontrado e selecionado o modelo YOLO, que cumpre com os requisitos de precisão e desempenho especificados pelo sistema.

4.4.2 Elaboração da arquitetura

A primeira etapa para elaboração da arquitetura foi definir os requisitos preliminares, a partir da estruturação do que o sistema precisaria fazer. Com isto, foi definido os principais módulos do sistema, como o módulo de coleta dos vídeos, processamento, detecção de eventos e módulo de ações, além dos protocolos de comunicação entre os módulos. Para isso, foi utilizado o modelo RM-ODP, na qual foi descrito cada visão do sistema.

4.4.3 Elaboração de maquetes para análise de viabilidade

Para análise de viabilidade da solução, foram realizados testes de detecção de objetos considerando cenários diversos para a posição da pessoa. Cada cenário consistiu de pessoas em diversos ângulos com o objetivo de verificar a detecção da pessoa. Os testes confirmaram a viabilidade em utilizar o modelo YOLO como base para detecção de objetos e a partir dele desenvolver o algoritmo de detecção de desmaios para as pessoas detectadas nos vídeos recebidos.

4.4.4 Definição e implementação da solução técnica

Com a arquitetura elaborada e a análise de viabilidade feita, foi definido um escopo adequado para o funcionamento do sistema, com os requisitos mínimos necessários para provar a viabilidade da ideia proposta. Para cada módulo proposto, foi implementado utilizando as tecnologias e ferramentas pesquisadas, tais como a linguagem Python, o

framework Django e a biblioteca OpenCV. Entre os principais módulos, o módulo de detecção de objetos foi implementado utilizando a arquitetura de detecção de objetos YOLO, o algoritmo detector de desmaio foi elaborado considerando os cenários possíveis para o evento acontecer. Um conjunto de parâmetros para o algoritmo foram elaboradas para serem configurados durante os testes com o objetivo de eliminar falsos positivos e falsos negativos.

4.4.5 Medição dos resultados obtidos

O processo de medição de resultados foi dividido em duas partes: coleta de dados e obtenção da taxa de acertos. O capítulo de implementação define o escopo do sistema que foi implementado para este trabalho, onde o evento a ser detectado é o de desmaio.

Na fase de coleta dados, foi escolhido um ambiente bem iluminado e com um piso adequado para simulação de desmaios. Foram coletados diversos vídeos, com duração média de 30 segundos, usando câmeras de qualidades distintas, contendo desmaios em diversos ângulos, com e sem pessoas transitando.

Na fase de obtenção da taxa de acertos, uma parte dos dados foi utilizada para estimar os principais parâmetros do algoritmo, enquanto que a outra, para obtenção da taxa de acertos, relacionando o número de verdadeiros positivos com o de falsos positivos.

5 ESPECIFICAÇÃO DO SISTEMA

Para especificar o projeto, foi utilizado o modelo *Reference Model of Open Distributed Processing* (RM-ODP), que é estruturado em 5 visões: negócio, informação, computação, engenharia e tecnologia. As seções a seguir contém a descrição de cada uma das camadas.

5.1 Visão empresa

O sistema de Monitoramento Inteligente de Ambientes é um serviço que monitora ambientes, através de vídeos de câmeras de vigilância, detecta eventos relacionados ao bem estar das pessoas presentes no ambiente e presta assistência, quando necessário, realizando um conjunto de ações, em tempo real. Em outras palavras, o sistema recebe vídeos, detecta eventos e realiza ações, visando dar suporte imediato as pessoas contidas nos eventos.

Há uma grande abrangência para os tipos de ambientes, eventos e ações, no contexto do sistema. Os ambientes podem ser do tipo público ou privado, onde os vídeos podem ser provenientes de câmeras de trânsito, monitoramento interno e externo de residências, indústrias, comércios, escolas, hospitais, entre outros. Existem diversos eventos relacionados ao bem estar de pessoas, onde sua detecção pelo sistema se limita pela existência de um algoritmo ou método que seja capaz de detectar tal evento. Alguns exemplos de eventos são: acidente de trânsito, assalto, incêndio, desmaio, agressão física, entre outros. Cada evento necessita de um certo conjunto de ações, que podem ser acionar a polícia, acionar o hospital, acionar corpo de bombeiros, contactar responsável via SMS, chamada de voz ou e-mail, modificar o estado de semáforos de trânsito para isolar áreas, acionar possíveis dispositivos inteligentes para atuar em campo, entre outros.

O sistema conta com três tipos de participantes: clientes, colaboradores e usuários, que podem ser descritos como:

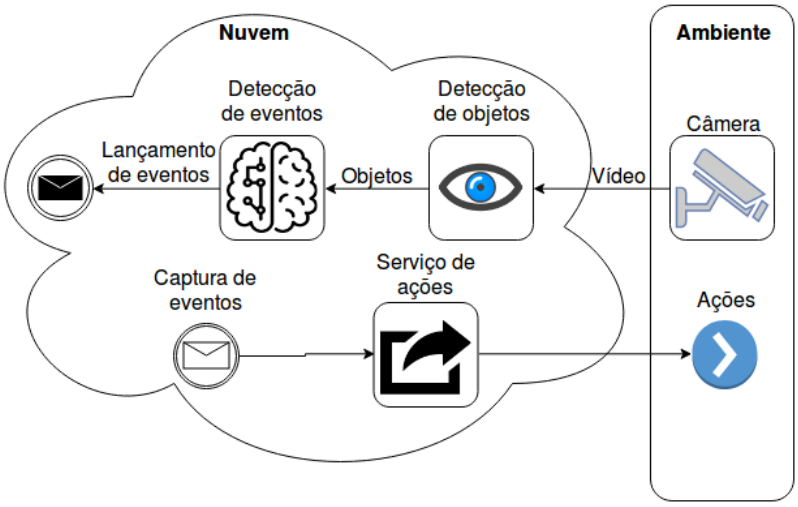
- Clientes:

  - Poder público: pode utilizar o sistema com a finalidade de aumentar a eficiência na prestação de serviços à população, utilizando seu grande número de câmeras para cobrir áreas públicas;
  - Particulares: podem utilizar o sistema com vídeos internos e externos, visando aumentar a sensação de segurança de seus clientes;

- Colaboradores: são os participantes que contribuem submetendo seus vídeos; eles podem ser atraídos através do oferecimento de bonificações, como prêmios, descontos em lojas, redução no valor de impostos, entre muitos outros;
- Usuários: são todos aqueles que estão presentes nos ambientes monitorados pelo sistema.

Para que o objetivo do sistema seja alcançado, há um conjunto básico de tarefas que o sistema precisa realizar. Primeiramente, é preciso submeter os vídeos ao sistema, para que seja possível detectar os objetos presentes nas imagens, através de técnicas de visão computacional. Os objetos vão para a fase de detecção de eventos, onde um conjunto de algoritmos analisam os objetos, para detectar os eventos. Os eventos detectados são capturados, onde realiza-se um levantamento do conjunto de ações que é necessário performar para cada evento. Uma vez que o conjunto de ações está definido, realiza-se todas elas. Para realizar essas tarefas, são necessários os componentes que estão representados esquematicamente pela Figura 14.

Figura 14: Componentes básicos do sistema

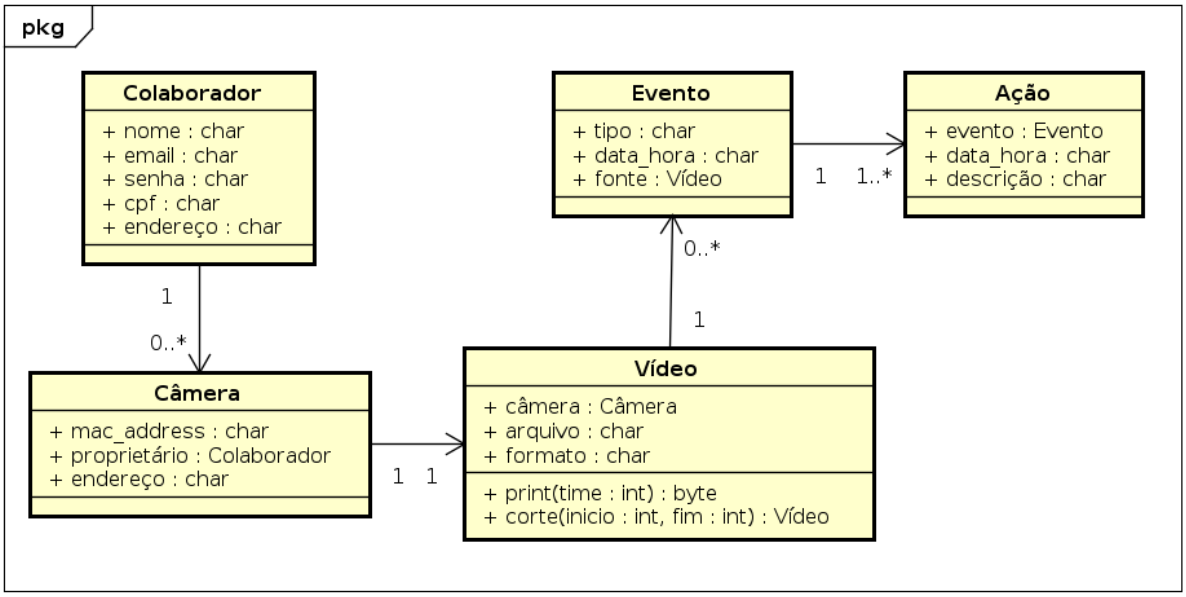


Fonte: Autores

5.2 Visão informação

O sistema é composto por cinco objetos de dados: Colaborador, Câmera, Vídeo, Evento e Ação. Os colaboradores se cadastram no sistema, para poderem cadastrar suas câmeras. As câmeras transmitem seus vídeos, que são comprimidos e armazenados. Os eventos detectados são registrados, assim como o conjunto de ações tomados.

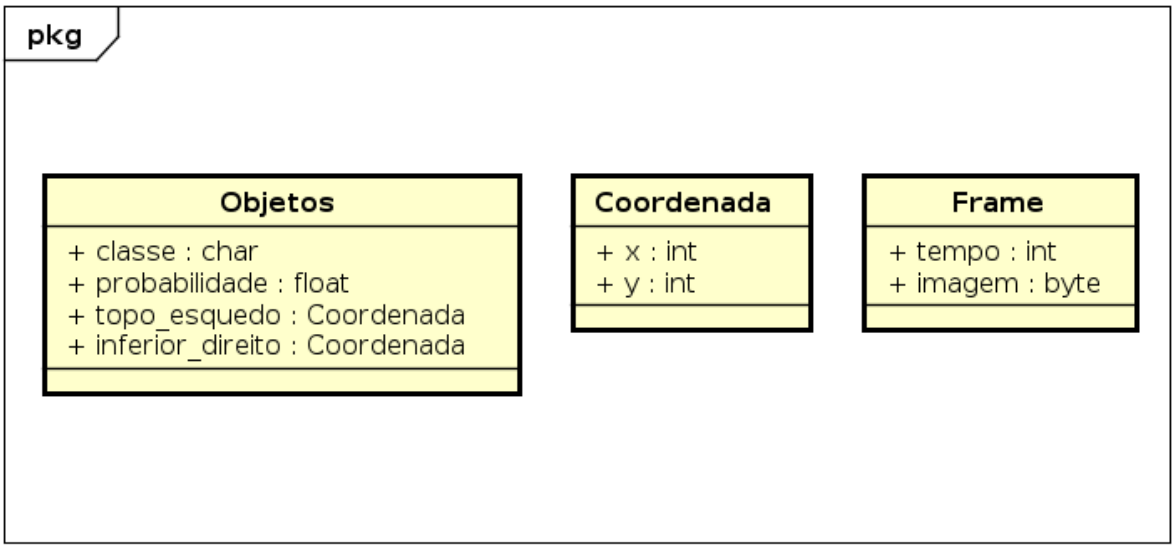
Figura 15: Diagrama de classes



Fonte: Autores

Existe ainda um pequeno conjunto de dados dinâmicos, que trafegam entre os componentes do sistema, e não são armazenados. A classe Objetos representa os objetos detectados pelo componente de detecção de objetos, que é a entrada do componente de detecção de eventos. A classe Frame representa os quadros do vídeo que são submetidos ao sistema, onde cada quadro deve ter, no máximo, 65 KB, por motivo de desempenho.

Figura 16: Dados dinâmicos

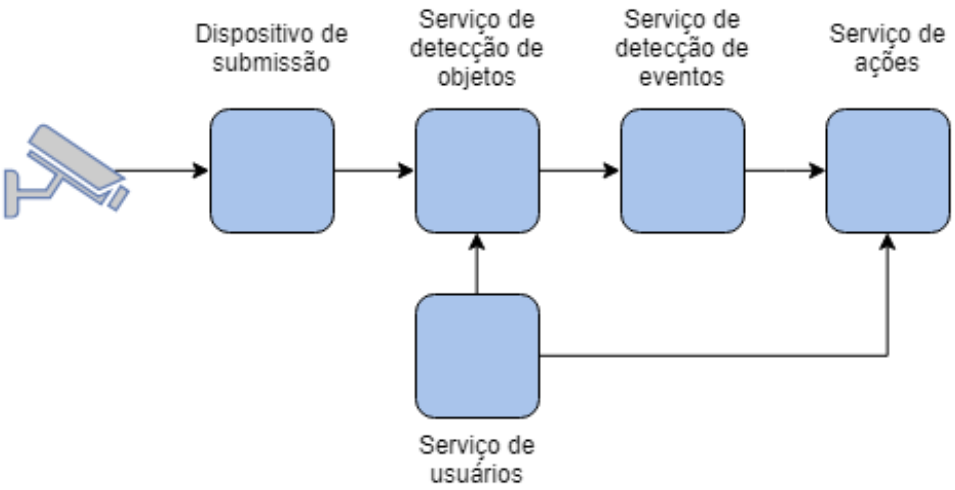


Fonte: Autores

5.3 Visão computação

Para cumprir com o objetivo proposto, o sistema deve possuir os módulos presentes na 17. Um dispositivo de submissão de vídeos deve instalado à câmera do colaborador, responsabilizando-se por requisitar permissão e submeter os vídeos de forma privada, confidencial e em tempo real. O módulo de detecção de objetos recebe a requisição de permissão do dispositivo, verifica se o mesmo está cadastrado, alerta o módulo de detecção de eventos, fornece a permissão ao dispositivo e passa a processar, em tempo real, o fluxo de quadros recebidos, detectando os objetos de cada quadro. O módulo de detecção de eventos recebe um alerta do módulo de detecção de objetos, instância os algoritmos disponíveis e passa a processar a lista de objetos contidos em cada frame. O módulo de ações é notificado sempre que um evento é detectado, coletando dados da câmera que gerou o evento e realizando as ações necessárias.

Figura 17: Visão computação

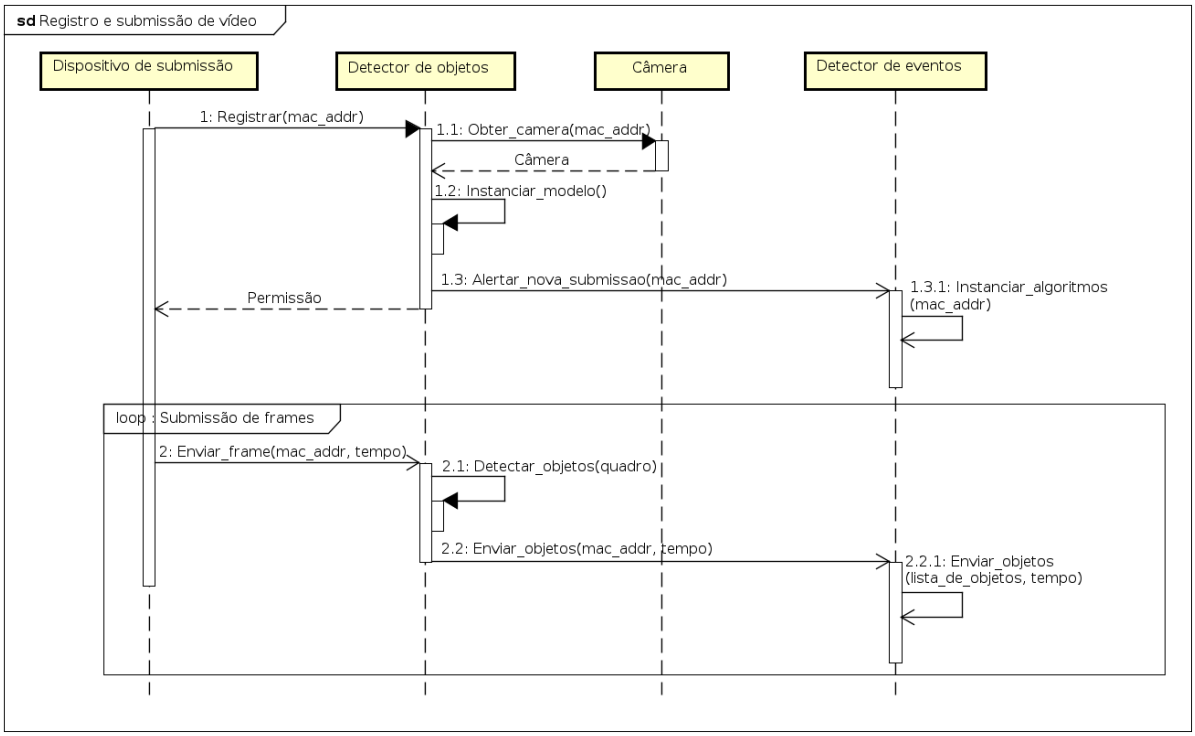


Fonte: Autores

Qualquer tipo de participante que queira participar do sistema, precisa se cadastrar. Se o participante for do tipo colaborador, ele poderá cadastrar as câmeras que deseja adicionar.

Uma vez que um dispositivo de submissão é instalado na câmera do colaborador, ele se registra no detector de objetos, informando seu identificador. O detector de objetos verifica a existência desse dispositivo nos registros, e em caso positivo, cria uma instância do modelo que fará as detecções e, emite um alerta ao detector de eventos. Sempre que o detector recebe um alerta, instância os algoritmos disponíveis, para que eles processem o fluxo de objetos que está para chegar. Quando o dispositivo de submissão recebe a permissão, inicia-se o seguinte laço: o dispositivo envia um frame, o detector de objetos detecta os objetos, o detector de eventos processa a lista de objetos.

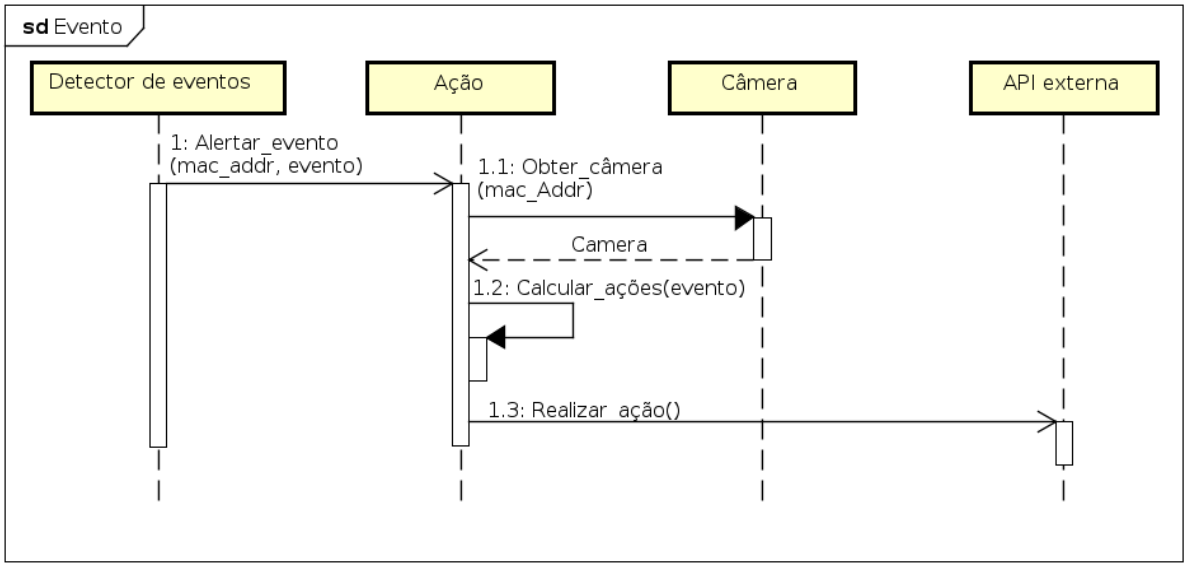
Figura 18: Diagrama de sequência do processo de submissão de vídeo



Fonte: Autores

Quando um evento é detectado, o detector de evento alerta ao módulo de ações, informando o identificador do vídeo gerador e o tipo de evento, que, por sua vez, obtém os dados da câmera, calcula quais ações devem ser tomadas, e realiza todas as ações calculadas.

Figura 19: Diagrama de sequência do processo de detecção de evento



Fonte: Autores

5.4 Visão engenharia

Esta seção contém a solução arquitetural do sistema, para atender aos requisitos levantados nas visões informação e computação. A visão informação definiu os módulos do sistema, que serão modelados usando a arquitetura de micro serviços. Os micro serviços do sistema são: submissão de vídeo, detecção de objetos, detecção de eventos, ações e usuários.

Para atender aos requisitos de privacidade e confidencialidade, toda a comunicação entre os serviços de sistema são feitos através de VPNs, e estão representados explicitamente nos diagramas.

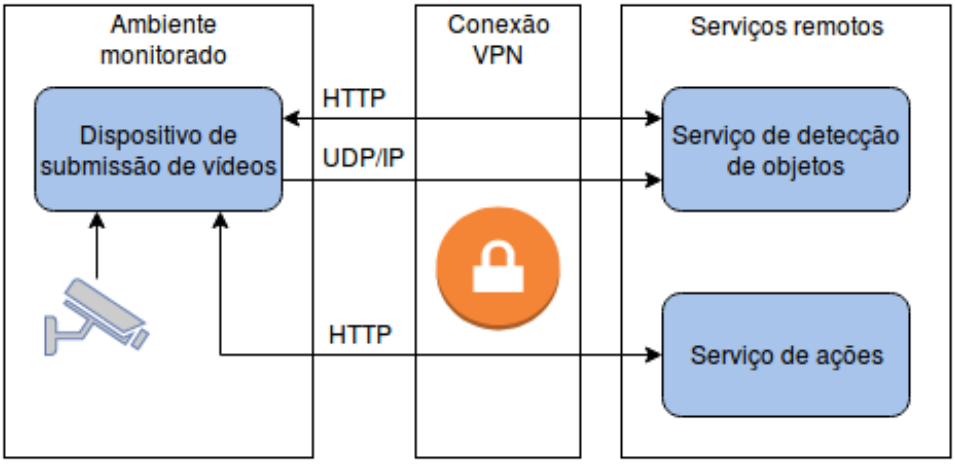
5.4.1 Submissão de vídeo

Dispositivos de submissão de vídeo são instalados às novas câmeras cadastradas pelos colaboradores. Ao serem ligados, solicitam permissão para submeter seus vídeos ao serviço de detecção de objetos, via protocolo HTTP. Quando recebem permissão, submetem seus vídeos em tempo real, usando o protocolo *User Datagram Protocol*/IP.

Para aumentar a disponibilidade, os dispositivos de submissão de vídeo podem detectar falhas no serviço de detecção de objetos, e passam a realizar o processamento

localmente, detectando objetos e rodando os algoritmos mais críticos em relação ao bem estar dos usuários. Quando um evento é detectado, ele alerta o serviço de ações por conta própria, via protocolo HTTP.

Figura 20: Arquitetura dos serviços envolvidos na submissão de vídeos



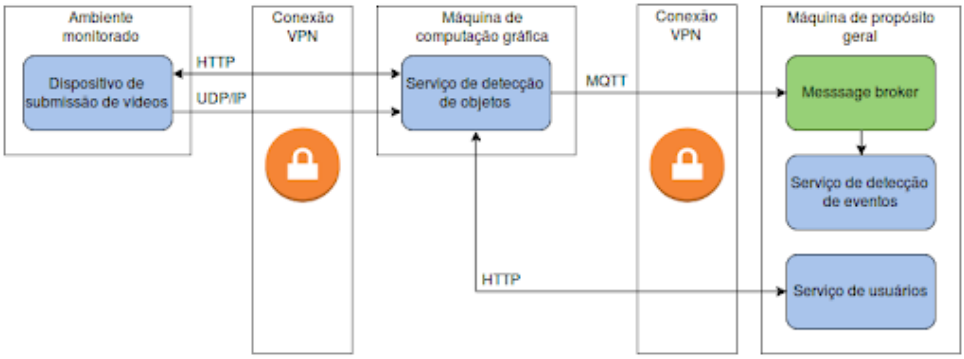
Fonte: Autores

5.4.2 Detecção de objetos

Sempre que um dispositivo de submissão de vídeo é iniciado ele solicita permissão para submissão. Para dar a permissão, o serviço de detecção de objetos se comunica, via HTTP, com o serviço de usuários, para verificar se o dispositivo solicitante está cadastrado.

O serviço de detecção de objetos fica constantemente processando todos os quadros recebidos de todas as câmeras, e envia os objetos detectados, no formato JSON, para o serviço de detecção de eventos. Essa troca de dados é assíncrona e utiliza o protocolo MQTT.

Figura 21: Arquitetura dos serviços envolvidos na detecção de objetos

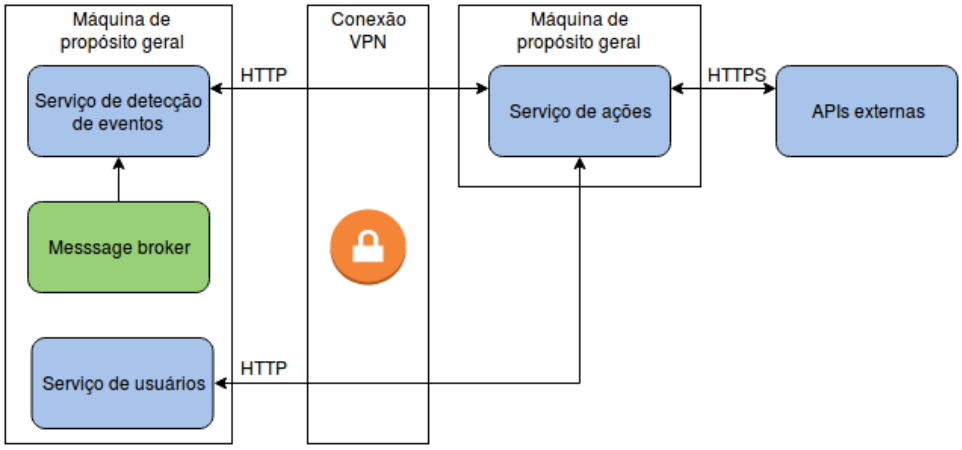


Fonte: Autores

5.4.3 Detecção de eventos e tomadas de ações

Quando o serviço de detecção de eventos detecta um evento, ele se comunica, via HTTP, com o serviço de ações, informando o tipo de evento e o identificador da câmera que originou o mesmo. Com essas informações, o serviço de ações coleta dados adicionais, como nome do dono na câmera e endereço, do serviço de usuários, via HTTP. Quando todas as ações tiverem sido calculadas, atua realizando requisições a APIs externas, via HTTPS.

Figura 22: Arquitetura dos serviços envolvidos na detecção de eventos e tomadas de ações



Fonte: Autores

5.5 Visão Tecnologia

5.5.1 Comunicação

Os protocolos de comunicação utilizados são:

- MQTT, onde utiliza-se a biblioteca Python Paho MQTT para implementar a comunicação e, como mensageiro central, o Mosquitto;
- *User Datagram Protocol*/IP, onde utiliza-se a biblioteca Python Socket, e;
- HTTP, onde utiliza-se a própria infraestrutura dos serviços, que são implementados com Django.

Para garantir uma comunicação segura, utiliza-se a ferramenta ZeroTier, que cria uma VPN entre as máquinas físicas e oferece uma boa interface de gerenciamento de rede.

5.5.2 Serviço de submissão de vídeo

Para a submissão dos vídeos, o hardware utilizado é um computador single-board, no caso, Raspberry Pi, que é conectado a uma câmera. As principais bibliotecas utilizadas são o OpenCV, para ter acesso à câmera do dispositivo e realizar processamento simples às imagens, como comprimir os quadros e o Paho MQTT para comunicação. Em um dos possíveis cenários, alternativo a mandar o vídeo para nuvem, utiliza para processamento local um Intel Movidius Neural Compute Stick, que se trata de um dispositivo de processamento de visão computacional, executando uma versão Yolo Tiny para o *framework* Caffe.

5.5.3 Serviço de detecção de objetos

O serviço de detecção de objetos utiliza a *web-framework* Django, para implementar sua interface. Para se comunicar com o serviço de usuários, utiliza-se a biblioteca Python Requests. A recepção de vídeos conta com o uso de Sockets e OpenCV, para descomprimir os quadros recebidos.

O detector de objetos implementa, em Python, a arquitetura de rede neural YOLO, utilizando a biblioteca Pytorch. Este serviço é hospedado nos serviços EC2 da AWS, na família de instâncias "*g3*", que possui a GPU Tesla M60 e o S.O. Ubuntu Server 16.04 LTS.

5.5.4 Serviço de detecção de eventos

O serviço de detecção de eventos utiliza a *web-framework* Django. A recepção dos objetos detectados nas imagens é feito através da biblioteca Paho MQTT.

5.5.5 Serviço de ações

O serviço de ações utiliza a *web-framework* Django para implementar sua interface. Os registros de eventos tomados são armazenados em banco de dados relacional, utilizando o MySQL. A integração com outras *APIs* é realizada com a biblioteca Requests.

5.5.6 Serviço de usuários

O serviço de usuários utiliza Django como *web-framework*, SQLite como banco de dados em ambiente de testes e MySQL no ambiente de produção. Nele é armazenado todas as informações necessárias relacionadas aos usuários e dispositivos cadastrados. O serviço é hospedado no serviço EC2 da AWS.

5.5.7 Ferramentas

Para desenvolvimento do código, utiliza a IDE, para python, PyCharm. Para dar suporte a criação de ambientes Python isolados, de fácil gerenciamento de dependências e bibliotecas, se utiliza a ferramenta Virtualenv. Para comunicação entre as máquinas na AWS, o Raspberry Pi e máquinas pessoais é utilizado o ZeroTier.

5.6 Requisitos não funcionais

A partir da análise das visões informação e computação, foi possível extrair os requisitos não funcionais do sistema, onde optou-se por descrevê-los através de tabelas, apresentadas a seguir.

5.6.1 Qualidade em uso

Tabela 1: Requisitos de qualidade em uso

Característica	Sub-característica	Grau	Métrica
Eficácia	Eficiência	Taxa de detecção de eventos de 95% e taxa de acerto na execução de ações de 99%	Levantamento de número de falsos e negativos; número de ações não executadas ou incorretas

continua na próxima página

Tabela 1: Requisitos de qualidade em uso (continuação)

Característica	Sub-característica	Grau	Métrica
Eficiência	Eficiência	99% dos quadros de vídeo devem ser de, no máximo, 65KB. 95% dos eventos críticos ocorridos devem ser detectados, em, no máximo, 120 segundos	Cálculo do tamanho dos quadros; quantidade de eventos que ultrapassaram o tempo limite
Satisfação	Confiança	90% dos colaboradores devem concordar com os termos de privacidade	Número de usuários que concordam com os termos de privacidade e aceitam enviar seus vídeos
Isenção de riscos	Mitigação de risco à saúde e segurança	Taxa de falsos positivos menor que 3%	Levantamento do número de falsos positivos através do relatório gerado pelas autoridades acionadas
Cobertura de contexto	Completeza de contexto	O sistema deve detectar eventos em ambientes iluminados (acima de 300 lux), sem chuva, em ambiente rural ou urbano, com resolução tal que a imagem tenha no máximo 65 KB, sem ruídos na imagem, de acordo com o requisito de correção funcional	Ambientes iluminado ou não, chuvoso ou não, rural ou urbano, alta qualidade de imagem ou baixa, imagem ruidosa ou não, poucas pessoas na imagem ou muitas devem respeitar o requisito de correção funcional

Fonte: Autores

5.6.2 Qualidade do produto

Tabela 2: Requisitos de qualidade do produto

Característica	Sub-característica	Grau	Métrica
Adequação funcional	Correção funcional	5% a 8% de falsos positivos e 1% a 3% de falsos negativos	Número de falsos positivos e negativos em relação aos eventos detectados
Eficiência de desempenho	Comportamento em relação ao tempo	O tempo de detecção de objetos em um quadro deve ser menor que 83,3 ms (12 FPS no mínimo)	Levantamento do número de vezes que a detecção ultrapassou o limite de tempo
	Capacidade	O sistema deve suportar o cadastro e submissão simultânea de no mínimo 1,5 milhões de câmeras	Quantidade de câmeras cadastradas no banco de dados e número de vídeos sendo processados
Compatibilidade	Coexistência	Local de instalação deve disponibilizar, no mínimo, 780 Kbps de rede (contando com o uso de outros dispositivos); possuir, no máximo, 5% de obstrução da câmera e o ambiente não deve possuir iluminação maior que 300 lux na direção da câmera	Número de câmeras submetendo vídeo a menos de 12 FPS; avaliação na hora da instalação das câmeras
Confiabilidade	Disponibilidade	O sistema deve responder a, no mínimo, 95% dos eventos	Número de vezes que o sistema respondeu, de acordo com a especificação, a um evento

continua na próxima página

Tabela 2: Requisitos de qualidade do produto (continuação)

Característica	Sub-característica	Grau	Métrica
Segurança de acesso	Confidencialidade	0% dos vídeos privados podem ser acessados por pessoa não autorizada	Número de vídeos visualizados por pessoa não autorizada
	Autenticidade	99% dos vídeos submetidos devem ser feitos por câmeras autorizadas	Quantidade de vídeos aceitos pelo sistema provenientes de câmeras não autorizadas

Fonte: Autores

6 PROJETO E IMPLEMENTAÇÃO

Uma vez concebida uma ideia de projeto, produto ou serviço, é preciso desenvolver uma arquitetura, para definir as funcionalidades, requisitos e organização de seus componentes. Na fase de implementação, uma etapa importante é definir o escopo, para explicitar quais funcionalidades devem ser implementadas e, quais requisitos devem ser atendidos em cada versão.

6.1 Escopo

O capítulo 5 define a arquitetura do sistema de forma abrangente, ou seja, como um serviço completo. Este trabalho tem como objetivo implementar as funcionalidades e, atender aos requisitos mínimos necessários para o funcionamento do sistema, para que seja possível provar a viabilidade da ideia proposta.

Os ambientes se restringem aos públicos, como ruas, praças, estações ferroviárias, entre outros. O principal fator para incluir apenas este tipo de ambiente ao escopo está no fato de que não será implementado mecanismos para garantir a confidencialidade, que é um requisito importante para ambientes privados.

Apenas o evento de desmaio será detectado pelo sistema. Este evento está fortemente ligado ao bem estar das pessoas presentes no ambiente, pois, dependendo do motivo que levou ao desmaio, o rápido atendimento pode ser crucial na sobrevivência da vítima.

A ação realizada para o evento de desmaio é o envio de e-mail para um entidade responsável, contendo informações relevantes sobre o evento.

Os participante se restringem apenas a colaborador e usuário. Os colaboradores podem se cadastrar e cadastrar suas câmeras. Não haverá interface para clientes, existindo apenas uma interface de monitoramento do sistema para administradores.

A seção 5.6 define todos os requisitos não funcionais do projeto. Com base nesses requisitos não funcionais e com o escopo, a implementação atende todos os itens da

Tabela 3.

Tabela 3: Tabela de requisitos não funcionais atendidos pela implementação do projeto

Característica	Sub-característica	Grau
Eficiência	Eficiência	99% dos quadros de vídeo devem ser de, no máximo, 65 KB. 95% dos eventos críticos ocorridos devem ser detectados em, no máximo, 120 segundos
Eficiência de desempenho	Comportamento em relação ao tempo	O tempo de detecção de objetos em um quadro deve ser menor que 83,3 ms (12 FPS no mínimo)

Fonte: Autores

6.2 Módulos implementados

6.2.1 Serviço de detecção de objetos

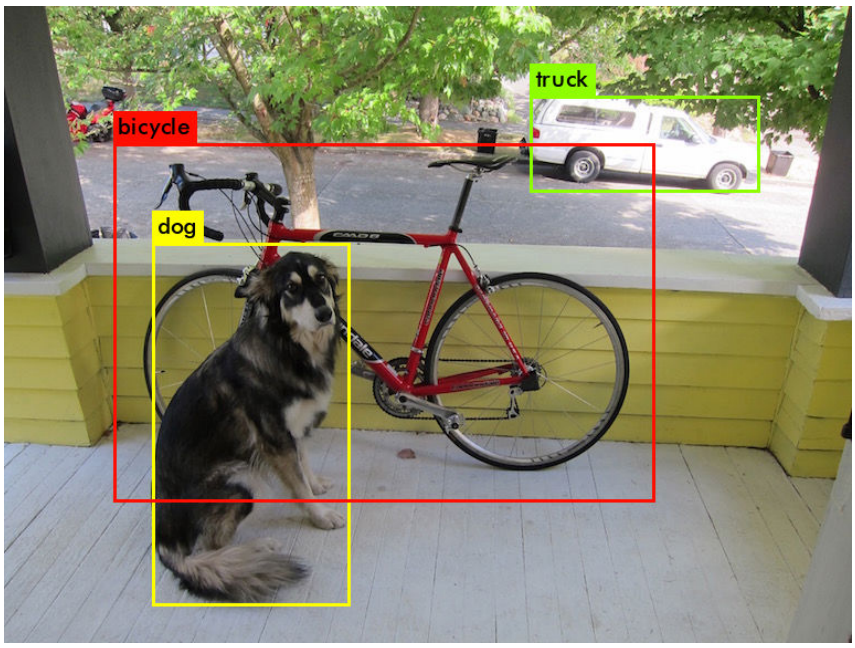
O serviço de detecção de objetos é a base todo o projeto, de forma que seu correto funcionamento é extremamente importante para os serviços de detecção de objetos e o serviço de ações. Devido sua importância, a detecção de objetos foi o primeiro módulo a ser implementado.

6.2.1.1 Módulo de detecção de objetos

Para implementar o módulo de detecção de objetos, foi utilizada a arquitetura de detecção de objetos YOLO, descrita na seção 3.1. A implementação oficial da arquitetura foi feita na linguagem C e CUDA, que resultou no *framework* de código aberto Darknet. A execução do *framework* dá uma amostra de seu funcionamento, como é possível ver na

Figura 23.

Figura 23: Visualização gráfica da execução do *framework* Darknet



Fonte: (YOLO, 2016)

A implementação YOLO utilizada foi a descrita por (Kathuria, 2018), que apresenta e detalha a arquitetura da rede neural do YOLO e, como implementá-la utilizando a ferramenta Pytorch, que é uma biblioteca Python para computação científica, muito usada na computação de matrizes e, portanto, possui integração com a biblioteca cuDNN (CUDA *Deep Neural Network*), que é responsável pelo interfaceamento com a unidade de processamento gráfico (GPU).

Para utilizar essa implementação no projeto, foi preciso criar um interface, onde fosse possível passar imagens como parâmetro e receber a lista de todos os objetos detectados, no formato JSON, para que os algoritmos de detecção de eventos possam consumir esses dados. Com essa interface, a execução do módulo fornece uma saída como a apresentada na Figura 24.

Figura 24: Exemplo de saída da interface do módulo de detecção de objetos


```
[
    {
        "score": 0.999735414981842, "label": "bicycle",
        "x": 164, "y": 108,
        "height": 339, "width": 396
    },
    {
        "score": 0.942956268787384, "label": "truck",
        "x": 473, "y": 85,
        "height": 85, "width": 216
    },
    {
        "score": 0.9884791970252991, "label": "dog",
        "x": 128, "y": 223,
        "height": 314, "width": 186
    }
]
```


Fonte: Autores

Devido a variação de iluminação dos ambientes, o módulo aplica a técnica de equalização de histogramas, descrito na seção 2.1.1, utilizando um método presente na biblioteca do OpenCV, que, aumenta o contraste da imagem. Como a detecção de objetos utiliza, em grande parte, informações sobre o contorno dos elementos na imagem, o aumento do contraste ajuda explicitar os contornos, resultando em um aprimoramento na detecção de objetos.

6.2.1.2 API

Para que os outros serviços pudessem utilizar o serviço de detecção de objetos, foi implementado uma API REST, utilizando o *web-framework* Django, conforme os itens a seguir:

- *POST*: *register*: método utilizado dispositivos que querem iniciar a submissão de vídeos, onde elas informam seu identificador. O serviço verifica se o dispositivo possui permissão e inicia duas *threads*, uma para receber os quadros e outra para consumir os quadros, realizando a detecção de objetos;
- *POST*: *monitor*: método utilizado pela interface do administrador, quando se deseja visualizar o vídeo que algum dispositivo está submetendo;
- *POST*: *unregister*: método utilizado para sinalizar que um dispositivo não irá mais submeter seu vídeo;

- *GET*: *event_print*: método utilizado para se obter uma captura do vídeo de um dispositivo específico;
- *GET*: *status*: método utilizado para obter o status do sistema, com informações de quantos vídeos estão sendo processados, a taxa de detecção e taxa de recepção de quadros.

6.2.2 Serviço de detecção de eventos

O serviço de detecção de eventos é responsável por receber objetos detectados em diversas fontes de vídeos e, detectar os eventos capturados por eles. No escopo do projeto, o evento a ser detectado é o desmaio.

6.2.2.1 Comunicação

Para receber os objetos detectados nas imagens, o serviço utiliza o protocolo MQTT, bastando inscrever-se em um tópico para receber os dados. Toda vez que um dispositivo inicia a submissão de vídeo é criada uma instância do algoritmo de detecção de eventos, para processar especificamente essa fonte de vídeo.

Os algoritmos consomem os objetos detectados em cada frame, e quando detectam um evento, informam o tipo de evento. O serviço obtém a identificação da câmera através da instância que detectou o evento, já que cada instância é atribuída a um fluxo de quadros proveniente de um dispositivo específico. O serviço de ações é alertado via protocolo HTTP, com o auxílio da biblioteca Python, Requests.

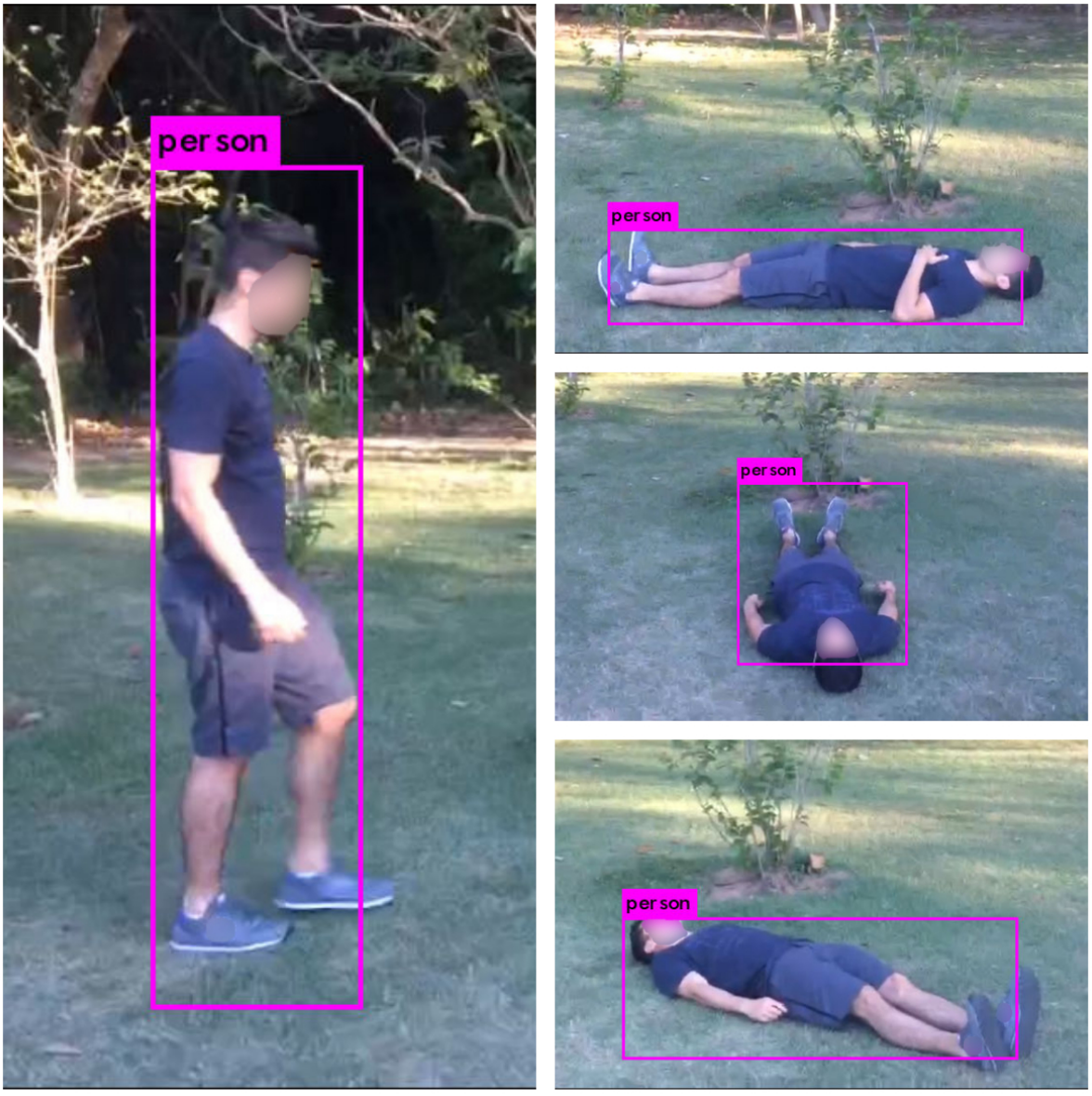
6.2.2.2 Algoritmo detector de desmaios

O algoritmo detector de desmaios recebe uma lista com todos os objetos detectados em um quadro, no formato de texto, como exemplificado na Figura 24, e, a partir disso, determina se uma pessoa está desmaiada.

Para este algoritmo, foram classificados três tipos de estados possíveis para uma pessoa: normal, desmaio horizontal e desmaio vertical. Há ainda uma quarta situação, que é o desmaio diagonal, que consiste na combinação dos desmaios horizontal e vertical (ver Figura 25). Observando graficamente algumas detecções nos três estados, foi possível identificar alguns padrões e definir as seguintes hipóteses:

- Estado normal: a altura da caixa é sempre maior que a largura, ou seja, $height > width$;
- Estado desmaio horizontal: a altura da caixa é sempre menor que a largura, ou seja, $height < width$;
- Estado desmaio vertical: a largura da caixa varia pouco em relação a largura do estado normal e a altura diminui um certo percentual em relação a altura do estado normal.

Figura 25: Categoria de estados: normal, desmaio horizontal, desmaio vertical e desmaio diagonal, respectivamente

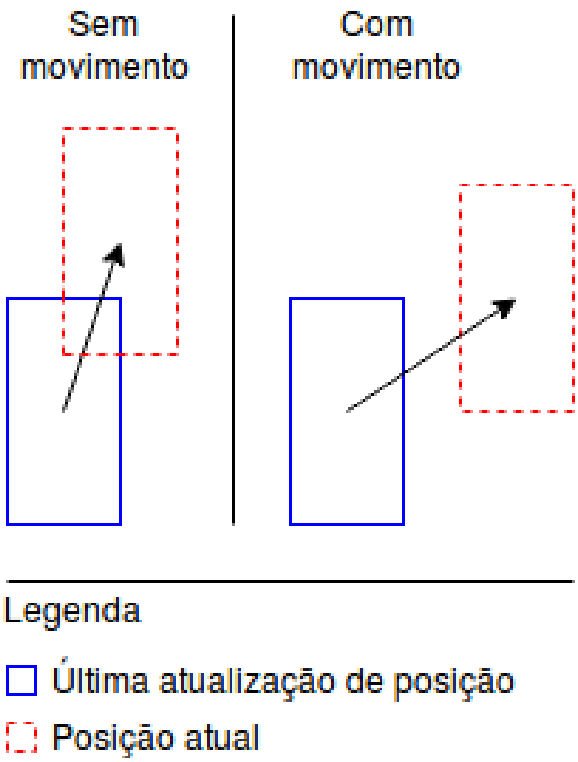


Fonte: Autores

Dado que $width$ é a largura e $height$ é a altura da caixa de detecção, então o estado de desmaio horizontal é atingido quando $\frac{height}{width} < \alpha$, onde α é o parâmetro que representa o quanto maior a largura deve ser em relação à altura, e o estado de desmaio vertical, quando $\frac{height}{highest\ height} < \beta$, onde β é o parâmetro que representa a relação entre altura atual e a maior altura registrada. Através de observações, foi possível notar que alguns tipos de ações de curto período de duração, como amarrar os cadarços do tênis, eram detectados como desmaio vertical. Para solucionar esse problema, foram introduzidos dois parâmetros, $time_\alpha$ e $time_\beta$, que definem quanto tempo é necessário permanecer nos estados de desmaio horizontal e vertical, respectivamente, para se determinar um evento. O estado de desmaio horizontal apresenta uma forma bem definida, e não foi observado nenhuma ação que gerasse falso positivo. Portanto, é possível afirmar que $time_\alpha < time_\beta$, ou seja, é preciso esperar menos tempo para determinar que um desmaio horizontal foi detectado, enquanto que no outro caso, é preciso esperar por mais tempo, para garantir que não se trata apenas de uma ação transitória.

Para cobrir possíveis erros, de forma a diminuir os falsos negativos, outra hipótese do algoritmo é que pessoas paradas por muito tempo podem estar com algum problema. Então, se por alguma falha, for atribuído de forma incorreta o estado normal a uma pessoa, quando o correto seria o de desmaio, será detectado falta de movimento, colocando esta pessoa no estado de "sem movimento". Após um certo tempo, definido como $time_{without\ motion}$, neste estado, um alerta é gerado. As imagens que chegam ao detector de objetos apresentam pequenas variações na intensidade de cores de alguns pixeis, devido à qualidade da câmera e, a variação da iluminação do ambiente. Então mesmo que os elementos do ambiente não mudem de posição, dois quadros consecutivos tem um pequena, e quase nula, probabilidade de serem idênticos. Este fato, juntamente com erros de precisão do detector de objetos, fazem com que a caixa delimitadora de uma pessoa que não se movimenta, de um quadro para o consecutivo, apresente uma variação em sua posição ou tamanho. Levando em conta este fato, o conceito de movimento deve levar em conta que mesmo quando a pessoa está parada, sua caixa continua a se mover, através de pequenas oscilações. Portanto, o movimento é definido como a intersecção nula entre a última atualização de posição e a posição da caixa atual, ou seja, $IoU_{motion} = 0$. Sempre que o movimento é identificado, atualiza-se a posição da pessoa com o valor de posição da caixa que gerou o movimento. Vale notar que, se por um lado, uma pessoa sem movimento por um longo período de tempo pode estar com problemas, por outro, uma pessoa que se move é considerada como estando no estado normal.

Figura 26: Admite-se movimento quando a intersecção entre a última atualização de posição e a caixa atual é nula



Fonte: Autores

Como dito anteriormente, para eliminar falsos positivos, o algoritmo detecta um estado e apenas gera evento quando uma pessoa permanece naquele estado por um certo período de tempo, que varia de estado para estado. Quanto maior o tempo esperado, maior é o grau de certeza que se tem de que o estado é verdadeiro, porém, a depender do evento, o tempo pode ser um fator crítico, de forma que é preciso obter um compromisso entre grau de certeza e tempo de resposta. Devido aos mesmos ruídos presente nas imagens e erros do detector de objetos, citados anteriormente, algumas detecções não são realizadas, mesmo quando há pessoas na imagem. Esse fato faz com que o histórico da pessoa seja perdido, ou seja, os dados de posição e, principalmente, tempo em um determinado estado sejam perdidos. Para contornar esse problema, existe o conceito de persistência, onde os dados de uma pessoa detectada só são excluídos após não ser detectada consecutivamente em um certo número de vezes, representado pelo parâmetro *persisting number*.

Para modelar uma pessoa, que possui atributos de estado, tempo de permanência no estado, entre outros citados anteriormente, o algoritmo possui a classe *Person*. A medida em que as pessoas são detectadas, uma lista é preenchida com esses objetos. Além disso,

é preciso saber qual objeto *Person*, salvo e atualizado no quadro anterior, corresponde ao *Object* (ver Figura 16 e Figura 24) detectado no quadro atual, para que seja possível observar como os atributos de cada pessoa mudam ao longo dos quadros. Esse processo é chamado de rastreamento ou *tracking*. Para realizar esta tarefa, o algoritmo assume a seguinte hipótese: o *Object* detectado no quadro atual mais próximo de *Person* é o seu correspondente. Para que isso seja possível, *Person* possui dois atributos do tipo *Object*, um para o quadro anterior e outro para o atual.

O algoritmo de *tracking* consiste em calcular a distância entre todos os atributos *Object*, do objeto *Person*, e *Object*, detectados no quadro atual. As distâncias são salvas em uma matriz, que é "achatada"(*flattened*) e ordenada em ordem crescente. A correspondência entre objetos é obtida pelos índices dos primeiros elementos da matriz ordenada. Há três cenários distintos para o *tracking*: pessoas permanecem, entram e saem da imagem.

Uma vez que os principais conceitos do algoritmo foram descritos, é possível deduzir alguns casos em que provavelmente ocorrerão erros. O caso mais simples é quando uma pessoa estiver deitada no chão, seja um morador de rua ou alguém descansando em um parque. O estado de desmaio vertical é caracterizado pela diminuição da altura e pouca variação da largura da caixa de detecção. É possível alcançar essas características estando sentado ou abaixado, amarrando os cadarços do tênis ou pegando um objeto. O primeiro caso é problemático, pois uma pessoa pode permanecer sentada por um longo período de tempo, de forma a superar $time_{\beta}$, gerando um evento. Os outros casos não apresentam problemas, já que são ações de curta duração de tempo. Estes cenários devem ser encarados como restrições, pois o algoritmo não é capaz de diferenciar tais situações do evento de desmaio.

6.2.3 Serviço de ações

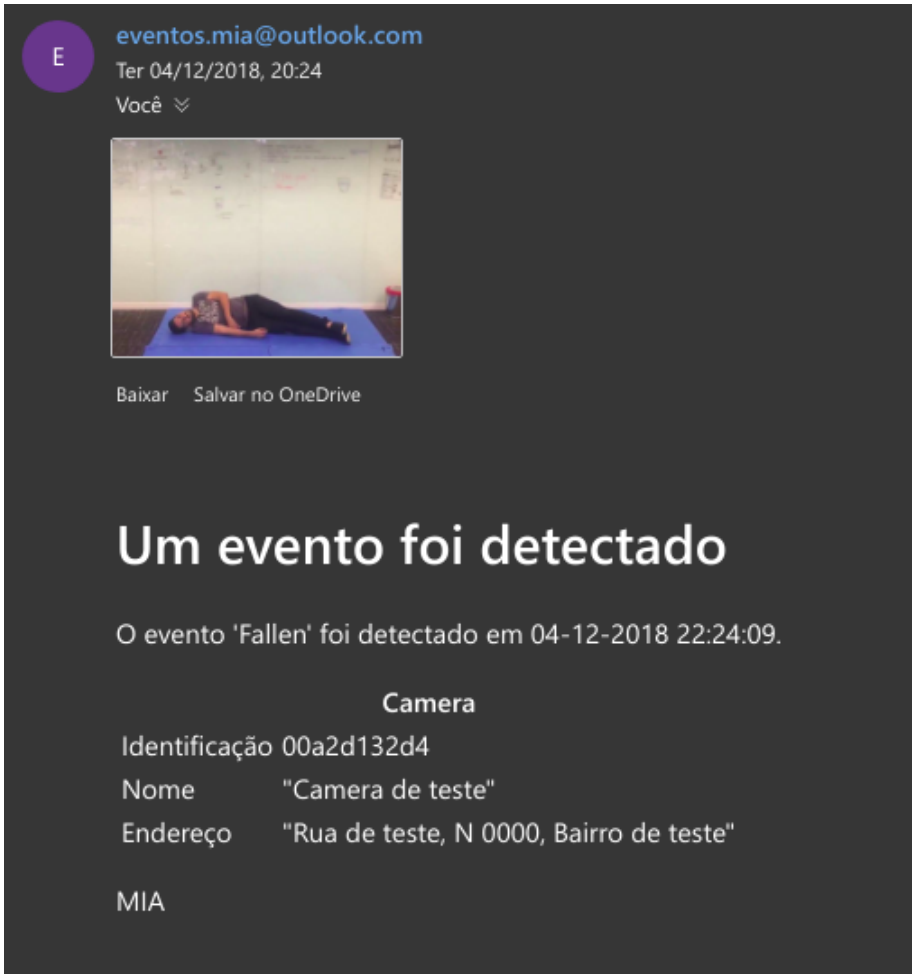
O serviço de ações é responsável por calcular todas as ações cabíveis a um evento e, realizar todas a ações. No escopo de implementação, a ação que deve ser tomada para o evento de desmaio é o envio de um e-mail para um endereço fixo.

Sua API consiste em um único endpoint, onde o identificador do dispositivo e o tipo de evento são passados como parâmetro. A partir do identificador do dispositivo, são obtidos os dados do mesmo, além de uma captura de imagem do instante do evento. Com essas informações, o e-mail é montado.

A transmissão do e-mail é realizada através da biblioteca Python Boto3, que é uma

interface dos serviços Simple Email Service (SES) da AWS.

Figura 27: E-mail gerado, contendo informações sobre a câmera e uma imagem do evento



Fonte: Autores

6.2.4 Submissão de vídeos

O serviço de submissão de vídeos é responsável por coletar e enviar o vídeo da câmera ao serviço de detecção de objetos. No escopo de implementação, um Raspberry Pi com uma câmera conectada é utilizado como dispositivo de submissão, porém, outros dispositivos, como computadores pessoal, também podem ser utilizados como dispositivo de submissão. Ele executa o software de coleta e envio de vídeo para o serviço de detecção de objetos. Ao ser ligado pela primeira vez, o dispositivo faz uma solicitação ao serviço de detecção de objetos, informando seu código de identificação, para então iniciar a submissão.

A coleta de vídeo é feita através da biblioteca OpenCV, que disponibiliza métodos de acesso à câmera, A solicitação de submissão feita para o serviço de detecção de objetos

é realizada utilizando-se a biblioteca *Requests*. Os quadros de vídeo são comprimidos, de forma que seu tamanho máximo seja de 65 KB, e são enviados utilizando-se a classe *sokets* da linguagem Python.

6.2.5 Serviço de usuários

O serviço de usuários é responsável por armazenar as informações relacionadas aos usuários do sistema bem como os dispositivos, câmeras, possuídos. Sua API consiste em endpoints que realizam as operações básicas no banco de dados, isto é, adição, busca, edição e remoção de usuários e dispositivos. Está contido nessas informações campos relevantes ao endereço dos usuários e onde estão localizados as câmeras, informações de contato que poderão ser utilizadas no serviço de ações, identificadores únicos tanto dos usuários quanto das câmeras para validação.

6.2.6 Dashboard de monitoramento

O serviço de dashboard de monitoramento é responsável por exibir as informações relevantes aos outros serviços. Entre essas informações consta os logs recebidos dos serviços através do broker MQTT, assim como é possível receber vídeo de depuração do detector de objetos. A sua interface é baseada em um *template* feito com o*web-framework* Angular, o *backend* onde é feito as interações com outros serviços utiliza a *microframework* em python Flask.

7 TESTES E AVALIAÇÃO

7.1 Infraestrutura

A infraestrutura do sistema consiste na submissão de vídeos, onde a câmera fica instalada no ambiente, no serviço de detecção de objetos, que é hospedado em uma máquina de processamento gráfico na nuvem, no serviço de detecção de eventos, no serviço de usuários e no serviço de ações, que são hospedados em máquinas de propósito geral na nuvem. Antes dos testes com algoritmos, foram realizados testes de integração entre os serviços, para validar comunicação.

Durante testes com submissão de vídeos, foi constatado que o serviço de detecção de objetos estava processando, em média, 18 FPS, na placa gráfica Tesla M60, da NVIDIA. O valor esperado era de 12 FPS, de forma que o resultado obtido superou as expectativas.

O envio dos objetos detectados para o serviço de detecção de eventos e a correta geração de e-mails de alerta pelo serviço de ações demonstrou o correto funcionamento do sistema.

7.2 O algoritmo

Como apresentado na seção 6.2.2.2, o algoritmo possui os seguintes parâmetros: α, β, $time_{\alpha}$, $time_{\beta}$, $time_{without\ motion}$, $persisting\ number$ e IOU_{motion}. Os mais críticos e difíceis de se determinar via bom senso são α e β. Por esse motivo, eles foram estimados através de testes.

Ao todo, foram coletados 31 vídeos, sendo 26 de qualidade média e 5 de qualidade baixa. Das 31 amostras, 12 foram escolhidos para a estimação dos parâmetros e os 19 restantes, para quantificar a taxa de acerto do algoritmo com os parâmetros estimados.

As 12 amostras escolhidas para a estimação tiveram que passar por um processo, onde foram extraídas duas imagens de cada vídeo, uma contendo o evento (desmaio) e outra,

o estado normal de uma pessoa (parada em pé ou caminhando). O processamento das imagens gerou uma tabela com 12 valores para cada parâmetro, cuja análise descritiva é mostrada na Tabela 4. Portanto, escolhendo o teto da média, os parâmetros ficaram definidos como $\alpha = 0,7$ e $\beta = 0.5$.

Tabela 4: Análise descritiva dos dados utilizados na estimação dos parâmetros

	α	β
Quantidade	12	12
Média	0,693661	0,475168
Desvio padrão	0,369198	0,087339
Mínimo	0,319239	0,341373
25%	0,394464	0,408522
50%	0,574568	0,491962
75%	0,904875	0,547293
Máximo	1,447059	0,577197

Fonte: Autores

Por fim, para obter medidas de acurácia do algoritmo com os valores estimados, foram submetidos ao sistema os 19 vídeos de teste. O resultado dos testes é apresentado na Tabela 5, onde é possível notar que a taxa de acerto do algoritmo está em torno de 80% para ambas as qualidades testadas. Foram utilizados para o cálculo das taxas os valores: $\alpha = 0,7$, $\beta = 0,5$, $time_{\alpha} = 4s$, $time_{\beta} = 6s$, $time_{without\ motion} = 60s$, $persisting\ number = 100$ e $IOU_{motion} = 0$.

Tabela 5: Taxa de acerto do algoritmo com os parâmetros definidos

Qualidade	Detectados	Não detectados	Taxa de acerto	Taxa de falsos negativos
Média	11	3	78,57%	21,43%
Baixa	4	1	80%	20%

Fonte: Autores

7.3 Análise dos resultados

O correto funcionamento da integração dos serviços mostra que a infraestrutura está pronta para receber novos algoritmos detectores de eventos, que é fundamental para a expansão do projeto.

O algoritmo implementado apresentou um taxa de acerto de aproximadamente 80% para vídeos com eventos contidos em seu conjunto de restrições. Esse valor é satisfatório para uma primeira versão do algoritmo, em um contexto acadêmico, porém, é um valor baixo para prestar serviço em linha de produção.

Os casos de restrição do algoritmo, citados em 6.2.2.2, foram testados e validados. De fato, nota-se uma grande taxa de falsos positivos para pessoas sentadas e que não aparecem de corpo inteiro na imagem.

8 CONSIDERAÇÕES FINAIS

8.1 Conclusões do projeto de formatura

Este trabalho contribuiu de forma positiva para o grupo, pois proporcionou a oportunidade de arquitetar, desenvolver e gerenciar um projeto do inicio ao fim. Esta experiência agregou um grande valor a base de conhecimento do grupo, dado que foi proposto um desafio completamente novo, que demandou a utilização das técnicas e teorias aprendidas durante o curso de graduação.

O principal objetivo deste trabalho foi demonstrar a viabilidade da arquitetura de um sistema capaz de detectar eventos e prestar assistência em tempo real. Segundo a seção 7.2, ele foi alcançado com uma precisão de aproximadamente 80%. Apesar de ser um valor consideravelmente bom para o contexto acadêmico, não foi possível desenvolver um algoritmo que atendesse as altas taxas de acerto compatíveis com o ambiente de produção.

8.2 Contribuições

A principal contribuição deste trabalho foi a de apresentar a arquitetura de um sistema capaz de fornecer assistência em tempo real para pessoas em ambientes monitorados, de forma automática e eficiente. As ideias apresentadas neste trabalho são de grande valor para a sociedade, pois, uma vez implantado o sistema nas cidades, haveria uma redução na taxa de mortes ou quadros graves de saúde causados por falta de assistência.

Para provar a viabilidade do sistema proposto, foi desenvolvido um algoritmo capaz de detectar o evento de desmaio, através de uma abordagem muito simples. Ele serve de inspiração para que novos desenvolvedores venham a contribuir com a criação de novos algoritmos, de forma que, se incorporados ao sistema, aumenta a sensação de segurança e bem-estar da população.

8.3 Perspectivas de Continuidade

Por se tratar de um projeto de formatura, as características de negócio foram apresentadas de forma superficial. Há diversos pontos a serem explorados, onde é possível desenvolver novas linhas de negócio, uma vez que se trata de uma plataforma que reúne grande quantidade de informações em forma de vídeo.

A arquitetura do projeto propõe que diversos tipos de algoritmos detectores de eventos estejam incorporados ao sistema. Neste trabalho, apenas um algoritmo foi desenvolvido. Existe a necessidade de se dar continuidade nessa área, seja melhorando o algoritmo que foi desenvolvido ou, criando novos.

As ações do sistema são de grande importância. É preciso realizar um estudo de quais ações devem ser tomadas em cada evento, realizando a integração dessas ações com os serviços externos.

REFERÊNCIAS

1 BERMÚDEZ, A. C. Câmeras de segurança inteligentes devem ser instaladas na usp em 2015. 2014. Disponível em: <http://www.jornaldocampus.usp.br/index.php/2014/11/cameras-de-seguranca-inteligentes-devem-ser-instaladas-na-usp-em-2015/>. Acesso em: 12 mar. 2018.

2 SHOTSPOTTER. The shotspotter solution. 2018. Disponível em: <https://www.shotspotter.com/technology/>. Acesso em: 12 mar. 2018.

3 EPO. European patent office, patente au2018100039. 2018. Disponível em: <https://worldwide.espacenet.com/publicationDetails/biblio?DB=EPODOC&II=0&ND=3&adjacent=true&locale=en_EP&FT=D&date=20180208&CC=AU&NR=2018100039A4&KC=A4>. Acesso em: 12 mar. 2018.

4 OPENCV-PYTHON TUTORIALS. Welcome to opencv-python tutorials's documentation! 2018. Disponível em: <https://opencv-python-tutroals.readthedocs.io/en/latest/index.html>. Acesso em: 23 set. 2018.

5 ANDREW NG. Convolutional neural network (cnns) by andrew ng [full course]. 2017. Disponível em: <https://www.youtube.com/playlist?list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF>. Acesso em: 26 set. 2018.

6 RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN 0136042597, 9780136042594.

7 REDMON, J. et al. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. Disponível em: <http://arxiv.org/abs/1506.02640>.

8 KOSUB, S. A note on the triangle inequality for the jaccard distance. *CoRR*, abs/1612.02696, 2016. Disponível em: <http://arxiv.org/abs/1612.02696>.

9 HOSANG, J. H.; BENENSON, R.; SCHIELE, B. Learning non-maximum suppression. *CoRR*, abs/1705.02950, 2017. Disponível em: <http://arxiv.org/abs/1705.02950>.

10 PIECH, C. K means. 2013. Disponível em: <http://stanford.edu/~cpiech/cs221/handouts/kmeans.html>. Acesso em: 15 set. 2018.

11 YADAV, V. (part 1) generating anchor boxes for yolo-like network for vehicle detection using kitti dataset. 2017. Disponível em: <https://medium.com/@vivek.yadav/part-1-generating-anchor-boxes-for-yolo-like-network-for-vehicle-detection-using-kitti-dataset-b2fe033e5807>. Acesso em: 15 set. 2018.

12 PUTMAN, J. *Architecting with RM-ODP*. Prentice Hall PTR, 2001. (Software architecture series). ISBN 9780130191168. Disponível em: <https://books.google.com.br/books?id=PA7BewHWqJQC>.

13 FOWLER, M.; LEWIS, J. Microservices. 2014. Disponível em: <http://martinfowler.com/articles/microservices.html>. Acesso em: 15 set. 2018.

14 NEWMAN, S. *Building Microservices*. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2015. ISBN 1491950358, 9781491950357.

15 BUYYA, R. et al. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 25, n. 6, p. 599–616, jun. 2009. ISSN 0167-739X. Disponível em: <http://dx.doi.org/10.1016/j.future.2008.12.001>.

16 MELL, P. M.; GRANCE, T. *SP 800-145. The NIST Definition of Cloud Computing*. Gaithersburg, MD, United States, 2011.

17 KELLY, S. D. T.; SURYADEVARA, N. K.; MUKHOPADHYAY, S. C. Towards the implementation of iot for environmental condition monitoring in homes. *IEEE Sensors Journal*, v. 13, n. 10, p. 3846–3853, Oct 2013. ISSN 1530-437X.

18 PATEL, K. K.; PATEL, S. M. et al. Internet of things-iot: definition, characteristics, architecture, enabling technologies, application & future challenges. *International Journal of Engineering Science and Computing*, v. 6, n. 5, 2016.

19 LEE, I.; LEE, K. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, v. 58, n. 4, p. 431 – 440, 2015. ISSN 0007-6813. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0007681315000373>.

20 REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. Disponível em: <http://arxiv.org/abs/1804.02767>.

21 KATHURIA, A. How to implement a yolo (v3) object detector from scratch in pytorch: Part 1. 2018. Disponível em: <https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/>. Acesso em: 15 nov. 2018.

22 ZEROTIER. Zerotier manual. 2018. Disponível em: <https://www.zerotier.com/manual.shtml>. Acesso em: 15 mar. 2018.

23 INTEL. Intel movidius neural compute stick. 2018. Disponível em: <https://software.intel.com/en-us/movidius-ncs>. Acesso em: 15 out. 2018.

24 YOLO. Yolo: Real-time object detection. 2016. Disponível em: <https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detection>. Acesso em: 10 jun. 2018.

25 SAINT-EXUPERY, A. de. *Voo noturno*. Difusão Européia do Livro, 1969. ISBN 9788567097046. Disponível em: <https://books.google.com.br/books?id=nvcVtwEACAAJ>.