UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MIKAEL LUCAS DE BRITO SOUSA

DESENVOLVIMENTO DE SISTEMA DE MONITORAMENTO DE CÂMERAS PARA DETECÇÃO DE AGLOMERAÇÃO E MÁSCARAS DE PROTEÇÃO INDIVIDUAL ATRAVÉS DE REDES NEURAIS

FORTALEZA

2022

MIKAEL LUCAS DE BRITO SOUSA

DESENVOLVIMENTO DE SISTEMA DE MONITORAMENTO DE CÂMERAS PARA DETECÇÃO DE AGLOMERAÇÃO E MÁSCARAS DE PROTEÇÃO INDIVIDUAL ATRAVÉS DE REDES NEURAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Fabrício Gonzalez Nogueira

Coorientador: Msc. Marcus Davi do Nascimento Forte

FORTALEZA

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S697d Sousa, Mikael Lucas de Brito.
Desenvolvimento de sistema de monitoramento de câmeras para detecção de aglomeração e máscaras de proteção individual através de redes neurais / Mikael Lucas de Brito Sousa. – 2022.
52 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2022.
Orientação: Prof. Dr. Fabrício Gonzalez Nogueira.
Coorientação: Prof. Me. Marcus Davi do Nascimento Forte.

1. Redes neurais. 2. Deep Learning. 3. Multithreading. 4. Visão computacional. 5. PyQt. I. Título.
CDD 621.3

MIKAEL LUCAS DE BRITO SOUSA

DESENVOLVIMENTO DE SISTEMA DE MONITORAMENTO DE CÂMERAS PARA DETECÇÃO DE AGLOMERAÇÃO E MÁSCARAS DE PROTEÇÃO INDIVIDUAL ATRAVÉS DE REDES NEURAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Fabrício Gonzalez Nogueira (Orientador)
Universidade Federal do Ceará (UFC)

Msc. Marcus Davi do Nascimento Forte (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Arthur Plínio de Sousa Braga
Universidade Federal do Ceará (UFC)

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

AGRADECIMENTOS

Primeiramente agradeço a Deus que sempre me acompanhou e me protegeu em todos os meus caminhos e projetos.

Aos meus pais, Jose Cleudes [*In Memoriam*] e Luzia Maria, por terem me criado bem e sempre se esforçado ao máximo para me ensinar valores, trabalho e fé, além de de sempre me incentivar a estudar buscando sempre mais. Em especial ao meu pai, sempre foi uma referência, um amigo presente e meu maior incentivador em todas as minhas batalhas. Eu queria muito que estivesse aqui comigo nesse momento mas o levo em meu coração e memória para sempre.

Aos meus amigos de longa data Germano Henrique, Fernando Pablo e Jessica Sousa, a amizade de vocês significa muito para mim e se fez presente desde os melhores aos mais sombrios tempos.

A minha namorada Carol, que me acompanhou e incentivou durante parte da minha graduação. Seu amor e carinho são muito importantes para mim e fazem parte da minha história.

Ao grupo PET Engenharia Elétrica, um grupo de excelência que me orgulho de ter feito parte, por todas as experiências, os aprendizados e por me incentivarem a ser um melhor aluno e profissional, mesmo antes de ser um petiano. "Uma vez petiano, sempre petiano"

Ao Grupo de Pesquisa Automação e Robótica (GPAR) pelas oportunidades e espaço de trabalho durante meu projeto de iniciação científica, assim como a equipe de excelência que o compõe.

A empresa RAV Tecnologia, onde realizei meu segundo estágio e pude trabalhar com uma equipe de excelência que me ensinou tanto na prática quanto na teoria.

Aos meus orientadores Dr. Fabrício Nogueira e Dr. Marcus Davi, pelos ensinamentos e orientações ao longo da trajetória no curso e deste trabalho.

Por fim, os amigos e colegas que conheci durante o curso, são vários que seria um erro esquecer de nominar algum e por isso deixo meus agradecimentos a todos. Vocês foram parte fundamental na minha graduação, desde os estudos e projetos aos momentos de diversão.

"*Si vis pacem, para bellum*"

(Flavius Vegetius)

RESUMO

Em virtude da crise sanitária do novo coronavírus, Covid-19, o mundo se pôs diante de uma situação de saúde pública onde cuidados como o uso de máscaras de proteção individual e o distanciamento social para evitar aglomerações se fizeram necessários para evitar a disseminação do vírus. No entanto, esses cuidados e precauções não são devidamente cumpridos por uma parte da população e, por conta disso, surgiu a necessidade de criação de mecanismos de monitoramento em prol do cumprimento das determinações de saúde pública e o alerta nos casos de não cumprimento das mesmas. Por isso, foi desenvolvido um sistema de monitoramento e gerenciamento de câmeras em tempo real que aplica *Processamento Digital de Imagem* (PDI) e algoritmos de visão computacional através de redes neurais artificiais para detecção do uso de máscaras e de aglomeração de pessoas.

O sistema foi desenvolvido em linguagem Python em conjunto ao framework PyQt para desenvolvimento da interface gráfica. O sistema foi construído sob paradigma de orientação a objetos para gerência e conexão entre o *backend* e o *frontend*. Além disso, foi criado um banco de dados MySql com 4 tabelas: uma para armazenamento das credenciais de acesso, uma para armazenamento das informações das câmeras usadas, uma para armazenamento das infrações de ausência de máscaras e uma para armazenamento das infrações de aglomeração.

Palavras-chave: Redes Neurais. Deep Learning. PyQt. Multithreading. Visão Computacional. Covid-19.

ABSTRACT

Due to the health crisis of the new coronavirus, Covid-19, the world is faced with a public health situation where care such as the use of personal protective masks and social distancing to avoid agglomerations were necessary to prevent the spread of the virus. However, these precautions and precautions are not properly complied with by part of the population and, because of this, the need arose to create monitoring mechanisms in order to comply with public health determinations and alert in cases of non-compliance with them. Therefore, a real-time camera monitoring and management system was developed that applies digital image processing and computer vision algorithms through artificial neural networks to detect the use of masks and crowding of people. The system was developed in Python language together with the PyQt framework for the development of the graphical interface. The system was built under an object-oriented paradigm to manage and connect the *backend* and the *frontend*. In addition, a MySql database was created with 4 tables: one for storing access credentials, one for storing used camera information, one for storing non-mask infractions and one for storing agglomeration infractions.

Keywords: Neural Networks. Deep Learning. PyQt. Multithreading. Computational Vision. Covid-19.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

PDI	*Processamento Digital de Imagem*
CNN	*Convolutional Neural Network*
POO	Programação Orientada a Objetos
RTSP	*Real Time Streaming Protocol*
ORM	*Object-Relational Mapper*

SUMÁRIO

1 INTRODUÇÃO

Em decorrência da crise sanitária causada pela pandemia do novo coronavirus, COVID-19, o mundo se viu em uma situação de calamidade pública e sistêmica em que diversos hábitos, costumes e práticas precisaram ser adaptados. O novo coronavirus tem sua disseminação de pessoa a pessoa pelo ar, por meio de tosse ou espirro, pelo toque ou aperto de mão ou pelo contato com objetos ou superfícies contaminadas, seguido pelo contato com a boca, nariz ou olhos (RMM DABOIN BEG; H., 2020). Uma das primeiras adaptações feitas para conter a disseminação do vírus foi o uso de máscaras de proteção individual tanto por profissionais da saúde quanto pela a população em geral (GIRARDI *et al.*, 2021) e a restrição de aglomerações entre pessoas. Em decorrência do aumento preocupante de casos, foram declarados *lockdowns* evitar o aumento de casos.

Com o passar dos meses, e com a crescente necessidade da população de sair de suas casas e retornar aos trabalhos presenciais, foram estabelecidas medidas de flexibilização como o fim dos *lockdowns*, o mantimento de uma política de distanciamento social e o uso de máscaras de proteção individual para todos. Contudo, sob o risco do surgimento de novos casos decorrentes do descumprimento das medidas de segurança estabelecidas, um sistema de monitoramento automatizado seria de grande valia e, a partir disso, veio a ideia do presente trabalho.

Para assegurar que as medidas de distanciamento social e uso de máscaras de proteção individual fossem obedecidas, foi desenvolvido um sistema de monitoramento e gerenciamento de câmeras para controle e processamento digital de imagem por redes neurais para detecção de aglomerações e uso de máscaras de proteção individual. Além disso, foi configurado em conjunto um banco de dados MySql para armazenamento dos eventos detectados pelo sistema de modo a manter um provas da alegação e recurso de controle quanto as verificações.

O sistema vai desenvolvido sobre ferramentas *Open Source* de modo que seu uso, replicação e melhoria é possível sem quaisquer custos excedentes de bibliotecas particulares. Além disso, para o presente sistema foi desenvolvido uma metodologia de aplicação e alocação dos métodos de processamento de imagem de modo que outros tipos de processamento de imagem pudessem ser aplicados para demais propósitos.

Por fim, o presente trabalho apresenta tanto a proposta e desenvolvimento de um sistema de gerenciamento de câmeras baseado em *threads* com aplicação de processamento digital de imagem através de redes neurais e que pode ser adaptado também a outras redes

neurais quanto um estudo de caso para as condições adversas decorrentes da pandemia do novo coronavírus.

1.1 Objetivos

O objetivo principal do presente trabalho encontra-se no desenvolvimento de um sistema para gerenciamento e monitoramento de câmeras que aplica técnicas de visão computacional através de redes neurais para detecção do uso de máscaras de proteção individual e aglomerações.

Contudo o objetivo principal pode ser dividido em alguns objetivos específicos, como:

1. Estudo de processamento digital de imagem com linguagem python e frameworks livres amplamente usados por empresas e pela comunidade acadêmica,
2. Estudo acerca do uso, desenvolvimento e treinamento de redes neurais artificiais,
3. Desenvolvimento de interfaces gráficas com framework PyQt e o software Qt Designer
4. Uso de banco de dados MySql e interação com interfaces gráficas,
5. Simular sistema em tempo real em diferentes ambientes.

1.2 Estrutura do trabalho

O trabalho se estrutura em 5 capítulos, discriminados da forma abaixo:

- Capítulo 1: Introduz os objetivos e perspectivas do trabalho;
- Capítulo 2: Aborda conceitos teóricos presentes no trabalho, abrangendo os princípios das redes neurais aplicadas ao processamento de imagem, o desenvolvimento das interfaces dos sistema, uso de banco dados e a base das tecnologias utilizadas;
- Capítulo 3: Apresenta a estrutura do sistema seguido dos algoritmos, telas e a integração de ambos com o banco de dados usado
- Capítulo 4: Mostra um compilado das telas do sistema em funcionamento e aplicando os algoritmos de processamento de imagem para detecção de máscaras e aglomerações em tempo real;
- Capitulo 5: Conclui o trabalho com as devidas considerações e apresenta sugestões de trabalhos futuros;

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo é apresentada a base teórica e prática necessária para a construção do sistema. O objetivo é apresentar ao leitor tecnologias e conceitos que permitissem o desenvolvimento de uma aplicação completa como é proposto.

O processamento digital de imagem é fundamental para o primeiro processo de padronização das imagens a serem processadas e a biblioteca OpenCV é uma ferramenta importantíssima, além de gratuita e aberta, que auxilia em todo o processo, sendo por tanto o inicio do trajeto. Contudo, para reconhecimento de padrões e criação de modelos que pudessem ser treinados para a detecção de aglomeração e detecção de máscaras seria necessário o uso de redes neurais voltadas para o processamento de imagem, como redes neurais convolucionais. Por fim, após o tratamento e ajuste dos métodos de processamento de imagem era necessário a construção de uma interface gráfica e uma estrutura de sistema que pudesse unir o uso da interface, conexão banco de dados e o processamento paralelo contínuo aplicado as câmeras conectadas.

2.1 Processamento Digital de Imagens

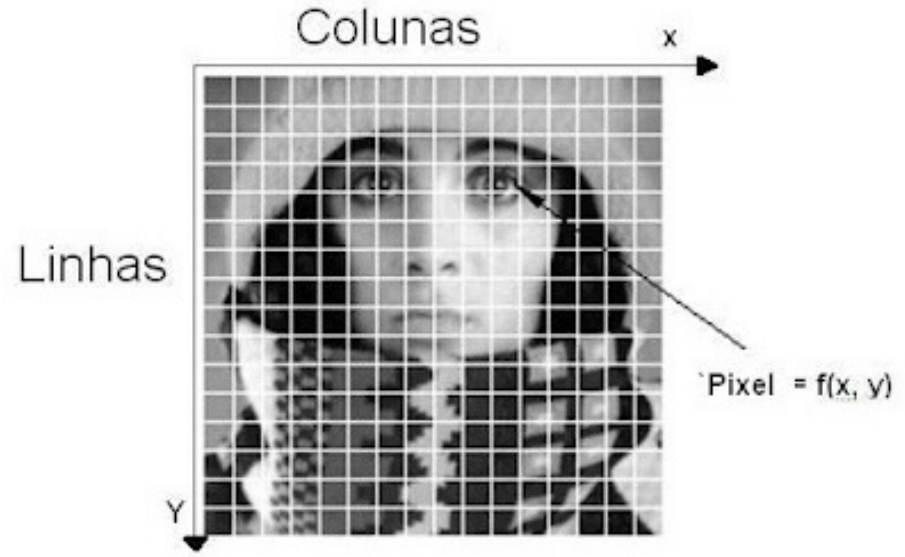
O processamento digital de imagens consiste em um conjunto de técnicas computacionais voltadas a análise e processamento multidimensional de dados. Ou seja, são métodos que recebem uma imagem como parâmetro e retornam uma imagem.

2.1.1 Modelo de Imagem digital

Uma imagem pode ser definida como uma função bidimensional f(x,y) onde x e y são coordenadas no plano cartesiano, e a amplitude f corresponde a intensidade da imagem naquele ponto. Quando x, y e os valores de amplitude f são todos finitos e discretos, a imagem é chamada de imagem digital (GONZALEZ; WOODS, 2002). A construção das imagens digitais, mais especificamente imagens matriciais, é o que permite o processamento por sistemas computacionais e armazenamento em mídias digitais. Cada unidade básica no plano cartesiano da função f(x,y) corresponde a um *pixel*. A figura 1 apresenta uma imagem em escala de cinza, onde cada *pixel* da imagem possui uma componente de cor que oscila de 0 a 255 em grau de intensidade.

Para imagens coloridas, cada *pixel* é composto das três componentes de cores

Figura 1 – Representação de imagem digital.



Fonte: (NEVES, 2009).

primárias do espectro visível humano: RGB (Red, Green e Blue), e através da combinação dessas três componentes obtemos outras cores.

2.2 OpenCV

OpenCV (*Open Source Computer Vision Library*) é uma biblioteca para processamento digital de imagem e visão computacional amplamente utilizada por hobistas e profissionais no mundo. Foi escrita originalmente em C/C++ mas atualmente possui adaptações para outras linguagens como python. A biblioteca além do suporte a manipulação e processamento de imagem também provê métodos de pré-processamento de imagens e suporte para aplicação de redes neurais como redes de aprendizagem profunda, ou em inglês *deep learning*.

No código abaixo é possível ver um exemplo da captura de vídeo de uma câmera em tempo real, no caso da webcam principal da máquina sinalizado pelo número 0 no método *cv2.VideoCapture()*, pela biblioteca opencv (no código, cv2).

Um vídeo nada mais é do que varias fotos, também chamados *frames*, tiradas por uma câmera e apresentadas em sequência a uma taxa definida. No código vemos esse processo na linha 4 onde a função *cv2.read()* retorna uma variável booleana, grab, sinalizando sucesso na captura e um *array*, frame, com as informações da foto, capturada. O método *cv2.imshow* da linha 7 abre uma janela que apresenta o frame capturado. Das linhas 9 a 12 é feita uma verificação para sair do loop de captura e apresentação clicando na letra 'e'.

Código-fonte 1 – Exemplo de uso da biblioteca openCV

```
import cv2
cap = cv2.VideoCapture(0)

while True and self.camOpen:
    grab, frame = cap.read()
    if grab:
        cv2.imshow("tela", frame)

    k = cv2.waitKey(10)
    if k == ord("e"):
        self.cap.release()
        sys.exit()
```

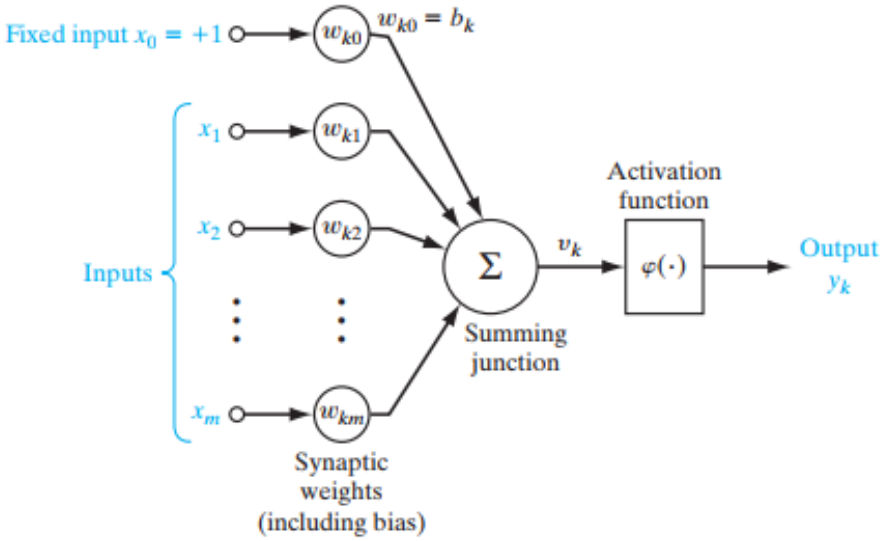
2.3 Redes Neurais Artificiais

Redes neurais Artificiais são sistemas complexos não lineares de processamento paralelo de informações (HAYKIN, 2001), que visam replicar o mecanismo de aprendizagem do cérebro humano por meio de uma rede de neurônios. Um neurônio humano funciona recebendo uma informação de entrada e processando-a por um processo de aprendizado, em seguida sendo armazenado em conexões interneurais nos chamados pesos sinápticos.

A estrutura básica, mostrado através de um perceptron simples na figura 2, é uma representação do modelo de um neurônio artificial que visa replicar o processo de pensamento humano. Sua construção, assim como o neurônio biológico, possui três estruturas básicas sobre as quais pode-se comentar:

1. Conjunto de sinapses((*Inputs*)*(*synaptic weights*)): Corresponde as ramificações de entrada do neurônio, as quais também possuem seus respectivos pesos sinápticos, em inglês (*synaptic weights*). Por exemplo, *Wkj* corresponde ao neurônio k e sinapse j.
2. Somador de Junção(*Summing junction*): Combinador linear que soma o produto de cada entrada ao seu respectivo peso sináptico como em cada ramo.
3. Função de Ativação(*Activation Function*): função, dependente do problema escolhido, que

Figura 2 – Modelo não-linear de um neurônio.



Fonte: (HAYKIN, 2001).

define um limiar para a resposta de saída.

A saída no neurônio é o resultado no conjunto de passos descritos acima: é feita a combinação linear entre os sinais de entrada com seus respectivos pesos da sinapse, incluindo o bias que tem peso 1, e em seguida aplica-se uma não-linearidade proveniente da função de ativação. A equação de saída explicada é mostrada na equação 2.1.

$$y_k = \Phi(\sum_{j=1}^{m}(w_{kj} * x_j) + b_k) \tag{2.1}$$

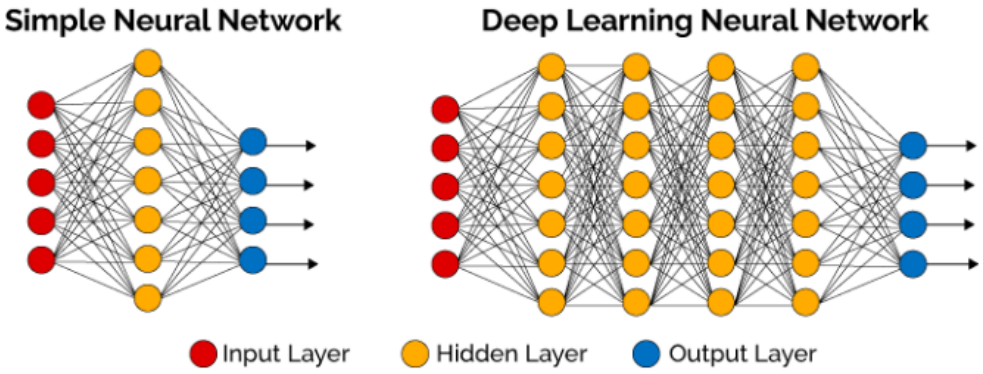
2.3.1 Perceptron Multicamadas

O perceptron multicamadas, ou em inglês *Multilayer Perceptron*, representa a evolução do uso do modelo do perceptron simples organizando um conjunto de perceptrons em camadas. A primeira camada de neurônios é chamada camada de entrada, a última camada é chamada camada de saída e as camadas internas entre a entrada e a saída são chamadas camadas ocultas, em inglês *hidden layers*.

Modelos mais complexos de redes neurais que possuem muitas camadas internas são também chamadas de redes neurais de aprendizagem profunda, conforme mostrado na figura 3.

Pode-se observar na representação que os círculos correspondem aos neurônios e cada linha corresponde a conexão que liga sinapses de diferentes neurônios em cadeia. A medida que a rede é treinada avaliando o processo de classificação com base nas entradas e nas saídas, os pesos correspondentes de cada neurônio são atualizados para reduzir o erro de classificação.

Figura 3 – Redes neurais simples e profundas.



Fonte: (DSA, 2021).

2.3.2 Função de ativação

A função de ativação tem um papel fundamental na constituição de neurônios artificiais pois são responsáveis por inserir não-linearidades no processo, visto que a entrada desse bloco é composta apenas da combinação linear dos elementos da entrada incluindo o bias. Podem ser aplicadas diversos tipos de funções de ativação de acordo com a arquitetura da rede e da natureza das entradas.

2.3.2.1 ReLU

A função de ativação ReLU(*Rectifier Linear Unit*), em português unidade de retificador linear, é um função simples, diferenciável e de derivada fácil. A equação é mostrada abaixo.

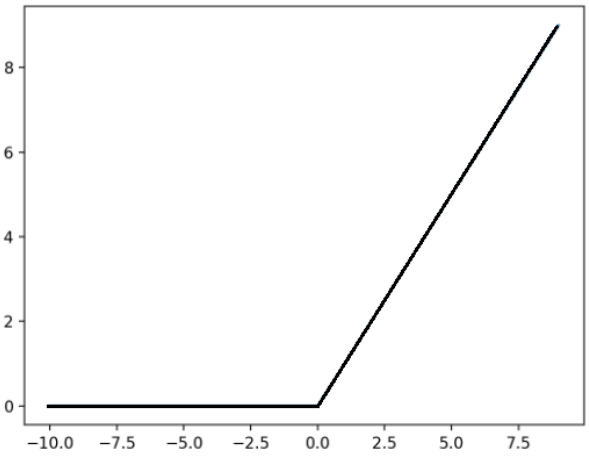
$$ReLU(x) = max(0, x) \tag{2.2}$$

Essa função basicamente zera todos os neurônios de saída menor que zero, conforme visto na figura 4.

2.3.3 Convolutional Neural Network (CNN)

A *Convolutional Neural Network* (CNN), ou em português Rede Neural Convolucional, é um modelo de algoritmo de aprendizagem profunda que pode captar uma imagem de entrada, atribuir pesos e vieses que podem ser aprendidos a vários aspectos da imagem e ser capaz de diferenciar um do outro. O pré-processamento exigido em uma CNN é muito menor em comparação com outros algoritmos de classificação. Enquanto nos métodos primitivos os

Figura 4 – Representação da função ReLU.



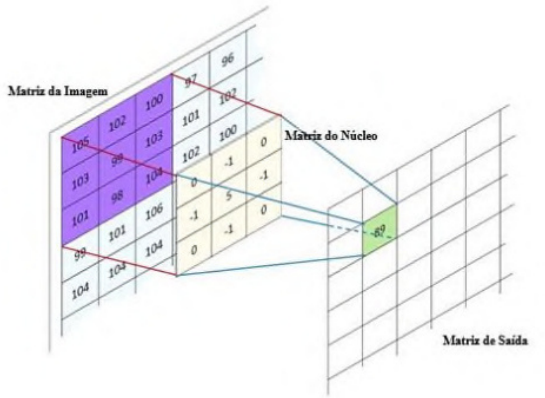
Fonte: O Autor (2021).

filtros são feitos à mão, com treinamento suficiente, as CNNs têm a capacidade de aprender esses filtros (DLB, 2019).

2.3.3.1 Convolução

A convolução de imagens consiste na aplicação do produto interno entre a matriz de pixels de uma imagem pelo matriz núcleo, ou kernel, adotado. O resultado dessa operação faz com que um conjunto de pixels da imagem original gere um pixel da imagem de saída, de modo a fazer uma espécie de compressão de pixels. Esse processo é ilustrado pela figura 5 onde o kernel converte uma parcela 3x3 de matriz de imagem qualquer em um pixel da matriz de saída.

Figura 5 – Exemplo Convolução em imagem.



Fonte: (KLEIN, 2013.)

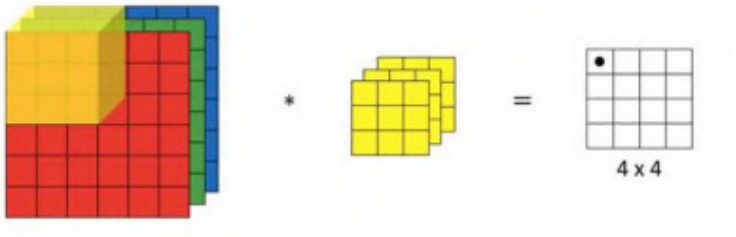
O processo de convolução é iterado para as porções, nesse caso 3x3, definidas gerando uma matriz de saída para ser processada. Kernels diferentes geram matrizes de saída diferentes e, a depender da arquitetura usada, pode piorar ou melhorar a eficiência da rede.

Imagens monocromáticas podem ser definidas por matrizes bidimensionais em que cada pixel contém a intensidade de cor, no caso em apenas uma banda, compreendendo entre branco e preto. Contudo, para imagens coloridas cada pixel possui três bandas de cores (RGB) que compõe o espectro de cor visível, 2.1.1, que são implementadas a imagem como camadas representadas por diferentes matrizes para cada cor.

Dessa forma, as imagens coloridas podem ser descritas como cubos $h * w * c$, onde h é a altura da imagem em pixels, w é a largura da imagem em pixels e c representa o número de canais da imagem. Cada canal corresponde a uma cor da imagem e o kernel também será usado será tridimensional para operar o conjunto de imagem com canais.

A convolução para imagens coloridas ocorre em volumes como mostrado na figura 6. O processo é iterado tal como para imagens monocromáticas apenas considerando a diferença de kernel e então é gerado uma matriz bidimensional de saída.

Figura 6 – Exemplo Convolução volumétrica em imagem.



Fonte: (SINHA, 2017.)

2.4 Qt

Qt é um *framework* de desenvolvimento de software completo escrito originalmente em C++ mas que atualmente possui interfaces para outras linguagens como python. Este possui ferramentas projetadas para simplificar a criação de aplicações com interfaces de usuário de alta interatividade para desktops, sistemas embarcados e plataformas móveis (QT DIGIA, 2013).

O Qt permite desenvolver aplicações multiplataforma, ou seja, foi projetado para desenvolver aplicações em vários sistemas operacionais, como Windows, Linux, Mac OS X e Android. Desta forma, o código-fonte de aplicações desenvolvidas em Qt é implementado uma única vez e pode ser compilado em várias plataformas sem a necessidade de alteração.

O framework Qt está disponível em duas licenças: uma licença para softwares comerciais, na qual os desenvolvedores pagam taxas de licenciamento para utilizar as ferra-

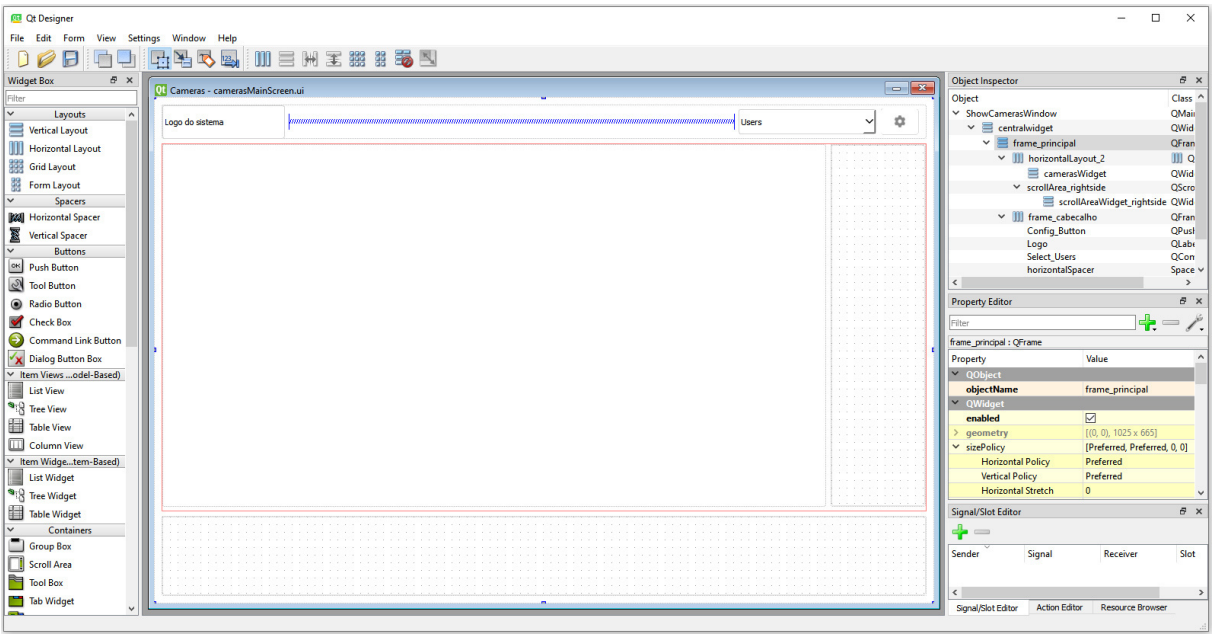
mentas oferecidas pelo Qt, e uma edição livre, na qual as versões possuem código-fonte aberto, podendo ser utilizadas por qualquer desenvolvedor sem cobrança de taxas (BLANCHETTE; SUMMERFIELD, 2006).

Para auxiliar o desenvolvimento Qt, existem aplicações como o Qt Designer que permitem o desenvolvimento por meio de uma interface gráfica interativa onde é possível arrastar/soltar elementos e configurar suas propriedades na tela.

2.4.1 Qt Designer

O *Qt Designer* é um software pertencente a Qt Foundation voltado para desenho e criação de interfaces gráficas. A estrutura do software é apresentada na figura 7.

Figura 7 – Estrutura Qt Designer na tela principal do sistema.



Fonte: O Autor (2021).

Dentro do sistema é possível usar os elementos, *widgets*, fazer conexões dentro das próprias telas através de *Signals* e *Slots*.

Por exemplo, se queremos criar o efeito de apagar o campo de texto de uma mensagem após o botão de envio ser acionado, podemos configurar um *signal* através do click do botão e conectar ao *slot*, que define uma ação, como limpeza do campo de texto, assim antecipando a necessidade de uma rotina de verificação via código para um efeito de design.

2.5 POO

Para o desenvolvimento da interface gráfica e estruturação das classes do sistema, era necessária a escolha de um paradigma de programação que permitisse o paralelismo de controle e capacidade de manipulação de interfaces, para isso foi escolhida a orientada a objetos.
A Programação Orientada a Objetos (POO) é um paradigma de programação baseado em "objetos", que contém dados e códigos em si. Os dados são alocados no formato de campos, ou atributos, de um objeto que definem os estados do objeto e funções em forma de processos, também chamados métodos, que tem o poder de alterar os estados de um objeto sob a devida permissão. Na programação são construídas classes com atributos e métodos para definirem objetos. No entanto, para o devido uso da orientação a objetos é necessário seguir 4 princípios: Encapsulamento, Herança, Abstração e Polimorfismo.

2.5.1 Herança

O principio base da herança é a transferência ou assimilação de atributos e funções a um ser ou indivíduo. Em POO é possível criar um objeto nulo (sem atributos ou métodos) ou baseado em um modelo de forma que este possa herdar atributos, métodos e características para operar conforme o modelo herdado, de modo a se tornar um objeto filho do qual foi herdado ou derivado. Dessa forma, o princípio da herança define uma hierarquia entre classes e objetos, além de permitir uma especialização dos objetos por meio de simples ou múltiplas heranças.

2.5.2 Abstração

A abstração é um modelo de trabalho em forma de "caixa-preta", ou seja, os detalhes internos e mecanismos de implementação usados nas classes são desconhecidos ao externo apenas sendo apresentada a interface de uso por meio dos métodos fornecidos.

2.5.3 Polimorfismo

O polimorfismo permite o tratamento uniforme de classes em uma hierarquia. Portanto, o código de chamada só precisa ser escrito para manipular objetos da raiz da hierarquia, e qualquer objeto instanciado por qualquer classe filha na hierarquia será tratado da mesma maneira.

2.5.4 Encapsulamento

O encapsulamento é a habilidade de "encapsular", semelhantemente a um medicamento, todo ou parte do conteúdo dentro de um invólucro. No contexto da orientação a objetos, atributos e métodos são encapsulados a uma classe. O encapsulamento protege informações internas de um objeto ao resto do código vinculando dados privados aos métodos da classe, assim ocultando os atributos e mostrando apenas as funcionalidades oferecidas da interface do objeto.

Dessa forma, todo o desenvolvimento do sistema, *front-end* e *back-end*, foi construído sob esse paradigma visto que cada classe de *thread*, acesso ao banco de dados e *buffers* de captura e processamento, são construídas na forma de objetos. Uma linguagem de fácil adaptação e em que se pode escrever códigos com orientação a objetos é a linguagem python.

2.6 Python

A linguagem python é uma linguagem de programação de alto nível e alta versatilidade com uma ampla e ativa comunidade de usuários. A linguagem foi feita para trabalhar tanto em paradigma procedural quanto no paradigma orientado a objetos, isso permite que a mesma seja seja usada em uma vasta gama de aplicações como ciência de dados, automação de processos, desenvolvimento de sistemas, processamento de áudio/vídeo, dentre outras.

Todo o sistema desde a interface gráfica, o processamento paralelo (via criação de threads e processos) e a estrutura de organização das classes do sistema foi desenvolvido sobre linguagem python. Essas partes são descritas abaixo por seus conceitos e os *frameworks* usados.

2.6.1 PyQt

PyQt é um *framework* multiplataforma baseado na biblioteca Qt5, Qt na 5ª versão, para linguagem python para desenvolvimento e edição de interfaces gráficas que traz a potencial do Qt junto com a versatilidade da linguagem python. A instalação da biblioteca pode ser feita pelo gerenciador de pacotes *pip* e o uso requer apenas a importação da biblioteca nos campos de código usados.

É possível trabalhar com telas desenvolvidas no Qt Designer juntamente com códigos em python exportando cada tela a um código python a partir da extensão *pyuic5*.
Pyuic5 corresponde ao utilitário uic do Qt. Ele converte GUIs baseadas em QtWidgets criadas usando Qt Designer para código Python. (PyQt5. . . ,)

Esse processo auxilia tanto a referência e manipulação de elementos via código conforme interação do usuário quanto a criação de novos elementos e até telas inteiras via código.

Código-fonte 2 – Exportar .uic em .py

```
1    pyuic5 exampleQtScreen.ui > exampleExportedCode.py
```

A linha de código acima exporta a tela *exampleQtScreen.ui* para o arquivo python *exampleExportedCode.py*.

Em conjunto ao desenvolvimento de telas são usadas figuras e ícones específicos no design que são condensados em um arquivo compilado de recursos *.qrc*. Esse arquivo deve ser exportado para um arquivo python, semelhantemente às telas como mostrado acima, para compatibilidade com o código e uso pelas telas a partir da extensão *pyrcc5*.
Pyrcc5 corresponde ao utilitário rcc do Qt. Ele incorpora recursos arbitrários (por exemplo, ícones, imagens, arquivos de tradução) descritos por um arquivo de coleção de recursos em um módulo Python (PyQt5...,).

Código-fonte 3 – Exportar .rcc em .py

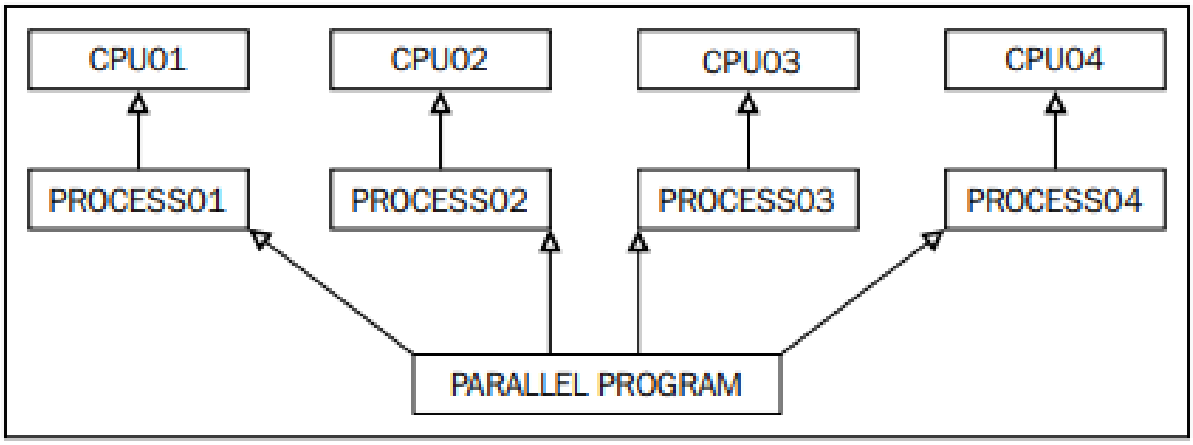
```
1    pyrcc5 figures.qrc > figures_rc.py
```

A linha de código acima exporta o conjunto de figuras e ícones *figures.rcc* para o arquivo python *figures_rc.py*.

2.6.2 Multiprocessamento

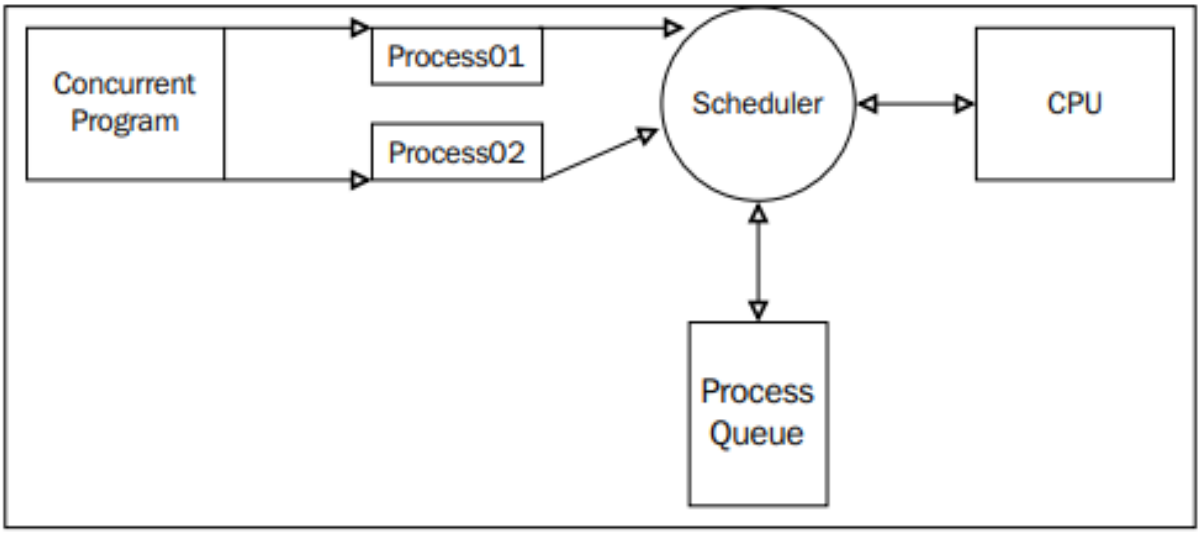
O multiprocessamento é uma técnica de programação paralela para execução de vários processos fazendo uso de dois ou mais núcleos de um processador para execução de forma pseudo-paralela e distribuída, assim aumentando a performance. As limitações acerca desse tipo de programação está relacionada à capacidade física, núcleos disponíveis no processador, e o correto agendamento de atividades. Os programas que não fazem uso de programação paralela denotam a execução de seus processos e tarefas a um agendador automático que compete pela execução em um núcleo. A diferença dos modelos são apresentados nas figuras 8 e 9.

Figura 8 – Comportamento de programação paralela.



Fonte: (PALACH, 2014).

Figura 9 – Comportamento de programação concorrente.



Fonte: (PALACH, 2014).

2.6.3 Multithreading

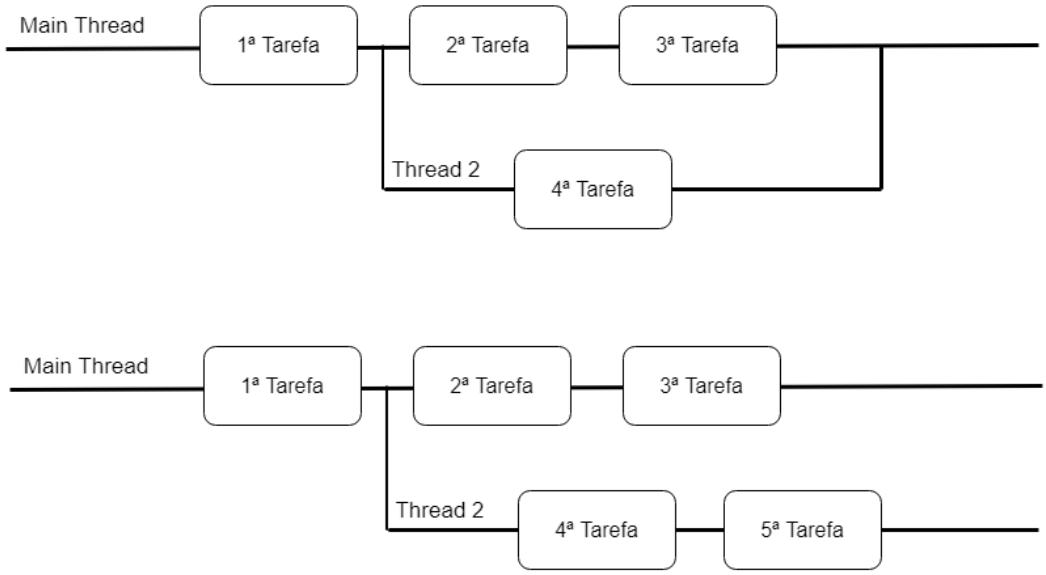
Threads são linhas de execução de código onde sequências de código são retidas. Ao trabalhar com códigos que executem tarefas de maior custo computacional onde não se faça necessária a espera pelo fim da realização de uma tarefa para execução de outra, é possível criar mais *threads*, ou seja novas linhas de execução de código, para execução de novas tarefas de forma pseudo-paralela e que possam conversar entre si.

Um conjunto de *threads* sob o mesmo processo compartilham recursos e memória, enquanto que cada *thread* possui registradores e uma pilha de tarefas exclusiva para si, conforme mostrado na figura 11.

Na figura 10 é mostrado um exemplo de criação de threads a partir da *thread* principal.

No primeiro uma *thread* é criada para resolução de uma tarefa e quando termina retorna a *thread* principal. No segundo uma *thread* e mantém seu fluxo de trabalho paralelamente a *thread* principal com duas tarefas em sequência.

Figura 10 – Exemplo de Fluxo de criação de *Threads*



Fonte: O Autor (2021).

É necessária atenção na hora de programar a criação e gerência de *threads* via código para que haja a correta gerência dos recursos compartilhados, caso contrário uma thread pode requisitar acesso a um recurso comum ao mesmo tempo que outra gerando atrasos e até quebra.

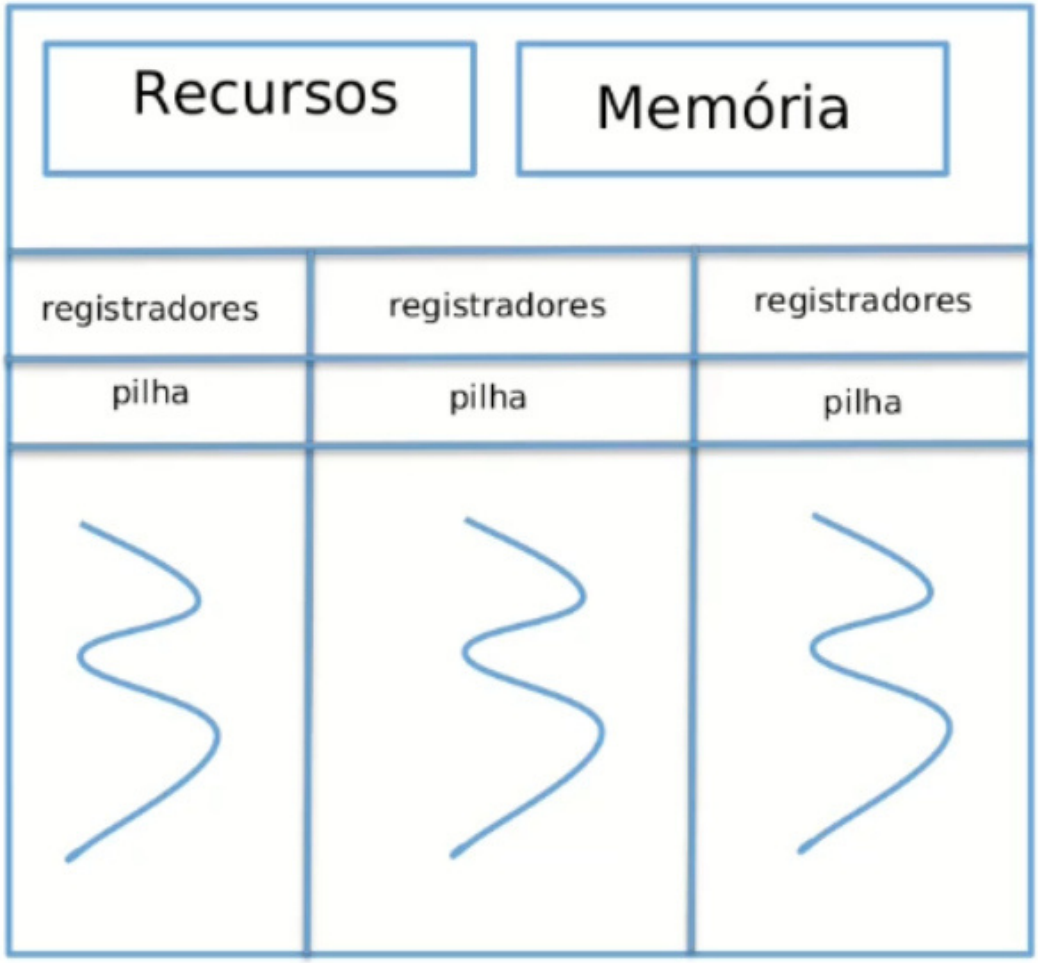
2.6.3.1 Gerenciamento de Recursos para Threads

Para que seja possível o compartilhamento de recursos é necessário um processo de sincronização de *threads* e uma das formas mais populares usados é por meio de *Mutexes*. Mutex é uma chave de recursos colocada em cada *thread* via código travada, em inglês *lock*, antes do uso de recursos e liberada, em inglês *release*, após o uso.

2.7 Protocolo RTSP

Para o controle e acesso as câmeras em rede era necessário um protocolo de *streaming* dedicado. *Real Time Streaming Protocol* (RTSP) é um protocolo de rede projetado para controle de *streaming* de mídia, preferencialmente para áudio e vídeo em tempo real por câmeras para sistemas de monitoramento e segurança (What...,). A porta de rede padrão para acesso é a 554

Figura 11 – Gestão de recursos por threads.



Fonte: (UNIVERSITY, 2021).

mas pode ser mudada por questão de segurança e/ou particular do sistema

2.7.1 Uso do stream via RTSP

Sabendo-se que os dispositivos devem ser conectados a mesma rede para acesso, a primeira informação necessária é o IP do dispositivo para construção do link de acesso.

2.7.1.1 Stream RTSP básico

O modelo básico de acesso a um dispositivo com o protocolo sem proteção de usuário e senha. Um exemplo é mostrado abaixo apenas com o IP e a porta.

rtsp://192.168.1.10:554

2.7.1.2 Stream RTSP com proteção

Para acesso a modelo de dispositivo protegido por usuário e senha pode-se acessar como mostrado no exemplo abaixo:

rtsp://USERNAME:PASSWORD@192.168.1.10:554

2.7.1.3 Configurações ou comandos adicionais de stream

Alguns modelos de câmera possuem acesso a mais de um modo de acesso como modo noturno, modo térmico ou apenas acesso a um canal de stream diferente. Para acesso a cada um dos modos fornecidos é necessário adicionar um caminho específico ao fim do link base conforme mostrado no exemplo abaixo

rtsp://USERNAME:PASSWORD@192.168.1.109/Streamings/channel/101

2.8 Considerações

Neste capítulo foram apresentados conceitos necessários ao desenvolvimento do sistema proposto. As seções apresentadas passam por processamento digital de imagens, reconhecimento de padrões via redes neurais e design de software como programação orientada a objetos, finalizando com protocolo RTSP para conexão das câmeras em rede.

Em posse dos conceitos apresentados neste capítulo, é possível iniciar os estudos para o desenvolvimento do sistema como um todo aplicando os algoritmos de processamento de imagem em tempo real sobre uma estrutura de sistema que permita o paralelismo de processamento mantendo a estabilidade da interface em conjunto as detecções propostas.

3 METODOLOGIA

Nesse capítulo são descritos os tópicos e passos para o desenvolvimento do sistema, considerando o uso das tecnologias já existentes e a junção das mesmas para uma aplicação que as una em sincronia. O uso de bibliotecas para captura de imagem, técnicas processamento digital de imagem e recursos de visão computacional como redes neurais para detecção de padrões são feitos a bastante tempo e cercam-se de uma vasta literatura, contudo o objetivo deste trabalho foca-se tanto no estudo de técnicas de inteligência artificial para reconhecimento de padrões quanto na construção de uma aplicação completa em forma de sistema que una esses conhecimentos para detecção de máscaras e aglomeração através de um conjunto de câmeras gerenciadas pelo mesmo sistema. Dessa forma, foram definidos 4 requisitos que formariam a base da aplicação e que são apresentados nesse capítulo pelas secções subsequentes:

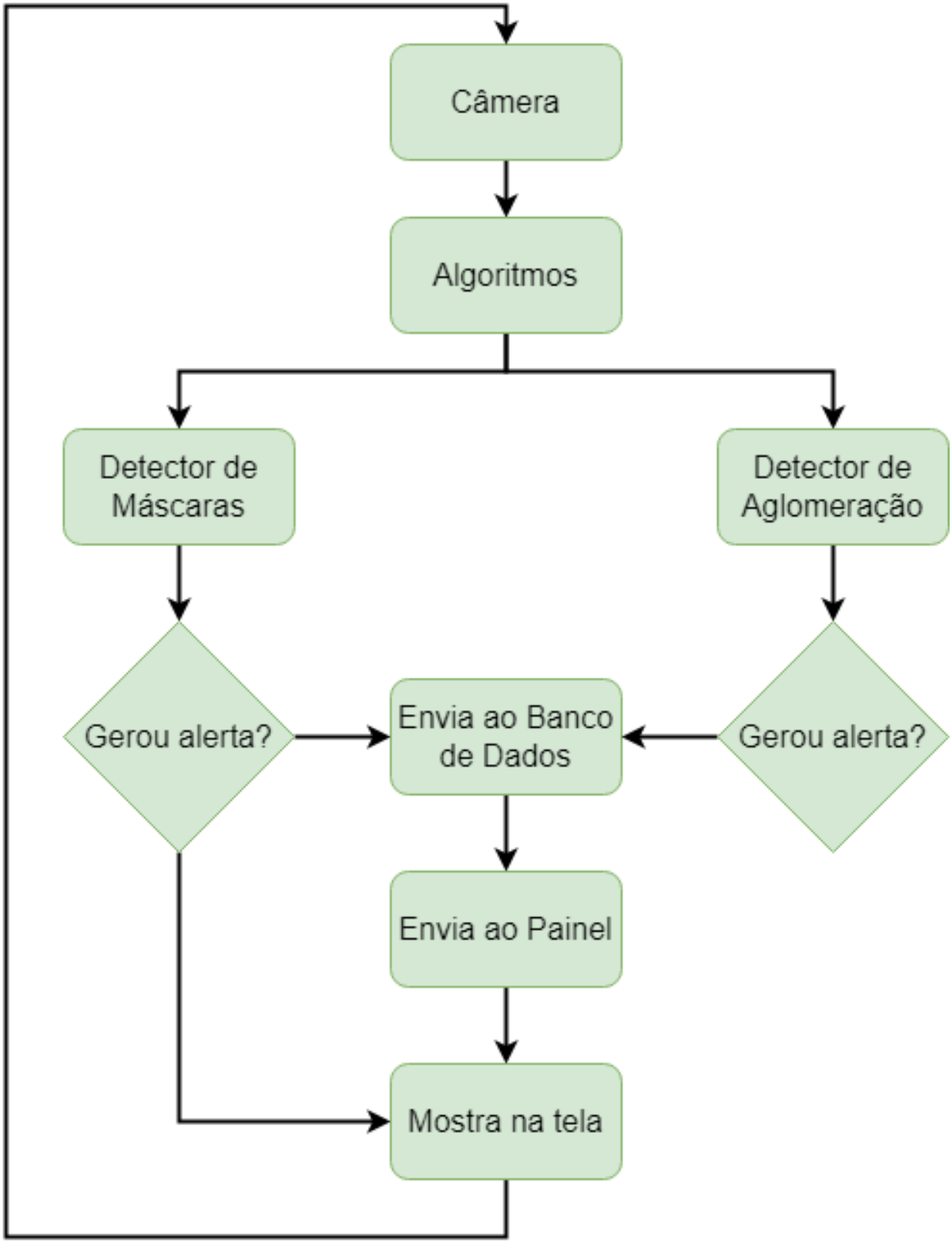
1. Threads, de modo que múltiplas câmeras pudessem trabalhar em conjunto
2. Persistência de dados, onde houvesse armazenamento e retirada de dados de forma prática e segura na máquina usada
3. Tratamento e aplicação de técnicas de reconhecimento de padrões para monitorar as situações propostas
4. Interface gráfica de fácil desenvolvimento e manutenção

3.1 Estrutura do Sistema

O sistema foi estruturado de modo que as câmeras conectadas seriam mostradas na tela e o processamento de imagem e detecção fossem feitos paralelamente em outras threads de modo a preservar a usabilidade da interface, que trabalha sob a *thread* principal de sistema. Para cada câmera conectada ao sistema são criadas duas *threads*: uma para captura e armazenamento de imagens em um *buffer* e uma para processamento dos frames capturados e envio ao banco de dados e aos painéis 3.5.1 em caso de alerta pelos algoritmos de detecção conforme mostrado na figura 12. Cada alerta e frame processado são enviados através de uma conexão de um *signal* específico conectado a um *slot*, seção 2.4, na *thread* correspondente, dessa forma, essas interagem entre si sem comprometer o fluxo de trabalho.

No apêndice B é apresentado o fluxograma completo de inicio do sistema.

Figura 12 – Fluxograma do processamento de imagem.



Fonte: O Autor (2021).

3.2 Algoritmos

Nesta sessão são explicados os algoritmos de processamento de imagem aplicados às câmeras, assim como as modificações feitas em algoritmos de terceiros.

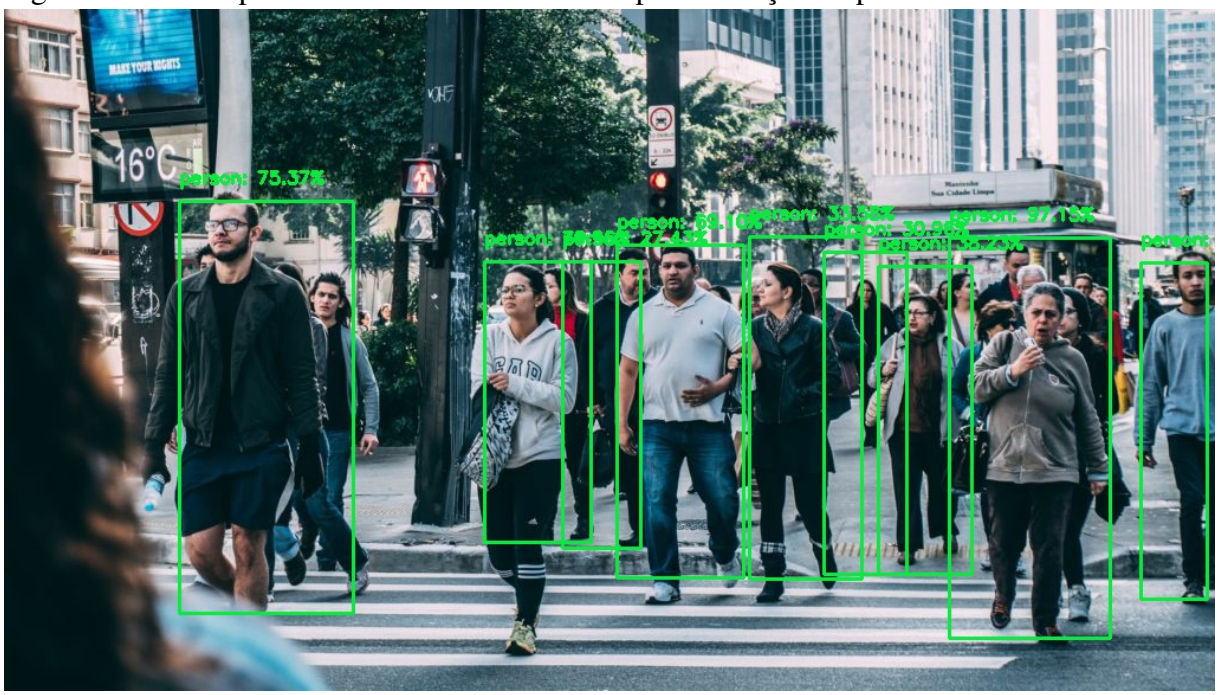
3.2.1 Detecção de Aglomeração

O distanciamento recomendado pelas autoridades era de 2 metros por pessoa e para manter essa medida era necessário que um algoritmo detectasse cada pessoa na foto e calculasse a distância entre as mesmas. Dessa forma, alcançar tal feito o algoritmo é dividido em duas partes: a detecção de pessoas e o cálculo da distância as mesmas.

3.2.1.1 Detecção de Pessoas

A detecção de pessoas foi feita por um modelo de rede neural convolucional profunda pré-treinada, o MobilenetV2-SSD (combinação dos modelos *MobilenetV2* e *Single Shot Multibox Detector*) sob arquitetura *Caffe* muito usada para detecção de objetos, em inglês *Object Detection*, podendo ser direcionada a detecção de pessoas, conforme mostrado na figura 13.

Figura 13 – Exemplo da rede MobilenetV2-SSD para detecção de pessoas.



Fonte: (TheNsBhasin, 2021)

A biblioteca opencv possui suporte para aplicação de modelos de DNN para di-

versas arquiteturas como *Caffe* usada neste trabalho em conjunto com um modelo do tipo MobilenetV2-SSD. Inicialmente, as redes são carregadas em cada thread aplicando o método *cv2.dnn.readNetFromCaffe(protoPath, modelPath)* que recebe como argumento o arquivo proto com as informações das camadas, em inglês *layers*, com seus parâmetros internos e o modelo, *.caffemodelo*, respectivamente.

Na aplicação do processamento de imagem são seguidos os seguintes passos:

1. O frame capturado passa pelo método *cv2.dnn.blobFromImage*
2. O arquivo .blob gerado é passado como parâmetro na rede
3. É aplicado o método *forward* na rede neural.

A método *forward* retorna um *array* contendo um vetor de proporções em função as dimensões *width* e *height* da imagem, por exemplo 640x480, associado a uma medida de probabilidade de detecção. Para cada detecção onde é verificado confiança associada a detecção maior que 50%, é feito o produto do vetor de proporções pelo vetor de dimensões da imagem, gerando assim um par de coordenadas que define o ROI na imagem como mostrado na equação 3.1. Essas coordenadas são armazenadas para ser enviadas para cálculo de distância entre as pessoas localizadas.

$$\begin{pmatrix} startXSSD \\ startYSSD \\ endXSSD \\ endYSSD \end{pmatrix} = \begin{pmatrix} p_{wi} \\ p_{hi} \\ p_{wf} \\ p_{hf} \end{pmatrix} . \begin{pmatrix} W \\ H \\ W \\ H \end{pmatrix} . \tag{3.1}$$

3.2.1.2 Cálculo de distância entre pessoas

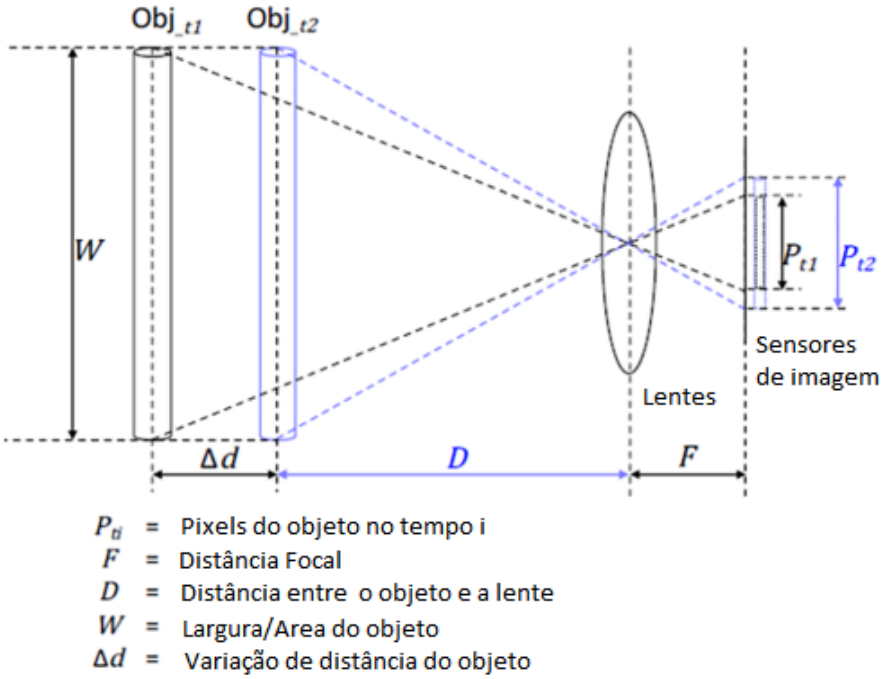
Para cada indivíduo detectado no frame é gerado um retângulo que delimita o espaço 2D ocupado pelo mesmo e em seguida calculado o ponto médio, em inglês *mid point*, desse quadrado dividindo a largura, em inglês *width* e a altura, em inglês *height*, ambas por 2 conforme a equação 3.2.

$$Midpoint = (\frac{W}{2}, \frac{H}{2}) \tag{3.2}$$

Após a detecção de uma pessoa e o cálculo do ponto médio da mesma na imagem, é calculada a distância da mesma à câmera. Uma imagem chega às lentes da câmera conforme

mostrado na imagem 14. Por meio de semelhança triangular e em posse constante de distância focal(constante obtida empiricamente ou fornecida pelo fabricante), do *width* do objeto(obtido pelo *frame* explicado anteriormente) e pelo número de pixels(obtido pela imagem convertida em um *array* pela biblioteca opencv) é calculada a distância entre o objeto e a câmera.

Figura 14 – Calculo de distância de um objeto a uma câmera.



Fonte: Adaptado de (D HAN K; Y., 2018)

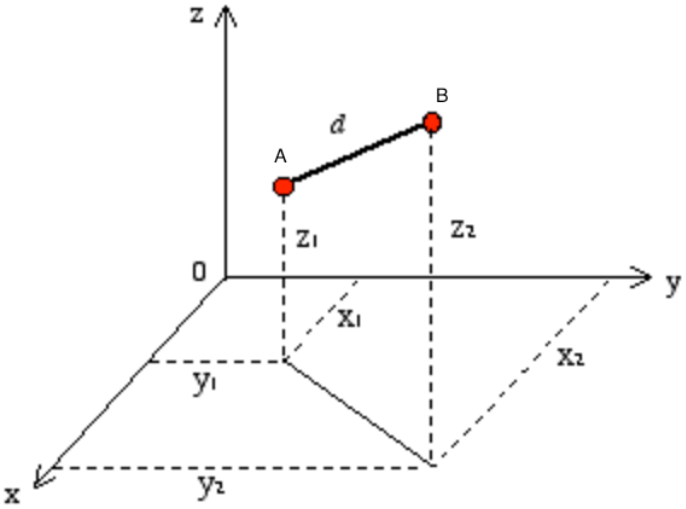
$$\frac{D}{W} = \frac{F}{P} \Longleftrightarrow D = \frac{(F.W)}{P} \tag{3.3}$$

Em posse do ponto médio e da profundidade de distância da câmera calculada de cada pessoa detectada, é calculada a distância de cada indivíduo entre si através da equação 3.4.

$$D_{obj} = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \tag{3.4}$$

Se a distância entre os indivíduos for menor que 2 metros é detectada aglomeração. Quando a aglomeração é detectada, um alerta via código é acionado enviando os dados correspondentes (nome da câmera, data/hora do ocorrido e título do evento) a um método de processamento, armazenados no banco de dados e então alocados no painel apresentado na seção 3.5.1.

Figura 15 – Calculo de distância euclidiana em 3 dimensões.



Fonte: O Autor(2021)

3.2.2 Detecção do uso de máscaras

A detecção do uso de máscaras de proteção individual já é um processo mais delicado visto que deve ser projetada uma rede neural de aprendizagem profunda, em inglês *deep learning*, que seja treinada sob um grande banco de dados com fotos de pessoas com e sem máscaras em diversas posições para uma predição de acurácia razoável para um vídeo em tempo real. A solução desse problema pode ser dividida em duas partes: a detecção de faces na imagem e a detecção da presença de máscaras.

3.2.2.1 Detecção de Faces

Para a detecção de faces foi usada uma biblioteca dentro da API da Google Mediapipe chamada *Face Detection* (GOOGLE, 2021). Esta é uma biblioteca de detecção de faces em tempo real que gera 6 *landmarks* e um quadrado ao redor da face, oferecendo também suporte a detecção de múltiplas faces no mesmo *frame*. É baseado no *BlazeFace* (BAZAREVSKY *et al.*, 2019), um detector de faces leve e de boa performance adaptado para inferência de GPU móvel. O detector usa uma leve rede de extração de recursos inspirada, mas distinto na arquitetura MobileNetV1/V2.(GOOGLE, 2021)

Dessa forma, para usar a biblioteca primeiro definimos uma instância do método de detecção e passamos por parâmetro o modelo selecionado (0 para alcance de até 2 metros e 1 para alcance a partir de 5 metros) e grau mínimo de confiança aplicado às detecções (entre 0 e 1), conforme apresentado no código abaixo.

Código-fonte 4 – Implementação da biblioteca Mediapipe FaceDetection

```
mpFaceDetection = mp.solutions.face_detection
faceDetection = mpFaceDetection.FaceDetection(
    model_selection=1, min_detection_confidence=0.5)
```

Em seguida, aplicamos o método *process* ao detector que retorna os resultados da detecção como mostrado no código abaixo. E por meio do método *detections* obtém-se uma lista dos objetos de detecções pelo método contendo as coordenadas relativas ao ROI da face detectada e uma métrica de confiança associada a detecção como mostrado no apêndice A.

Código-fonte 5 – Obtendo detecções de face

```
results = faceDetection.process(cv2.cvtColor(frame, cv2
    .COLOR_BGR2RGB)
if results.detections:
     # Espaco para processamento das deteccoes obtidas
```

A partir das detecções de face com confiabilidade assegurada que podemos prosseguir e passar ao método de verificação do uso de máscaras.

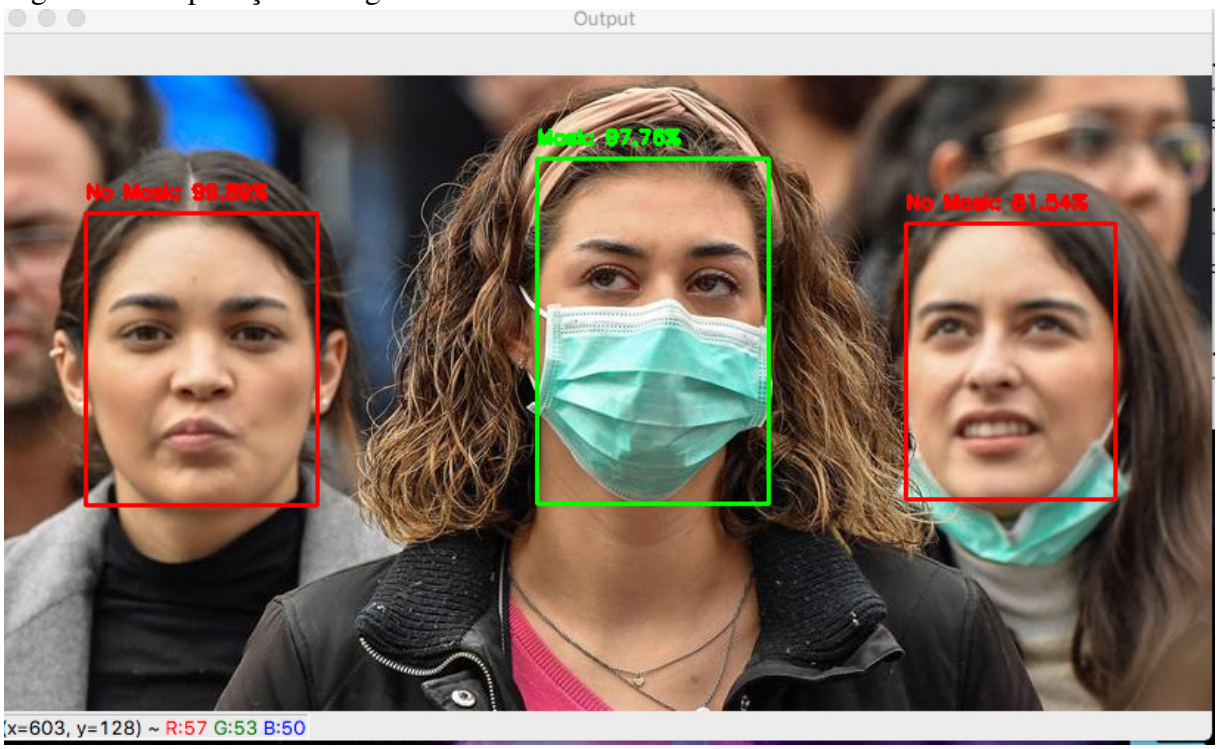
3.2.2.2 Detecção de máscaras

A priori foi usado o algoritmo Face-Mask-Detection (DEB, 2021) para detecção de faces e posterior detecção de máscaras, ambas através de uma rede neural de aprendizagem profunda, em inglês *deep learning*, com modelos projetados usando Keras/TensorFlow, sob arquitetura MobilenetV2. O uso da MobilenetV2 é bastante útil para a expansão da aplicação a sistemas embarcados, devido a sua eficiência computacional. O resultado da aplicação do algoritmo a uma imagem estática é apresentado na figura 16.

No entanto, conforme foi apresentado na seção anterior, foi avaliado um mecanismo mais eficiente de detecção de faces de modo a suplantar a necessidade de outra rede sob a mesma arquitetura como feito por (DEB, 2021) e fazendo uso da biblioteca mediapipe como descrito na seção anterior. Para a detecção de máscaras manteve-se o uso da rede *MaskNet*.

Dessa forma, foi aplicada a rede neural conforme descrito nos passos abaixo.

Figura 16 – Aplicação do algoritmo Face-Mask-Detection.



Fonte: (DEB, 2021)

São definidas duas listas com dados paralelos, uma com as coordenadas de cada face na imagem original e outra com as imagens recortadas de cada rosto detectado. Em seguida, a lista de faces passa 3 processos de adequação à rede construída sob a arquitetura Keras:

1. É feita a conversão de cor de todas as imagens de BGR para RGB
2. É feito o redimensionamento de todas as imagens para 224x224 pixels
3. As imagens são convertidas para um array numpy por um método próprio da biblioteca Keras
4. A lista passa por um método de pré-processamento de arrays numpy, ou tensores, que codifica um lote de imagens para predição.

Após esses processos de preparação, a lista de faces é enviada ao método de predição que retorna uma lista contendo uma tupla de dois elementos, porcentagem de máscara e porcentagem de sem máscara, referente a cada imagem de face enviada a rede. Em seguida, com base na resposta é visto se a predição indicou a falta de máscara e, nesse caso, é definida a cor vermelha para o ROI a ser desenhado na imagem e construído um dicionário com as informações correspondentes para envio ao painel de alertas, seção 3.5.1, pelo *signal* definido previamente. Por fim, o método descrito acima finaliza desenhando o ROI com a cor correspondente a detecção em sob cada face na imagem e retorna a imagem modificada para ser enviada a tela principal.

3.3 Banco de dados

Um dos requisitos definidos ao sistema é a persistência de dados aplicada visto que, para um sistema de monitoramento, é necessário o armazenamento das ocorrências e eventos de alerta tanto para uso contínuo quanto análise posterior, caso necessário. Dessa forma, foi feito o uso de um banco de dados relacional MySql. Esse banco de dados foi escolhido pela compatibilidade com a linguagem, facilidade de inserção e recuperação dos dados.
O banco configurado possui quatro tabelas descritas abaixo:

- *cameras*: Armazena as informações de cada câmera cadastrada no sistema a ser acessada pelo sistema.
- *credentials*: Armazena as credenciais de acesso ao sistema.
- *crowd_detection*: Armazena os alertas de detecção de pessoas em aglomeração preditos pelo algoritmo descrito na seção 3.2.1
- *mask_detection*: Armazena os alertas de detecção de pessoas sem máscara preditos pelo algoritmo descrito na seção 3.2.2

O conjunto de tabelas do banco de dados é apresentado na figura 17.

Figura 17 – Tabelas do Banco de Dados.

Table	Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra
cameras	cam_id	int		NO			select,insert,update,references	auto_increment
cameras	cam_ip	varchar(45)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
cameras	cam_maxtemp	int	0	YES			select,insert,update,references	
cameras	cam_name	varchar(30)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
cameras	cam_password	varchar(30)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
cameras	cam_path	varchar(255)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
cameras	cam_port	varchar(45)	554	NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
cameras	cam_type	enum('Térmica','Normal')		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
cameras	cam_user	varchar(30)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
credentials	email	varchar(100)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
credentials	id	int		NO			select,insert,update,references	auto_increment
credentials	password	varchar(30)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
credentials	perfil	enum('Administrador','Cliente')		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
credentials	user	varchar(100)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
crowd_detection	cam_name	varchar(45)		YES	utf8mb4	utf8mb4_genera...	select,insert,update,references	
crowd_detection	date	varchar(45)	00.00.00	YES	utf8mb4	utf8mb4_genera...	select,insert,update,references	
crowd_detection	description	varchar(255)		YES	utf8mb4	utf8mb4_genera...	select,insert,update,references	
crowd_detection	event	enum('FEBRE','AGLOMERAÇÃO','MÁSCARA')		YES	utf8mb4	utf8mb4_genera...	select,insert,update,references	
crowd_detection	id	int		NO			select,insert,update,references	auto_increment
crowd_detection	time	varchar(45)	00.00.00	YES	utf8mb4	utf8mb4_genera...	select,insert,update,references	
mask_detection	cam_name	varchar(100)		YES	utf8mb4	utf8mb4_genera...	select,insert,update,references	
mask_detection	date	varchar(12)		YES	utf8mb4	utf8mb4_genera...	select,insert,update,references	
mask_detection	mask_id	int		NO			select,insert,update,references	auto_increment
mask_detection	photo	longblob		YES			select,insert,update,references	
mask_detection	temperature	float		YES			select,insert,update,references	
mask_detection	time	varchar(10)		YES	utf8mb4	utf8mb4_genera...	select,insert,update,references	

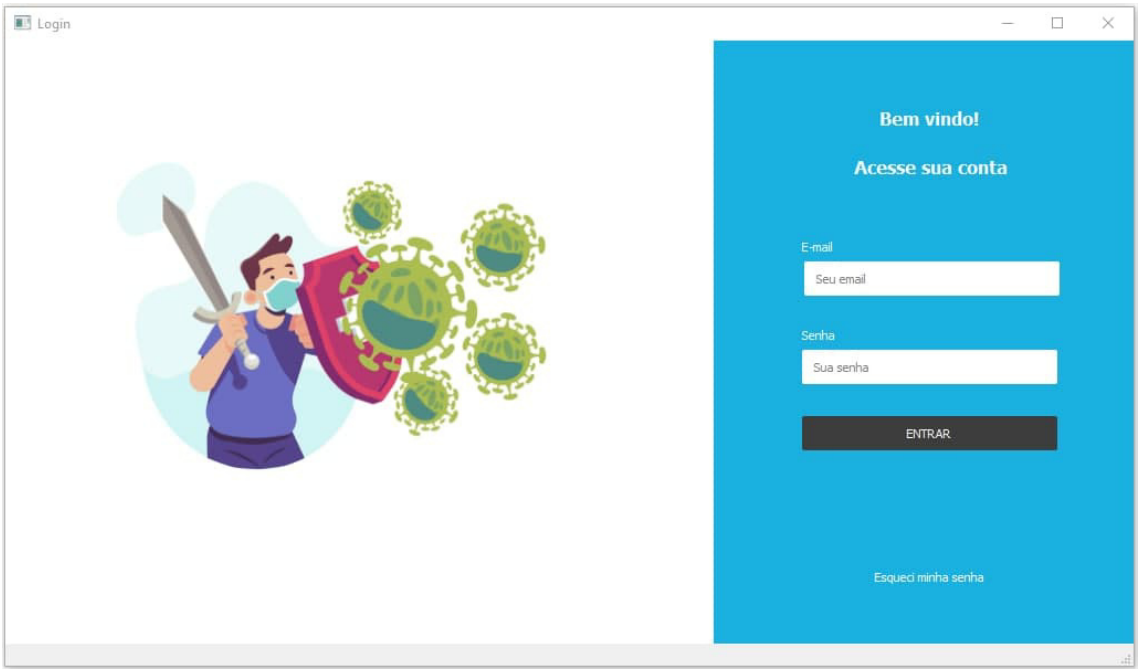
Fonte: O Autor (2021).

3.4 Tela de Login

O sistema começa pelo seu acesso através da tela de login apresentada na figura 18. O acesso é feito por meio de credenciais de *email* e de senha do usuário cadastrado.

As credenciais de cada usuário são armazenadas na tabela ***credentials*** com os dados de nome do usuário, email, senha e perfil de acesso, conforme mostrado na figura 26.

Figura 18 – Tela de Login.



Fonte: O Autor (2021).

Novas credenciais de acesso podem ser adicionadas, assim como outras existentes podem ser editadas, na tela de cadastro de novo usuário, como mostrado na seção 3.7, desde que se possua perfil de acesso de Administrador.

3.5 Tela Principal

Após a autenticação de login, o usuário é levado a tela principal do sistema.

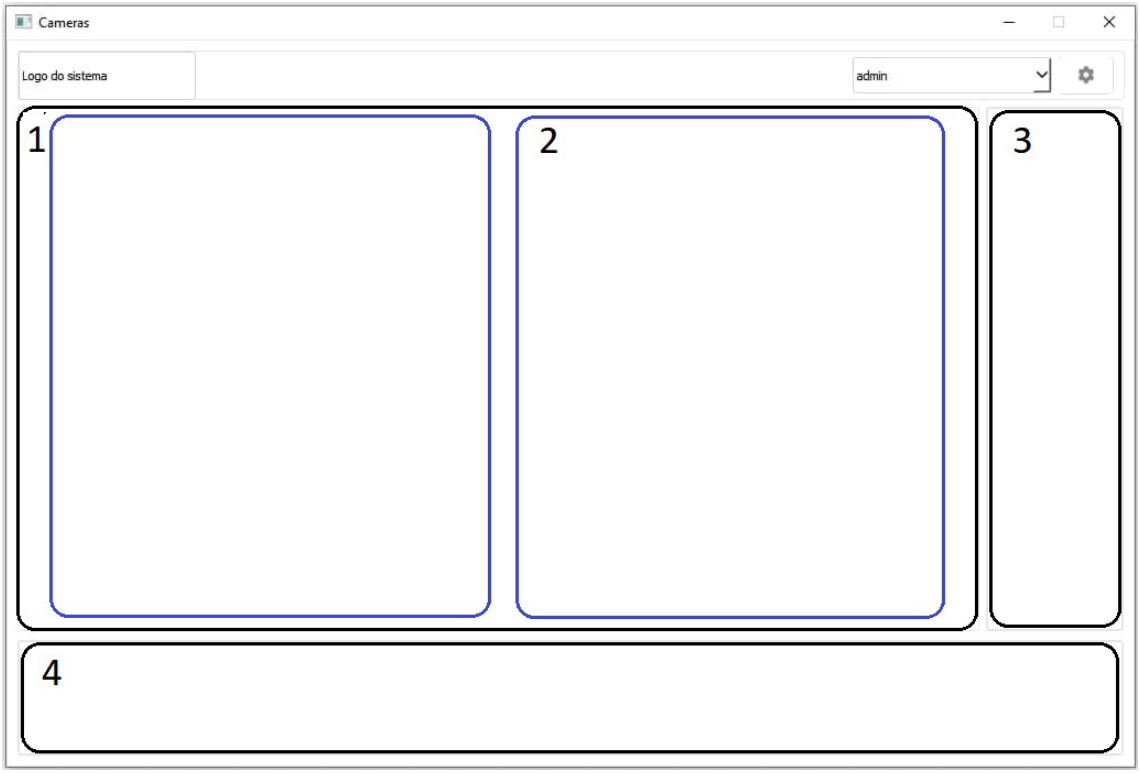
Após o acesso ao sistema são requisitadas ao banco de dados local as câmeras cadastradas e alertas anteriores. É verificado se há alertas anteriores e, em caso positivo, estes são inseridos nos respectivos painéis. Em seguida, as câmeras são verificadas, conectadas e enviadas ao layout das câmeras na tela apresentados abaixo.

A tela principal do sistema apresenta as câmeras monitoradas e os painéis de detecção de alertas enviados pelos algoritmos 3.2. Conforme visto na figura 19, é apresentado o modelo limpo da tela com as delimitações referentes a localização das câmeras e dos painéis. No primeiro bloco, mostrado no centro da tela, as câmeras são alocadas dinamicamente a medida que são conectadas. No segundo bloco é mostrado o espaço, em caso de duas câmeras usadas, por uma das câmeras. No terceiro e quarto bloco encontram-se os painéis de detecções de ausência de máscaras e presença de aglomerações, respectivamente.

No canto superior direito da tela pode ser visto uma opção de seleção, com o nome

"*admin*"em foco, e um botão com ícone de engrenagem. No campo de seleção se encontra o nível de acesso do usuário que, em caso de nível administrador, teria acesso a todas as configurações disponíveis e, no caso de nível cliente, teria acesso apenas a visualização do monitoramento das câmeras e painéis. O botão ao lado leva a tela de configurações do sistema.

Figura 19 – Tela Principal.

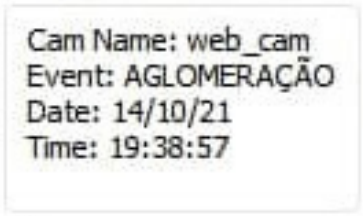


Fonte: O Autor (2021).

3.5.1 Painéis de Alerta

Os painéis de alerta de máscara e aglomeração podem ser vistos na lateral direita e abaixo do layout de câmeras, respectivamente. Estes são inicialmente criados para a tela principal, figura 19, e em seguida herdados na mesma localização para a tela de configurações, figuras 22 e 23. A medida que os eventos vão acontecendo, ambos os painéis são preenchidos ao mesmo tempo. Os algoritmos, 3.2, são processados em *threads* separadas da principal, logo os eventos são enviados em *background* para os painéis independente de qual tela esteja em foco. As figuras 20 e 21 mostram os *widgets* criados e alocados nos respectivos painéis para eventos de aglomeração e detecção de ausência de máscaras. É possível ver o uso dos painéis nos casos de testes apresentados no próximo capítulo.

Figura 20 – *Widget* de evento de aglomeração detectada



Fonte: O Autor (2021).

Figura 21 – *Widget* de evento de ausência de mascara detectado

Fonte: O Autor (2021).

3.6 Tela de Configurações

Nas figuras 22 e 23 encontra-se a tela de configurações do sistema onde usuários e câmeras cadastradas podem ser vistos, editados ou excluídos, tal como o cadastro de novos. Veja que os painéis de alerta ainda são mostrados de modo que mesmo em um instante de configuração ou edição do sistema, o monitoramento continua.

Na tela de configurações são apresentados os usuários, figura 23, e as câmeras cadastradas, figura 22, com base no foco representado pelo botão de câmeras ou usuários na cor azul. Por padrão as câmeras são mostradas primeiro ao entrar na tela. Para cada câmera ou usuário cadastrado é possível editar ou excluir, desde que se tenha os devidos privilégios, através dos botões "Editar"e "Excluir", respectivamente.

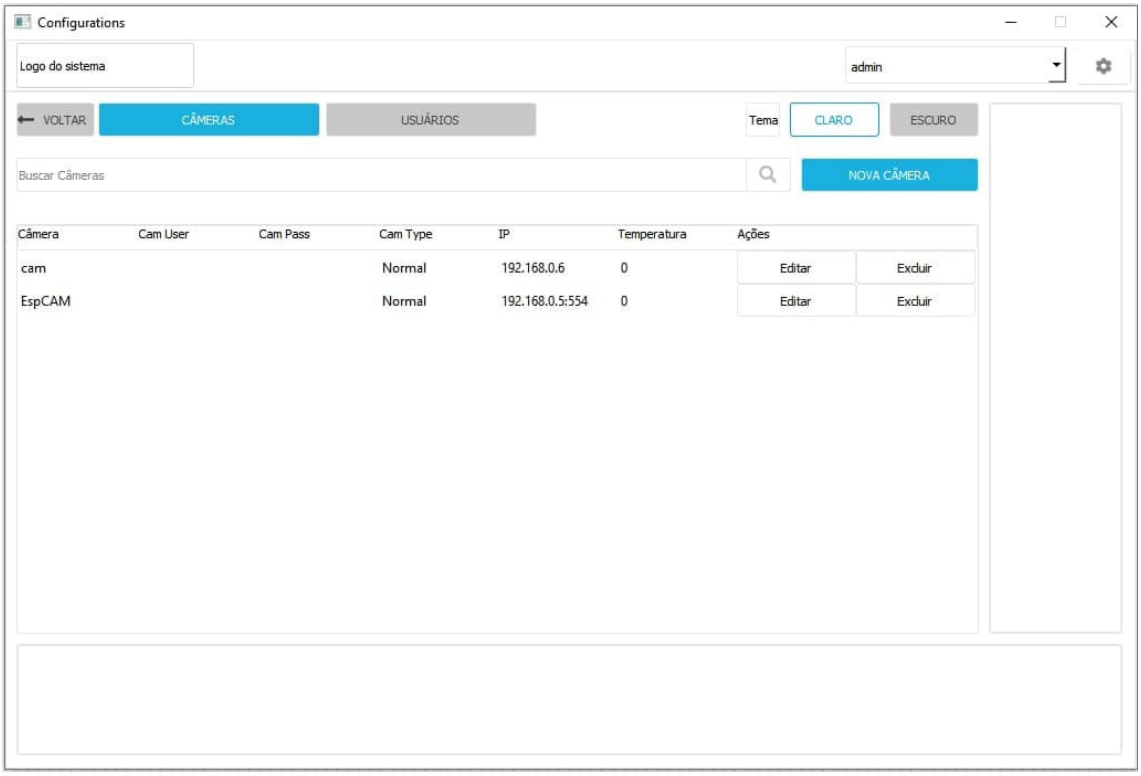
Para o cadastro de novas câmeras deve-se clicar no botão *"Nova Câmera"* que levará a tela de cadastro, figura 27. Para o cadastro de novos usuários deve-se clicar no botão *"Novo Usuário"* que levará a tela de cadastro, figura 24.

3.7 Tela de Cadastro de Usuário

Na figura 24 é apresentada a tela de cadastro de novos usuários. A mesma também é usada para edição de usuários como mostrado na figura 25.

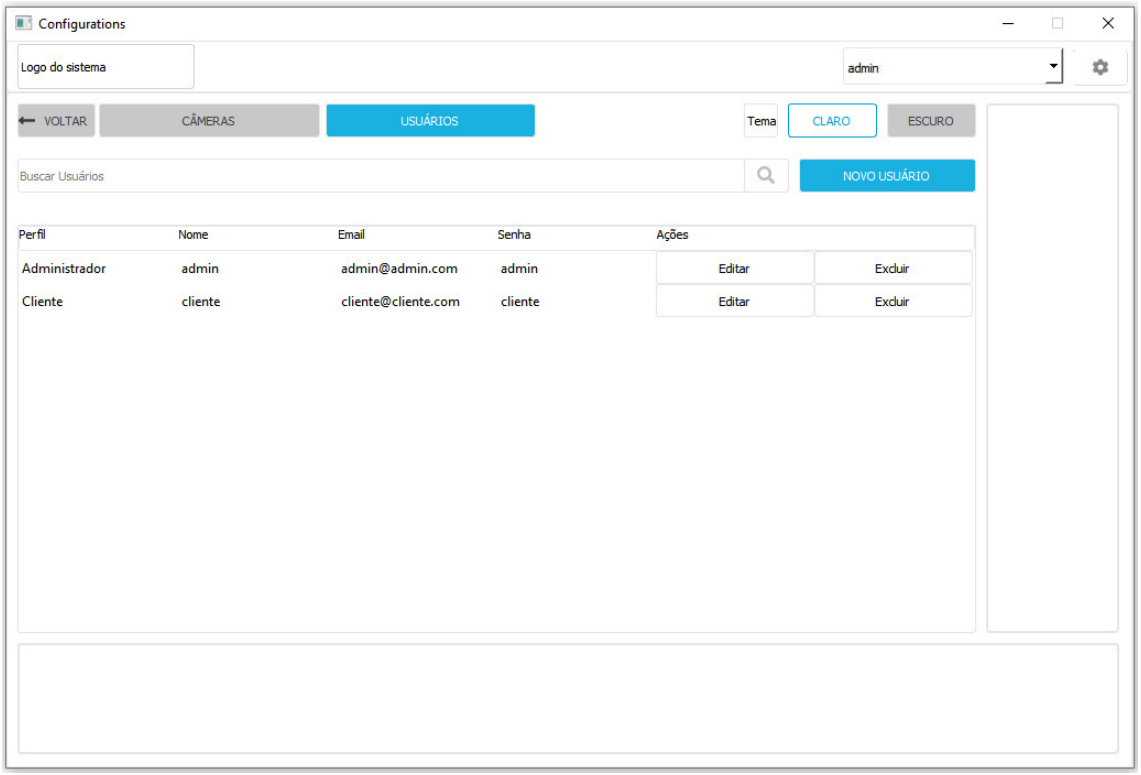
Após cadastrado de um usuário, é gerado um processo separado onde as informações do usuário são verificadas e enviadas ao banco de dados, seção 3.3. Os usuários cadastrados são apresentados na figura 26.

Figura 22 – Tela de Configurações com foco nas câmeras.



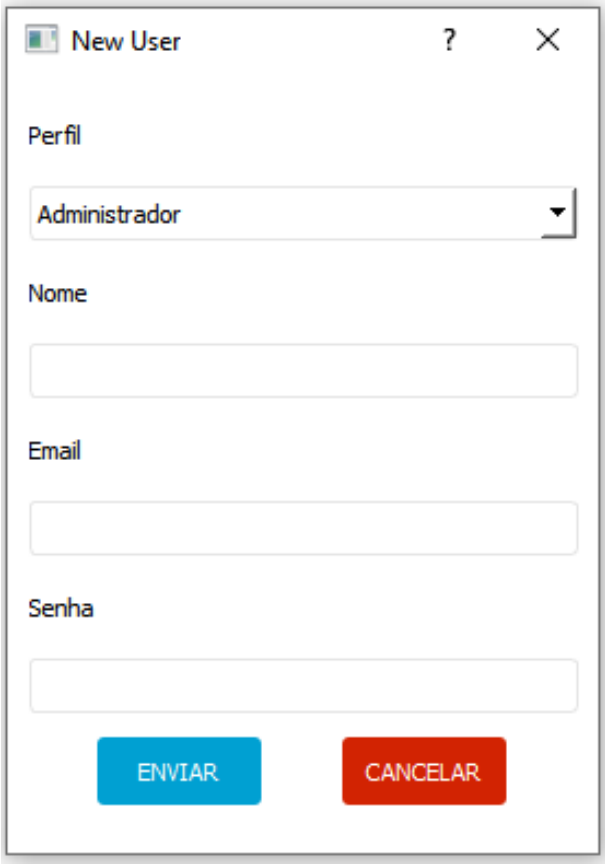
Fonte: O Autor (2021).

Figura 23 – Tela de Configurações com foco nos usuários.



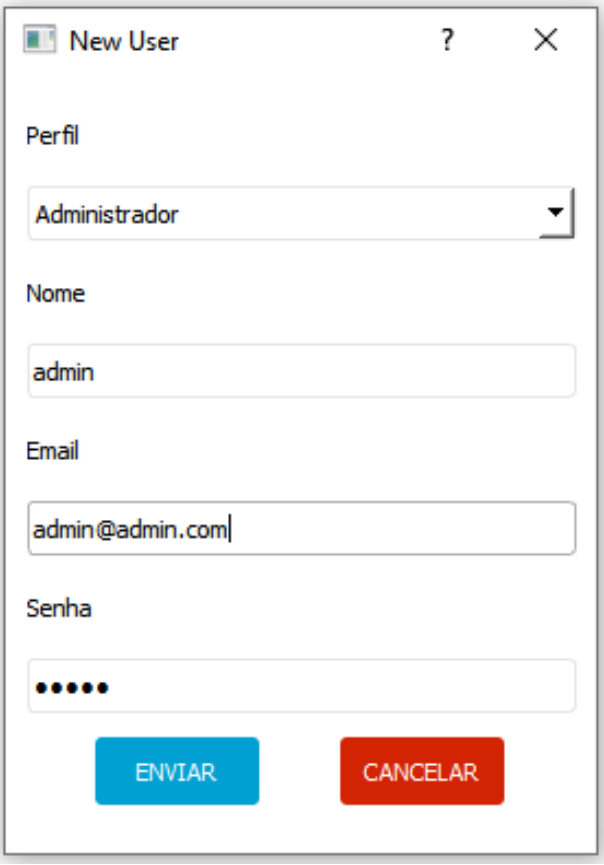
Fonte: O Autor (2021).

Figura 24 – Tela de Cadastro de Usuário



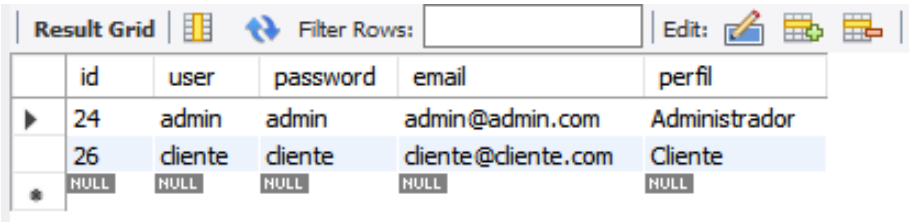
Fonte: O Autor (2021).

Figura 25 – Tela de Edição de Usuário.



Fonte: O Autor (2021).

Figura 26 – Usuários cadastrados no banco de dados.



Fonte: O Autor (2021).

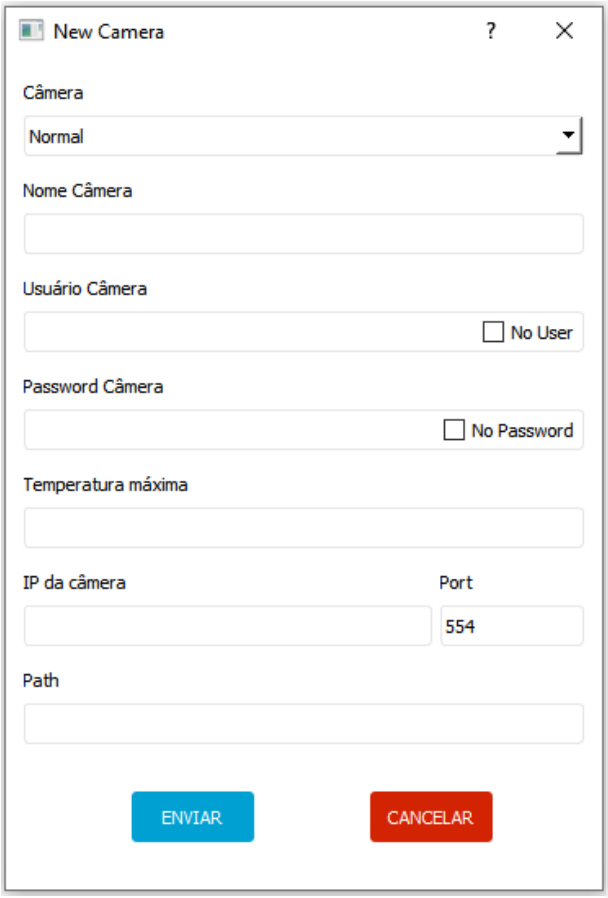
3.8 Tela de Cadastro de Câmera

Na figura 27 é mostrada a tela de cadastro de uma nova câmera. A mesma também é usada para edição dos dados de uma câmera como mostrado na figura 22, onde os dados da câmera são preenchidos nas lacunas em branco para modificação.

Na figura 28 vemos assinalados dois *checkbox* sinalizando que a câmera cadastrada não possui usuário e senha, uma prática não comum em sistemas reais mas útil para testes com câmeras mais simples e para fins de validação.

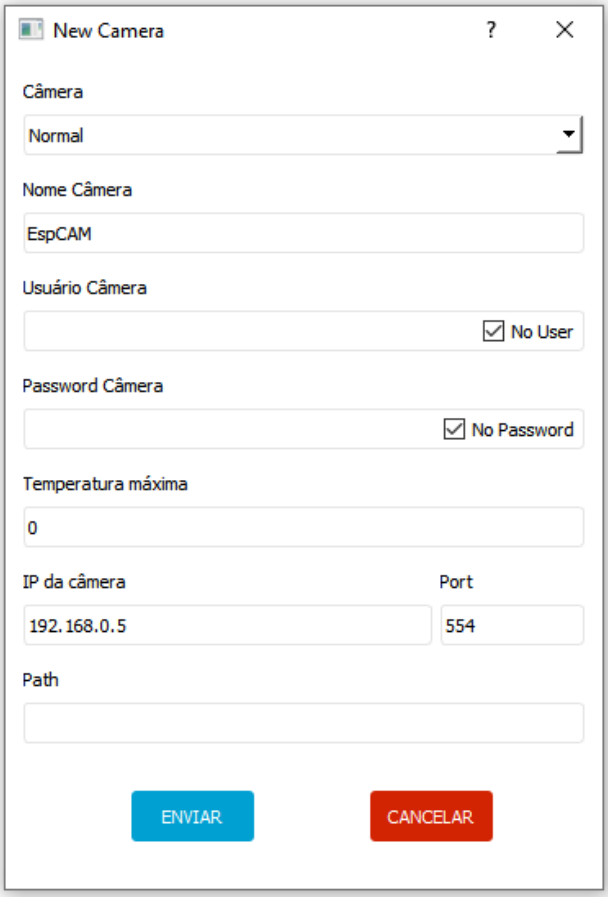
Após cadastrada uma câmera e enviada pelo botão *"Enviar"*, é gerado um processo

Figura 27 – Tela de Cadastro de Câmera



Fonte: O Autor (2021).

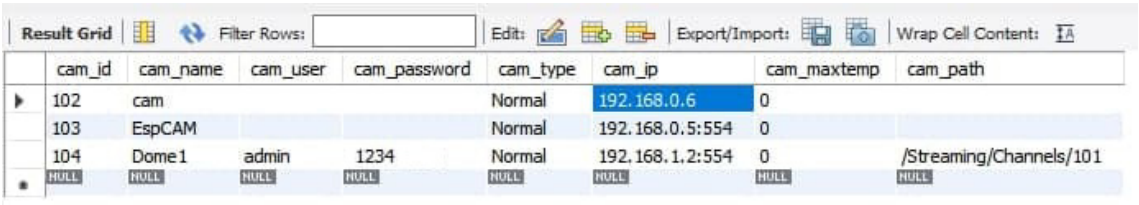
Figura 28 – Tela de Edição de câmera.



Fonte: O Autor (2021).

separado onde as informações da câmera são verificadas e enviadas ao banco de dados na tabela *cameras*, seção 3.3. As câmeras cadastradas são apresentadas na figura 29.

Figura 29 – Câmeras cadastradas no banco de dados.



cam_id	cam_name	cam_user	cam_password	cam_type	cam_ip	cam_maxtemp	cam_path
102	cam			Normal	192.168.0.6	0	
103	EspCAM			Normal	192.168.0.5:554	0	
104	Dome1	admin	1234	Normal	192.168.1.2:554	0	/Streaming/Channels/101
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: O Autor (2021).

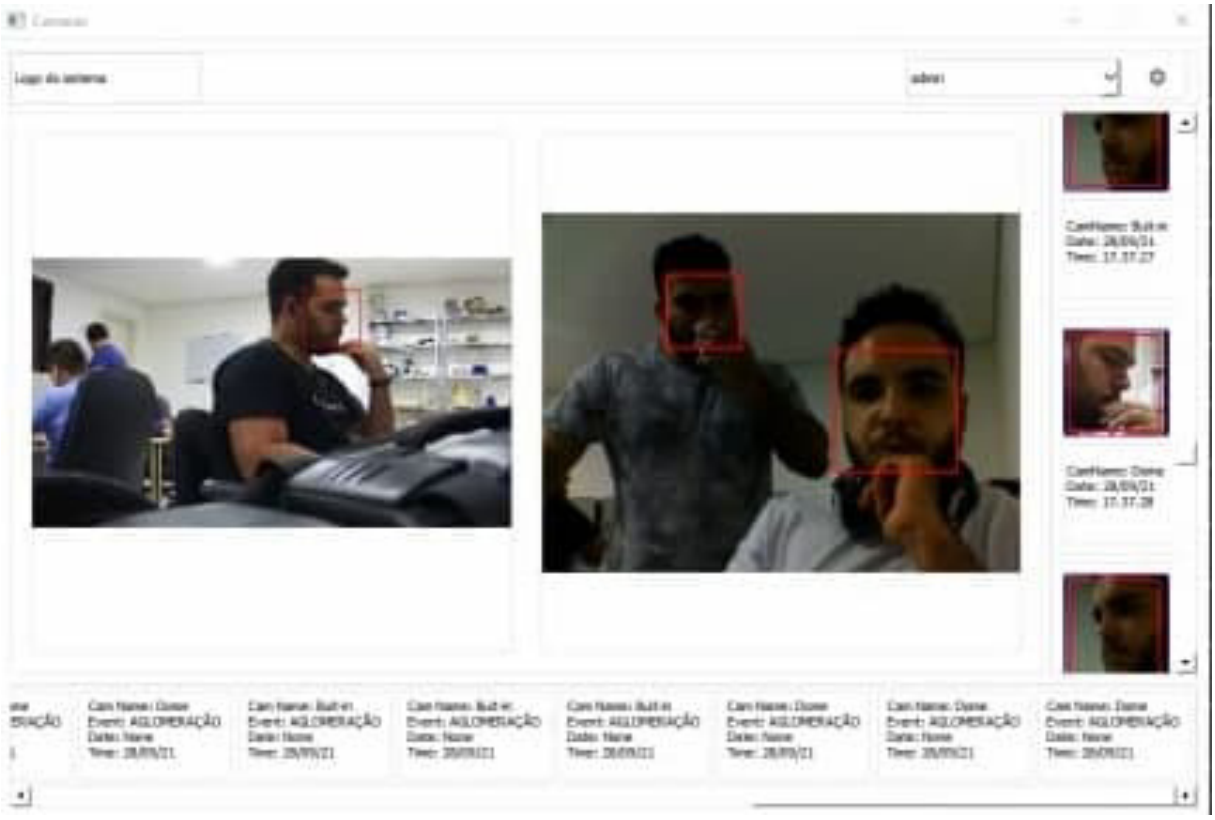
4 RESULTADOS

Esse capítulo apresenta e descreve casos de teste onde os algoritmos, seção 3.2, são aplicados juntos e separadamente nas câmeras para verificação de performance e análise de cada caso considerando condições que podem auxiliar ou piorar o processo de monitoramento. Ao final são descritas algumas considerações dos testes assim como recomendações para melhor uso.

4.1 1º Teste: aplicação do algoritmo de detecção de máscaras

No primeiro teste foi aplicado apenas o algoritmo de detecção de máscaras, 3.2.2, em duas câmeras como mostrado na figura 31. Observa-se que 3 indivíduos são detectados e ambos estão sem máscara, um na câmera da esquerda e dois na câmera da esquerda. Como os 3 indivíduos estão sem máscaras, o ROI delimitado no espaço de suas faces fica na cor vermelha, denotando alerta, e suas fotos são encaminhadas para o painel lateral direito seguidos do nome da câmera, data e hora do evento. São mostradas também detecções de testes anteriores no painel, podendo ser vistas através da rolagem do campo.

Figura 30 – Teste com duas câmeras para detecção de máscara.



Fonte: O Autor (2021).

Observa-se que há mais dois indivíduos na câmera à esquerda os quais não tiveram suas faces detectadas pelo algoritmo dado a pouca exposição da parte frontal do rosto a câmera.

4.2 2º Teste: aplicação dos dois algoritmos

No terceiro teste são apresentados casos de testes com aplicação dos dois algoritmos, 3.2.2 e 3.2.1.

Na figura 31 o algoritmo de aglomeração detecta a presença de dois indivíduos, calcula a distância entre eles e por ser maior que 2 metros, delimita um ROI no entorno desses na cor verde. Além disso, é detectado um dos indivíduos sem máscara e por isso o ROI que delimita seu rosto está na cor vermelha e um alerta foi enviado ao banco e ao painel lateral com as informações correspondentes.

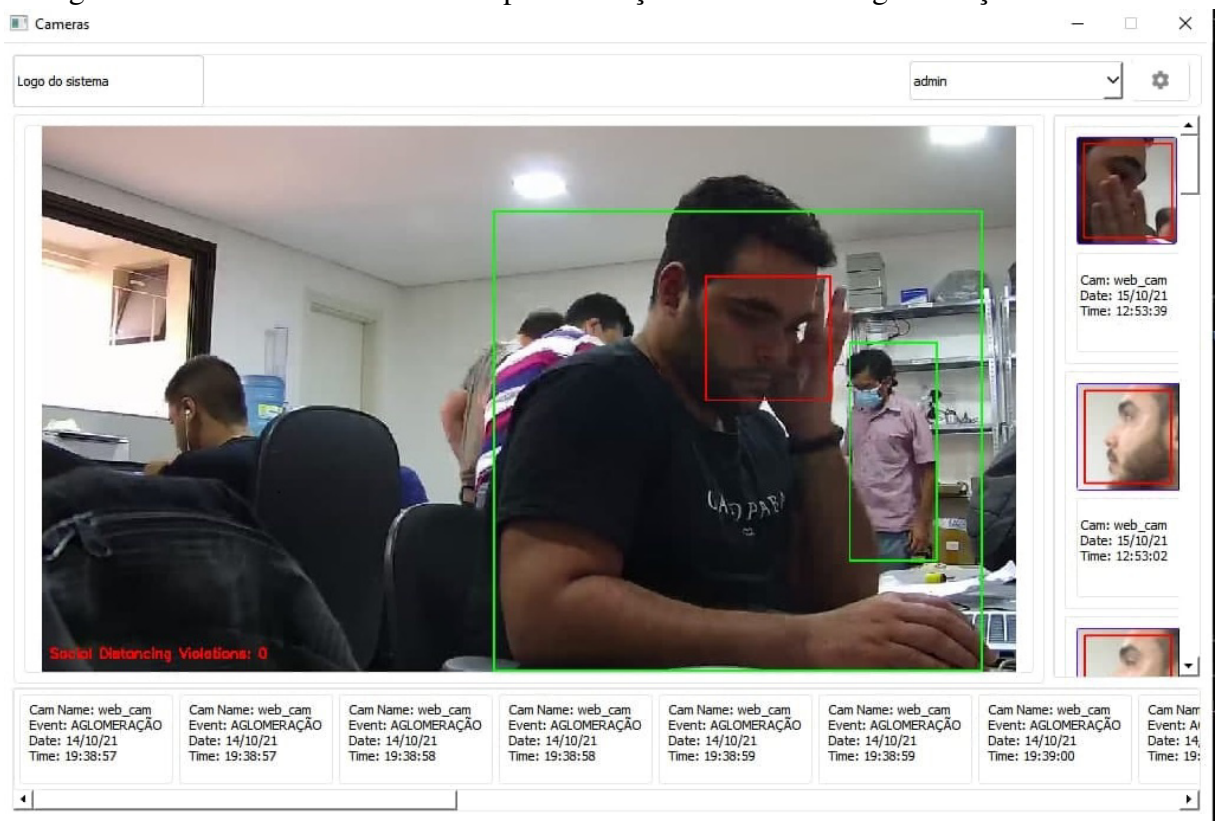
Pode-se observar que há mais dois 3 indivíduos no ambiente, um de camisa listrada, um de camisa verde e um sentado de camisa preta. Para os dois indivíduos em pé, o algoritmo de aglomeração não os detecta pois o espaço que os mesmos ocupam na foto é pequeno e gera confiança inferior a 50% no processo de detecção. Já o indivíduo de sentado de camisa preta não é detectado justamente pelo contraste entre sua camisa preta, a cadeira preta e a mochila preta em frente a câmera.

Isso mostra que se um indivíduo na imagem estiver entre elementos de cor semelhante não houver espaço suficiente e iluminação para distingui-lo, há uma probabilidade que o algoritmo possa errar.

Já na figura 32 o algoritmo de aglomeração detecta a presença de 3 indivíduos, todos no ambiente, calcula a distância entre eles e por ser menor que 2 metros, delimita os ROIs no entorno dos indivíduos em cor vermelha. Além disso, é detectado que um indivíduo sem máscara ao qual o algoritmo de máscaras delimita um ROI de cor vermelha e um alerta do evento foi enviado ao banco e ao painel lateral. Vale a pena notar que, em comparação a figura 31, um indivíduo está na mesma posição sentado mas com uma camisa azul e em ambiente melhor iluminado de modo que o algoritmo detecta sua presença assim como a dos outros e aglomeração ocorrida.

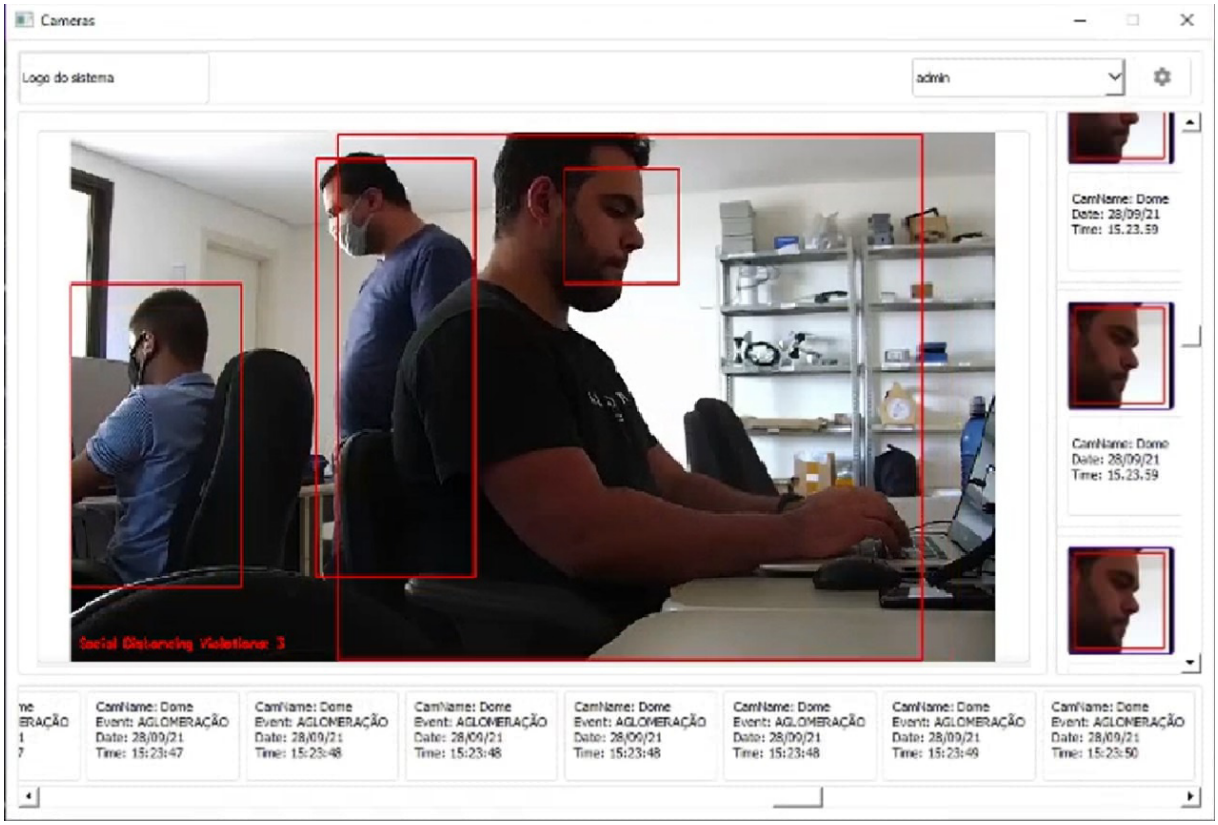
Um problema observado e ainda não resolvido cerca-se na filtragem dos eventos para preenchimento dos painéis de alerta, 3.5.1, de modo que se uma situação detectada pelos algoritmos persiste por um tempo, até curto devida a alta taxa de captura por alguns dispositivos, há repetição de alertas enviados que pode poluir a visualização dos painéis.

Figura 31 – Teste com uma câmera para detecção de máscara e aglomeração.



Fonte: O Autor (2021).

Figura 32 – Teste com uma câmera para detecção de máscara e aglomeração.



Fonte: O Autor (2021).

4.3 Considerações

A acurácia obtida pelos testes com ambos os algoritmos se mostrou boa, mesmo com eventuais condições de falhas apresentados nos testes. Como poderia se esperar, ao ativar o processamento as câmeras tiveram taxa de captura de *frames* por segundo(*fps*) reduzida em até 50%, mas mantiveram um bom fluxo de processamento.

Por fim, alguns cuidados podem ser tomados para que possam ser feitas detecções e predições de maior garantia, evitando falsos positivos ou falsos negativos. Dentre as recomendações para melhor funcionamento dos algoritmos estão:

1. Garantir que as câmeras sejam instaladas há uma distância segura de modo que a presença de um indivíduo mais próximo não se oponha a outros e permita uma omissão de casos
2. Garantir que as câmeras sejam instaladas a uma altura acima 2 metros pode fornecer um alcance de visão maior.
3. Garantir uma boa iluminação do ambiente, seja ela natural ou não, para melhor o processamento das imagens.

5 CONCLUSÕES E TRABALHOS FUTUROS

O trabalho apresentado conclui com sucesso o desenvolvimento de um sistema de gerenciamento e monitoramento de câmeras aplicando processamento digital de imagem e métodos de visão computacional através de redes neurais para detecção de máscaras de proteção individual e detecção de aglomeração de pessoas. Os algoritmos mostraram bons resultados nos testes e pode-se obter considerações pertinentes como o posicionamento das câmeras para melhor uso do sistema prevenindo falhas.

A interface desenvolvida foi desenvolvida sob o *framework* Qt e com linguagem python para criação e manipulação de elementos gráficos. Esta mostrou-se intuitiva e de fácil uso pelo usuário tanto para o monitoramento simples quanto para o cadastro de novas câmeras e usuários.

Foram realizados testes de usabilidade da interface e de performance dos algoritmos. Dado o uso de *threads* e a correta separação de conceitos foi possível a interação da interface e do processamento algoritmos sem que ambos se interferissem causando quebra da aplicação.

Por fim, de modo geral foi possível a junção de conceitos tanto integrados a engenharia elétrica principalmente aos estudos de inteligência artificial e reconhecimento de padrões quanto conceitos aplicados de forma constante ao mercado como uso de banco de dados relacionais e desenvolvimento de software.

5.1 Trabalhos Futuros

Pelos objetivos alcançados no desenvolvimento deste trabalho e através dos estudos necessários para desenvolvimento do mesmo, foram pensadas extensões e modificadas ao escopo da interface e algoritmos de modo a continuar os estudos. Dentre os tais podem ser destacados:

1. Implementar algoritmos de detecção com bibliotecas YOLO via rede *darknet* para verificar performance no sistema desenvolvido considerando o estrutura de software com threads.
2. Implementação de banco de dados online para aplicar um pós processamento descentralizado e possibilidade de obter alertas de detecção e relatórios fora do ambiente local
3. Adaptar o sistema para plataformas mobile considerando a versatilidade e processamento em GPUs de menor porte
4. Adaptar os códigos do banco de dados para uma *Object-Relational Mapper* (ORM) como *SQLAlchemy* com objetivo de facilitar a interação com o banco de dados.

REFERÊNCIAS

BAZAREVSKY, V.; KARTYNNIK, Y.; VAKUNOV, A.; RAVEENDRAN, K.; GRUNDMANN, M. **BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs**. 2019.

D HAN K, S. J. K.; Y., N. **Smombie Guardian: We watch for potential obstacles while you are walking and conducting smartphone activities.** 2018.

DEB, C. **Face Mask Detection**. 2021. Disponível em: https://github.com/chandrikadeb7/Face-Mask-Detection.

DSA. **Deep Learning Book**. 2021. Acessado em 15/01/22. Disponível em: https://www.deeplearningbook.com.br/o-que-sao-redes-neurais-artificiais-profundas/.

GIRARDI, J. d. M.; ANDRADE, A. M.; RAMOS, M. C.; OLIVEIRA, L. E. d. S.; PEREIRA, D. C. R.; SILVA, E. T. Uso de máscaras para a redução da transmissão da covid-19: revisão integrativa. **Comunicação em Ciências da Saúde**, v. 32, n. 01, jun. 2021. Disponível em: http://www.escs.edu.br/revistaccs/index.php/comunicacaoemcienciasdasaude/article/view/800.

GONZALEZ, R.; WOODS, R. **Digital Image Processing**. Prentice Hall, 2002. ISBN 9780201180756. Disponível em: https://books.google.com.br/books?id=738oAQAAMAAJ.

GOOGLE. **MediaPipe Face Detection**. 2021. Disponível em: https://google.github.io/mediapipe/solutions/face_detection.

HAYKIN, S. **Redes Neurais - 2ed.** Bookman, 2001. ISBN 9788573077186. Disponível em: https://books.google.com.br/books?id=lBp0X5qfyjUC.

KLEIN, B. **DEEP Learning Notes: Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization.** 2013. Acessado em 25/01/2022. Disponível em: https://github.com/brunoklein99/deep-learning-notes/blob/master/README.md.

NEVES, V. **Imagem Digital - Teoria.** 2009. Acessado em 15/01/22. Disponível em: http://sdbmcc.blogspot.com/2009/06/imagem-digitalteoria.html.

PALACH, J. **Parallel programming with Python**. [*S. l.*]: Packt Publishing Ltd, 2014.

PyQt5 Reference Guide. RiverBank Computing. Acessado em 12/01/22. Disponível em: https://www.riverbankcomputing.com/static/Docs/PyQt5/introduction.html.

RMM DABOIN BEG, O. A. P.; H., M. J. The dissemination of covid-19: an expectant and preventive role in global health. **Journal of Human Growth and Development**, p. 135–140, 2020.

SINHA, U. **CONVOLUTIONS.** 2017. Acessado em 25/01/2022. Disponível em: http://aishack.in/tutorials/image-convolution-examples/.

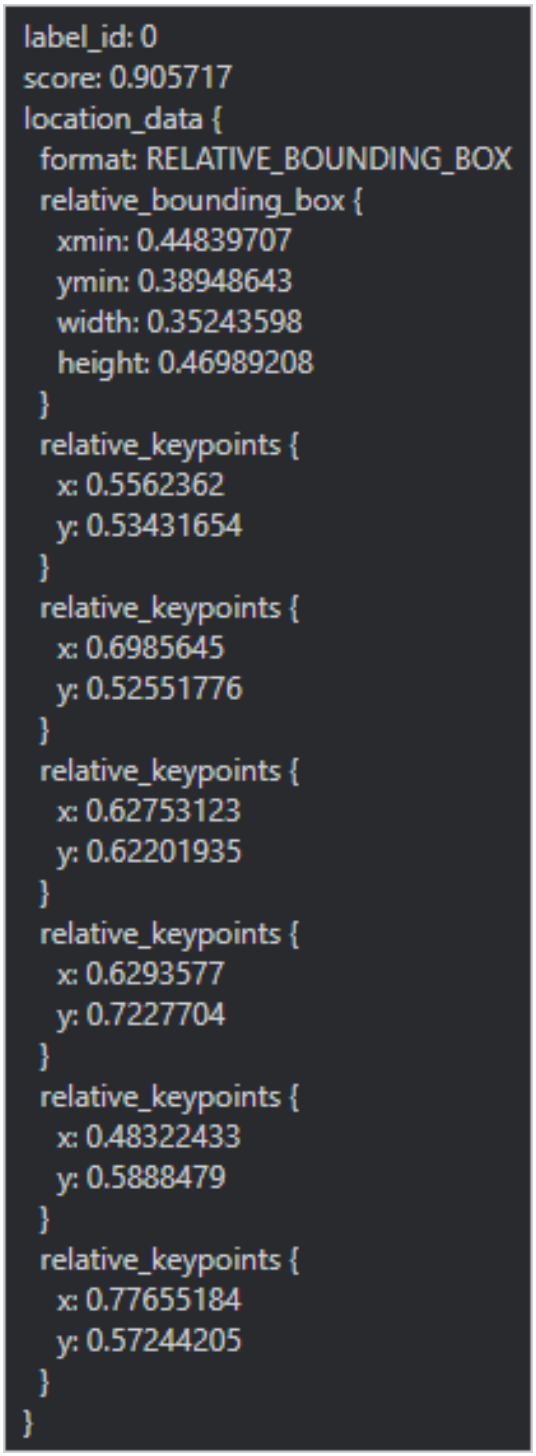
TheNsBhasin. **DNN_Object_Detection**. 2021. Disponível em: https://github.com/TheNsBhasin/DNN_Object_Detection.

UNIVERSITY, G. **Curso: Programação Concorrente e Assíncrona com Python**. [*S. l.*]: Udemy, 2021.

What is RTSP? Acessado em 12/01/22. Disponível em: https://www.cctvcameraworld.com/what-is-rtsp/.

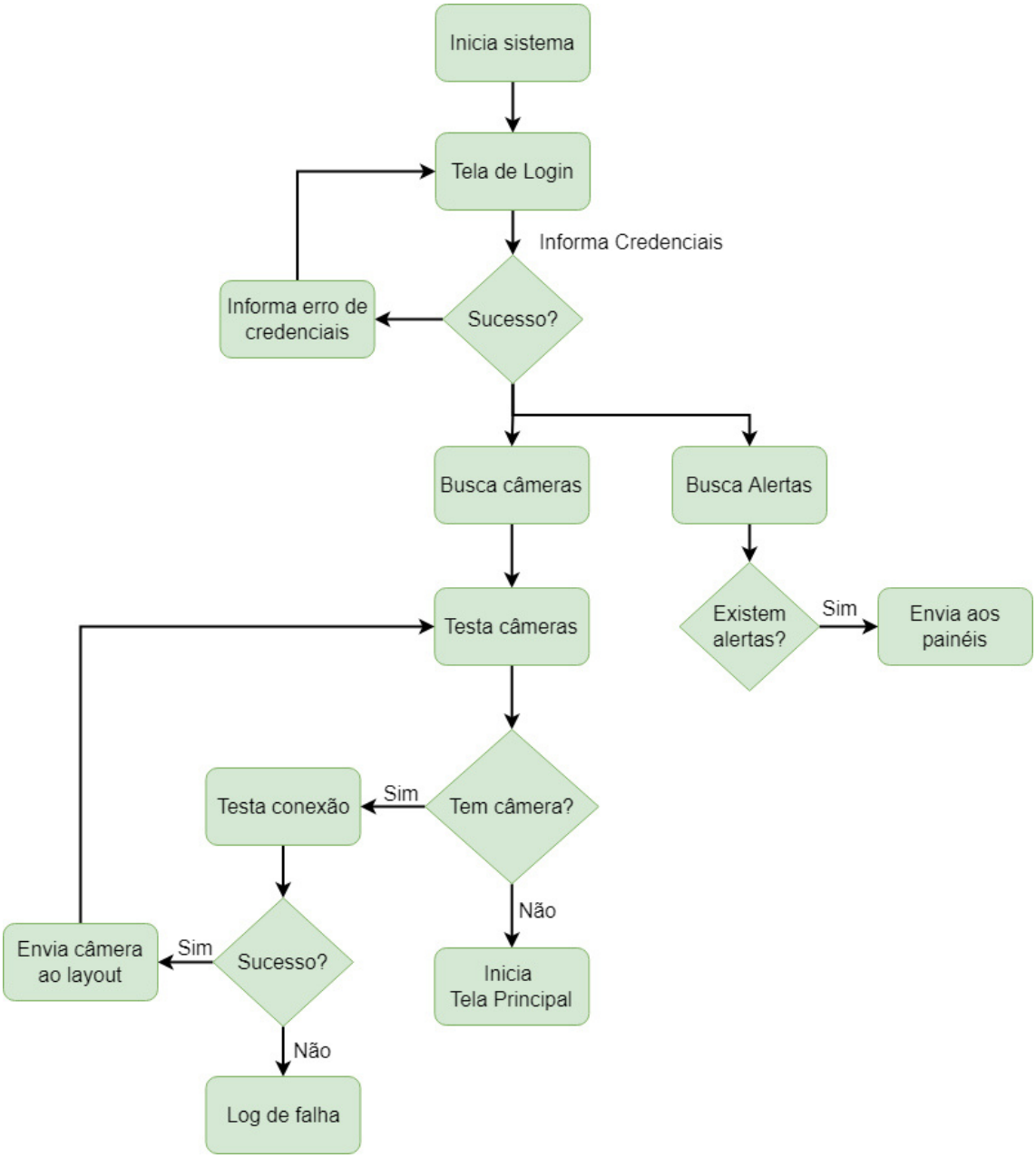
APÊNDICE A – RESULTADO POR ELEMENTO NA DETECÇÃO DE FACES

Figura 33 – FaceDetector result.



Fonte: O Autor (2021).

APÊNDICE B – FLUXOGRAMA DE INICIO DE INÍCIO DE SISTEMA



Fonte: O Autor (2021).