UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

RAFAEL HORA RAMOS

MOISES DE PAULO DIAS

SISTEMA DE CONTAGEM, CLASSIFICAÇÃO DE GRUPO DE IDADE E SEGMENTAÇÃO DE VESTIMENTA DE PESSOAS EM VÍDEOS UTILIZANDO DEEP LEARNING

CURITIBA

2022

RAFAEL HORA RAMOS
MOISES DE PAULO DIAS

SISTEMA DE CONTAGEM, CLASSIFICAÇÃO DE GRUPO DE IDADE E SEGMENTAÇÃO DE VESTIMENTA DE PESSOAS EM VÍDEOS UTILIZANDO DEEP LEARNING

Counting, age group classification and clothing segmentation system of people in videos using deep learning

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Heitor Silvério Lopes

CURITIBA
2022

RAFAEL HORA RAMOS
MOISES DE PAULO DIAS

SISTEMA DE CONTAGEM, CLASSIFICAÇÃO DE GRUPO DE IDADE E SEGMENTAÇÃO DE VESTIMENTA DE PESSOAS EM VÍDEOS UTILIZANDO DEEP LEARNING

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 12/dezembro/2022

Dr. Heitor Silvério Lopes
Universidade Tecnológica Federal do Paraná

Dra. Ana Cristina Barreiras Kochem Vendramin
Universidade Tecnológica Federal do Paraná

Dr. Guilherme de Santi Peron
Universidade Tecnológica Federal do Paraná

Dr. Manassés Ribeiro
Instituto Federal Catarinense

CURITIBA
2022

Dedicamos este trabalho às nossas famílias,
amigos e professores, pelo apoio prestado.

AGRADECIMENTOS

Agradecemos ao nosso orientador Prof. Dr. Heitor Silvério Lopes por todo apoio durante a graduação e, em específico, no desenvolvimento desse projeto, sempre nos auxiliando com sabedoria.

Gostaríamos de agradecer de forma geral aos docentes da UTFPR por todo conhecimento passado a nós, bem como a todos os servidores por tornarem viável o funcionamento da universidade.

Aos nossos pais, irmãos, avós e à nossa família como um todo, principalmente os mais próximos, que participaram do início ao fim dessa trajetória e deram todo o apoio necessário para vencer os obstáculos, que não foram poucos.

Às nossas namoradas e amigos.

A Deus por ter nos dado a vida, conhecimento e força para concluir mais uma etapa muito importante das nossas vidas.

A todos envolvidos nas nossas vidas e na nossa trajetória, somos muito gratos!

“A computer would deserve to be called intelligent if it could deceive a human into believing that it was human”. (TURING, 1950)

RESUMO

Devido ao crescimento do número de estabelecimentos comerciais e dos métodos de inteligência e visão artificiais, tem surgido interesse na aplicação de tais métodos para entender o comportamento de clientes e do mercado, objetivando vender mais e/ou vender melhor. O processo de automatização da obtenção de informações, como grupo de idade e vestimenta, pode ser muito relevante do ponto de vista econômico. Através da análise destes dados, um estabelecimento comercial pode obter informações interessantes sobre o seu público-alvo de tal maneira a impulsionar vendas e gerar retorno monetário ao direcionar o conteúdo do estabelecimento para este público. O presente trabalho propõe um sistema de ponta-a-ponta para auxiliar a obtenção de informações relevantes do público-alvo. São utilizados métodos de visão computacional e *deep learning* para realizar a contagem de pessoas, a classificação de grupo de idade, bem como a segmentação e classificação de vestimenta de pessoas em vídeos. Para isto, foi utilizado o algoritmo StrongSORT em conjunto com o YOLO-v5 para o *tracking* de pessoas, bem como métodos de detecção e reconhecimento facial, além de alinhamento das faces para remover ruídos de *background*. Os módulos foram implementados sobre um sistema complexo de comunicação e processamento distribuídos utilizando o Apache Kafka. Para a persistência dos dados foi utilizado um banco de dados MongoDB, que fornece as informações consumidas na última etapa do sistema, que é a visualização dos dados processados em um *dashboard* de forma amigável ao usuário final. Ao fim do trabalho foram obtidos resultados bastante satisfatórios em todos os módulos propostos com diferentes condições de contorno aplicadas aos mesmos, além de resultar em um sistema com uma arquitetura de dados robusta e facilmente escalável.

Palavras-chave: redes neurais; deep-learning; rastreamento de pessoas; segmentação de roupas; big data.

ABSTRACT

Due to the growth in the number of commercial establishments and artificial intelligence and vision methods, there has been interest in the application of such methods to understand the behavior of customers and the market, aiming to sell more and/or sell better. The process of automating obtaining information, such as age group and clothing, can be very relevant from an economic point of view. Through the analysis of this data, a commercial establishment can obtain interesting information about its target audience in such a way as to boost sales and generate monetary return by directing the establishment's content to this audience. This work proposes an end-to-end system to help obtain relevant information from the target audience. Computer vision and deep learning methods are used to perform people counting, age group classification, as well as segmentation and clothing classification of people in videos. For this, the StrongSORT algorithm was used together with YOLO-v5 for the tracking of people, as well as methods of detection and facial recognition, in addition to face alignment to remove noise from the background. The modules were implemented on a complex distributed communication and processing system using Apache Kafka. For the persistence of the data, a MongoDB database was used, which provides the information consumed in the last stage of the system, which is the visualization of the processed data in a dashboard in a friendly way for the end user. At the end of the work, very satisfactory results were obtained in all proposed modules with different boundary conditions applied to them, in addition to resulting in a system with a robust and easily scalable data architecture.

Keywords: neural-networks; deep-learning; people tracking; cloth segmentation; big data.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Siglas

CORS	*Cross-Origin Resource Sharing*
FPN	*Feature Pyramid Network*
HOG	*Histogram of Oriented Gradient*
HTTP	*Hypertext Transfer Protocol*
IoT	*Internet of Things*
JSON	*JavaScript Object Notation*
LABIC	Laboratório de Inteligência Computacional
LGPD	Lei Geral de Proteção de Dados
NoSQL	Não apenas SQL, do inglês *Not only SQL*
OSNet	Omni-Scale Network
RCNN	*Region-based Convolutional Neural Network*
RNC	Rede Neural Convolucional
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*
SSD	*Single Shot Detector*
StrongSORT	*Strong Simple Online Realtime Tracking*
SVM	*Support Vector Machine*
UTFPR	Universidade Tecnológica Federal do Paraná
YOLO	*You Only Look Once* (Você só olha uma vez)

SUMÁRIO

1 INTRODUÇÃO

1.1 Considerações iniciais

Com o passar dos anos, a coleta e o uso de dados em geral têm crescido de forma exponencial (STATISTA, 2022) e, concomitantemente a isso, houve o aumento no número de câmeras espalhadas por todos os lugares, seja em residências, shoppings ou até vias públicas de cidades inteiras. A partir do aumento da população mundial, a contagem e identificação de pessoas, entre outras informações relevantes voltadas às suas características, tornaram-se cada vez mais relevantes tanto no que diz respeito à área de segurança, incluindo controle de tráfego e identificação de pessoas (PETERSON, 2000), quanto à área comercial, como segmentação de clientes e identificação de público (CHRISTY *et al.*, 2021), assim, podendo trazer informações relevantes para diversos fins.

Em função deste vertiginoso crescimento dos dados disponíveis, métodos de *Deep Learning* tem sido utilizados intensamente, atingindo resultados muito satisfatórios nas mais diversas tarefas, tais como: classificação de objetos em imagens/vídeos, segmentação de imagens, extração e classificação de características, entre outras. Apesar de todo o avanço citado, ainda há problemas principalmente quando se trata de integração e processamento de dados em grande escala, uma vez que esses modelos e métodos têm alto custo computacional, demandando uma arquitetura poderosa de processamento para viabilizar o funcionamento do processo como um todo. Com a associação das necessidades de alto poder de processamento, além de melhor desempenho na transferência dos dados processados, a área de *Big Data* entrou em cena e com ela surgiram ferramentas robustas que conseguem lidar com um alto tráfego de dados em tempo real (RODRíGUEZ-MAZAHUA *et al.*, 2016).

Baseando-se na afirmação da empresária americana Carly Fiorina, “O objetivo é transformar dados em informação e informação em entendimento”, muitas empresas, e o mercado como um todo, têm entendido cada vez mais esta mensagem no sentido de inovar e entender melhor os seus clientes e, assim, utilizando grandes volumes de dados para alavancar os negócios. Devido a esta visão de mercado, muitas aplicações estão sendo desenvolvidas e têm mostrado bons resultados, por exemplo: predição de *churn*[1] (SPANOUDES; NGUYEN, 2017), rastreamento e análise comportamental de clientes no interior de lojas (GENEROSI; CECCACCI; MENGONI, 2018), clusterização (separação em grupos) de clientes (SYAKUR *et al.*, 2018) entre outros. O presente trabalho segue a ideia de transformar dados em informação, com o desenvolvimento de um sistema que, a partir da contagem de pessoas, classificação de grupo de idade e segmentação de vestimenta das mesmas, consegue apresentar informações relevantes, no formato de *dashboard*, sobre o público alvo de um estabelecimento comercial. Com o uso de uma arquitetura robusta e escalável, este sistema pode trazer a possibilidade

[1] churn é uma métrica utilizada para mostrar o número de clientes que cancelam um serviço em um determinado período de tempo.

de direcionar produtos e propagandas a partir da análise dos resultados, se usado de forma correta.

1.2 Objetivos

1.2.1 Objetivo geral

O objetivo geral do presente trabalho é desenvolver um sistema de ponta-a-ponta para auxiliar a obtenção de informações relevantes de um público-alvo. O sistema processa em tempo-real um fluxo de vídeo, realiza a contagem de pessoas e sua classificação por grupo de idade, bem como segmenta as imagens e classifica o tipo de vestimenta das pessoas. O resultado final é sintetizado em um *dashboard* compreensível para o usuário.

1.2.2 Objetivos específicos

Os objetivos específicos do presente trabalho são:

- Desenvolver um módulo para realizar o rastreamento de pessoas em um dado vídeo.
- Desenvolver um módulo para realizar detecção e reconhecimento facial de pessoas em um vídeo.
- Desenvolver um módulo para classificar pessoas por grupo de idade.
- Desenvolver um módulo para segmentar e classificar vestimentas de pessoas.
- Desenvolver um módulo para tratar e persistir os dados das identificações em um banco de dados não relacional.
- Gerar um *pipeline* para integrar os módulos de forma a realizar as tarefas em um fluxo de vídeo.
- Armazenar os resultados obtidos em um banco de dados não relacional.
- Gerar resultados experimentais do sistema como um todo.
- Gerar um *dashboard* para apresentar de forma simples e intuitiva os resultados obtidos.

1.3 Justificativa

O processo de automatização da obtenção de informações, como grupo de idade e vestimenta, pode ser muito relevante do ponto de vista econômico. Através da análise destes dados, um estabelecimento comercial pode obter informações interessantes sobre o seu público-alvo

de tal maneira a impulsionar vendas e gerar retorno monetário ao direcionar o conteúdo do estabelecimento para este público.

Dada a crescente utilização de métodos de *deep-learning* para análise de imagens e predição, bem como o aumento de tecnologias de reconhecimento facial e métodos de segmentação, fica clara a importância de obter conhecimentos nesta área. Além disto, a orquestração de diversos módulos de processamento em um ambiente de paralelismo computacional é uma tecnologia muito importante para a escalabilidade de sistemas.

1.4 Estrutura do trabalho

O presente trabalho foi estruturado em 5 capítulos da seguinte forma: No Capítulo 2 são apresentados os referenciais teóricos dos temas pertinentes ao trabalho realizado. No Capítulo 3 são abordados os métodos que foram utilizados no desenvolvimento. O Capítulo 4 trata dos resultados obtidos a partir da aplicação do sistema criado em testes. Por fim, no Capítulo 5, são apresentadas as conclusões a partir dos resultados obtidos.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os conceitos teóricos utilizados no trabalho.

2.1 Sistemas Distribuídos

Sistemas distribuídos são sistemas onde o processamento é distribuído em diferentes processos que coordenam suas ações através de envio de mensagens e cooperam para atingir uma tarefa.

Os sistemas distribuídos permitem a escalabilidade horizontal, ou seja, adicionando mais máquinas com um poder de processamento semelhante às existentes, diferente da escalabilidade vertical, que seria aumentar o poder de processamento das máquinas já existentes. Isto é vantajoso, pois utilizando a escalabilidade vertical pode-se atingir rapidamente o limite de recursos para uma máquina, impedindo um maior crescimento do sistema. Por outro lado, a escalabilidade horizontal permite a clonagem de uma máquina já existente e a distribuição das cargas de trabalho entre as máquinas criadas.

Apesar de ser uma arquitetura completamente diferente de um sistema centralizado, esta diferença de arquitetura é transparente, ou seja, não se manifesta para o usuário final, que o vê como um computador único e integrado.

As principais vantagens de um sistema distribuído em comparação com um sistema centralizado são o seu menor custo, a facilidade para escalar a aplicação, a tolerância a falhas, uma vez que erros de *software* ou *hardware* ficam isolados àquela parte do todo, diferentemente de um sistema monolítico, e a possibilidade de ter um sistema heterogêneo, isto é, cada parte do todo pode ter diferente *hardware* e ambiente de sistema dos demais. Por outro lado, suas principais desvantagens são possíveis atrasos decorrentes de falha na rede e a complexidade de orquestrar todos os subsistemas.

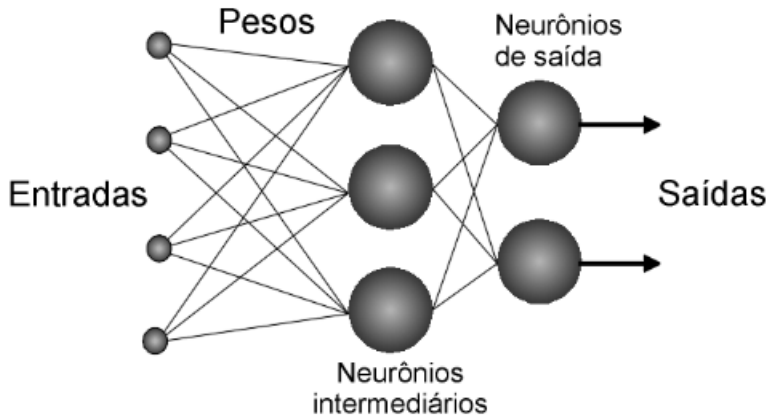
2.2 Redes Neurais Artificiais

Redes Neurais Artificiais (RNAs), também chamadas de *feed-forward*, são algoritmos de inteligência artificial utilizados para reconhecimento de padrões, regressão e classificação de dados. As mesmas são utilizadas para resolver problemas tais como reconhecimento de fala, imagens e vídeos, processamento de linguagem natural e previsão de valores. Sua técnica é uma abstração da rede neural biológica.

RNAs utilizam conjuntos de nós ou neurônios em camadas que se interconectam com outras camadas. Estas podem ser divididas em três tipos: camada de entrada, camada oculta e camada de saída, conforme mostrado na Figura 1.

Os nós da camada de entrada recebem os dados brutos. Em um exemplo para predição de preços de casas, os dados brutos poderiam ser características como área construída, acabamento, localização, quantidade de móveis. Neste exemplo, como a amplitude das classes dos valores variam em ordens de grandeza, é realizada uma normalização nos valores, de forma a manter uma escala comum entre todos os valores de entrada.

Figura 1 – Exemplo de uma Rede Neural Artificial com 4 entradas e 2 saídas.



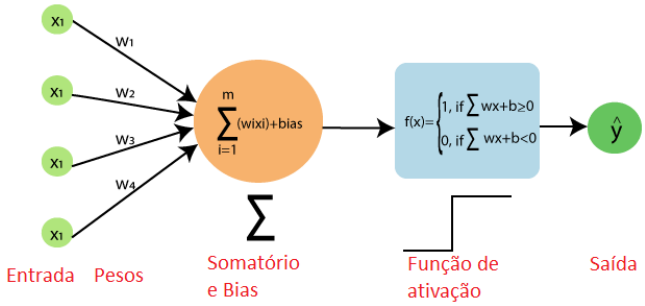
Fonte: Imagem retirada da internet.

Cada nó de uma camada possui valores inicialmente aleatórios (exceto a camada inicial, que possui os valores dos dados brutos) e é conectado a todos os nós da camada posterior e anterior. Cada aresta entre estes nós possui um peso ou multiplicador, de forma que, durante a execução da rede neural, o valor de um nó é definido da seguinte forma: o primeiro valor é referente a camada anterior, é realizada a multiplicação de cada nó de uma camada anterior pelo peso da aresta que liga este nó da camada anterior a um nó da camada posterior. Os valores das multiplicações são então somados e atribuídos ao nó da camada posterior. Além disto, cada nó tem também uma segunda entrada, que é uma constante chamada de polarização ou *bias* normalmente abreviada para a letra b. O resultado da soma dos valores de entrada do neurônio é então somado ao *bias* e passado por uma função de ativação, que realiza uma limitação na amplitude do valor de saída do nó. Isto é importante para garantir que mudanças nos pesos e *bias* dos nós não causem uma mudança drástica indesejada na saída. Este procedimento é mostrado na Figura 2.

A camada de saída é composta de um ou mais nós. Porém, diferentemente das camadas ocultas, os valores de saída possuem um significado: o resultado da classificação ou regressão a partir dos dados inseridos na camada de entrada.

Para que uma rede neural possa gerar resultados satisfatórios, é necessário o treinamento da mesma. Para tanto, são utilizados dados rotulados, ou seja, um conjunto de dados de entrada com os valores de saída correspondentes. Este treinamento é feito a partir da retropropagação (*backpropagation*) das informações pela rede, isto é, a partir da execução da rede neural em dados rotulados, a saída é comparada com o resultado esperado e a retropropaga-

Figura 2 – Exemplo da passagem de valor dos nós de uma camada para o nó de uma camada posterior.



Fonte: Imagem retirada da internet.

ção ajusta o valor de todos os pesos a partir de uma função de custo, baseada na diferença entre o resultado esperado e o obtido.

Devido ao alto número de parâmetros ajustáveis (valores dos pesos das camadas ocultas, que não podem ser preditos), redes neurais conseguem se adaptar para resolver problemas complexos que outras abordagens de aprendizado de máquina, tais como regressão linear e regressão logística, são simples demais para resolver.

2.3 Redes Neurais Convolucionais

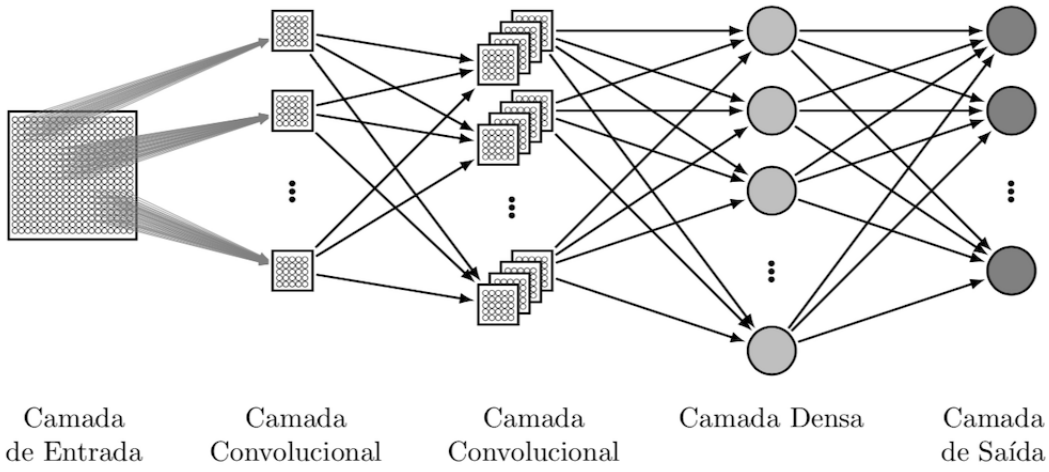
Uma Rede Neural Convolucional (RNC), mostrada na Figura 3, é uma variação da rede neural *feed-forward* que possui um desempenho melhor quando a entrada da rede neural é uma imagem.

Suas principais diferenças estão na estrutura do formato da entrada, na estrutura das camadas e na passagem de valor de uma camada para outra. Enquanto na rede neural *feed-forward* tem-se nós conectados a todos os nós das camadas anterior e posterior, nas RNCs tem-se filtros que são matrizes que realizam operações de convolução nos *pixels* de uma imagem de entrada. O resultado destas operações são filtros (ou *feature maps*), que também são matrizes. Os *feature maps*, por sua vez, também serão submetidos à operação de convolução em outros filtros, sendo que cada camada é composta pelos *feature maps* resultantes e a passagem de valor de uma camada para outra é feita a partir da convolução. Na RNA, os parâmetros que são ajustados durante o treinamento são os pesos das arestas que conectam os nós da rede neural, já na RNC os parâmetros ajustados são os valores do filtro que é utilizado para realizar a convolução na imagem.

Após utilizar as camadas de convolução para extrair padrões de imagens, a última camada de convolução é conectada em uma rede neural *feed-forward*, também chamada de camada densa, com as características descritas anteriormente. Isto é feito pois a primeira etapa, com as camadas de convolução, são capazes de se aproveitar da localidade ou relação espa-

cial dos *pixels* das imagens, sendo que esta informação poderia ser perdida se a imagem fosse convertida em um vetor e passada para uma rede neural *feed-forward*.

Figura 3 – Rede neural convolucional com camadas de convolução e *feed-forward*.



Fonte: Imagem retirada da internet.

2.3.1 YOLO

You Only Look Once (Você só olha uma vez) (YOLO) (JOCHER *et al.*, 2022) é uma arquitetura de RNC utilizada para a detecção de objetos em imagens, onde a entrada da rede neural é uma imagem e a saída é, não só uma classificação, mas sim, uma ou mais classificações junto com suas respectivas coordenadas na imagem (*bounding-box*). Diferentemente de arquiteturas *Region-based Convolutional Neural Network* (RCNN) e *Single Shot Detector* (SSD) (LIU *et al.*, 2016), que realizam este processo de detecção em duas etapas (identificação e classificação das áreas), a YOLO realiza estas duas tarefas em uma etapa apenas (daí a origem do nome *You Only Look Once*), sem comprometer a acurácia das detecções, o que faz desta arquitetura a melhor escolha para detecção de objetos em tempo-real.

Existem diversas versões da rede YOLO que são publicadas por pesquisadores após realizarem melhorias nas arquiteturas precedentes, a versão normalmente está presente no nome da rede, por exemplo, YOLO-v5 é a quinta versão do YOLO, lançada em 2020.

2.3.2 YOLOv5-Face

YOLOv5-Face (QI *et al.*, 2021) é uma arquitetura de rede neural convolucional feita a partir do YOLOv5 com algumas modificações centrais, tais como adição de detector de pontos de referências (*landkmarks*) faciais de 5 pontos, para otimizar a rede para a tarefa de detecção facial. YOLOv5-Face pode ser usado como um modelo com mais camadas, de forma a atingir a melhor performance nas detecções, ou um modelo de tamanho reduzido para realizar detecções em tempo real.

2.3.3 OSNet

Omni-Scale Network (OSNet) (ZHOU *et al.*, 2019) é uma arquitetura de RNC especializada em re-identificação de pessoas (ReID). Esta rede neural atribui uma identificação para cada pessoa a partir de características da mesma. Estas características podem ser de várias escalas, desde uma estampa em uma camisa até um vestido inteiro. Além disto, a rede não necessariamente utiliza todas as características em uma detecção pois, se a pessoa identificada possuir uma estampa apenas na parte frontal na camisa, reconhecida pela rede, em uma imagem onde esta pessoa esteja de costas, ainda assim a OSNet pode utilizar outras características (como a cor da camisa) para identificar tal pessoa.

2.3.4 StrongSORT

Strong Simple Online Realtime Tracking (StrongSORT) (DU *et al.*, 2022) é um algoritmo de rastreamento por detecção, utilizado para tarefas de rastreamento de múltiplos objetos. Este algoritmo também utiliza o YOLO para realizar detecções em imagens que são passadas para a rede OSNet para realizar o rastreamento destas detecções.

2.3.5 EfficientNet

EfficientNet (TAN; LE, 2019) é uma arquitetura de RNC com um método de escalar ou expandir uniformemente a profundidade (quantidade de camadas), largura (quantidade de nós nas camadas) e resolução (tamanho da imagem de entrada) utilizando um coeficiente comum a estas três dimensões. Este método é chamado de coeficiente composto e para cada uma das dimensões da rede neural é atribuído um coeficiente. A expansão da rede neural é sempre feita nas três dimensões simultaneamente, a partir da multiplicação de cada dimensão por seu respectivo coeficiente elevado a um expoente, no lugar de arbitrariamente alterar uma dimensão. Atingindo o estado da arte com menos *FLOPs* (*Floating Point Operations per second*), ou seja, de forma mais eficiente que as demais arquiteturas, a EfficientNet é amplamente utilizada para

classificação de imagens por meio do chamado *transfer-learning*, que congela a maior parte da rede e inclui uma camada de classificação para um novo conjunto de dados, aproveitando, assim, a rede pré-treinada para novas aplicações. Tendo isso em vista, neste trabalho a EfficientNet é utilizada justamente para este fim. Com ela é realizada a classificação de grupo de idade por meio de imagens, além de a mesma ser responsável pela extração de *features* (*backbone*) do módulo de segmentação de vestimenta.

2.4 EPYNET

EPYNET (INÁCIO; LOPES, 2020) é um *framework* para a segmentação de vestimentas e é baseada no detector SSD e no *Feature Pyramid Network* (FPN) (LIN *et al.*, 2017) com a já citada EfficientNet. Este *framework* é focado na identificação e segmentação de imagens de atributos humanos (como pele e cabelo), bem como de vestimentas e acessórios (como blusa, calça, óculos, bolsa). Este *framework* também utiliza a técnica de *data augmentation* para melhorar o desempenho do sistema. Esta técnica envolve gerar imagens com distorções, tais como mudança de rotação e orientação, ampliação, recorte e mudança dos níveis de brilho e cor, durante o treino para melhorar o desempenho na tarefa de segmentação. O modelo foi treinado utilizando o *dataset* UTFPR-SBD3 (UTFPR *Soft Biometrics Dataset*[1]), contendo 4500 imagens rotuladas manualmente em 18 classes distintas de roupas além do fundo da imagem, a Figura 4 ilustra alguns exemplos de imagens presentes em um dos *datasets* que originaram o UTFPR-SDB3.

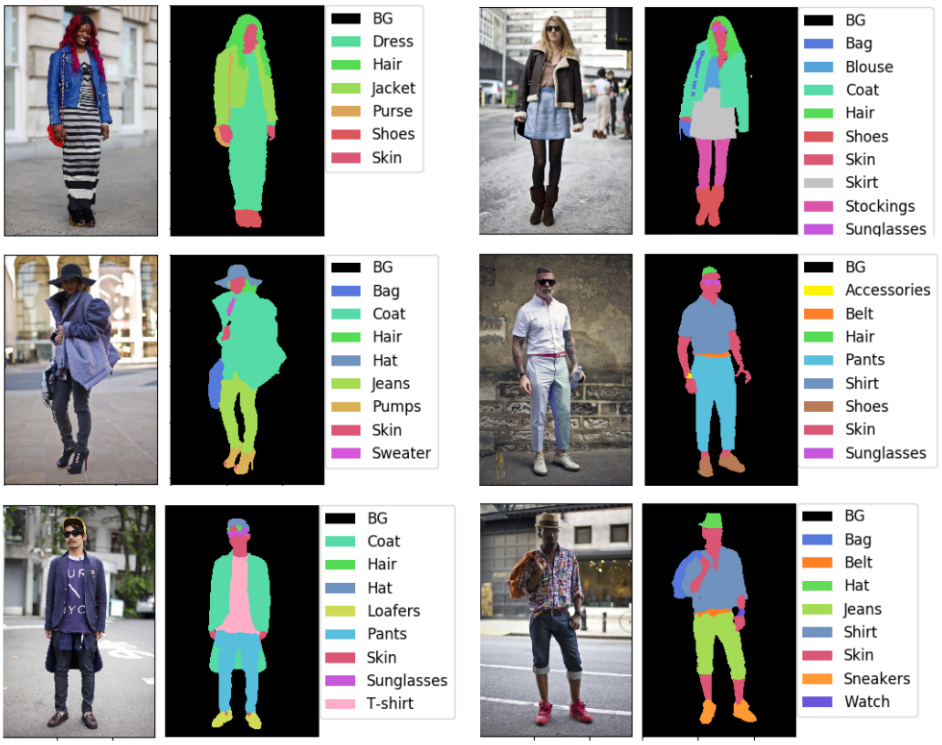
As classes distintas anotadas no *dataset* UTFPR-SBD3 e, portanto, passíveis de serem identificadas são: bolsa, cinto, casaco, vestido, óculos, calçado, cabelo, chapéu, gravata, calças, macacão, camisa, shorts, pele, saias, meias, meia-calça e suéter.

2.5 *Streaming* de Eventos

Um *Streaming* de eventos é caracterizado pela coleta de dados em tempo-real de fontes (ou produtores) como sensores, softwares, dispositivos móveis ou *Internet of Things* (IoT), seguida da armazenagem ou enfileiramento e processamento destes dados. Posteriormente, estes dados são enviados para distintos destinatários (ou consumidores). O processamento destes dados, ou processamento de *streaming* de eventos, é o processo de tomar uma ação a partir do dado gerado em uma arquitetura orientada a eventos.

[1] Disponível em: https://labic.utfpr.edu.br/datasets/UTFPR-SBD3.html

Figura 4 – Exemplos de imagens e anotações de segmentações de vestimenta.



Fonte: (INÁCIO; BRILHADOR; LOPES, 2019).

2.6 Apache Kafka

O sistema Apache Kafka (KREPS; NARKHEDE; RAO, 2011) é uma plataforma *open-source* de *streaming* de eventos que tem como principal atribuição publicar, subscrever, armazenar e processar fluxos de registros ou eventos em tempo-real. A plataforma pode lidar com diversas fontes de eventos e diversos clientes de destino simultaneamente, lidando com um fluxo constante de dados e processando-os de forma sequencial e incremental. Para atingir este objetivo, o Kafka utiliza uma combinação de uma fila de mensagens e a arquitetura *publish-subscribe*, enfileirando as mensagens provenientes dos produtores e enviando não para um consumidor, mas para todos os publicadores (*publishers*), que enviam para todos os consumidores assinantes deste publicador. O Kafka vem sendo utilizado cada vez mais em empresas que necessitam de *streaming* de dados em grande volume como, por exemplo, a Coinbase que possui um número muito grande de transações diárias de criptomoedas (COINBASE, 2022).

2.7 Banco de dados

Banco de dados é um sistema responsável por armazenar informações estruturadas em um sistema computacional, de forma a permitir fácil acesso, manutenção e gerenciamento destas informações. Os dados são comumente modelados em linhas e colunas em uma série

de tabelas para tornar eficiente as consultas a estas informações. A maioria os bancos de dados usam a linguagem *Structured Query Language* (SQL) para a manipulação e consulta de dados. Este sistema é composto por informações a serem armazenadas (dados) e um Sistema de Gerenciamento de Banco de Dados (SGBD), que é uma interface entre o banco de dados e o usuário final.

O MongoDB (GUO, 2017) é um banco de dados orientado a documentos, categorizado como Não apenas SQL, do inglês *Not only SQL* (NoSQL), que são arquiteturas específicas para um alto volume de dados em uma velocidade alta, além de suportar dados não estruturados, semi-estruturados ou estruturados. Ele possui código aberto e suporte para as principais linguagens de programação, tais como C, C++, C#, Java e Python.

Além do banco de dados em si, os desenvolvedores do mesmo fizeram o chamado MongoDB Atlas que, em linhas gerais, é um serviço em nuvem que visa facilitar o lançamento, gerenciamento e observação dos bancos de dados MongoDB.

2.8 Docker

Docker (MERKEL, 2014) é uma plataforma para criação de ambientes para execução de uma aplicação e, para isto, o Docker utiliza contêineres (um ambiente isolado do sistema operacional que possui pacotes de software que contém todos os elementos necessários para serem executados) que podem ser utilizados em qualquer sistema que possua o Docker instalado. Isto permite a possibilidade de migrar uma aplicação para diferentes sistemas, seja num computador local ou na nuvem mas, principalmente, facilita a criação de módulos em sistemas distribuídos de forma a utilizar o mínimo de recursos necessários para uma aplicação e possibilita uma fácil escalabilidade de um sistema. A plataforma também oferece uma interface gráfica e de linha de comando para a criação e administração de contêineres.

2.9 Kubernetes

Kubernetes é uma plataforma utilizada para gerenciar a carga de trabalho em serviços distribuídos em contêineres (MEDEL *et al.*, 2016), tais como contêineres Docker. A plataforma também é usada para realizar manutenção, *deploy*, monitoramento e escala de aplicações em contêineres. O Kubernetes normalmente é utilizado quando há a necessidade de gerenciar serviços distribuídos que realizam processamento em tempo real, de forma a poder escalar a aplicação sem que isso afete as demais partes do sistema e de forma transparente para quem vê o sistema de fora.

2.10 OpenFace

OpenFace (AMOS; LUDWICZUK; SATYANARAYANAN, 2016) é uma implementação em código aberto de redes neurais para reconhecimento facial, baseado no FaceNet (SCHROFF; KALENICHENKO; PHILBIN, 2015). O OpenFace pode ser usado para detectar faces (*facial landmark detection*), estimar o local onde a pessoa está olhando (*eye-gaze estimation*), detectar mudanças na expressão de uma face (*facial action unit recognition*) e detecção da posição da cabeça na imagem (*head pose estimation*). Com estas funcionalidades, o OpenFace se torna uma ferramenta muito útil para extração de características faciais em uma imagem. O OpenFace também oferece um ótimo desempenho mesmo sendo executado em tempo-real e pode ser integrado com outros serviços utilizando um sistema de *streaming* de eventos.

2.11 Dlib

Dlib[2] é uma biblioteca de código aberto desenvolvida em C++ contendo uma coleção de algoritmos e ferramentas de *Machine Learning* com uma extensa documentação e modos de depuração. Esta biblioteca permite o uso de detectores de objetos baseados em *Histogram of Oriented Gradient* (HOG) (DALAL; TRIGGS, 2005), que é uma técnica para detecção de objetos semi-rígidos, ou seja, objetos cujo formato e as características não possuem uma ampla gama de variabilidade. O processo de treinamento do detector baseado em HOG utiliza um algoritmo baseado na *Support Vector Machine* (SVM) da própria biblioteca Dlib, o que permite o detector ser treinado em todas as sub-janelas em cada imagem, o que diminui a quantidade de dados necessários para realizar o treinamento do detector.

2.12 Cvlib

Cvlib (*Computer vision library*)[3] é uma biblioteca de código aberto, simples e de alto nível, com funções de visão computacional para a linguagem Python. A biblioteca foi desenvolvida para permitir implementações simples e rápidas não exigindo muito conhecimento de visão computacional para utilizar suas funcionalidades. A Cvlib utiliza duas bibliotecas de visão computacional para executar suas funções: Tensorflow[4] e OpenCV[5].

[2] Disponível em: http://dlib.net/
[3] Disponível em: https://www.cvlib.net/
[4] Disponível em: https://www.tensorflow.org/
[5] Disponível em: https://opencv.org/

2.13 Trabalhos relacionados

Nesta seção são apresentados dois trabalhos que têm algumas similaridades ao presente trabalho.

2.13.1 *A Scalable Analytics Pipeline for COVID-19 Face Mask Surveillance*

Este trabalho, publicado por Kossoski *et al.* (2022), apresenta um sistema chamado SFMDP (*Scalable Face Mask Detection Pipeline*) utilizado para a detecção de máscaras faciais de COVID-19 em fluxos de vídeo. De forma semelhante ao presente trabalho, possui a proposta de utilização de métodos de *deep learning* para identificação de pessoas e faces em um fluxo de vídeo. O sistema apresenta uma estrutura semelhante para o processamento dos vídeos, também utilizando o Kafka para a ingestão de dados a partir do fluxo de vídeo, bem como o Docker para cada um dos módulos do sistema distribuído, Tensorflow e Keras para as detecções das faces e MongoDB para armazenamento das detecções.

2.13.2 Sistema de Processamento de Big Data Aplicado à Monitoração de Automóveis em Tempo Real

O trabalho de conclusão de curso, realizado por Lima e Agostinho (2022), propõe um sistema para identificação de automóveis em tempo real e a indexação dos mesmos a partir de características como placa, cor, marca e modelo do veículo. As tecnologias utilizadas se assemelham muito ao presente trabalho, foi utilizado o YOLO para a detecção dos veículos, processamento distribuído utilizando Kubernetes e Apache Kafka, além do MongoDB para armazenar os registros.

3 MÉTODOS

Nas Seções deste capítulo é detalhada cada uma das partes que compõem o trabalho.

3.1 Definição da Arquitetura

No decorrer de todo o curso foi aprendido que esta é a parte mais importante de todo projeto ou trabalho, uma vez que um erro na definição da arquitetura, identificado depois ou durante o desenvolvimento, pode acarretar em uma reestruturação completa e muito trabalho extra. Dito isto, uma atenção especial foi dada na execução desta definição.

Após toda a pesquisa de ferramentas e estudos teóricos, foi decidido utilizar módulos com tarefas específicas que juntos fariam o funcionamento de todo o sistema. Estas tarefas são:

a) Identificação e rastreamento de pessoas;

b) Detecção e reconhecimento facial;

c) Classificação de grupo de idade de pessoas;

d) Segmentação e classificação de vestimenta de pessoas;

e) Interface com *dashboard* para visualização sintética dos resultados obtidos.

Além disto, uma vez que os módulos precisam de um meio de comunicação entre si, foi decidido utilizar o Kafka como ferramenta de comunicação do sistema.

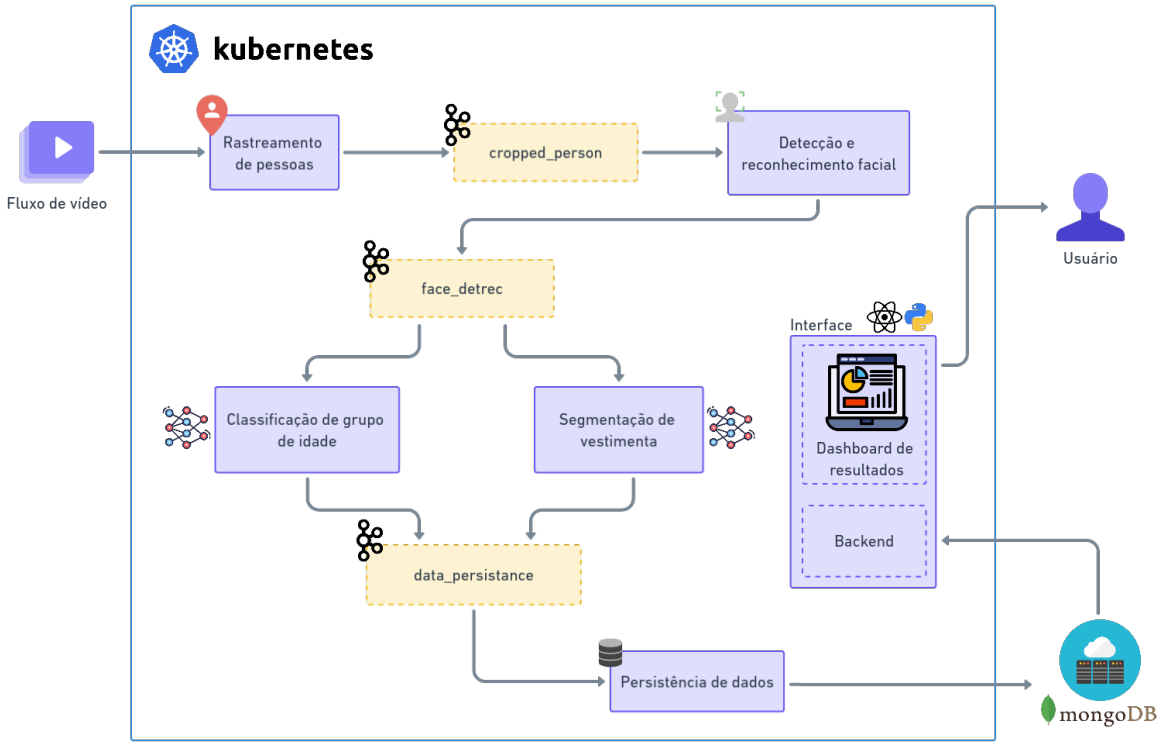
Para atingir o objetivo proposto foi necessário treinar modelos de redes neurais e trabalhar com processamentos custosos. Assim, foi utilizada a infraestrutura do Laboratório de Inteligência Computacional (LABIC) da Universidade Tecnológica Federal do Paraná (UTFPR), que conta com um *cluster* de alto desempenho para fins acadêmicos. Para o treinamento ficou definida a utilização de uma das máquinas do *cluster* equipada com um processador Intel Core i7 9700K, 64 gigabytes de memória RAM e uma placa GPU NVIDIA Titan-Xp, que possibilitou todo o processo de treinamento e testes.

Para o carregamento dos modelos e utilização do sistema, já tendo sido feito o treinamento, não seria mais necessário um poder de processamento tão robusto, portanto surgiu a ideia de utilizar um *cluster* do Kubernetes, também já disponível no laboratório, contendo placas gráficas menos potentes onde seriam alocados cada um dos módulos em forma de *containers*, deixando toda a tarefa de gerenciamento de recursos para a ferramenta.

Apesar de funcionar de forma bem automatizada, o Kubernetes, além dos conhecimentos específicos da ferramenta, também necessita de outros conhecimentos como Linux e Docker. Embora estas ferramentas tenham uma curva de aprendizado um tanto lenta, uma vez com

tudo configurado e rodando, o Kubernetes gerencia os recursos do sistema, desde a reinicialização automática em caso de erros, até a replicação de instâncias caso assim seja configurado o *pod*. O diagrama completo de arquitetura utilizada no trabalho é mostrado na Figura 5, onde pode ser visto em amarelo os tópicos do kafka e lilás os módulos do sistema, que serão detalhados nas próximas seções.

Figura 5 – Arquitetura completa do sistema desenvolvido.



Fonte: Autoria própria (2022).

3.2 Módulos

Nessa seção são apresentados os módulos desenvolvidos para o sistema.

3.2.1 Rastreamento de pessoas

Um dos grandes desafios do projeto, além da identificação de uma ou mais pessoas em um dado *frame*, uma vez que se trata de vídeos, é como saber se uma pessoa detectada em presente *frame* já foi detectada em *frames* anteriores. Neste contexto que o módulo de rastreamento de pessoas, ou *people tracking*, entra em cena. Em linhas gerais, este módulo trata primeiramente da identificação das pessoas, atribuindo a cada uma delas uma identificação única, além de conseguir fazer a predição da posição de cada uma delas em *frames* subsequentes, de forma a seguir atribuindo esta identificação única ao longo dos *frames*.

Para realizar esta tarefa de identificação de pessoas foi utilizado o YOLOv5. Ele permite a detecção de múltiplos objetos (e pessoas) com uma performance muito boa, tanto em tempo de execução quanto em assertividade. Para a execução das detecções foi usado o modelo pré-treinado no COCO (*Commom Objects in Context*) dataset[1] e foi escolhida a classe *"person"*. Portanto, este módulo foi configurado para detectar apenas pessoas.

Para realizar o rastreamento (*"tracking"*) propriamente dito foi utilizado o StrongSORT. Este módulo possui um desempenho muito bom e atua até em cenários em que o detector não funciona. Por exemplo, quando uma pessoa passa por trás de algum objeto como uma árvore, ainda assim o rastreamento tem uma boa chance de identificar esta pessoa mesmo durante a transição.

Importante ressaltar que este módulo é o primeiro do *pipeline* de processamento do sistema, portanto, ele já é o gerador dos dados, ou seja, dado um vídeo, diretório com *frames* ou até mesmo uma *webcam* como entrada, o mesmo faz as devidas conversões dos dados possibilitando o processamento posterior. Além disto, após um *frame* de vídeo ser processado, a saída obtida são as imagens já cortadas de cada uma das pessoas presentes no *frame* com os respectivos números de identificação. Na Figura 6 é mostrado um exemplo de um momento da execução do rastreamento, tendo sido identificados os dois *bounding-boxes* das pessoas presentes no vídeo, junto com a identificação da pessoa, seguida da classe (que, no caso, é sempre *"person"*) e a medida de confiança da detecção desta classe.

Figura 6 – Exemplo de execução do rastreamento de pessoas.

Fonte: Autoria própria (2022).

[1] https://cocodataset.org/

3.2.2 Detecção e reconhecimento facial (Detrec)

Mesmo com toda a tecnologia utilizada no módulo de rastreamento de pessoas, incluindo o StrongSORT, experimentos preliminares mostraram que em alguns casos duas pessoas eram vinculadas a uma única identificação ou, de forma ainda mais frequente, eram atribuídas duas ou mais identificações únicas a uma mesma pessoa. Uma vez que isto poderia levar a diversos problemas nos estágios seguintes do *pipeline*, foi necessário uma adaptação no sistema, incluindo um módulo de reconhecimento facial na saída do rastreamento, de forma a validar a informação passada pelo mesmo.

Para iniciar todo o processamento deste módulo, foi feita a parte de detecção propriamente dita e, para isto, foram utilizadas duas abordagens: utilizando a função *detect_face* da biblioteca cvlib (PONNUSAMY, 2018) e, novamente, utilizando o YOLOv5 na sua especialização para faces. Embora ambas as abordagens tenham trazido resultados interessantes, o primeiro também trouxe alguns falsos positivos, ou seja, detectou faces onde não existiam de fato. Este problema que foi logo sanado utilizando o YOLOv5-Face com os devidos ajustes.

Após assegurar a correta detecção da pessoa e com a obtenção do *bounding-box* da face, foi aplicada uma função de corte na imagem na posição deste *bounding-box*. Desta maneira foi obtida a imagem contendo apenas a face da pessoa, como mostrado na Figura 7.

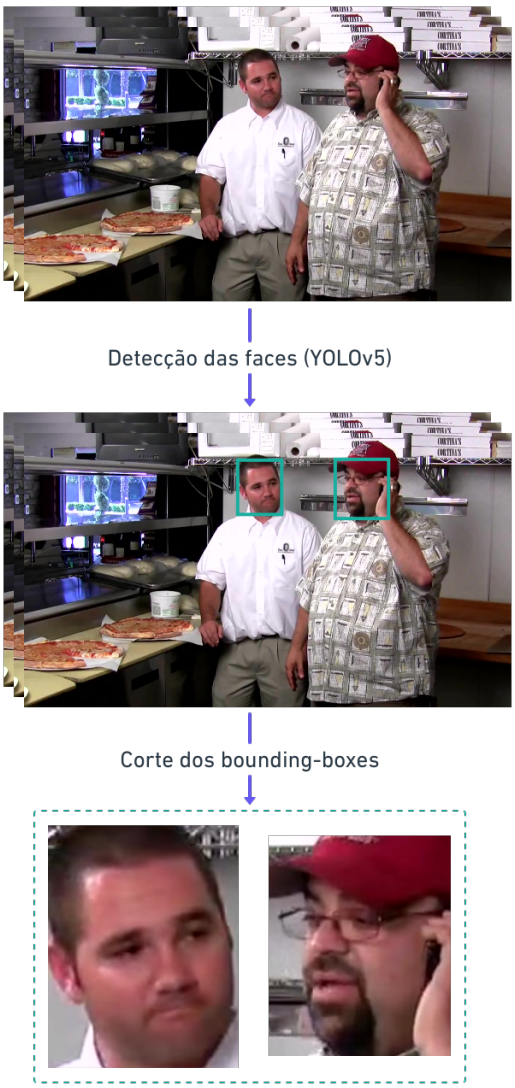
Uma vez obtida a imagem contendo a face de cada pessoa, pode-se passar para o próximo passo do *pipeline*, que é o reconhecimento facial. Para isto, antes de tudo, deve ser feito o alinhamento das faces que se deseja verificar a semelhança, ou seja, remover o máximo possível do ruído de fundo (*background*), que pode causar uma grande divergência no reconhecimento. Este alinhamento foi feito utilizando a biblioteca Dlib (KING, 2009), que utiliza o extrator de *features* HOG e o classificador SVM para, dada uma imagem de face, remover este *background* indesejado e alinhar os olhos e lábios, conforme mostrado na primeira etapa da Figura 8.

Com as faces já alinhadas, foi realizada a vetorização das mesmas, que é a extração de características faciais com o modelo do OpenFace nn4.small2.v1 [2], pré-treinado a partir dos *datasets* FaceScrub (NG; WINKLER, 2014) e CASIA-WebFace (YI *et al.*, 2014), fazendo com que cada imagem de face se transforme em um vetor de 128 dimensões. A partir destes vetores, pode-se utilizar uma função para calcular a distância entre estes vetores, como a distância Euclidiana (Equação 1) e da similaridade de cosseno (Equação 2). Ambas foram testadas nesta fase do desenvolvimento, porém a última obteve resultados significativamente melhores do que a primeira e, por este motivo, foi a escolhida. O processo de reconhecimento facial é mostrado na Figura 8.

$$dist = ||(vet1 - vet2)|| = \sum (vet1 - vet2)^2 \quad (1)$$

[2] https://cmusatyalab.github.io/openface/models-and-accuracies/

Figura 7 – Processo de detecção facial e corte da imagem.



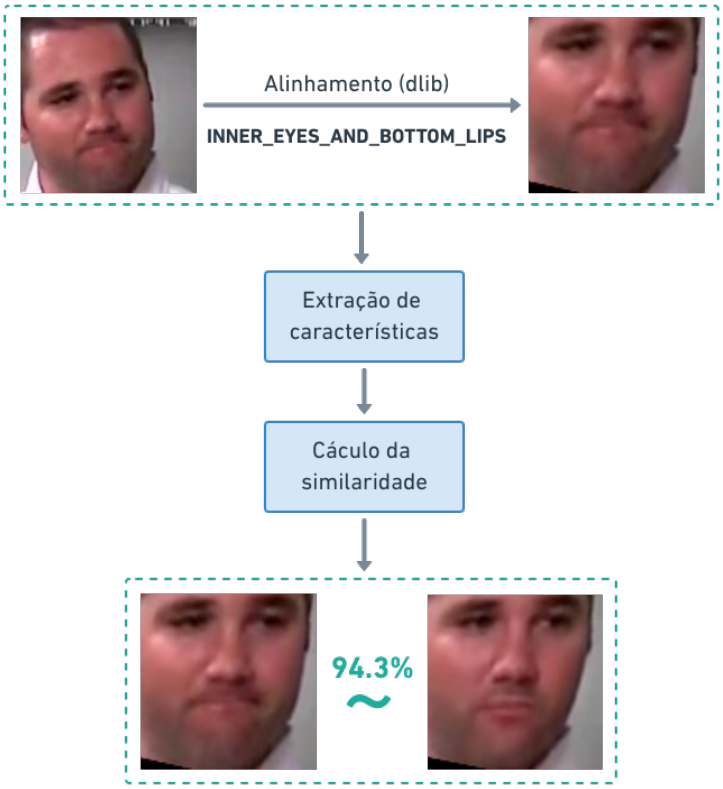
Fonte: Autoria própria (2022).

$$cos(\theta) = \frac{vet1 \cdot vet2}{\sqrt{vet1 \cdot vet2}\sqrt{vet1 \cdot vet2}} \tag{2}$$

A partir deste ponto o módulo está pronto para receber imagens de pessoas e passar pelo processo de re-identificação, ou seja, verificar se ela já foi detectada anteriormente ou não. As imagens das pessoas são vetorizadas e estes vetores são salvos em tempo de execução em um dicionário para ser comparado posteriormente. Neste dicionário, a chave é a identificação da pessoa. Esta abordagem previne erros no processo de re-identificação, conforme ilustrado no diagrama da Figura 9.

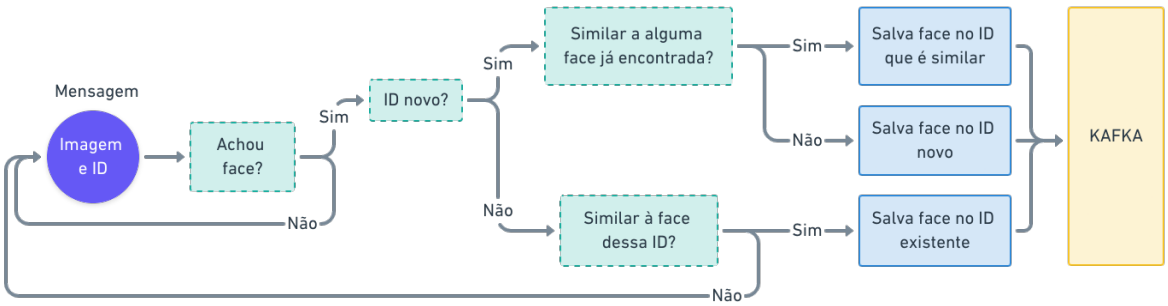
Conforme proposto, o processo de re-identificação requer um alto custo computacional, tanto em memória quanto em velocidade de processamento. Então, foram tomadas algumas medidas para mitigar esta questão. A primeira delas foi utilizar um estrutura de dados de fila com tamanho fixo de forma a salvar apenas uma quantidade fixa de vetores de faces, assim,

Figura 8 – Processo de reconhecimento facial.



Fonte: Autoria própria (2022).

Figura 9 – Processo de re-identificação de pessoas.



Fonte: Autoria própria (2022).

sempre que um novo vetor for inserido e este valor limite for ultrapassado, o vetor mais antigo é eliminado, reduzindo o número de vetores a serem comparados e minimizando o processamento. A segunda medida foi utilizar uma *timer thread* que, a cada estouro de tempo, limpa todas as identificações e todos vetores salvos localmente, já que não são necessários para intervalos longos de tempo. Para esta limpeza dos vetores é utilizado um tempo padronizado de 1 minuto, parametrizado no módulo, o que torna fácil o ajuste a cada *deploy* do mesmo.

É importante ressaltar que também foi testado um outro método baseado em álgebra para fazer a média dos vetores a cada inserção de forma a salvar apenas um vetor. Entretanto, os resultados foram inferiores à abordagem inicial. Em particular, quando há uma taxa de *frames* baixa, uma variação muito grande de posição da face pode afetar a similaridade com o vetor salvo e apresentar um falso negativo. Por esta razão este método não foi utilizado.

Outro ponto relevante é que este módulo também serve como um filtro, uma vez que nem todos os *frames* provenientes do módulo de rastreamento são processados, posto que, ao verificar que não possui face em um dado recorte de pessoa (pessoa de costas, por exemplo) ou possui mais de uma face (prejudicando os resultados do resto do sistema), este módulo apenas pula este *frame*, poupando processamento tanto deste módulo quanto dos módulos subsequentes.

3.2.3 Classificação de grupo de idade

3.2.3.1 Criação do *dataset*

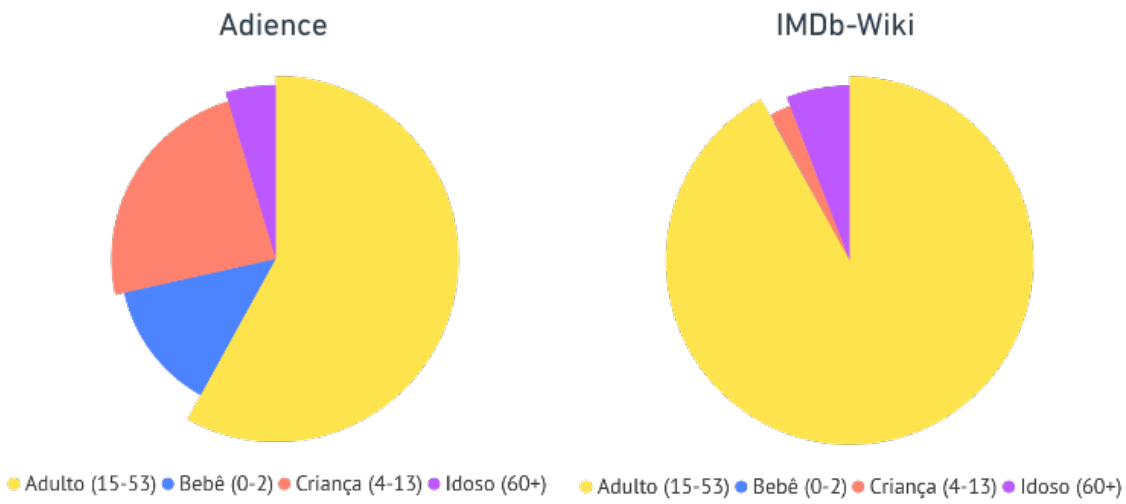
Antes de realizar o treinamento de uma RNC é necessário escolher um *dataset* adequado para o treinamento e teste do modelo. Foram escolhidos dois *datasets* já bastante utilizados na literatura para realizar o treinamento: o Adience *dataset* (HASSNER *et al.*, 2015) e o IMDb-Wiki *dataset* (ROTHE; TIMOFTE; van Gool, 2018).

O Adience *dataset* possui cerca de 26.500 imagens de rostos de uma antiga rede social chamada Flickr. O IMDb-Wiki Dataset, por sua vez, possui consideravelmente mais imagens, chegando a ter 100.000 imagens de pessoas famosas do mundo artístico presentes no site do IMDb bem como de pessoas que se encontram no site Wikipedia. Ambos os *datasets* estão disponibilizados para auxiliar o estudo de classificação de grupo de idade de pessoas em imagens.

A primeira coisa a ser feita foi modificar a forma que os grupos eram organizados nos *datasets* para, além de facilitar, melhorar a quantidade de imagens por grupo, assim, tal modificação resultou nos seguintes grupos: *Baby* ou bebê (0 a 2 anos), *child* ou criança (4 a 13 anos), *adult* ou adulto (15 a 53 anos), *elderly* ou idoso (60 anos ou mais). Ao analisar ambos os *datasets*, foi percebido que embora o IMDb-Wiki tenha significativamente um número maior de imagens, o Adience tem um balanceamento mais interessante (Figura 10), então, por balanceamento ser relevante no processo de treinamento, o Adience *dataset* foi escolhido como principal

para o trabalho. Outro fato interessante é que, como pode-se observar na Figura 11, o Adience possui imagens de pessoas com óculos, pessoas pintadas, utilizando chapéus, em condições adversas de luminosidade, entre outros e isso pode ser considerado ruído e pode dificultar o processo de treinamento e classificação, mas, em contrapartida, auxilia a deixar o mais próximo da realidade possível, onde imagens podem ser capturadas nas mais diversas condições e não necessariamente de forma tão controlada.

Figura 10 – Balanceamento dos *datasets* Adience e IMDb-Wiki.



Fonte: Autoria própria (2022).

Figura 11 – Amostra de imagens do Adience Dataset.

Fonte: Autoria própria (2022).

3.2.3.2 Treinamento e teste

Uma vez escolhido o *dataset* principal do trabalho, foi dado o início do treinamento. Para isso foi utilizada a técnica de *hold-out*, ou seja, o *dataset* foi dividido em treinamento e teste de forma que o conjunto de treinamento foi utilizado especificamente para o treinamento e o de teste foi feito para avaliar como o modelo treinado desempenha com imagens nunca vistas. A proporção utilizada para isto foi 80% do *dataset* para treinamento e 20% para o teste.

As arquiteturas de redes neurais testadas foram: EfficientNet, EfficientNetV2, InceptionV3, MobileNet e VGG16. Quanto aos otimizadores, foram testados o Adam e o RMSprop. Para o treinamento foi utilizada a biblioteca Keras[3] com o Tensorflow[4] como *backend*. O treinamento foi realizado durante 100 épocas, ou até que o *loss* não diminua por 5 épocas consecutivas. Além disto, a cada duas destas vezes em que não tenha ocorrido a diminuição do *loss*, foi reduzida a *learning rate* por um fator de 0,1, de modo a permitir que o modelo possa se ajustar mais facilmente.

Após a divisão e definições foram realizados dois experimentos. O primeiro deles (Tabela 1) foi para avaliar o desempenho das diferentes arquiteturas de redes neurais, fixando o otimizador utilizado. O segundo experimento (Tabela 2) foi feito para avaliar qual otimizador conseguiria obter um resultado melhor. Desta vez foi mantida fixa a arquitetura do modelo que teve o melhor desempenho no primeiro experimento. Para escolher o melhor resultado de cada experimento foi levado em consideração majoritariamente o F1-Score, que é uma métrica que relaciona as métricas *precision* e *recall* a partir da sua média harmônica, ou seja, a mesma considera a proporção de predições relevantes em relação com o total analisado com a proporção de predições relevantes que forma corretamente classificadas.

A partir desta análise em ambos os experimentos foi decidido utilizar a arquitetura Efficientnet em conjunto com o otimizador Adam. A matriz de confusão dos resultados deste modelo é mostrada na Figura 12. Os gráficos de *loss* e *accuracy* correspondentes são mostrados na Figura 13. Nestes gráficos, observando-se a escala e o baixo número de épocas, é possível verificar uma rápida convergência no decorrer do treino.

Tabela 1 – Experimento 1 - Escolha da arquitetura de rede neural.

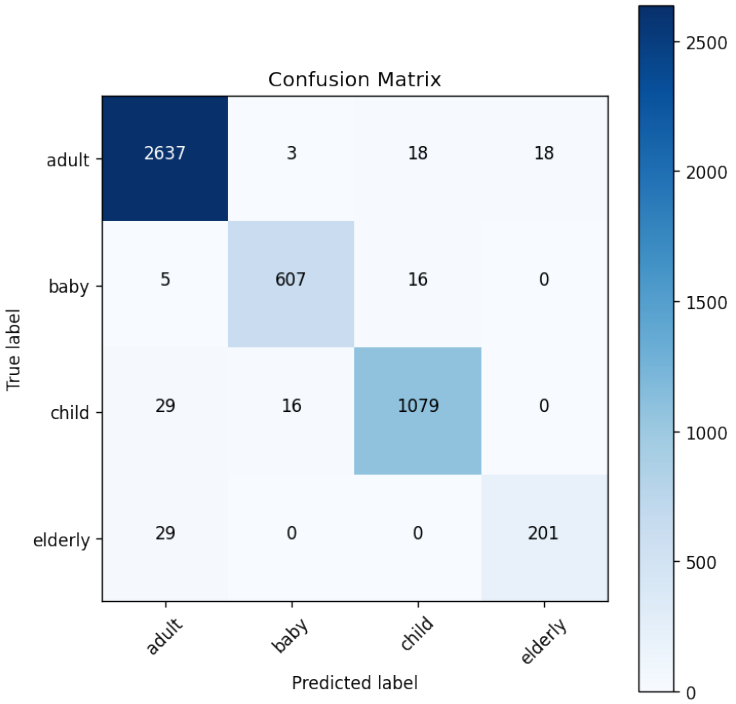
Arquitetura	Otimizador	Accuracy	Precision	Recall	F1-Score
Efficientnet	Adam	0.971	0.958	0.946	0.952
Efficientnet-V2	Adam	0.965	0.949	0.931	0.940
InceptionV3	Adam	0.963	0.948	0.926	0.936
MobileNet	Adam	0.950	0.927	0.950	0.923
VGG16	Adam	0.930	0.918	0.868	0.890

Fonte: Autoria própria (2022).

[3] https://keras.io/

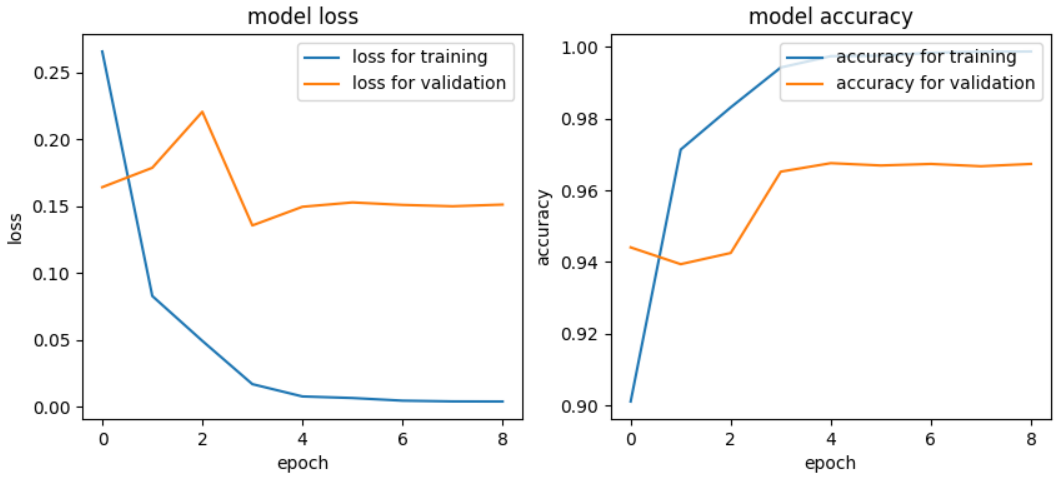
[4] https://www.tensorflow.org/

Figura 12 – Matriz de confusão do melhor modelo treinado.



Fonte: Autoria própria (2022).

Figura 13 – Gráficos de *loss* e *accuracy* do melhor modelo treinado.



Fonte: Autoria própria (2022).

Tabela 2 – Experimento 2 - Escolha do otimizador.

Arquitetura	Otimizador	Accuracy	Precision	Recall	F1-Score
Efficientnet	Adam	0.971	0.958	0.946	0.952
Efficientnet	RMSprop	0.958	0.934	0.929	0.932

Fonte: Autoria própria (2022).

3.2.4 Segmentação de Vestimenta

A especialidade deste módulo é segmentar e classificar as vestimentas de uma determinada pessoa dada uma imagem de corpo inteiro. Esta segmentação consiste em atribuir, a nível de pixel da imagem, uma classe referente à vestimenta identificada. Isto levanta o requisito de a câmera estar posicionada de forma a capturar a pessoa por completo.

Para esta tarefa, foi utilizado o *framework* EPYNET (INÁCIO; LOPES, 2020) desenvolvido no LABIC[5] da UTFPR, câmpus Curitiba. Neste projeto o *framework* EPYNET foi utilizado para realizar apenas a segmentação de vestimentas. É importante ressaltar que, embora o projeto original utilize o SSD para detectar uma pessoa em uma imagem e então realizar o corte desta mesma, no presente trabalho esta função não foi necessária, uma vez que a pessoa já foi previamente detectada no módulo de rastreamento de pessoas. De fato o módulo de segmentação de vestimenta já recebe a imagem cortada no *bounding-box* da pessoa e, assim, economiza tempo de processamento ao longo do *pipeline*, reduzindo a latência geral do sistema.

Outra alteração realizada foi na minimização de falsos positivos. Em alguns testes a segmentação apresentava vestimentas com poucos *pixels* detectados e, ao verificar a segmentação, foi perceptível o erro nesta etapa. A solução para tal problema foi colocar um valor mínimo de *pixels* para as vestimentas detectadas, removendo os falsos positivos e deixando os resultados mais consistentes.

3.2.5 Persistência de dados

Quando se trata de persistência de dados, a primeira coisa que vem em mente é banco de dados. Assim, o banco de dados escolhido para este trabalho foi o NoSQL MongoDB devido à sua praticidade de conexão bem como o seu modelo flexível de documentos e o seu desempenho ao lidar com grande fluxo de dados.

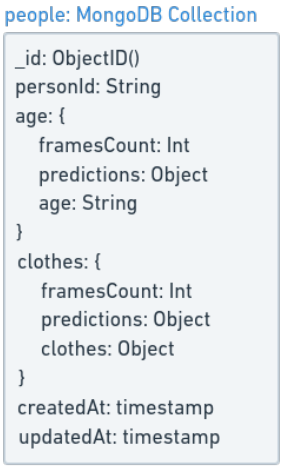
Por ser uma análise *frame* a *frame* e se tratar de um sistema distribuído, surge a necessidade de um módulo especializado no tratamento e na persistência de dados. Um exemplo prático do trabalho que retrata tal necessidade é: uma pessoa com identificação de número 14 é classificada como criança em um dado *frame*. Porém, em todos os outros *frames* é classificada como adulta, desta forma o módulo tem que ser capaz de fazer uma correção e classificar a

[5] http://labic.utfpr.edu.br

pessoa em questão como adulta e não como criança. Dado este exemplo, algumas abordagens foram realizadas com o intuito de melhorar a confiança dos resultados.

O banco de dados foi estruturado de forma a existir uma coleção de pessoas e, nesta coleção, cada documento representa uma pessoa distinta (Figura 14), com a sua identificação única bem como a suas "possíveis" classificações. Tendo isto em mente, fica claro que uma simples contagem dos documentos salvos retorna a contagem de pessoas, deixando bem resolvido um dos objetivos do trabalho.

Figura 14 – Modelo dos documentos do MongoDB



Fonte: Autoria própria (2022).

Quanto à classificação de grupo de idade e da segmentação e classificação de vestimenta, por se tratarem de modelos preditivos que não têm taxa de acerto absoluta, basicamente salva-se a contagem de cada uma das ocorrências de cada predição e, a cada evento recebido pelo módulo, a contagem é atualizada no banco de dados como a classificação predita, obtida a partir do máximo dos valores registrados. Especificamente para a segmentação de vestimentas, é feita uma separação em classes como Superior (Camisas, casacos, moletons e afins), Inferior (Calça, saia, shorts e afins), Única (Vestido, macacão e afins) e Acessórios (Bonés, colares, meias, cintos e afins). Desta forma é possível excluir casos com grande probabilidade de não acontecer como, por exemplo, o uso de vestido e saia ao mesmo tempo.

Alguns casos de roupas de uma mesma classe são permitidos como, por exemplo, usar camisa e casaco ao mesmo tempo. Para contabilizar isto, desconsiderando os erros de predição, usa-se uma margem de segurança de 75%. Por exemplo: a pessoa de identificação 14 teve camisa detectada em 19 de 20 *frames* processados. Porém, nestes 20 *frames*, em apenas 3 foi detectado jaqueta. Portanto, é classificada apenas a vestimenta camisa. Por outro lado, se além da camisa fosse classificada jaqueta em 15 dos 20 *frames*, a probabilidade desta pessoa estar usando camisa e jaqueta ao mesmo tempo é alta. Assim, a pessoa é classificada como portando camisa e jaqueta. Nesta mesma linha de raciocínio é feita a análise de vestido e jaqueta. Especificamente para acessórios, cada um dos acessórios classificados são validados dentro

uma margem de 40%, dos *frames* processados. Se a contagem for muito baixa, a probabilidade de ter sido um falso positivo é alta e, portanto, este acessório é desconsiderado.

Vale enfatizar que, respeitando a Lei Geral de Proteção de Dados (LGPD), nenhuma imagem ou qualquer informação que vincule a pessoa aos dados são persistidas, são armazenadas apenas as informações utilizadas para calcular o desempenho do sistema.

3.2.6 Interface

Foi necessário a criação de um módulo de interface para apresentar os resultados obtidos de forma clara e intuitiva para o usuário final. Assim, o módulo de interface foi dividido em *Front-end* e *Back-end*, que serão detalhados a seguir.

3.2.6.1 *Back-end*

O *back-end* é responsável pelo o que ocorre “por trás dos panos” na interface, ou seja, é onde um servidor executa todas as operações de controle e serviços, como banco de dados e APIs (*Application Programming Interface*).

Neste trabalho foi utilizada a linguagem Python para a execução do *back-end* e para a API como um todo foi utilizada a biblioteca FastAPI [6], permitindo a criação das rotas e funções necessárias para o funcionamento da interface.

No *back-end* há a conexão com o banco de dados MongoDB, de onde são obtidos os dados processados pelos módulos anteriores. Além disto, para simplificar as coisas, foram criadas apenas duas rotas. A primeira é para a contagem de pessoas, incluindo: Total, do dia, média por dia e total por mês. A segunda é para retornar os dados gerais de cada documento para, assim, serem mostrados pelo *front-end*.

Um detalhe importante é que o *back-end* já retorna os dados tratados e formatados em *JavaScript Object Notation* (JSON) de modo a facilitar o uso dos mesmos pelo *front-end*, além de facilitar a manutenção, precisando alterar as operações apenas no *back-end* em caso de mudanças. Esta etapa de estruturação e tratamento dos dados é um tanto quando complexa pois a mesma requer consultas ao banco de dados NoSQL. Tais consultas incluem contagens, buscas e agregações para viabilizar as métricas desejadas. Ademais, foi necessário criar um *middleware* para configurar o *Cross-Origin Resource Sharing* (CORS), que atua diretamente sobre o protocolo *Hypertext Transfer Protocol* (HTTP). Em linhas gerais, o CORS permite uma forma segura de requisitar dados entre o navegador que apresenta as operações ao usuário e o servidor que fornece os dados.

Devido à importância do *back-end* para levar os dados ao usuário na ponta, ele está inserido no Kubernetes, pois, ao aumentar a carga de acesso, o mesmo pode ser configurado

[6] https://fastapi.tiangolo.com/

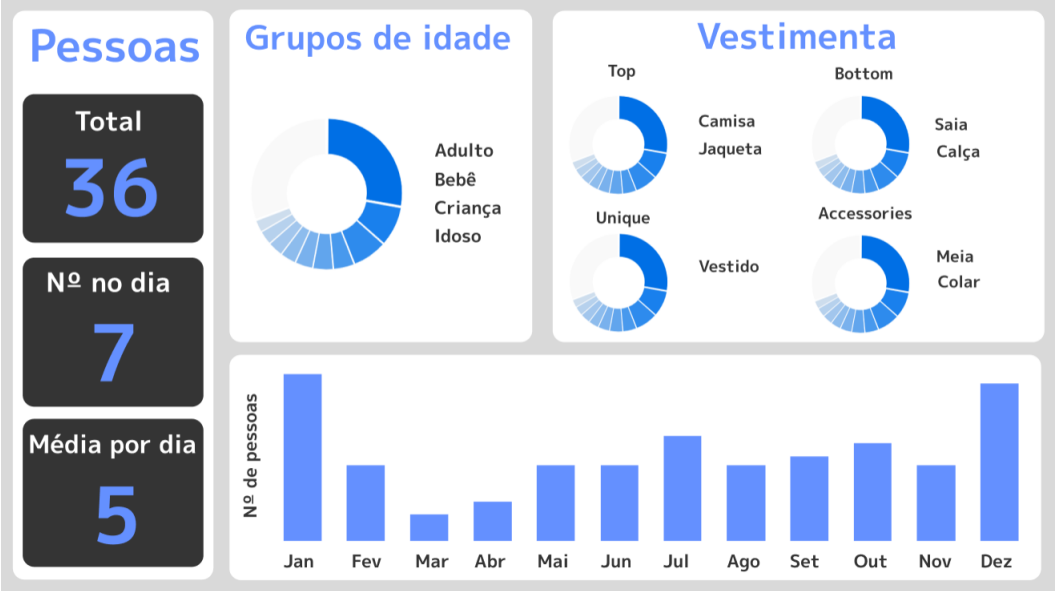
para realizar o *load-balancing*, dividindo a carga de processamento para diversos *containers* de forma transparente ao usuário.

3.2.6.2 *Front-end*

O *front-end* tem a tarefa de materializar os resultados, ou seja, apresentar os gráficos e números de forma intuitiva para o usuário final compreender com facilidade o resultado do processamento do *stream* de vídeo.

Antes de começar o desenvolvimento propriamente dito, foi utilizada a ferramenta Figma[7] para idealizar e realizar o protótipo do design gráfico com um possível visual para o *front-end* (Figura 15).

Figura 15 – Protótipo do design gráfico do *front-end*.



Fonte: Autoria própria (2022).

O desenvolvimento *front-end* foi feito na linguagem JavaScript, utilizando o ReactJS[8] e o MaterialUI[9], desenvolvido pela Google. Em linhas gerais, a parte visual possui uma barra lateral com os *"big numbers"*, além de *cards* com gráficos em forma de *doughnut* com as informações de grupos de idade e vestimentas utilizados e um gráfico de barra com o número de pessoas agrupados por mês. Além disto, para realizar as requisições para o *back-end* foi utilizada a biblioteca Axios[10], possibilitando chamar as rotas necessárias para obter as informações processadas.

De forma similar ao *back-end*, o *front-end* também está inserido no Kubernetes para ter um gerenciamento mais automatizado, evitando que o usuário na ponta seja prejudicado.

[7] https://www.figma.com/
[8] https://reactjs.org/
[9] https://mui.com/pt/
[10] https://axios-http.com/

Somado a isto, durante o desenvolvimento, foi necessária a preocupação com a disposição das informações e quais gráficos seriam usados, justamente para atingir o objetivo de deixar o *dashboard* simples e intuitivo para o usuário.

Como mencionado anteriormente, o *back-end* já consegue fornecer os dados de forma estruturada para a utilização no *front-end*. Desta forma, ao realizar requisições HTTP para as rotas do *back-end*, as informações já são retornadas em JSON e utilizadas nos componentes do React. Quanto à atualização dos dados no *dashboard*, o mesmo é realizado a cada atualização da página, forçando uma nova requisição ao *back-end* e atualizando os componentes da interface, uma vez que o banco de dados é atualizado em tempo real com os dados produzidos pelos outros módulos.

3.3 Comunicação entre os módulos

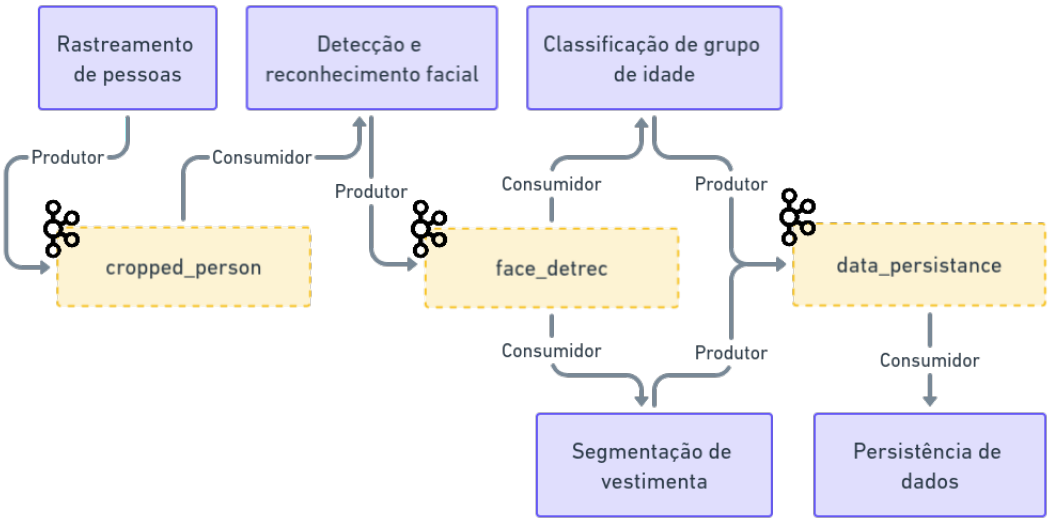
Um dos pontos de grande importância neste trabalho é a comunicação entre os módulos. Considerando que estes são distribuídos, isto é, não estão utilizando os recursos de uma mesma máquina, uma boa orquestração entre os módulos é essencial para o funcionamento adequado do *pipeline*. Para esta finalidade foi utilizada a ferramenta Apache Kafka, que neste trabalho atua como um gerenciador de eventos, assim, todas as mensagens enviadas ou recebidas por um módulo necessariamente devem passar por ela. Os módulos deste trabalho podem ser classificados como produtores, consumidores ou ambos simultaneamente. Isto é, um módulo pode apenas enviar mensagens para o Kafka (produzindo dados), pode apenas receber mensagens do Kafka (consumindo dados), ou pode enviar e receber mensagens do Kafka (produtor e consumidor de dados). Cada uma destas relações pode ser observada no diagrama da Figura 16, onde os blocos das extremidades (lilás) são os módulos e os blocos internos (amarelos) são os tópicos do Kafka, que são as categorias únicas da ferramenta que organizam as mensagens produzidas e consumidas na ferramenta.

Para orquestrar as mensagens foram criados tópicos e grupos de consumidores. Os tópicos servem para organizar as mensagens, um produtor escreve as mensagens no tópico e um consumidor lê as mensagens do tópico. Os grupos de consumidores, são grupos de módulos que compartilham uma mesma identificação. Apesar de parecer simples, esta definição é de muita importância para o trabalho.

De forma a otimizar a execução do *pipeline*, foi decidido que os módulos de classificação de grupo de idade e segmentação de vestimenta deveriam ser executados em paralelo, ou seja, estes módulos devem ser executados ao mesmo tempo de forma independente. É nesta parte que a relevância do Kafka e dos grupos de consumidores entram em cena.

Para ilustrar o funcionamento do Kafka, suponha a seguinte situação hipotética em que um módulo M envia um evento E1 e um evento E2 para um tópico T e, além disto, existem dois consumidores C1 e C2. De forma geral, pode-se ter dois cenários para o recebimento de um evento por um consumidor, observados na Figura 17:

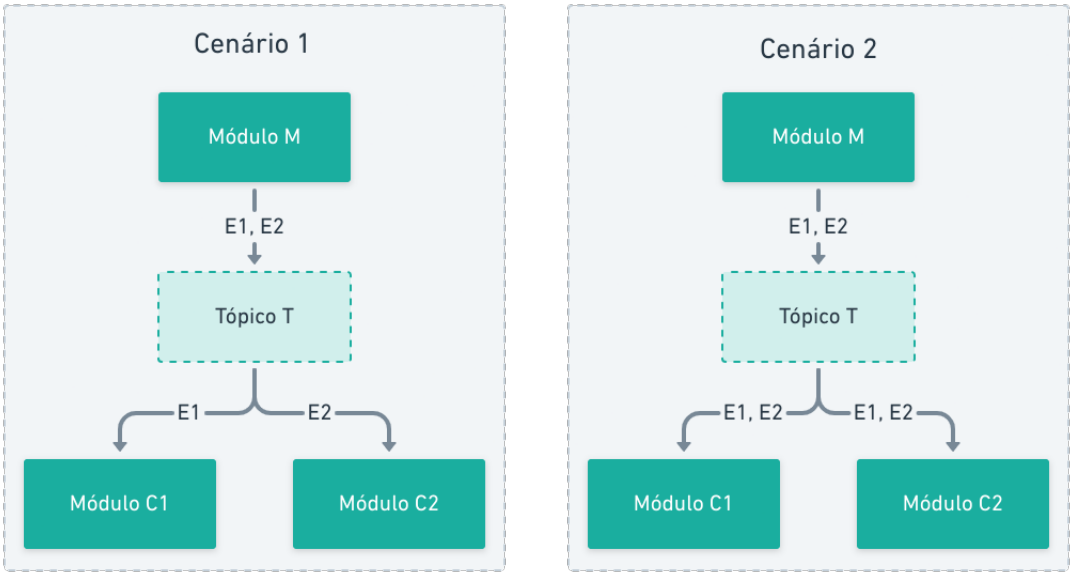
Figura 16 – Diagrama de módulos produtores e consumidores.



Fonte: Autoria própria (2022).

- O primeiro cenário acontece se C1 e C2 possuírem a mesma identificação de grupo de consumidores. Neste caso o Kafka realiza uma espécie de *Load-Balancing*, ou seja, C1 recebe apenas E1 e C2 recebe apenas E2, por exemplo.
- O segundo cenário ocorre se C1 e C2 possuírem identificações de grupo de consumidores diferentes. Neste caso tanto C1 quanto C2 recebem E1 e E2.

Figura 17 – Representação dos cenários de grupos de consumidores exemplificados.



Fonte: Autoria própria (2022).

Para o caso do presente trabalho, para conseguir executar em paralelo os módulos de classificação de grupo de idade e o de segmentação de vestimenta, o módulo de detecção e

reconhecimento facial foi configurado para enviar a mensagem para o seu respectivo tópico (face_detrec). Então, o módulo de classificação de grupo de idade e o módulo de segmentação de vestimenta lêem as mensagem deste tópico (face_detrec), porém, cada um deles com identificação de grupo de consumidores distintos. Desta forma ambos os módulos recebem a mesma mensagem e executam o seu processamento de forma independente sem impactar na persistência de dados, pois essa tarefa é de competência do módulo especializado nisto, como mencionado anteriormente.

Outro ponto relevante é que, com a escolha do Kafka como ferramenta de comunicação entre os módulos, o sistema se torna tolerante a falhas individuais dos módulos, uma vez que o Kafka utiliza uma semântica chamada *"at-least once delivery"*, ou seja, caso um dos módulos seja interrompido, ao ser reiniciado, o mesmo recebe as mensagens que não conseguiram ser entregues no seu período de falha, garantindo assim que as mensagens cheguem ao consumidor.

4 RESULTADOS

Apesar de ser uma tarefa complexa principalmente por se tratar de módulos tão distintos inseridos em um sistema distribuído, no presente capítulo são apresentados os resultados do trabalho como um todo.

Durante a execução do sistema ficou claro que os resultados podem variar bastante em função da qualidade do vídeo, posição da câmera, iluminação, fluxo de pessoas, entre outros fatores que se enquadram nas chamadas condições de contorno. Neste sentido, para apresentar tais resultados, foram utilizados dois cenários: (a) ambiente controlado e (b) ambiente não-controlado, e nestes cenários são variadas as condições de contorno. O *hardware* utilizado nos testes foi o *cluster* do Kubernetes (Tabela 3), que cuidou do gerenciamento destes recursos. Todas as máquinas utilizadas estavam executando o sistema operacional Ubuntu versão 18.04.6 LTS.

Tabela 3 – *Hardware* utilizado nos testes.

Máquina	CPU	RAM	GPU
CPU10	Intel(R) Xeon(R) E5-2450 v2	32GB	Não disponível
GPU10	Intel(R) Core(TM) i7-5820K	32GB	NVIDIA GeForce RTX 2060
GPU11	Intel(R) Core(TM) i7-6700	16GB	NVIDIA GeForce RTX 2060

Fonte: Autoria própria (2022).

No experimento de ambiente controlado, alguns vídeos foram coletados da internet e dois deles foram gravados pelos próprios autores do presente trabalho. Neste cenário as condições são mais favoráveis, com boa iluminação, boa qualidade do vídeo, e baixo fluxo de pessoas.

Por outro lado, no experimento de ambiente não-controlado há uma condição ruim de iluminação, bem como um alto fluxo de pessoas em múltiplas direções e com bastante sobreposição. Este vídeo foi obtido de filmagem no corredor que interliga os blocos A, B e C, a partir do bloco D, da UTFPR sede Centro.

4.1 Arquitetura e comunicação

A arquitetura e o método de comunicação utilizados foram adequados para o desenvolvimento do trabalho, tanto na parte de treinamento e validação dos módulos de forma separada, quanto para a execução do sistema como um todo.

A ferramenta Kafka conseguiu suportar toda a carga de um fluxo de dados e eventos de forma eficaz e com latência de comunicação muito baixa, ou seja, abaixo de 1 milissegundo. Isso faz sentido, visto que o mesmo é feito para suportar operações em tempo real. Além disto, a decisão de processar os módulos de classificação de grupo de idade e segmentação de vestimenta em paralelo foi bem sucedida, ajudando bastante no processo.

Outra ferramenta que foi bem sucedida no trabalho foi o Kubernetes, permitindo que os módulos sejam executados em *containers*. O Kubernetes garante robustez ao sistema, principalmente caso haja necessidade de reinicialização devido a falhas. Portanto, a arquitetura resultante do sistema bem como a comunicação inter-módulos foi mantida conforme a Figura 5.

4.2 Módulos

4.2.1 Rastreamento de pessoas

Com a utilização do algoritmo StrongSORT em conjunto com o YOLOv5, este módulo teve resultados muito bons em velocidade, visto que no pior caso testado o tempo total foi de 75 milissegundos, significativamente abaixo de alguns dos módulos subsequentes. Além disso, tanto o YOLOv5 quanto o StrongSORT são o estado da arte nas suas respectivas funções. Os tempos de cada etapa podem ser observados na Tabela 4. Porém, quanto à assertividade no rastreamento, apesar desta solução ter parecido suficiente para tratar de toda a identificação de pessoas bem como a sua contagem, o resultado de alguns testes não foi o esperado. Em particular, no momento em que pessoas entram e saem de cena ou quando há o cruzamento de pessoas, o algoritmo algumas vezes se comportou de forma oscilatória. Em algumas situações o algoritmo identificou uma mesma pessoa como múltiplas pessoas distintas ou, ainda, múltiplas pessoas distintas como uma mesma pessoa. Isto é mostrado na contagem de pessoas da Tabela 5, onde a contagem aproximada foi realizada de forma visual. É importante ressaltar que este problema não tira de forma alguma o mérito ou os resultados positivos deste módulo, sendo um módulo primordial para o funcionamento correto do sistema.

Tabela 4 – Tempos médios de processamento para detecção e rastreamento de pessoas.

N° de pessoas no frame	Tempo médio YOLOv5 (ms)	Tempo médio StrongSORT (ms)	Tempo total (ms)
1	10	27	37
5	10	39	49
12	10	65	75

Fonte: Autoria própria (2022).

Tabela 5 – Contagem de pessoas no rastreamento.

Experimento	N° aproximado de pessoas	N° predito de pessoas	Erro
Controlado	18	33	83,33%
Não controlado	163	735	350,92%

Fonte: Autoria própria (2022).

Outro ponto importante de ser mencionado é que, por realizar uma análise *frame* a *frame* no mesmo processo, este módulo é o único que não pode ser distribuído e escalável de forma

simples, ao contrário dos outros módulos. Consequentemente, uma das GPUs presentes na infraestrutura do projeto ficou dedicada somente a esta função, com o objetivo de oferecer o máximo de desempenho, buscando diminuir, assim, a latência total do *pipeline*.

4.2.2 Detecção e reconhecimento facial

Para iniciar os testes e o presente módulo fazer sentido no que diz respeito à re-identificação de pessoas, além do filtro, foi necessário introduzir o conceito de "Pessoa Identificável" que, em linhas gerais, é aquela pessoa cuja face é visível. Então, se uma determinada pessoa se encontrar de costas, por exemplo, a mesma não será identificada pelo módulo.

Retomando o problema do módulo de rastreamento, onde são atribuídas múltiplas identificações a uma mesma pessoa ou onde são atribuídas múltiplas pessoas a uma identificação, este módulo cuidou com sucesso da parte de validação destas atribuições. Um exemplo disto foi observado durante os testes, como mostrado na Tabela 6, onde, após a re-identificação das pessoa, a contagem ficou muito mais próxima da quantidade aproximada visualmente, diminuindo o erro significativamente.

Tabela 6 – Comparação da contagem antes e após a re-identificação das pessoas.

Experimento	N° aproximado de pessoas identificáveis	Contagem antes da re-identificação	Contagem após re-identificação	Erro após a re-identificação
Controlado	18	33	18	0%
Não controlado	63	735	50	26%

Fonte: Autoria própria (2022).

O tempo de processamento do presente módulo também é considerado bom, ainda mais levando em conta que o mesmo pode ser replicado e escalado de forma simples devido à arquitetura proposta. Os resultados são mostrados na Tabela 7.

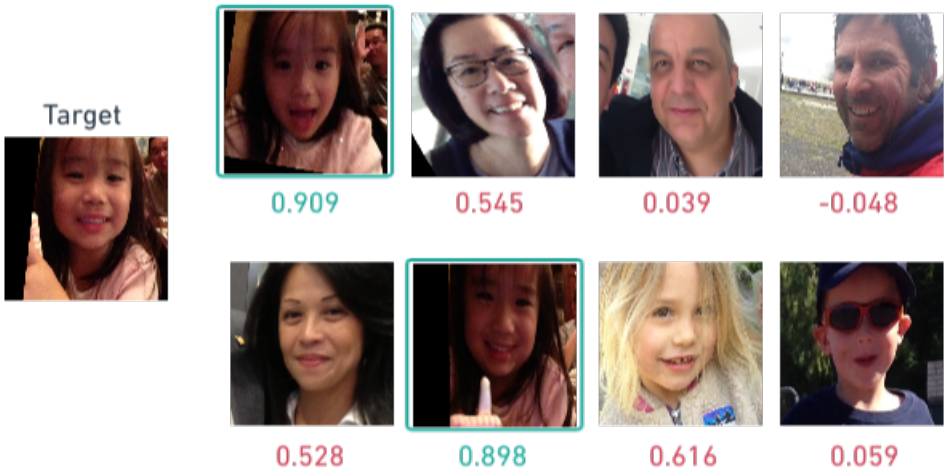
Tabela 7 – Tempo médio de processamento da re-identificação de pessoas.

Número de faces	Tempo médio (ms)
1	179
0 ou >1	<10

Fonte: Autoria própria (2022).

Ao realizar um teste unitário da função de reconhecimento facial foram obtidos resultados consistentes, como mostrado na Figura 18, onde um *threshold* de 0,8 foi configurado para classificar duas faces como similares. A face a ser comparada é a face com o *label "target"* e as outras compõem a amostra onde a face *"target"* é buscada. Abaixo de cada imagem da amostra está o resultado da similaridade cosseno após todo o processo já mostrado anteriormente, considerando que quanto mais próximo de 1, maior a similaridade.

Figura 18 – Exemplo de resultado da etapa de reconhecimento facial.



Fonte: Autoria própria (2022).

Por se tratar de um módulo que demanda elevado custo computacional, foi decidido utilizar a outra GPU restante na infraestrutura do projeto para processar de forma integral este módulo.

4.2.3 Classificação de grupo de idade

Como observado no Capítulo 3, o módulo de classificação de grupos de idade mostrou um desempenho muito bom nos experimentos de treinamento e validação. Nos testes realizados não foi diferente, sendo que o mesmo se mostrou excelente, não errando a classificação de nenhuma das pessoas identificadas tanto em ambiente controlado quanto em ambiente não controlado. No cenário de ambiente não-controlado, por se tratar de um vídeo no corredor da universidade, o número de adultos foi majoritário. No ambiente controlado, por sua vez, foi possível fazer uma variação mais ampla para testar o presente módulo e os resultados em ambos os cenários são mostrados nas Tabelas 8 e 9, respectivamente. Nas tabelas de resultados, para melhorar a compreensibilidade, as classes foram limitadas apenas àquelas presentes ou preditas.

É importante ressaltar que o tempo médio de processamento do módulo ficou em aproximadamente 322 milissegundos, embora o mesmo seja executado em CPU e não em GPU como os módulos anteriores. Para fins de comparação, o tempo de processamento deste módulo em GPU ficou em aproximadamente 219 milissegundos.

Tabela 8 – Resultado de classificação de grupo de idade em ambiente controlado.

Classe	Total Real	Total Predito	*True Positive*	*False Positive*	*False Negative*	*True Negative*	*Precision*	*Recall*	*F1-Score*
Adulto	6	6	6	0	0	12	1.00	1.00	1.00
Bebê	6	6	6	0	0	12	1.00	1.00	1.00
Criança	4	4	4	0	0	14	1.00	1.00	1.00
Idoso	2	2	2	0	0	16	1.00	1.00	1.00
Total	18	18	18	0	0	54			
Média							1.00	1.00	1.00

Fonte: Autoria própria (2022).

Tabela 9 – Resultado de classificação de grupo de idade em ambiente não-controlado.

Classe	Total Real	Total Predito	*True Positive*	*False Positive*	*False Negative*	*True Negative*	*Precision*	*Recall*	*F1-Score*
Adulto	49	49	49	0	0	1	1.00	1.00	1.00
Idoso	1	1	1	0	0	49	1.00	1.00	1.00
Total	50	50	50	0	0	50			
Média							1.00	1.00	1.00

Fonte: Autoria própria (2022).

4.2.4 Segmentação e classificação de vestimenta

Inicialmente surgiram alguns desafios e inconsistências no funcionamento deste módulo, que foram reduzidos significativamente depois de compreender que no *dataset* onde o modelo utilizado foi treinado havia apenas pessoas com o corpo completo aparecendo nas imagens. Assim, ao se adicionar este requisito ao projeto os resultados melhoraram de forma significativa.

Além disto, se tratando de um sistema que processa vídeos, ao fim do desenvolvimento eventuais erros pontuais em *frames* são devidamente corrigidos à medida que novos *frames* são processados. Isto foi discutido anteriormente na descrição do módulo de persistência de dados. Tal procedimento aumenta a confiabilidade e qualidade dos dados armazenados no banco de dados.

Porém, mesmo com todas as melhorias, foram percebidos alguns falsos positivos e falsos negativos quanto às classes de vestimenta, o que seria esperado devido à alta complexidade quando se refere à segmentação de imagens. Contudo, tais erros não impactaram significativamente o resultado do módulo, como mostrado nas Tabelas 10 e 11.

De forma análoga ao módulo anterior, nas tabelas, as classes foram limitadas àquelas presentes ou preditas. Quanto aos valores em si, ressalta-se que o esperado seria o ambiente controlado apresentar resultados mais assertivos do que o ambiente não controlado. Entretanto, no caso controlado, entre os erros haviam bebês que não estavam utilizando roupas que pudessem ser efetivamente classificadas pelo modelo (Figura 19), o que levou a erros nestes casos. O restante dos resultados foram bem similares ao ambiente não-controlado.

Para casos de elementos difíceis de serem detectados, como óculos, cintos e mochilas vistas de frente (classificadas como bolsa), uma vez que aparecem apenas as alças, os resultados foram surpreendentemente satisfatórios.

Tabela 10 – Resultado de segmentação de vestimenta em ambiente controlado.

Classe	Total Real	Total Predito	*True Positive*	*False Positive*	*False Negative*	*True Negative*	*Precision*	*Recall*	*F1-Score*
Camisa	10	11	9	2	1	6	0.82	0.9	0.86
Calça	11	12	11	1	0	6	0.92	1.00	0.96
Shorts	2	2	2	0	0	16	1.00	1.00	1.00
Casaco	2	4	1	3	1	13	0.25	0.5	0.33
Óculos	4	3	3	0	1	14	1.00	0.75	0.86
Calçado	13	15	13	2	0	3	0.87	1.00	0.93
Acessório de cabeça	1	3	1	2	0	15	0.33	1.00	0.50
Vestido	4	4	4	0	0	14	1.00	1.00	1.00
Bolsa	0	1	0	1	0	17	0.00	0.00	0.00
Cinto	1	1	1	0	0	17	1.00	1.00	1.00
Sweater	2	2	2	0	0	16	1.00	1.00	1.00
Total	50	58	47	11	3	137			
Média							0.74	0.83	0.77

Fonte: Autoria própria (2022).

Tabela 11 – Resultado de segmentação de vestimenta em ambiente não controlado.

Classe	Total Real	Total Predito	*True Positive*	*False Positive*	*False Negative*	*True Negative*	*Precision*	*Recall*	*F1-Score*
Camisa	47	48	46	2	1	1	0.96	0.98	0.97
Calça	43	41	41	0	2	7	1.00	0.95	0.98
Shorts	6	4	4	0	2	44	1.00	0.67	0.80
Casaco	4	10	4	6	0	40	0.4	1	0.57
Óculos	18	14	13	1	5	31	0.93	0.72	0.81
Calçado	49	49	49	0	0	1	1.00	1.00	1.00
Acessório de cabeça	0	6	0	6	0	44	0.00	0.00	0.00
Vestido	1	1	1	0	0	49	1.00	1.00	1.00
Bolsa	37	28	27	1	10	12	0.96	0.73	0.83
Cinto	1	1	1	0	0	49	1.00	1.00	1.00
Total	206	202	186	16	20	278			
Média							0.83	0.81	0.80

Fonte: Autoria própria (2022).

Outros erros interessantes e justificáveis aconteceram durante os experimentos. Entre eles, duas pessoas que estavam com a mochila/bolsa de forma muito difícil de ser visualizada, como mostrado na Figura 20.

Este módulo foi configurado para ser processado em CPU, uma vez que as GPUs já foram alocadas para processar módulos anteriores e o tempo médio de processamento foi aproximadamente 700 milissegundos.

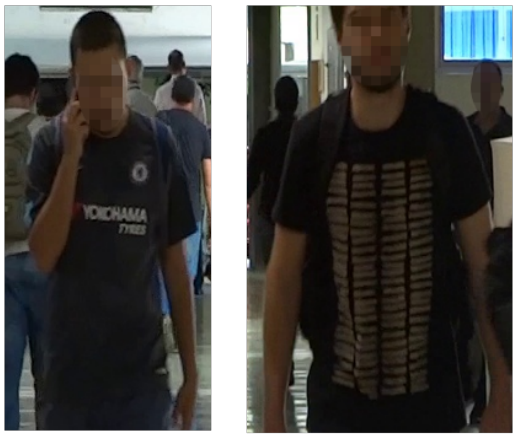
4.2.5 Persistência de dados

Sendo de extrema importância para possibilitar a recuperação e visualização dos dados processados, o módulo de persistência de dados conseguiu cumprir a sua função de conectar no banco de dados MongoDB e salvar com sucesso as informações referentes a cada pessoa presente em um dado *frame*. Além disto, a forma como os dados foram estruturados ajudou de forma direta na contagem, uma vez que cada identificação única de pessoa corresponde a um documento do MongoDB. Desta forma, a contagem destes documentos corresponde à contagem de pessoas, podendo ser combinado com outros filtros, tais como data de criação do documento, atualização, entre outros.

Figura 19 – Exemplo roupas que não são classificáveis pelo modelo.

Fonte: Autoria própria (2022).

Figura 20 – Exemplo de mochilas/bolsas difíceis de identificar.

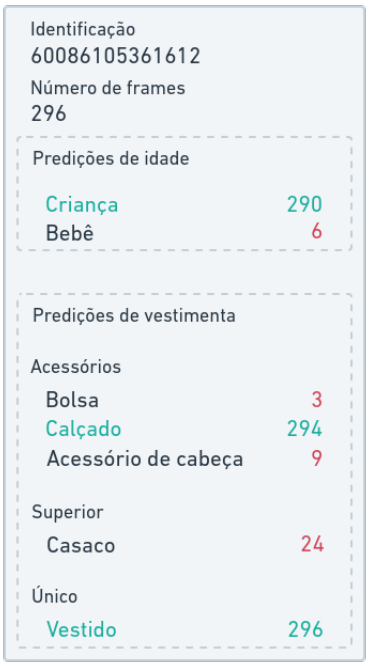


Fonte: Autoria própria (2022).

Além do processo básico de salvar as informações, o módulo também conseguiu garantir a qualidade dos dados através das operações descritas no Capítulo 3, utilizando proporções para eliminar falsos positivos e selecionar a classe predita mais provável para cada uma das características.

O resultado da estrutura dos documentos do banco de dados bem como o processo de seleção das melhores predições descrito no Capítulo 3 é mostrado na Figura 21, onde no documento há todas as classes preditas nos *frames* individualmente e em verde as classes escolhidas pelo sistema. Além disto, é importante pontuar que o todo o processo do módulo de persistência de dados dura em média 14 milissegundos para cada evento. Assim, o mesmo consegue lidar muito bem com a demanda proveniente dos outros módulos.

Figura 21 – Exemplo de documento de pessoa no MongoDB.



Fonte: Autoria própria (2022).

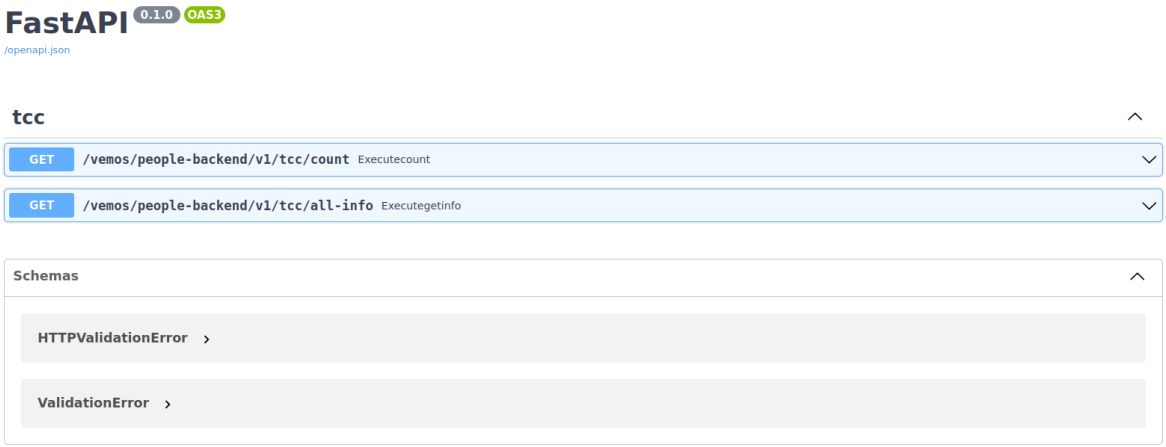
4.2.6 Interface

O módulo de interface é o último do sistema, fazendo a interação com o usuário final. A realização do *back-end* utilizando a ferramenta FastAPI foi de grande ajuda pois, utilizando as boas práticas de desenvolvimento foi possível gerar uma documentação da API de forma automática, facilitando o entendimento das rotas da mesma por outros desenvolvedores, tanto *front-end* quanto *back-end*, inclusive permitindo testes rápidos pela própria interface. Esta interface de documentação pode ser observada na Figura 22. Ademais, cada uma das rotas conseguiu coletar as informações do banco, organizar a estrutura e enviar de forma pronta, restando ao *front-end* apenas apresentar as mesmas.

A rota *count* cuidou da obtenção e estruturação das informações de número total de pessoas, número de pessoas no dia, média de pessoas por dia, além da contagem de pessoas por mês para o seu respectivo gráfico. Já a rota *all-info* ficou responsável pelas informações de idade e vestimenta. Essa divisão foi realizada, pois a contagem é uma consulta menos custosa e, consequentemente, mais rápida. Dessa forma, o *front-end* pode requisitar a contagem e mostrar essas informações enquanto a outra rota é processada para ser mostrada posteriormente.

O *front-end*, por sua vez, cumpriu o objetivo de trazer um *dashboard* simples que permite uma fácil interpretação do mesmo pelo usuário final. Somado a isso, a parte visual desenvolvida ficou bem similar ao design projetado no Figma durante a idealização do projeto. O resultado final da interface gráfica é mostrado na Figura 23.

Figura 22 – Interface de documentação da API do back-end.



Fonte: Autoria própria (2022).

Figura 23 – Versão final do *front-end*.

Fonte: Autoria própria (2022).

5 CONCLUSÃO

O aumento populacional, lado a lado com a crescente da chamada “Era dos dados”, faz com que as mais diversas áreas, incluindo a área empresarial, invistam em tecnologias guiadas por dados para auxiliar na identificação do seu público. O objetivo é direcionar anúncios e produtos a grupos específicos, de modo a aumentar as vendas.

Assim, o presente trabalho fornece uma ferramenta para a identificação do público-alvo de estabelecimentos a partir da contagem, classificação de grupo de idade e da segmentação e classificação de vestimenta das pessoas. O sistema proposto utiliza métodos de reconhecimento facial para, a partir da re-identificação de pessoas, trazer resultados mais precisos ao processo. A integração do funcionamento dos módulos foi feita de maneira robusta com ferramentas como Kafka, Kubernetes e MongoDB, capazes de orquestrar dinamicamente os diversos módulos que compõem o trabalho.

Arquitetar o sistema proposto neste trabalho como um sistema distribuído e utilizar as ferramentas e tecnologias já citadas trouxe a vantagem de ter um sistema facilmente escalável e automatizado, além de possuir uma baixa latência. Porém, em contrapartida, houve grande complexidade para fazer os módulos funcionarem de forma correta. A facilidade em escalabilidade reduziu o impacto até dos módulos que processam em CPU, uma vez que, com a necessidade de um tempo de resposta ainda menor, pode-se replicar os módulos e o Kafka aplicando *load-balacing*, o que poderia reduzir ainda mais a latência total do sistema.

Deixando de lado questões operacionais, o trabalho é considerado um sucesso pois, além de cumprir com os objetivos propostos desenvolvendo um sistema de ponta-a-ponta, ou seja, desde o carregamento dos dados até a apresentação dos mesmos, conseguiu uma solução com resultados bem satisfatórios para todos os módulos desenvolvidos e para diferentes condições de contorno. É importante ressaltar que, para o funcionamento adequado do mesmo, é importante que o fluxo de vídeo seja obtido em um ambiente com boas condições de iluminação, com equipamento de boa qualidade de imagem (resolução e *frames* por segundo), e posicionado de forma a capturar a pessoa por inteiro, de preferência na entrada dos estabelecimentos.

Por ser um sistema completo de ponta-a-ponta, projetos futuros podem ser facilitados a partir do mesmo. Tendo isto em vista, uma das recomendações para projetos futuros é criar réplicas dos módulos que permitam essa opção ou até a inclusão de mais GPUs, de modo a reduzir ainda mais a latência em cada módulo. Outro ponto interessante é trazer análises mais complexas ao *dashboard* tais como vestimentas por grupo de idade, abrindo um maior leque de possibilidades de forma simples para o usuário final.

REFERÊNCIAS

AMOS, B.; LUDWICZUK, B.; SATYANARAYANAN, M. **OpenFace: A general-purpose face recognition library with mobile applications**. CMU School of Computer Science, 2016.

CHRISTY, A. J. *et al.* RFM ranking — an effective approach to customer segmentation. **Journal of King Saud University – Computer and Information Sciences**, v. 33, n. 10, p. 1251–1257, 2021. ISSN 1319-1578.

COINBASE. **Kafka infrastructure renovation at Coinbase**. 2022. Disponível em: https://www.coinbase.com/blog/kafka-infrastructure-renovation. Acesso em: 16 nov. 2022.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. *In*: **Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition**. [*S.l.*: *s.n.*], 2005. v. 1, p. 886–893.

DU, Y. *et al.* Strongsort: Make deepsort great again. **arXiv preprint**, arXiv:2202.13514v1, 2022. Disponível em: https://arxiv.org/abs/2202.13514.

GENEROSI, A.; CECCACCI, S.; MENGONI, M. A deep learning-based system to track and analyze customer behavior in retail store. *In*: **Proc. IEEE 8th International Conference on Consumer Electronics**. [*S.l.*: *s.n.*], 2018. p. 1–6.

GUO, R. MongoDB's javascript fuzzer. **Communications of the ACM**, New York, USA, v. 60, n. 5, p. 43–47, 2017. Disponível em: https://doi.org/10.1145/3052937.

HASSNER, T. *et al.* Effective face frontalization in unconstrained images. *In*: **Proc. IEEE Conf. on Computer Vision and Pattern Recognition**. [*S.l.*: *s.n.*], 2015.

INÁCIO, A. D. S.; LOPES, H. S. EPYNET: Efficient pyramidal network for clothing segmentation. **IEEE Access**, v. 8, p. 187882–187892, 2020.

INÁCIO, A. de S.; BRILHADOR, A.; LOPES, H. S. Semantic segmentation of clothes in the context of soft biometrics using deep learning methods. *In*: FERNANDES, B. J. T.; {Pereira Júnior}, A. (Ed.). **Anais do 14 Congresso Brasileiro de Inteligência Computacional**. Curitiba, PR: ABRICOM, 2019. p. 1–7.

JOCHER, G. *et al.* **ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation**. Zenodo, 2022. Disponível em: https://doi.org/10.5281/zenodo.7347926.

KING, D. E. Dlib-Ml: a machine learning toolkit. **Journal of Machine Learning Research**, v. 10, p. 1755–1758, dec 2009.

KOSSOSKI, C. *et al.* A scalable analytics pipeline for COVID-19 face mask surveillance. **Learning & Nonlinear Models**, SBIC, v. 20, n. 1, p. 62–73, 2022.

KREPS, J.; NARKHEDE, N.; RAO, J. Kafka: a distributed messaging system for log processing. *In*: **Proc. of the Netdb**. [*S.l.*: *s.n.*], 2011. p. 1–7.

LIMA, A. S. de; AGOSTINHO, W. R. U. **Sistema de Processamento de Big Data aplicado à monitoração de automóveis em tempo real**. Trabalho de Conclusão de Curso, Universidade Tecnológica Federal do Paraná, Curitiba, PR: [*s.n.*], 2022.

LIN, T.-Y. *et al.* Feature pyramid networks for object detection. *In*: **Proc. IEEE Conference on Computer Vision and Pattern Recognition**. [*S.l.*: *s.n.*], 2017. p. 936–944.

LIU, W. *et al.* SSD: Single shot multibox detector. *In*: **Proc. European Conference on Computer Vision**. [*S.l.*]: Springer International Publishing, 2016. p. 21–37.

MEDEL, V. *et al.* Modelling performance resource management in kubernetes. *In*: . [*S.l.*: *s.n.*], 2016. p. 257–262.

MERKEL, D. Docker: lightweigh linux containers for consistent development and deployment. **Linux Journal**, v. 2014, n. 239, p. 2, 2014.

NG, H.-W.; WINKLER, S. A data-driven approach to cleaning large face datasets. *In*: **Proc. of the IEEE International Conference on Image Processing**. [*S.l.*: *s.n.*], 2014. p. 343–347.

PETERSON, B. **Are You Ready for Your Face to Be the Only Travel Document You Need?** 2000. Disponível em: https://www.afar.com/magazine/new-facial-recognition-tech-is-coming-to-airports-in-the-u-s-and-abroad. Acesso em: 15 nov. 2022.

PONNUSAMY, A. **CVlib - high level Computer Vision library for Python**. 2018. Disponível em: https://github.com/arunponnusamy/cvlib.

QI, D. *et al.* YOLO5Face: Why reinventing a face detector. **ArXiv preprint**, ArXiv:2105.12931, 2021.

RODRíGUEZ-MAZAHUA, L. *et al.* A general perspective of big data: applications, tools, challenges and trends. **Journal of Supercomputing**, v. 72, n. 8, p. 3073–3113, 2016.

ROTHE, R.; TIMOFTE, R.; van Gool, L. Deep expectation of real and apparent age from a single image without facial landmarks. **International Journal of Computer Vision**, v. 126, n. 2-4, p. 144–157, 2018.

SCHROFF, F.; KALENICHENKO, D.; PHILBIN, J. FaceNet: A unified embedding for face recognition and clustering. *In*: **Proc. IEEE Conference on Computer Vision and Pattern Recognition**. [*S.l.*: *s.n.*], 2015. p. 815–823.

SPANOUDES, P.; NGUYEN, T. Deep learning in customer churn prediction: Unsupervised feature learning on abstract company independent feature vectors. **CoRR**, abs/1703.03869, 2017. Disponível em: http://arxiv.org/abs/1703.03869.

STATISTA. **Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025**. 2022. Disponível em: https://www.statista.com/statistics/871513/worldwide-data-created/. Acesso em: 15 nov. 2022.

SYAKUR, M. A. *et al.* Integration K-means clustering method and elbow method for identification of the best customer profile cluster. **Materials Science and Engineering**, v. 336, n. 1, p. 012017, 2018. Disponível em: https://dx.doi.org/10.1088/1757-899X/336/1/012017.

TAN, M.; LE, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. *In*: CHAUDHURI, K.; SALAKHUTDINOV, R. (Ed.). **Proc. of 36th International Conference on Machine Learning**. [*s.n.*], 2019. v. 97, p. 6105–6114. Disponível em: https://proceedings.mlr.press/v97/tan19a.html.

YI, D. *et al.* Learning face representation from scratch. **arXiv preprint**, v. 1411.7923, 2014.

ZHOU, K. *et al.* Omni-scale feature learning for person re-identification. **arXiv preprint**, v. 1905.00953, 2019. Disponível em: https://arxiv.org/abs/1905.00953.