

**GUSTAVO TORRES MENDES
LUÍS HENRIQUE DAS NEVES HANSEN
MICHAEL MAGALHÃES SCHARDOSIM**

**TRACKING DE PESSOAS EM MOVIMENTO EM
MÚLTIPLAS CÂMERAS**

São Paulo
17 de janeiro de 2019

**GUSTAVO TORRES MENDES
LUÍS HENRIQUE DAS NEVES HANSEN
MICHAEL MAGALHÃES SCHARDOSIM**

**TRACKING DE PESSOAS EM MOVIMENTO EM
MÚLTIPLAS CÂMERAS**

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro de Computação.

São Paulo
17 de janeiro de 2019

**GUSTAVO TORRES MENDES
LUÍS HENRIQUE DAS NEVES HANSEN
MICHAEL MAGALHÃES SCHARDOSIM**

**TRACKING DE PESSOAS EM MOVIMENTO EM
MÚLTIPLAS CÂMERAS**

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro de Computação.

Orientador:

Prof. Dr. Edson Satoshi Gomi

São Paulo
17 de janeiro de 2019

Dedicamos esse trabalho aos nossos pais, familiares, amigos e animais de estimação, que estavam conosco durante todo o processo criativo.

AGRADECIMENTOS

Ao Professor Doutor André Fujita (IME-USP), pela ajuda na concepção do trabalho.

Ao Professor Doutor Edson Satoshi Gomi (EPUSP), pela orientação e apoio na elaboração do trabalho.

Aos Professores Doutores José Coelho de Pina Jr. (IME-USP) e Fábio Levy Siqueira (EPUSP), pela dedicação e empenho na nossa formação em Engenharia de Computação, mantendo acesa a chama da busca pelo conhecimento.

À Universidade de São Paulo, pela oportunidade de estudar em uma das mais renomadas instituições de ensino do mundo.

RESUMO

Este trabalho é focado no reconhecimento e rastreamento de pessoas em diferentes ambientes, com diferentes câmeras com reidentificação de um mesmo sujeito. Inspirado pela loja de varejo *Amazon GO*, o desenvolvimento se iniciou com o estudo de técnicas atuais. Após a realização de testes e *benchmarks*, foi escolhida uma arquitetura que utiliza redes neurais com *datasets* específicos, além do famoso *framework* de visão computacional *OpenCV*. O resultado final é um sistema que, apesar de dependente de hardware moderno, pode ser utilizado com câmeras simples, como webcams ou câmeras de segurança, até mesmo aquelas já instaladas e em funcionamento.

Palavras-chave – Multi-tracking, perda triade, yolo, re-identificação, multi-câmera.

ABSTRACT

This paper is focused on identifying and tracking people in different environments, with different cameras and same-subject re-identification. Inspired by the retail store *Amazon Go*, the development began with a study and research of the modern related technologies. After making some tests and benchmarks, an architecture which uses neural network with specific datasets was chosen, besides the famous computer vision framework *OpenCV*. The final result is a system that, despite being very hardware dependant, can be used in existing camera setups, even with simple cameras, such as webcams or security cameras.

Keywords – Multi-tracking, triplet loss, yolo, re-identification, multi-camera.

LISTA DE FIGURAS

1	Modelo de rede neural convolucional de 7 camadas	19
2	Camada de <i>spatial pyramid pooling</i>	22
3	Exemplo de funcionamento do <i>YOLO</i>	26
4	Arquitetura da rede neural do <i>YOLO</i>	26
5	Diagrama dos módulos do sistema	30
6	Exemplo do funcionamento do sistema.	31
7	Comparação da identificação de pessoas com <i>OpenCV</i> e <i>YOLO</i>	35
8	Exemplo do funcionamento do algoritmo de rastreamento.	36
9	Representação visual das cores no espaço de cores CIELAB	38
10	Diagrama com as partes do sistema.	40
11	Resultados dos testes de (1) com 364 imagens do <i>dataset</i> de <i>Duke</i>	42

LISTA DE TABELAS

1	Benchmark de algoritmo de re-identificação de pessoas	39
2	Benchmark de algoritmo de re-identificação de pessoas com remoção de <i>background</i>	41
3	Resultados obtidos a partir dos testes com o sistema final	45

SUMÁRIO

1	Introdução	11
1.1	Objetivo	11
1.2	Motivação	11
1.3	Justificativa	12
2	Aspectos Conceituais	13
2.1	Identificação de Pessoas	13
2.2	<i>Tracking</i> em Vídeo	14
2.2.1	<i>Boosting Tracker</i>	14
2.2.2	<i>MIL Tracker</i>	14
2.2.3	<i>KCF Tracker</i>	15
2.2.4	<i>TLD Tracker</i>	15
2.2.5	<i>MedianFlow Tracker</i>	15
2.2.6	<i>GOTURN Tracker</i>	15
2.2.7	<i>MOSSE Tracker</i>	16
2.3	Correspondência de Imagens	17
2.3.1	Redes neurais convolucionais e deep-learning	18
2.3.2	Rede residual e <i>ResNet</i>	19
2.3.3	Triplet loss	20
2.3.4	<i>Pooling</i> espacial	21
3	Tecnologias utilizadas	23
3.1	Python	23
3.2	<i>Frameworks</i> de <i>machine-learning</i>	23

3.3	OpenCV	24
3.3.1	Leitura de imagens	24
3.3.2	Remoção de <i>background</i>	24
3.3.3	Identificação de pessoas	24
3.3.4	<i>Tracking</i> de pessoas	25
3.4	<i>Darknet</i>	25
3.5	YOLO	25
3.6	Trinet	27
3.7	Google Cloud Platform	28
4	Metodologia de trabalho	29
4.1	Concepção	29
4.2	Projeto	29
4.2.1	Readequação de escopo	30
4.3	Implementação	31
4.4	Testes	32
5	Requisitos do sistema	33
5.1	Requisitos	33
5.1.1	Entrada e saída de vídeo	33
5.1.2	Cliente	34
6	Projeto e implementação	35
6.1	Testes	35
6.1.1	Identificação de pessoas	35
6.1.2	<i>Tracking</i>	36
6.1.3	Re-identificação de pessoas	36
6.1.3.1	Algoritmos tradicionais	36

6.1.3.2	Algoritmos com redes neurais	41
6.2	Projeto e implementação	42
6.2.1	Sistema de <i>tracking</i>	42
6.2.2	Sistema de detecção de pessoas	43
6.2.3	Sistema de re-identificação	43
6.2.3.1	<i>Threshold</i> adaptativo	43
6.2.4	Implementação	44
7	Testes e avaliação	45
7.1	Testes de desempenho	45
7.2	Avaliações posteriores	46
8	Considerações finais	47
8.1	Conclusões do projeto de formatura	47
8.2	Contribuições	47
8.3	Perspectivas de continuidade	48
Referências		49

1 INTRODUÇÃO

Our movements and feelings are constantly monitored, because surveillance is the business model of the digital age.

-- Katharine Viner

1.1 Objetivo

O objetivo deste trabalho é realizar o projeto de um sistema capaz de identificar e rastrear pessoas em diferentes fontes de vídeo de forma autônoma, realizando a correspondência entre diferentes imagens da mesma pessoa (reidentificação). O sistema deve cumprir uma série de requisitos definidos e sua adequação ser avaliada por uma série de métricas específicas para a aplicação.

1.2 Motivação

No início do ano de 2018, muito se comentou nos grandes veículos de comunicação sobre as novas lojas físicas da grande rede de varejo *online Amazon*. Nomeadas de *Amazon GO*, teriam sua primeira loja na cidade de *Seattle*, nos EUA. A ideia da loja é que pessoas possam entrar aproximando o celular de uma espécie de *catraca digital*, que registra o usuário e o associa à sua respectiva conta na *Amazon*. Após passar pela catraca, o consumidor não precisa se preocupar com mais nada: basta retirar os produtos desejados das prateleiras e sair da loja, e tudo será cobrado automaticamente em seu cartão de crédito. Para a realização disso, a loja é equipada com diversas câmeras, e alguns produtos específicos com *QR Codes*. Não existem, porém, outros tipos de sensores, como *RFID* ou sensores de posição de produtos em prateleiras, soluções já existentes e aplicadas em outros cenários.

Tal tecnologia despertou o interesse de muitos na comunidade, e foi objeto de diversas reportagens devido à novidade que era a aplicação de uma tecnologia tão desenvolvida num caso de uso real, e com possibilidade real de grande expansão. Analisando superficialmente o funcionamento da *Amazon Go*, nota-se que uma parte essencial de seu funcionamento

depende do rastreamento de pessoas e sua reidentificação em diferentes câmeras. Este trabalho, nesse contexto, entra como um estudo dessa parte importante da tecnologia.

Apesar de existirem algoritmos satisfatórios neste campo (2), eles ainda são restritos e não são capazes de, por exemplo, rastrear um mesmo objeto em diferentes capturas. Para tal, neste trabalho decidiu-se por focar no estudo de técnicas envolvendo redes neurais, uma tecnologia que vem sendo muito desenvolvida e utilizada, principalmente no campo de visão computacional .

1.3 Justificativa

Este projeto se baseia na tarefa de *tracking* de pessoas, que é importantes para muitas aplicações modernas, como na área de segurança, no qual se deve identificar e rastrear um indivíduo que cometeu alguma infração nas múltiplas câmeras de vigilância do sistema, ou então na área comercial, no qual o caminho percorrido por uma pessoa em seu estabelecimento pode gerar dados para análises posteriores a fim de se extrair comportamentos que possam otimizar o caminho percorrido.

Além disso, o modo no qual esta tarefa é abordada neste trabalho, de maneira completamente autônoma e sem intervenção humana, é com poucos precedentes, e requer técnicas modernas de visão computacional para executá-la. Assim, também se faz importante neste trabalho discutir as maiores dificuldades e soluções encontradas em se tratando das técnicas e tecnologias aqui discutidas.

Finalmente, o sistema final serve como base para uma futura aplicação no qual seja integrada com os sistemas anteriormente citados, para que a tarefa até então manual, possa ser realizada de maneira automática.

2 ASPECTOS CONCEITUAIS

O rastreamento de pessoas em um vídeo passa por três principais atividades, como descrito em (3):

- 1 Detecção de objetos: atividade que se consiste em identificar objetos de interesse em imagens e agrupar os pixels desse objeto, separando-o do plano de fundo;
- 2 Classificação de objetos: se trata de classificar os objetos encontrados em classes de interesse, como pessoas, automóveis ou pássaros;
- 3 Rastreamento de objeto: registrar o caminho de um objeto enquanto ele se move no vídeo;

Para cada uma dessas atividades existem diversos algoritmos capazes de executá-las, como pode ser visto em (3–7). Como o intuito desse trabalho é o rastreamento de pessoas, as atividades de detecção e classificação de objetos foram aglutinadas em uma só, aqui chamadas de identificação de pessoas. A seguir, serão abordados os principais tipos de algoritmos para cada uma das atividades, com a última seção do presente capítulo tratando da correspondência de uma mesma pessoa em diferentes vídeos.

2.1 Identificação de Pessoas

O estudo sobre identificação de pessoas foi focado em algoritmos com implementações já feitas e encontradas nas bibliotecas utilizadas. Dessa forma, foram testados dois algoritmos implementados no *OpenCV*, descritos nos próximos parágrafos, e também o *framework YOLO*, descrito na seção 3.5.

O algoritmo *Haar Cascade* (8) se baseia na criação de *features* formadas pela subtração da soma dos pixels de áreas próximas. Essas *features* são então selecionadas utilizando o algoritmo *AdaBoost* (9), que selecionará aquelas com maiores correlações com a variável resposta (a presença do objeto a ser detectado, no caso, humanos), por meio da verificação

em rodadas onde, a cada rodada, as observações erroneamente classificadas na rodada anterior ganham um peso maior. Dessa forma, para detectar um rosto em uma região, calculam-se as *features* para ela e avalia se é o objeto desejado. Isso pode ser uma tarefa temporalmente custosa dependendo da quantidade de *features* e do tamanho do quadro. Por isso, as *features* são aplicadas em cascata, onde um grupo de maior prioridade é aplicado antes e, em caso de falha, descarta-se o quadro sem se aplicar as demais.

O algoritmo *HOG*, ou *Histogram of Oriented Gradients* (10) utiliza *features* criadas a partir do gradiente em um grupo de pixels, dessa forma, sendo úteis para a detecção de mudanças em esquemas de cores, ou seja, bordas, e deixando de lado as regiões de cores uniformes. Então, esses gradientes são colocados em um histograma onde os ângulos definem a classe e a intensidade do gradiente é adicionada à frequência da classe. Esses histogramas são as *features* utilizadas então em uma implementação de *Support Vector Machine* (11), que será aplicada às novas observações com as mesmas *features* calculadas.

2.2 *Tracking* em Vídeo

Para o *tracking* de pessoas, também foram buscados algoritmos com implementações em Python já existentes. Dessa forma, os algoritmos utilizados para *tracking* e *multi-tracking* na versão 3.4 do OpenCV foram os estudados para a utilização no trabalho e se encontram explicados a seguir.

2.2.1 *Boosting Tracker*

O *Boosting Tracker* (12) é um algoritmo que é treinado em tempo de execução, utilizando a caixa de delimitação do objeto anotado como observação positiva e outras caixas retiradas da redondeza como observações negativas. Dado um novo quadro, o algoritmo é utilizado em cada pixel da redondeza da anotação do quadro anterior e escolhe aquele com maior pontuação. Então, a nova localização do objeto é incluída na base de treino como uma observação positiva e suas redondezas como negativas, e o algoritmo é retreinado.

2.2.2 *MIL Tracker*

O *MIL Tracker*, ou *Multiple Instance Learning* (13), funciona de forma parecida, mas trata a caixa do objeto anotado e sua redondeza mais próxima como observações positivas. Inicialmente, isso pode ser prejudicial ao modelo, porém, por utilizar o algoritmo de

Multiple Instance Learning, as observações positivas não são tratadas individualmente, mas como uma coleção de observações onde dentre elas pelo menos uma é realmente positiva. Com isso, o algoritmo acaba sendo mais eficaz, conseguindo corrigir pequenos desvios da caixa de anotação passada a ele.

2.2.3 *KCF Tracker*

O *KCF*, ou *Kernelized Correlation Filter*, *Tracker* (14) também se baseia na ideia de coleção de observações onde há uma positiva, mas se utiliza de propriedades da coleção, como a existência de áreas de intersecção entre seus componentes, para tornar o algoritmo mais rápido e ainda mais eficaz.

2.2.4 *TLD Tracker*

O *TLD Tracker*, ou *Tracking, Learning and Detection* (15), decompõe a tarefa de rastreio em três atividades, rastreio, aprendizado e detecção. A primeira segue o objeto denotado de quadro em quadro. A detecção localiza o objeto e corrige o rastreio se necessário. Por fim, o aprendizado estima os erros da detecção e os corrige para tentar evitá-los no futuro.

2.2.5 *MedianFlow Tracker*

O *MedianFlow Tracker* (16) se baseia em fazer o rastreio nos quadros em sentido temporal e no sentido inverso. Com isso, calcula a diferença entre as duas abordagens, es- colhendo o caso em que há menores discrepâncias. Isso possibilita o algoritmo a encontrar erros de *tracking* ao longo da execução.

2.2.6 *GOTURN Tracker*

O *GOTURN* (17) é o único algoritmo implementado no *OpenCV 3.2* que utiliza redes neurais convolucionais. Ele funciona cortando o quadro anterior na região do objeto a ser rastreado com o dobro do tamanho do retângulo de demarcação e o quadro atual, onde se quer fazer o rastreio, na mesma posição. Então, ambos os cortes são dados como entrada na rede neural, que devolve quatro números que representam a posição do retângulo de demarcação no quadro atual.

2.2.7 MOSSE Tracker

O MOSSE, *Minimum Output Sum of Squared Error* (18), é um algoritmo de *tracking* baseado na utilização de um filtro aplicado por correlação, o conceito matemático relacionado à filtragem espacial. Tal filtro, quando aplicado por produto escalar em uma região de uma imagem, produz uma resposta positiva se houver o objeto de interesse ou zero caso contrário. A aplicação do filtro é feita no domínio da frequência, isto é, aplica-se a Transformada de Fourier sobre a imagem e o filtro, fazendo com que o produto escalar possa ser feito por uma simples multiplicação (de acordo com o Teorema da Convolução), acelerando a aplicação do algoritmo.

Pode-se representar a aplicação do filtro pela fórmula $G = F \odot H^*$, sendo F a imagem, H^* o filtro e G o resultado, já no domínio da frequência e \odot a multiplicação elemento a elemento. Dessa forma, dadas imagens de treino f_i e suas saídas esperadas g_i , o algoritmo busca o filtro H^* que minimize a soma dos erros da aplicação da correlação de f_i e do resultado esperado g_i , como mostra a Equação 2.1, já no domínio da frequência.

$$\min_{H^*} \sum_i |F_i \odot H^* - G_i|^2 \quad (2.1)$$

Esse problema de minimização pode ser simplificado, pois as operações envolvidas são todas elemento a elemento, fazendo com que seja possível aplicar a minimização a cada elemento, obtendo a Equação 2.2, com w e v sendo os índices dos elementos de H . Então, partindo do pressuposto de que essa equação é real, positiva e convexa, tem-se que ela só terá um mínimo, que pode ser obtido igualando a derivada parcial a zero, como na Equação 2.3. Com algumas operações matemáticas simples é possível chegar a Equação 2.4 elemento a elemento, que também pode ser reescrita na forma da Equação 2.5.

$$\min_{H_{wv}^*} \sum_i |F_{iwv} \odot H_{wv}^* - G_{iwv}|^2 \quad (2.2)$$

$$0 = \frac{\partial}{\partial H_{wv}^*} \sum_i |F_{iwv} \odot H_{wv}^* - G_{iwv}|^2 \quad (2.3)$$

$$H_{wv} = \frac{\sum_i F_{iwv} G_{iwv}^*}{\sum_i F_{iwv} F_{iwv}^*} \quad (2.4)$$

$$H_{wv} = \frac{\sum_i F_i \odot G_i^*}{\sum_i F_i \odot F_i^*} \quad (2.5)$$

Então, os filtros são inicializados como na Equação 2.5, utilizando oito observações

do quadro inicial do vídeo com pequenas perturbações, e atualizados de acordo com a Equação 2.6, que utiliza uma média com peso η para os quadros recentes, fazendo com que o filtro se adapte a mudanças do objeto sendo rastreado.

$$H_i^* = \frac{A_i}{B_i} \quad (2.6)$$

$$A_i = \eta G_i \odot F_i^* + (1 - \eta) A_{i-1} \quad (2.7)$$

$$B_i = \eta F_i \odot F_i^* + (1 - \eta) B_{i-1} \quad (2.8)$$

O valor padrão para η é 0,125, considerado pelos autores como bom por possibilitar uma rápida adaptação às mudanças da imagem ainda mantendo o filtro robusto.

2.3 Correspondência de Imagens

A correspondência de imagem é a tarefa mais complexa do trabalho. Isso porque exige técnicas mais robustas de reconhecimento, ante os algoritmos clássicos já existentes de detecção de objetos. Sua solução se baseia em sistemas de *re-identificação*, muito conhecido na literatura como *Re-ID*.

No estado da arte atual, a classe de algoritmos que melhor desempenha essa função é baseada em redes neurais convolucionais e *deep learning*(19)(20)(21), e por isso essa foi a solução adotada neste projeto.

O primeiro e principal problema a ser resolvido em se tratando de reconhecimento envolve, a partir de um par de imagens já detectadas e categorizadas, identificar se elas apresentam um mesmo objeto ou não. Esse é o fundamento básico do problema de reconhecimento de imagens, e técnicas já maduras existem para resolvê-lo, como serão apresentadas adiante.

Outro problema inerente a algoritmos de visão computacional e que também foi explorado neste projeto é o tratamento de imagens com tamanhos e proporções diferentes. Enquanto nas arquiteturas clássicas a solução se baseia em redimensionar e escalar a imagem para que todas possuam o mesmo tamanho, há artigos mais recentes que propõem técnicas para o uso de redes neurais de natureza convolucional que podem trabalhar com imagens de qualquer tamanho e proporção.

2.3.1 Redes neurais convolucionais e deep-learning

A técnica de redes neurais surgiu com o crescimento inteligência artificial clássica, ainda na década de 1940, baseada no modelo de *perceptron* baseada na analogia das células nervosas de animais. Seu uso, no entanto, só começou a se difundir na década de 1980, com o uso do algoritmo de retropropagação para seu aprendizado.

Em 1998 surge o conceito de *rede neural convolucional*, como cunhado por Yann LeCun e sua implementação do modelo LeNet-5 (22). Sua teoria se baseia nos união dos trabalhos e avanços anteriores na área de redes neurais, e na sua utilização para desempenhar um papel similar ao córtex visual de primatas.

Surge com isso também o termo *deep-learning*, devido às profundidades profundas de camadas características das redes convolucionais. Devido a isso, no entanto, o treinamento é computacionalmente mais custoso que as redes clássicas e só difundiu-se no meio tecnológico como opção viável na década corrente, com o advento dos coprocessadores gráficos.

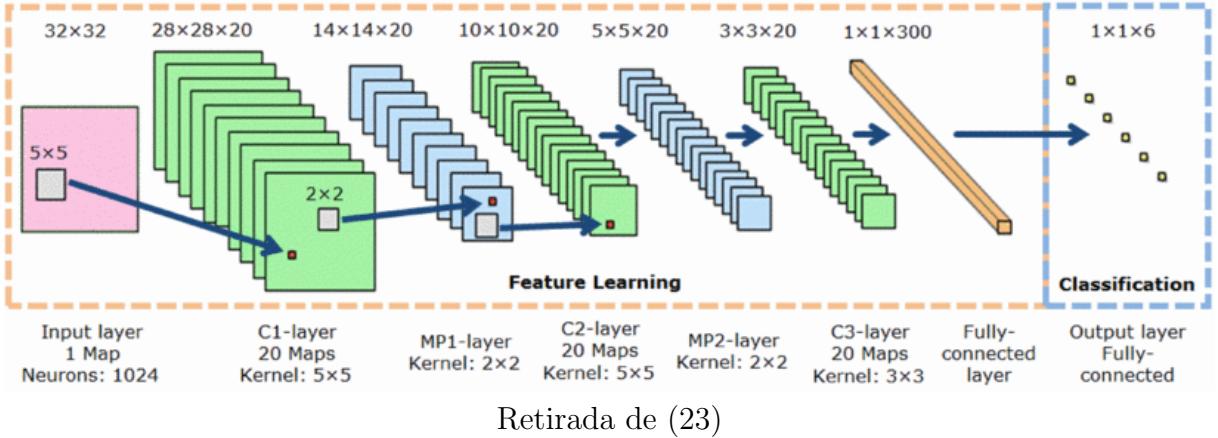
O princípio teórico das redes convolucionais se baseia em dois tipos de camadas neurais existente em sua arquitetura: as camadas convolucionais e as camadas de subamostragem, ou *pooling*, como costuma-se denominá-la na literatura.

Intuitivamente, a camada convolucional tem como objetivo extrair *features* dos dados de entrada, utilizando para isso a operação de *convolução discreta* sobre a entrada, utilizando uma função de *filtro* que contém parâmetros treináveis. No caso de imagem, esses filtros são mapas tridimensionais, que aplicam a operação sobre os múltiplos canais da imagem de entrada. Geralmente aplicam-se múltiplos desses filtros sobre uma mesma entrada, obtendo-se assim, múltiplas saídas, que representam, idealmente, diferentes *features* da imagem de entrada.

A camada de *pooling*, por sua vez, não possui parâmetros treináveis e tem como objetivo reduzir a dimensionalidade dos dados, utilizando para isso, usualmente, a média aritmética ou o valor máximo em um mapa sobre a entrada.

Nos modelos tradicionais, utiliza-se camadas de *pooling* sobre a saída das camadas de convolução e, quando a dimensionalidade está suficientemente reduzida, rearranja-se o resultado para uma única dimensão, que é aplicado para uma rede neural artificial clássica. Uma arquitetura muito utilizada para problemas de classificação e que no geral produz resultados suficientemente bons é o “modelo de 7 camada”, que se emprega de 3 camadas convolucionais e 3 camadas de *pooling*, seguido por uma camada *fully-connected*.

Figura 1: Modelo de rede neural convolucional de 7 camadas



Sobre a saída desta, aplica-se uma função de classificação, que possibilite o cálculo de um erro para o seu treinamento, como a função de *softmax*. A figura 1 ilustra tal arquitetura.

É importante notar que tanto as camadas de convolução quanto de *pooling* são diferenciáveis e deriváveis, portanto, o algoritmo de aprendizado de *backpropagation* pode ser utilizado.

2.3.2 Rede residual e *ResNet*

Um problema existente nas redes neurais profundas é que, à medida que o número de camadas aumenta, também o faz o tempo de processamento. Isso é um comportamento esperado, visto que mais camadas significa mais passos para o treino e teste da rede. O que surpreende, porém, é que há determinado ponto em que quando a profundidade aumenta a acurácia do aprendizado fica saturado, fazendo com que o erro de treinamento seja superior à uma arquitetura menos profunda. Esse fenômeno de degradação foi observado em diversos experimentos nos últimos anos e sua origem não se baseia em *overfitting*(24).

Este erro ocorre devido à estrutura das redes, no modelo *feed-forward*, no qual a saída de uma camada deve produzir um significado semântico para um dado de entrada, mas a única referência que esta camada possui para esta entrada é sua representação gerada pelas camadas anteriores. Isso gera um erro intrínseco à camada, que será propagado para as camadas posteriores. Ora, para uma rede profunda, com centenas ou milhares de camadas, esse erro se acumula e se potencializa nas camadas finais, fazendo com que o erro global seja limitado por este tipo de erro. Uma solução para este problema foi proposta em um artigo da Microsoft, utilizando uma arquitetura de redes neurais denominada *residual*, ou *ResNet*(25). Sua arquitetura se baseia nos princípios de *representação residual* e *shortcut*.

connection.

Representação residual se refere a uma nova forma de representação da saída de uma rede, no qual ela gera uma saída que quando subtraída do resultado da rede, obtém-se a própria entrada. Daí o nome “residual”. A rede, durante o treinamento, aprende o resíduo e não o valor integral da saída. A equação 2.9 representa este tipo de rede, no qual $O(x)$ representa a saída da rede para uma entrada x , e $N_w(x)$ é a saída de uma rede neural, com parâmetros treináveis, para a entrada x .

$$O(x) = N_w(x) + x \quad (2.9)$$

Shortcut connection é uma técnica que consiste na alimentação de um dado x diretamente para uma camada l_{i_1} da rede, sem que este dado seja saída da camada imediatamente anterior l_i . Assim, entende-se que x está curto-circuitando a camada l_i , ou, de forma mais geral, um conjunto de camadas l_i, \dots, l_{i+n} . Esta técnica facilita à representação residual em uma rede por permitir que a entrada seja diretamente alimentada às camadas de saídas.

A *ResNet* é uma categoria de redes que se utilizam destas duas técnicas para criar arquiteturas que utilizem centenas de camadas sem que haja degradação da acurácia.

2.3.3 Triplet loss

O problema de reconhecimento possui peculiaridades que dificultam seu uso em redes neurais, cujos principais objetivos são classificação e regressão. Por isso, por muito tempo utilizavam-se outros métodos para solucioná-lo.

Em 2015, no entanto, a corporação *Google* desenvolveu uma rede denominada *FaceNet*, que possui um artigo associado (26), propondo uma solução para este problema com redes neurais convolucionais. A solução se baseia no conceito de *triplet loss*, ou perda tríade.

O problema do uso de redes neurais convolucionais para reconhecimento de imagens oriunda-se da natureza de sua arquitetura, que tem por objetivo produzir uma saída no qual se associa um significado semântico. Para o reconhecimento, por outro lado, a saída não possui valor semântico, mas a relação entre diferentes saídas possuem, i.e., a saída para duas imagens de um mesmo objeto deve ser interpretada como tal, assim como duas imagens de objetos diferentes devem ser devidamente dissociados.

A técnica do *triplet loss* visa sanar este problema, e para isso, durante a fase de

treinamento, para cada iteração, obtém a saída para duas imagens de um mesmo objeto e um diferente, e sua perda é calculada de tal forma que reduza a diferença entre as saídas semelhantes e aumente a diferença entre as saídas das imagens diferentes.

As imagens de entradas são chamadas de âncora (a), que é a imagem cuja relação entre as outras duas tenta-se otimizar, a positiva (p), que se refere ao mesmo objeto que a âncora e a negativa (n), que se refere a um objeto diferente da âncora e, consequentemente, da positiva. A esse conjunto de imagens para uma mesma iteração de treino dá-se o nome de tríade.

A função de perda para uma tríade é calculada através da equação 2.10, no qual d é chamado de função distância, que deve ser diferenciável. Geralmente utiliza-se a distância euclidiana. ϵ é a denominada margem de tolerância cujo valor deve ser positivo e representa a distância mínima que se deseja que dois pontos de imagens de objetos diferentes estejam.

$$\mathcal{L} = \max(d(a, p) - d(a, n) + \epsilon, 0) \quad (2.10)$$

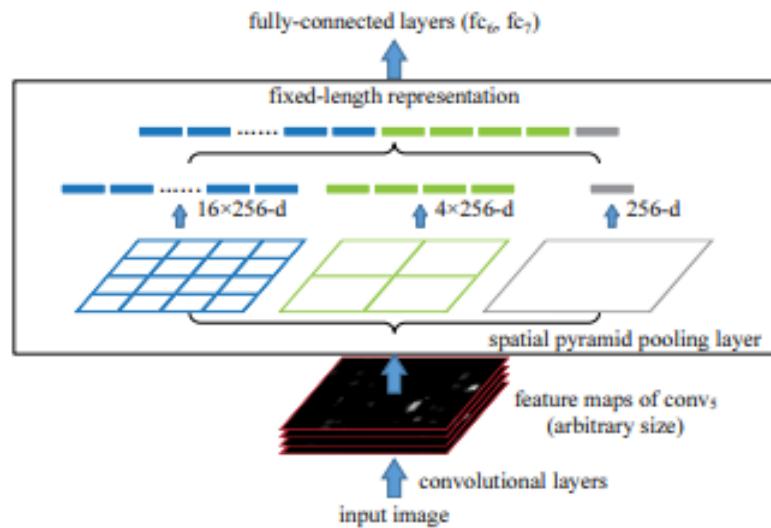
Percebe-se que esta função possui saturação negativa no 0, indicando que a imagem positiva está mais próxima da âncora do que a negativa.

2.3.4 *Pooling* espacial

Outro conceito explorado neste projeto são camadas de *pooling* espacial, também denominadas globais. Estas camadas, assim como as camadas de *pooling* convencionais, reduzem a dimensionalidade dos dados de entrada. No entanto, enquanto estas aplicam um filtro para extrair os valores locais em uma pequena região da entrada, as camadas globais aplicam um filtro sobre todos os dados de entrada, com o intuito de se obter os valores mais relevantes para o problema e desprezar regiões da entrada que não são relevantes para a saída final (27).

As duas formas mais comuns de se implementar esta camada são a camada de *max global pooling*, que extraí os n maiores valores da entrada, para cada um dos c canais, o que resulta em $c \times n$ valores na saída; e as camadas de *spatial pyramid pooling*, que apesar de mais complexas, empregam a mesma técnica de *max global pooling*, mas para diferentes valores de n em uma mesma camada, assim, obtém-se maior flexibilidade para se representar dados em diferentes granularidades e níveis de abstração, como demonstrado em estudos anteriores.(27) A figura 2 ilustra a ação deste tipo de camada.

Figura 2: Camada de *spatial pyramid pooling*



Retirada de (27)

Outra vantagem de se usar camadas de *pooling* espacial é que as dimensões de sua saída dependem apenas do número de canais, e não do tamanho de imagens. Portanto, pode-se utilizá-las para imagens de qualquer tamanho que sua saída terá o mesmo tamanho. Devido à esta propriedade, é comum utilizá-las após as camadas de convolução, para se obter um vetor de tamanho fixo para ser passado para a camada *fully-connected*.

3 TECNOLOGIAS UTILIZADAS

3.1 Python

A linguagem Python é hoje talvez a mais famosa e utilizada quando se trata de ciência de dados e inteligência artificial, especialmente *machine learning*. Desse modo, muitos *frameworks* específicos para as aplicações requeridas estão disponíveis, e são mantidos por uma grande comunidade. Analisando o desempenho da linguagem, percebe-se que Python está muito abaixo de outras linguagens mais eficientes, como C++. Existe, porém, a possibilidade de se utilizar, dentro do Python, bibliotecas compiladas em outras linguagens, e um exemplo disso é o OpenCV.

Em termos de facilidade de desenvolvimento, o grupo levou em conta não apenas facilidade de desenvolvimento e leitura do código, mas também as ferramentas disponíveis em diferentes plataformas. Nesse quesito, pode-se perceber que as vantagens do Python são várias: utilizando o *framework* “Anaconda”, e o gerenciador de pacotes “pip”, obteve-se padronização entre todas as plataformas de desenvolvimento (MacOS, Linux e Windows). Para realização de testes, utilizou-se mais um facilitador disponível e estável apenas para o *Python*, chamado “*Jupyter Notebook*”: uma ferramenta com interface web simples onde pode-se escrever código, versionar, executar e separá-lo para execução em diferentes trechos.

Considerados todos os fatores, o grupo optou por utilizar a linguagem *Python*, que se mostrou adequada para o problema apresentado.

3.2 *Frameworks* de *machine-learning*

Os *frameworks* utilizadas para se trabalhar com redes neurais artificiais em *Python* foram *tensorflow* e *theano*, e outras bibliotecas que se utilizam destas para fornecer maior nível de abstração, como *keras* e *lasagne*. Elas foram utilizadas tanto durante a realização dos testes das tecnologias mencionadas a seguir, quando na implementação das redes e

seu treinamento.

3.3 OpenCV

O *OpenCV* é uma robusta e amplamente utilizada biblioteca de visão computacional. Ela comprehende um grande conjunto de funções bem documentadas e abrange uma variada gama de problemas de visão computacional. Ela foi escolhida por ter o código aberto, ser sustentada por uma grande comunidade e implementar suas funções de maneira muito eficiente. A biblioteca é escrita nas linguagens C e C++, porém possui uma interface em *Python*, sendo assim facilmente integrada com o restante do projeto. Foi utilizada a versão 3.4, disponível para *download* no link (28).

3.3.1 Leitura de imagens

O *OpenCV* possui os *codecs* necessários para a leitura e interpretação de diversos tipos de arquivos de imagens e vídeos. Foram utilizadas funções para a abertura de imagens *JPG*, *PNG* e de vídeos no formato *AVI* e em *stream*. A biblioteca é compatível com outras bibliotecas muito utilizadas e nativas do *Python*, como o *numpy*. As imagens são lidas e os *pixels* são transformadas em matrizes, onde cada elemento representa uma cor RGB. A partir daí, foi possível utilizar tais informações para realizar a identificação de pessoas e seu posterior *tracking*.

3.3.2 Remoção de *background*

Foi utilizada também a função de remoção de *background* em vídeos *BackgroundSubtractorMOG*. Mais detalhes de seu funcionamento estão descritos na seção 6.1.3.

3.3.3 Identificação de pessoas

Durante a fase de teste, principalmente durante os *benchmarks* dos algoritmos de *tracking*, foi muito utilizada a função nativa de identificação de pessoas presente no *OpenCV*. Ela utiliza o algoritmo tradicional mais utilizado atualmente para detecção de pessoas, o *HOG* (*Histogram of Oriented Gradients*, ou Histograma de gradientes orientados). Ele utiliza janelas deslizantes de diferentes tamanhos que buscam por *features* específicas a partir de descritores testados previamente. Sua acurácia não é tão boa quanto a de redes neurais (*RNNs*), porém é muito eficiente e rápido. Variações desse algoritmo estão

presentes, por exemplo, nas câmeras digitais que apresentam detecção de rostos humanos.

3.3.4 *Tracking* de pessoas

Também principalmente durante a fase de testes e *benchmarks*, foram utilizados os algoritmos presentes no *OpenCV* para a realização do *tracking* de pessoas. Foi realizado, com o auxílio das implementações de algoritmos de *tracking* do *OpenCV*, o *benchmark* para se decidir qual deles seria utilizado. Todos os algoritmos testados, bem como uma análise de suas vantagens e desvantagens, estão descritos na seção 2.2.

3.4 *Darknet*

Darknet (29) é um *framework* de código aberto de redes neurais escrito em C e CUDA, ou seja, com suporte a GPUs. Sua utilização nesse trabalho está intimamente ligada à utilização do *YOLO*, pois este utiliza o *Darknet*.

3.5 YOLO

O *YOLO* - *You Only Look Once* (30) é um sistema de detecção parrudo de objetos para uso em tempo real. Isso é feito através de um algoritmo que passa cada quadro apenas uma vez pela rede neural convolucional, enquanto outros algoritmos, em geral, passam diversas partes de um quadro em suas redes para detectar diversos objetos.

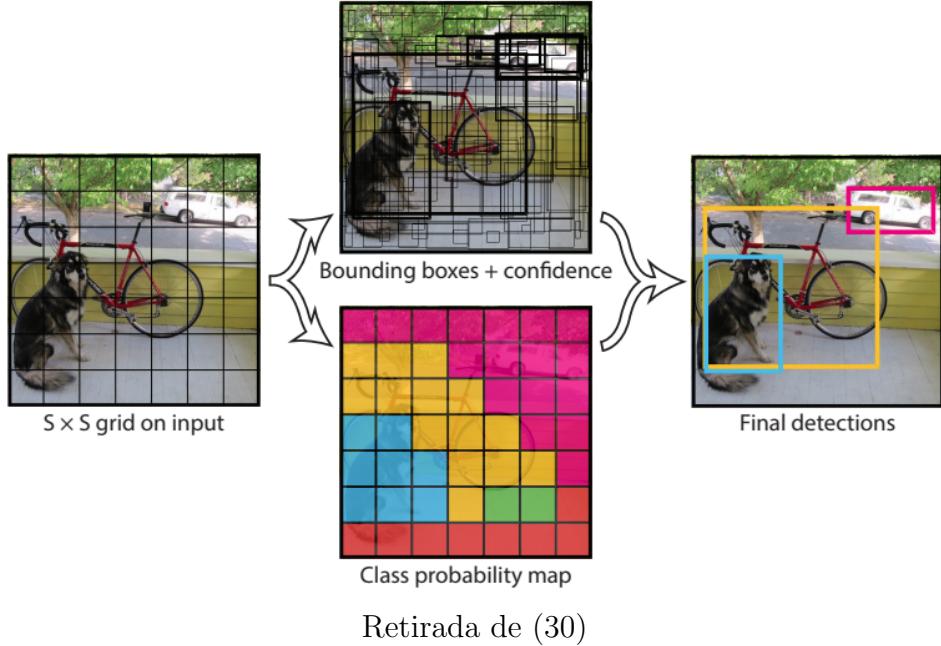
Para isso, inicialmente ele divide o quadro de entrada em uma grade de tamanho $S \times S$. Então, para cada célula, o *YOLO*: prevê B retângulos demarcadores de objetos, cada um com os valores (x , y , w , h), para posição nas horizontal e vertical, largura e altura respectivamente, e um valor de confiança, que indica a probabilidade do retângulo conter um objeto e quão certo ela está; detecta um único objeto independentemente da quantidade B de retângulos; e prevê a probabilidade do objeto ser de cada uma das C possíveis classes.

Para entender melhor, podemos usar o exemplo para o treino no *dataset Pascal VOC* (31). Nele foi utilizada uma grade 7×7 , com dois retângulos de demarcação e 20 classes de objetos. Dessa forma, a previsão deve ter um formato $(S, S, B \times 5 + C) = (7, 7, 2 \times 5 + 20) = (7, 7, 30)$.

Dessa forma, a imagem é dividida na grade, passa pela rede neural que devolve os

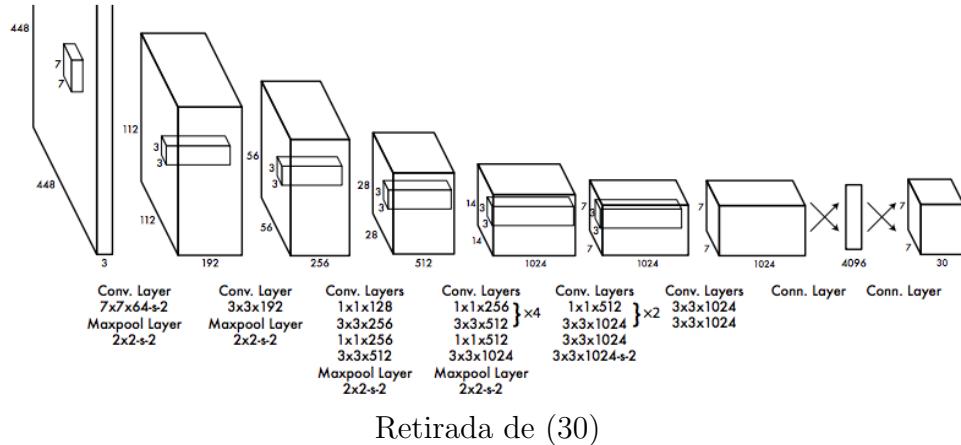
dois retângulos para cada célula, e filtra a imagem com os retângulos cuja confiança seja maior que um limiar, como mostra a Figura 3.

Figura 3: Exemplo de funcionamento do (YOLO). Na esquerda, a imagem inicial dividida em uma grade de $S \times S$. No meio, os $7 \times 7 \times 2$ retângulos de demarcação em cima e o mapa de probabilidade de classes abaixo. Na direita, as detecções finais.



A arquitetura da rede do *YOLO* pode ser vista na figura 4. Trata-se de 24 camadas de rede convolucional, que extraem as *features* da imagem, seguidas por duas camadas totalmente conectadas, que calculam as probabilidades e posições.

Figura 4: Arquitetura da rede neural do *YOLO*. São 24 camadas convolucionais seguidas por duas camadas totalmente conectadas.



A função de perda utilizada está representada na equação 3.1 e é composta de três

componentes: perda de classificação, de localização e de confiança.

$$\begin{aligned}
 Loss = & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
 & + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
 & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
 & + \sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
 \end{aligned} \tag{3.1}$$

O erro de classificação é a soma dos erros quadráticos de classificação para cada célula e classe, quando um objeto é encontrado.

O erro de localização é a soma dos quadráticos das posições x e y e da raiz dos tamanhos w e h para cada retângulo de demarcação, quando há a detecção de um objeto, multiplicados por um fator. A raiz é usada nos tamanhos para que um erro absoluto, de por exemplo dois pixels, não seja igualmente tratado num retângulo grande e um pequeno. O fator, de valor padrão 5, é utilizado para dar mais ênfase ao erro de localização.

O erro de confiança é calculado como a soma dos erros quadráticos das confianças de cada retângulo de marcação, com um fator multiplicando quando um objeto não é detectado nele. Esse fator, de valor padrão 0,5, é utilizado pois a frequência de retângulos sem objetos é maior que com, causando um desbalanceamento que pode levar o modelo a aprender mais a detectar o plano de fundo do que os objetos.

Por fim, como podem haver mais de um retângulo demarcando o mesmo objeto, é seguido o seguinte procedimento: os retângulos são ordenados pelo valor de confiança; a partir do maior valor, são ignorados os retângulos que tem mesma marcação e razão de intersecção sobre união com outro retângulo maior que 0,5.

3.6 Trinet

A *trinet* (1) é uma arquitetura de rede neural proposta para solucionar o problema de re-identificação de pessoas em câmeras de segurança, de código aberto (32) e, portanto, amplamente testado e avaliado. A arquitetura da rede segue o modelo *ResNet50*,

possuindo, portanto 50 camadas treináveis, como mostrado na seção 2.3.2.

No repositório em questão, há *scripts* para realizar seu treinamento a partir de qualquer *dataset*, desde que previamente formatado. Há também a possibilidade de se utilizar a rede com pesos resultantes de treinamentos anteriores, sendo fornecido os valores para o treinamento realizado sobre os datasets *Market1501* e *MARS*, sendo o primeiro adotado para a realização do projeto.

Esta rede é implementada em *Python*, em conjunto com os *frameworks* *Theano* e *Lasagne* de *machine-learning* para a execução da rede, e *tensorflow* para seu treinamento. Seu treinamento se utiliza da função de perda tríade, descrita em 2.3.3, com *mining online* de *hard triplets*.

A rede, como originalmente concebida, recebe como entrada uma imagem de 3 canais (*RGB*), de dimensões fixas 288×144 , sendo qualquer outra imagem fornecida distorcida e reescalada para estas dimensões. Sua saída são 128 valores, *embeddings*, que idealmente são únicos por pessoas.

3.7 Google Cloud Platform

A execução do treinamento e da avaliação das tecnologias foi eventualmente realizada em nuvem, devido ao alto poder de processamento requerido, optando-se assim a utilização da plataforma de computação em nuvem do *Google*, a *Google Cloud Platform*. Utilizando-se nele uma instância de máquina virtual situada fisicamente no *data-center* de *West Virginia*, com 8 núcleos processamento, 16 GB de memória e uma placa gráfica *NVIDIA Tesla K80*. Nesta máquina também hospeda-se o sistema final para a demonstração de seu funcionamento.

4 METODOLOGIA DE TRABALHO

4.1 Concepção

A primeira parte do desenvolvimento do projeto é realizar o estudo das tecnologias já existem e o estado da arte no que se refere a *visão computacional* e a identificação e rastreamento de objetos em imagens, com uma abordagem voltada para o uso de técnicas de *machine-learning* e *deep learning*.

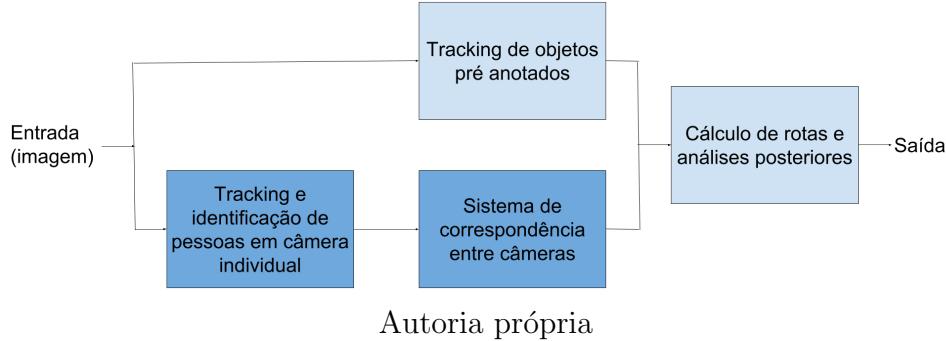
A partir da leitura, foram realizadas *proof of concepts* de uma seleção de tecnologias e técnicas que se mostraram mais promissoras nos estudos anteriores, para que se obtivesse uma rápida comparação entre elas, envolvendo tanto desempenho e acurácia na resposta, quanto na facilidade de uso e de integração com as demais tecnologias para o desenvolvimento do sistema fina.

O método utilizado para se avaliar esses testes foi selecionando um conjunto de bibliotecas e algoritmos para o rastreamento de pessoas e um outro conjunto para a identificação de pessoas em uma imagem. As métricas foram obtidas a partir do seu desempenho em alguns datasets pré-anotados (33, 34) e então escolheu-se qual tecnologia para cada uma dessas funções seriam utilizadas no projeto.

4.2 Projeto

O sistema foi dividido em diferentes partes descritas abaixo, sendo as duas primeiras dadas como essenciais e as duas últimas tidas como extra para aplicação do sistema ao caso em análise de comportamento de clientes. A Figura 5 mostra como as partes estão relacionadas.

Figura 5: Diagrama dos módulos do sistema. Em azul escuro, as partes essenciais. Em azul claro, possíveis partes extras para a aplicação em análise de comportamento de cliente.



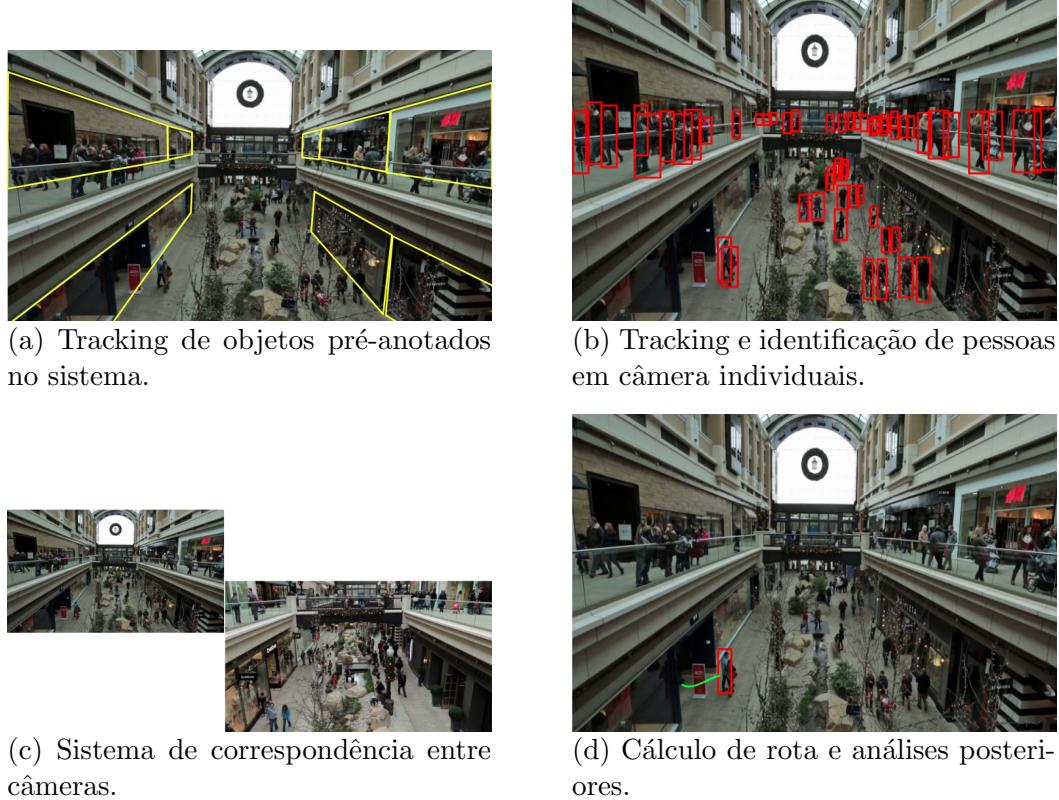
- 1 *Tracking* e identificação de pessoas em câmera individual: Nessa parte ocorrerá a identificação das pessoas na imagem (realizado a cada intervalo de X frames) e o rastreio das pessoas identificadas ao longo da cena;
- 2 Sistema de correspondência entre câmeras: Aqui será feita a comparação de cada pessoa identificada com o histórico das pessoas que já passaram pela câmera guardado em um *buffer*;
- 3 Cálculo de rotas e análises posteriores: Análise das saídas das partes anteriores para a retirada de informação útil;

A Figura 6 mostra um exemplo teórico do funcionamento do sistema, com o rastreamento dos objetos pré-anotados em 6a, o rastreamento de pessoas em câmeras individuais em 6b, a correspondência entre câmeras em 6c e o cálculo de rotas em 6d.

4.2.1 Readequação de escopo

Durante a implementação, foi observado que a parte do sistema responsável pelo *tracking* de objetos pré-anotados não traria muito valor ao sistema como um todo. Isso porque sua implementação técnica não traria nenhum desafio adicional, visto que o *tracking* de objetos pré-anotados já é extensamente abordado na literatura. Para alguns casos de uso do sistema, tal *tracking* faria sentido, mas foi decidido retirá-lo do escopo do projeto, visto que ele poderia ser facilmente implementado em caso de necessidade.

Figura 6: Exemplo do funcionamento do sistema.



Autoria própria baseada em (35)

4.3 Implementação

A metodologia de implementação utilizada por todo o projeto foi baseada no conceito de *MVP* (*Minimum Viable Product*, ou produto minimamente viável), muito utilizado na metodologia *agile*. Assim, cada parte do sistema começava com um teste em ambiente de *debug*, aqui, o *Jupyter Notebook*. Nesse ambiente, era possível realizar diferentes testes e *benchmarks* de diferentes implementações de cada módulo do sistema. Assim que determinado módulo estivesse funcionando de acordo com o esperado, sua implementação era refinada e integrada com as outras partes já implementadas. No mesmo ambiente foi possível testar a interação entre os diferentes módulos do sistema.

A vantagem de se utilizar um ambiente de *debug* controlado, juntamente com um gerenciador de ambientes virtuais (*virtual environments*) foi a possibilidade de continuar o desenvolvimento e testar seu funcionamento em diversas plataformas - diferentes computadores com diferentes sistemas operacionais.

4.4 Testes

Os testes para a avaliação da corretude da parte programática do sistema será feita por meio de testes unitários e testes de integração automatizados.

A avaliação do desempenho do sistema de rastreamento e análise de dados, será feita iterativamente, durante cada fase de treinamento, a partir de *datasets* já existentes adaptados para o caso de uso. E finalmente, após julgado suficiente seu desempenho, será testado em um ambiente real, com duas câmeras.

O teste para a integração do sistema com os dispositivos de entrada e saída será feito manualmente, devido à simplicidade e baixa relevância para o projeto em questão.

5 REQUISITOS DO SISTEMA

5.1 Requisitos

O sistema em questão consiste em um serviço que deve ser capaz de receber entradas de vídeos e detectar, identificar e rastrear pessoas nesse vídeo, exibindo ao usuário o vídeo anotado seguindo as especificações descritas adiante. Esta visualização ocorre no cliente local, enquanto que o processamento dos dados ocorre em um servidor remoto. Para este projeto considera-se que o sistema principal seja o serviço remoto e o cliente aqui desenvolvido tem como único objetivo auxiliar na demonstração de seu funcionamento, permitindo assim maior flexibilidade para a criação de outras formas de interação com o sistema para projetos futuros, bastando apenas integrar com sua *API*.

Nas seções seguintes descrevemos os requisitos e restrições deste sistema principal.

5.1.1 Entrada e saída de vídeo

O sistema deve ser capaz de processar até 3 fontes diferentes de vídeos, com uma resolução máxima de 1280x720 px, garantindo uma resposta a uma taxa média de 30 fps. A resposta deve conter o vídeo enviado, acrescido de metadados, podendo eles ser:

- Uma lista de demarcações de retângulos (coordenada superior esquerda e coordenada inferior direita) contidos na imagem, referentes ao trajeto de uma única pessoa, com o número do frame a qual ela se relaciona.
- Uma lista de demarcação retangulares de diferentes pessoas para um único *frame*, com a identificação de cada uma. Esta lista é enviada a cada 15 quadros.
- Uma lista com um número de *frame* e um identificador de vídeo. Esta informação é utilizada quando se deseja saber em quais vídeos e em quais instantes uma pessoa específica foi detectada.

O sistema deve ser capaz de responder com uma latência máxima de 1 segundo para cada *frame*, sem considerar o atraso da rede. Os vídeos devem obedecer as especificações de resolução e taxa de reprodução, e os diferentes vídeos devem ter suas imagens regularizadas para uma mesma temperatura de cor e luminosidade para que sejam garantidas o tempo de resposta e precisão.

5.1.2 Cliente

O cliente, desenvolvido para a demonstração do funcionamento do sistema no servidor remoto, mostrará as imagens das câmeras, com retângulos de demarcação das pessoas identificadas na imagem. A identificação será atualizada a cada período de alguns segundos, para que encontre novas pessoas na imagem. Então, o usuário poderá clicar em algum dos retângulos, fazendo com que o algoritmo de *tracking* e re-identificação passem a ser executados para ela, mostrando seu caminho nas câmeras. Dessa forma, os requisitos são os seguintes:

- Exibir as imagens das câmeras conectadas
- Exibir os retângulos de demarcação para cada pessoa identificada
- Atualizar os retângulos de demarcação quando os mesmos forem atualizados
- Identificar o clique do usuário em um dos retângulos de demarcação, selecionando-o e gerando a chamada para execução do *tracking* e da re-identificação para aquela pessoa
- Exibir o rastro da pessoa selecionada para o *tracking*
- Exibir a re-identificação em diversas câmeras da pessoa selecionada

Vale notar que esse cliente poderia ser adaptado para a aplicação do sistema a outros casos de uso, como vigilância ou estudo de comportamento, fazendo com que os requisitos fossem diferentes dos levantados para a demonstração.

6 PROJETO E IMPLEMENTAÇÃO

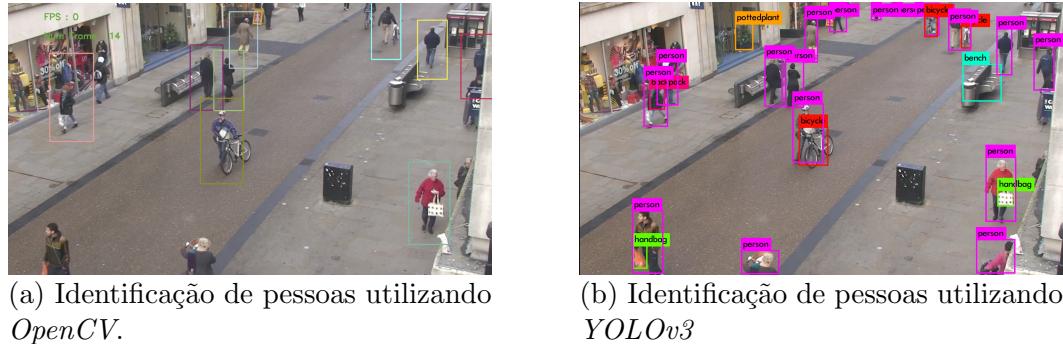
6.1 Testes

6.1.1 Identificação de pessoas

Para a identificação de pessoas nos vídeos, foi testado o sistema *YOLOv3*, em sua implementação padrão, comparado à implementação de identificação de pessoas do *OpenCV*, ambos explicados em 2.1. As métricas extraídas foram de acurácia na identificação e no tempo de execução. Apesar de maior tempo de execução, o *YOLOv3* se mostrou significativamente mais preciso.

Um exemplo do resultado utilizando-se estes algoritmos é apresentado na figura 7. Percebe-se que o *YOLO* é mais parrudo que o *OpenCV*, capaz de detectar pessoas apenas parcialmente visíveis e até outros objetos que não sejam pessoas. Por isso decidiu-se utilizá-lo, adaptando-se o restante do sistema para que acomode o tempo de processamento de sua classificação.

Figura 7: Comparação da identificação de pessoas com *OpenCV* e *YOLO*

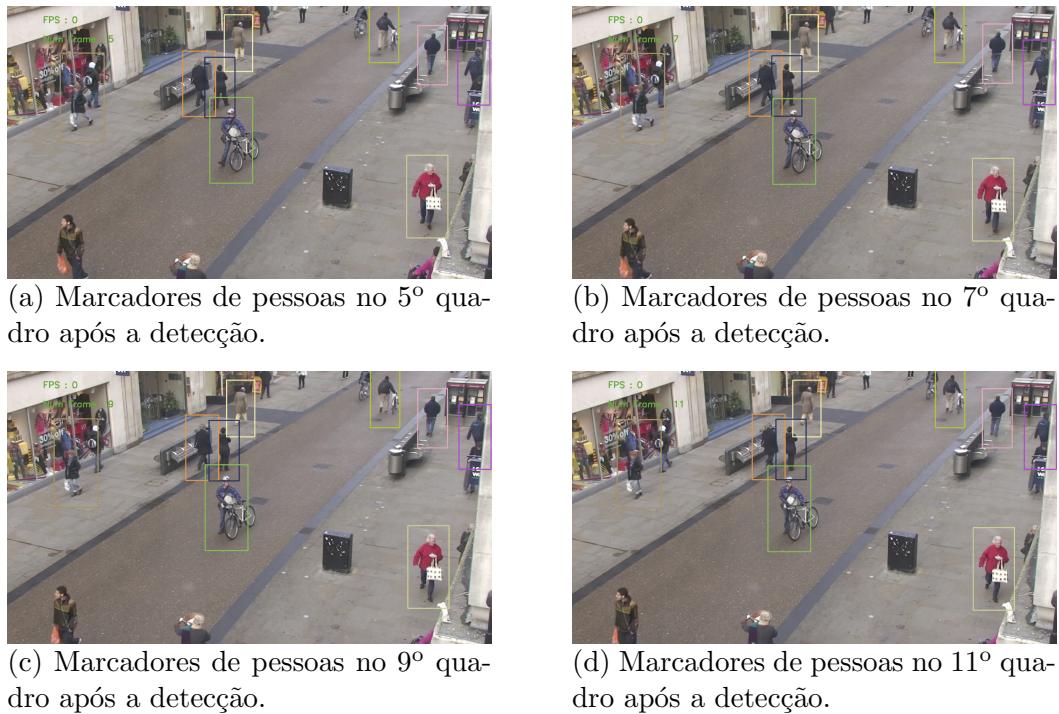


Autoria própria baseada em (34)

6.1.2 Tracking

O *tracking* foi testado utilizando os diferentes algoritmos implementados pelo *OpenCV* citados em 2.2. Os que visualmente apresentaram melhor resultado foram o *KCF* e o *CSRT*, e dentre eles, foi escolhido o *CSRT*, que de acordo com a literatura é mais preciso que o anterior, mas requer maior processamento. Novamente, a preferência por precisão sobre performance teve impacto no modelo do sistema, como será apresentado em 6.2.1.

Figura 8: Exemplo do funcionamento do algoritmo de rastreamento.



Autoria própria baseada em (34)

Na figura 8a é apresentado um exemplo de funcionamento do sistema de *tracking*, no qual são exibidas as caixas de demarcação das pessoas no 5º quadro após a identificação das mesmas. Nas imagens seguintes, ocorre a passagem dos quadros, com as pessoas em posições levemente diferentes na cena, mas com as caixas de demarcação acompanhando-as devido ao algoritmo de *CSRT* do *OpenCV*.

6.1.3 Re-identificação de pessoas

6.1.3.1 Algoritmos tradicionais

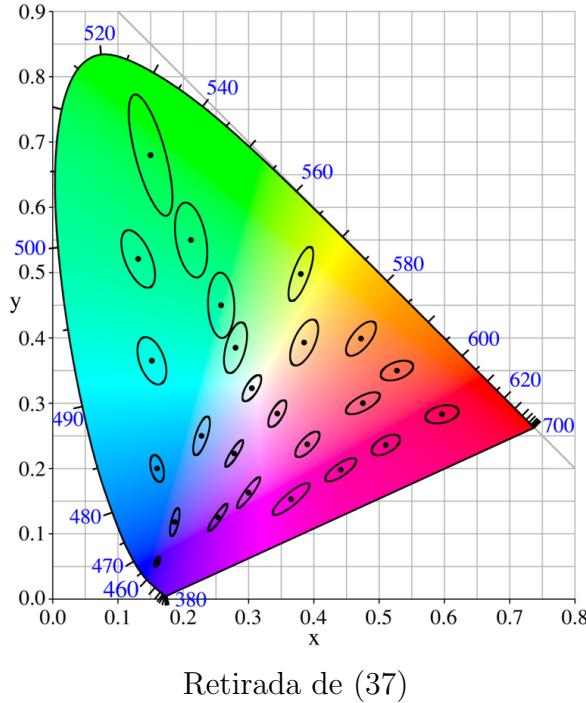
A parte do sistema responsável por realizar a correspondência entre câmeras foi a que mais gerou códigos e implementações diferentes. Os testes foram iniciados utilizando

combinações de diferentes algoritmos já existentes, sem o uso de redes neurais ou inteligência artificial, e sim algoritmos tradicionais implementados em *Python*. A ideia primária baseava-se na comparação de cores presentes em cada pessoa identificada. Assim, o primeiro teste utilizou a comparação da média das cores de cada indivíduo. Tal técnica rapidamente se mostrou inefetiva, principalmente pelo fato da média das cores não ser um bom caracterizador de uma imagem. Por exemplo, se uma imagem contém a mesma quantidade de cores complementares, a cor média será sempre cinza, independentemente das cores iniciais. Com esse sistema, por exemplo, uma imagem com 50% azul - RGB(00,00,FF) e 50% amarelo - RGB(FF,FF,00), seria identificada como sendo igual a outra com 33% ciano - RGB(00,FF,FF), 33% magenta - RGB(FF,00,FF) e 33% amarelo, já que as duas médias seriam um cinza - RGB(7F,7F,7F).

Após pesquisas sobre comparação de imagens, percebeu-se que uma melhor caracterização das cores da imagem poderia ser feita pela determinação de cores dominantes. Os testes concentraram-se, então, em utilizar a função já implementada no *OpenCV* de agrupamento de cores por k-médias, um algoritmo já testado e amplamente utilizado em problemas de agrupamentos. Os resultados obtidos foram considerados muito bons, tendo as "n" cores principais das figuras extraídas com sucesso. Um outro conceito introduzido, também amplamente utilizado, foi o de distância de cores por delta-e.

Esse conceito foi criado pelo CIE (International Commission on Illumination), juntamente com o espaço de cores CIELAB. Assim como o espaço RGB, as cores são divididas em três componentes, sendo eles *L*, para luminosidade, *a*, para as cores dicotômicas vermelho-verde e *b*, para as cores dicotômicas amarelo-azul. O CIE baseou-se no funcionamento do olho humano para sua criação, então esse sistema é muito utilizado ao analisar as cores como percebidas pelo olho humano. A fórmula do delta-e foi sofrendo atualizações ao longo dos anos, e a fórmula atualmente utilizada, atualizada em 2000, está descrita em (36).

Figura 9: Representação visual das cores no espaço de cores CIELAB



Aplicando os conceitos já provados em (38), dividiu-se as imagens de cada indivíduo em algumas faixas, e aplicou-se o algoritmo de agrupamento em cada uma dessas faixas. Durante os testes, foram definidos alguns parâmetros para o algoritmo. Esses parâmetros foram: o número de faixas em que cada imagem era subdividida, o número de cores principais a serem extraídas de cada faixa, a distância de cores (dentro do delta-e) para duas cores serem consideradas idênticas (aqui chamado de *threshold* de cores) e o número de *matches* de cor/faixas de cores para se admitir duas pessoas como sendo a mesma. Foram feitos um *benchmark*, variando os parâmetros e utilizando figuras do dataset DukeMTMC(39). Os resultados estão reportados na tabela 1.

Os testes do *benchmark* descrito na tabela 1 foram realizados com 15 imagens de 4 pessoas diferentes. O tempo de cada execução foi de, em média, 8,15 segundos.

Analizando os resultados obtidos, bem como seu processo de obtenção, ficou claro que o algoritmo, como implementado no momento, não seria viável. Seu tempo de execução para poucas pessoas era muito grande e, considerando a métrica mais importante a de positivos corretos, no melhor dos casos a acurácia foi de apenas 51,56%, e positivos corretos 70,18% (parâmetros 6F 3M 1C T=30). Numa tentativa de melhorar a eficácia do algoritmo, testou-se a aplicação do mesmo com as imagens das pessoas isoladas, ou seja, sem *background*. Tal configuração seria possível de se alcançar na prática, visto que

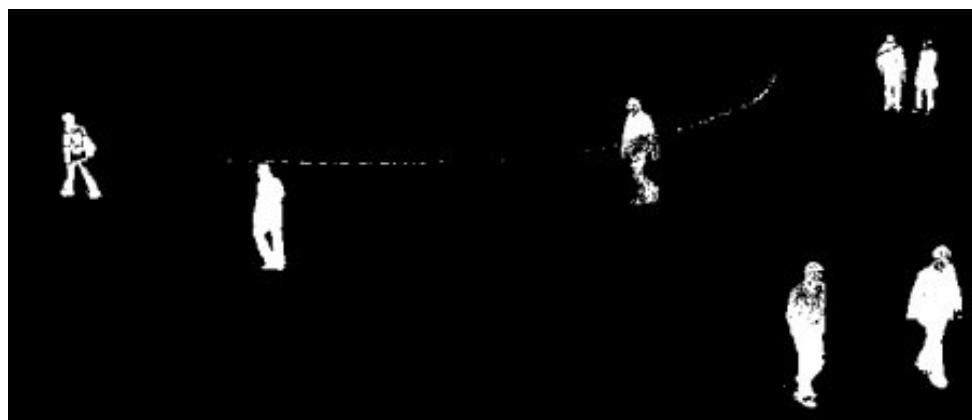
	8F 5M 1C	6F 3M 1C	8F 3M 2C	6F 2M 2C	5F 2M 2C
T=5	Acur: 81.33 %	Acur: 80.89 %	Acur: 81.33 %	Acur: 81.33 %	Acur: 81.33 %
	TP: 26.32 %				
	TN: 100.0 %	TN: 99.4 %	TN: 100.0 %	TN: 100.0 %	TN: 100.0 %
	FP: 0.0 %	FP: 0.6 %	FP: 0.0 %	FP: 0.0 %	FP: 0.0 %
	FN: 73.68 %				
T=15	Acur: 79.56 %	Acur: 77.33 %	Acur: 79.56 %	Acur: 79.56 %	Acur: 80.44 %
	TP: 26.32 %	TP: 29.82 %	TP: 26.32 %	TP: 28.07 %	TP: 26.32 %
	TN: 97.62 %	TN: 93.45 %	TN: 97.62 %	TN: 97.02 %	TN: 98.81 %
	FP: 2.38 %	FP: 6.55 %	FP: 2.38 %	FP: 2.98 %	FP: 1.19 %
	FN: 73.68 %	FN: 70.18 %	FN: 73.68 %	FN: 71.93 %	FN: 73.68 %
T=20	Acur: 78.22 %	Acur: 69.78 %	Acur: 79.56 %	Acur: 76.44 %	Acur: 79.11 %
	TP: 33.33 %	TP: 40.35 %	TP: 26.32 %	TP: 29.82 %	TP: 28.07 %
	TN: 93.45 %	TN: 79.76 %	TN: 97.62 %	TN: 92.26 %	TN: 96.43 %
	FP: 6.55 %	FP: 20.24 %	FP: 2.38 %	FP: 7.74 %	FP: 3.57 %
	FN: 66.67 %	FN: 59.65 %	FN: 73.68 %	FN: 70.18 %	FN: 71.93 %
T=30	Acur: 63.11 %	Acur: 51.56 %	Acur: 65.33 %	Acur: 60.44 %	Acur: 64.44 %
	TP: 61.4 %	TP: 70.18 %	TP: 52.63 %	TP: 47.37 %	TP: 50.88 %
	TN: 63.69 %	TN: 45.24 %	TN: 69.64 %	TN: 64.88 %	TN: 69.05 %
	FP: 36.31 %	FP: 54.76 %	FP: 30.36 %	FP: 35.12 %	FP: 30.95 %
	FN: 38.6 %	FN: 29.82 %	FN: 47.37 %	FN: 52.63 %	FN: 49.12 %

Tabela 1: Benchmark - acurácia, positivos corretos, negativos corretos, falsos positivos e falsos negativos para diferentes parâmetros. Número de cores por faixa: 3.

F = número de faixas; M = *matches* de faixas; C = número de cores para *match*; T = *threshold* de cor

Acur = acurácia; TP = positivos corretos; TN = negativos corretos; FP = falso positivos; FN = falso negativos

Figura 10: Diagrama com as partes do sistema.



Autoria própria

os algoritmos existentes de remoção dinâmica de fundos funcionam em vídeos - caso de uso do projeto. Novamente, focou-se em utilizar funções já implementadas na biblioteca *OpenCV*.

Realizando testes visuais de remoção de *background*, foi considerada como melhor função a "*BackgroundSubtractorMOG*", que implementa o algoritmo introduzido no artigo (40), utilizando segmentação de *background/foreground* baseado em misturas Gaussianas. Utilizou-se também as funções de morfologia presentes no *OpenCV*, *OPEN*, *CLOSE* e *EXPAND*, para melhorar a acurácia da máscara gerada. A figura 10 mostra a máscara gerada pelo programa.

Utilizando as mesmas imagens do *benchmark* anterior, porém com o *background* removido, foi realizado mais um *benchmark*, partindo do melhor cenário dos testes anteriores. O resultado está reportado na tabela 2.

	6F 3M 1C	8F 3M 2C	5F 2M 2C
T=20	Acur: 71.56 %	Acur: 79.11 %	Acur: 79.11 %
	TP: 40.35 %	TP: 26.32 %	TP: 28.07 %
	TN: 82.14 %	TN: 97.02 %	TN: 96.43 %
	FP: 17.86 %	FP: 2.98 %	FP: 3.57 %
T=30	FN: 59.65 %	FN: 73.68 %	FN: 71.93 %
	Acur: 52.89 %	Acur: 65.78 %	Acur: 64.44 %
	TP: 66.67 %	TP: 52.63 %	TP: 50.88 %
	TN: 48.21 %	TN: 70.24 %	TN: 69.05 %
	FP: 51.79 %	FP: 29.76 %	FP: 30.95 %
	FN: 33.33 %	FN: 47.37 %	FN: 49.12 %

Tabela 2: Benchmark com remoção de *background* - acurácia, positivos corretos, negativos corretos, falsos positivos e falsos negativos para diferentes parâmetros. Número de cores por faixa: 3.

F = número de faixas; M = *matches* de faixas; C = número de cores para *match*; T = *threshold* de cor

Acur = acurácia; TP = positivos corretos; TN = negativos corretos; FP = falso positivos; FN = falso negativos

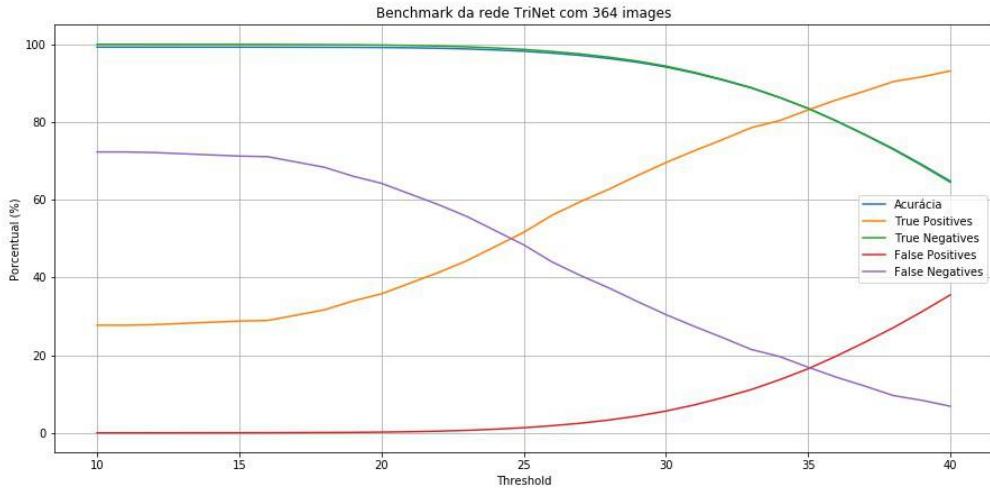
Analizando os dados obtidos no *benchmark*, conclui-se que não havia melhora significativa na performance do algoritmo utilizando a remoção de *background*. O único parâmetro que se mostrou realmente apto a influenciar na razão de positivos corretos foi o *threshold* de distância de cores. Pelo design do sistema CIELAB, uma distância acima de 20 já é muito visível ao olho humano, e não se pode mais dizer que as cores estão realmente próximas. Por estas razões, foi decidido abandonar sua implementação e estudos e focar os esforços em outras técnicas, como a de redes neurais - que se mostrou muito eficiente.

6.1.3.2 Algoritmos com redes neurais

Por fim, foi testada uma implementação do algoritmo em (1) disponível em (32). O teste foi feito utilizando 364 imagens do *dataset* de *Duke* (??) que foram aplicada à rede. A partir do resultado, foram testados diversos limiares para se considerar uma re-identificação positiva e foram calculadas as métricas de acurácia (azul), verdadeiro positivo (amarelo), verdadeiro negativo (verde), falso positivo (vermelho) e falso negativo (roxo). O resultado pode ser visto na Figura 11.

Dados os resultados obtidos no teste, o algoritmo foi considerado satisfatório, com um

Figura 11: Resultados dos testes de (1) com 364 imagens do *dataset* de *Duke*.



Autoria Própria

bom valor de limiar entre 30 e 35, de acordo com o balanceamento de verdadeiros e falsos positivos, mantendo uma acurácia geral superior à 80%.

6.2 Projeto e implementação

Considerando-se os testes realizados descritos na seção anterior, juntou-se os resultados obtidos para se decidir quais tecnologias seriam utilizadas no sistema final. O sistema foi modelado em três módulos: o subsistema de detecção de pessoas, o subsistema de re-identificação de pessoas e o subsistema de *tracking* de pessoas. Os dois primeiros devem operar continuamente conforme o fluxo de entrada de imagens, podendo ser em tempo real, quando utilizada na máquina virtual descrita em 3.7

6.2.1 Sistema de *tracking*

A partir das avaliações dos algoritmos de *tracking* foi escolhido o algoritmo de *MOSSE*, em sua implementação padrão na biblioteca do *OpenCV*. Devido a restrições de performance, este módulo não é executado em tempo real, e só é utilizado na visão detalhada. Ele também rastreia apenas uma pessoa por vez, por causa desta mesma *restrição*.

Este subsistema recebe como entrada um vídeo e um retângulo de marcação, e sua saída será o valor desta marcação, atualizando-o para os quadros seguintes de acordo com o algoritmo de *tracking*. Ele é executado continuamente por 30 *frames* e então aciona o

módulo de identificação para garantir a precisão do resultado.

6.2.2 Sistema de detecção de pessoas

Este subsistema tem por objetivo apenas detectar a existência de pessoas no quadro, para que se possa, posteriormente, atribuí-la uma identificação e traçar-se sua rota seguinte. Ele se utiliza do *YOLOv3* para tal identificação, a partir de quadro de entrada que lhe é passado. O módulo então retorna os retângulos de marcação no qual detectou uma pessoa. Este sistema é acionado a cada 30 quadros, ou seja, a cada 1 segundo se mantida a execução em tempo real com 30fps especificados.

6.2.3 Sistema de re-identificação

Este módulo recebe as pessoas detectadas pelo subsistema de detecção e tem como objetivo atribuir uma identidade a ela. Para isso, ele alimenta a imagem à *trinet* para se obter o *embedding* que identificará a pessoa, e após essa verificação retorna um identificador único para representá-la, seja ele já utilizado anteriormente se a pessoa já estava presente em um instante anterior, ou um valor novo se ela não foi correlacionada com nenhuma outra pessoa.

A correlação é feita através da clusterização dos *embeddings*, normalizados para um desvio padrão unitário, i.e., dividindo-se todos os valores pelo desvio padrão. Para cada dado novo, tenta-se classificá-lo em um dos *clusters*, ou caso ele seja distante de todos, é criado um novo grupo, com uma nova identificação única. As condições de contorno são quando há menos que 5 pessoas na base de identificação, e então as categorias são geradas comparando-se todos os dados entre si e agrupando os que possuem distância menor que um *threshold*. Para os demais casos, utiliza-se o algoritmo de *k-means* para a clusterização. O valor de *threshold* adotado para o sistema é 10.0, inclusivo, e aplicado sobre os valores normalizados. Assim, quando os *embeddings*, divididos pelo desvio padrão de seus 128 valores, entre duas pessoas, distam um valor menor ou igual a 10, considera-se que estas pessoas possuem a mesma identidade.

6.2.3.1 *Threshold* adaptativo

Durante o desenvolvimento do trabalho, outra pesquisa foi publicada tratando de um tema similar. Trata-se do "DukeMTMC" (*Duke Multi-Target, Multi-Camera Tracking*) (39) (41) (42). Este trabalho lida com o mesmo problema aqui abordado, o *trackig* e

reidentificação de pessoas. Sua abordagem, porém difere em alguns pontos. Um destes pontos é a forma de reidentificação de pessoas. Para tal, "DukeMTMC" utiliza algoritmos de *clusterização* de *features* extraídas de pessoas, e utiliza como critério de enquadramento de uma nova imagem de pessoa dentro em um *cluster* existente a distância euclidiana das *features* extraídas dessa nova imagem com o centro do cluster. Como critério, ao invés de utilizar um *threshold* fixo, como foi proposto inicialmente aqui, utiliza-se o desvio padrão de cada *cluster*.

Baseado na solução de "DukeMTMC", foi pensado em se utilizar um *threshold* adaptativo, ou seja, que não fosse o mesmo independentemente de qualquer outro fator. Assim, foi criado um *threshold* que varia com a distância entre as *caixas* de identificação de cada pessoa no *frame* anteriormente analisado e no atual. Dessa forma, o *threshold* diminui proporcionalmente à distância, partindo do pressuposto que, se duas figuras de pessoas foram identificadas em *frames* próximos e estão localizadas também próximas uma a outra, então a avaliação de distância pode ser mais rigorosa, já que não houve mudanças significativas nas condições de luz, câmera, vestuário e posição.

Após a implementação do *threshold* adaptativo, foram realizados testes visuais de percepção de melhoria, e determinou-se que a precisão havia melhorado significativamente. Assim, sem *benchmarks* formais, essa solução foi utilizada no sistema final.

6.2.4 Implementação

Estes módulos foram unidos em um único serviço, que é executado em uma instância de máquina virtual na nuvem, como especificado em 3.7, em um sistema *Linux* baseado em *Debian*. A interação com o usuário é realizada em um cliente, cujo software é capaz de ser executado através de um interpretador *Python*, sendo testado apenas em *Linux*. A este cliente deve-se também ser conectado uma ou mais câmeras que realizarão a captura dos vídeos. A comunicação entre os clientes e o servidor é feita através da rede internet, utilizando os protocolos TCP e UDP sobre IP.

7 TESTES E AVALIAÇÃO

7.1 Testes de desempenho

Após a implementação de cada uma das partes do sistema, seguindo a arquitetura do sistema proposta na seção 4.2, a integração dos três módulos foi testada em um vídeo de autoria própria, gravado com duas câmeras no corredor de uma universidade. O vídeo foi pré-anotado manualmente, em 400 diferentes quadros sequenciais. Os resultados foram obtidos utilizando-se este vídeo no sistema, variando-se o valor do *threshold* no módulo de reidentificação, e então comparou-se o resultado do sistema com o que foi pré-anotado. As métricas extraídas deste teste foram os IDs recebidos por uma mesma pessoa, a porcentagem de tempo de uma mesma pessoa com um mesmo ID e confusões entre IDs de pessoas diferentes. Os resultados podem ser observados na tabela 3.

	Média	Desvio padrão
IDs recebidos por uma mesma pessoa	2	1,155
% de tempo com um mesmo ID	51,44%	39%
Confusões entre IDs de pessoas	1	0

Tabela 3: Resultados obtidos a partir dos testes com o sistema final

A partir deste vídeo também foi gerado um outro vídeo no qual foi acrescido uma marcação para as pessoas identificadas pelo sistema, a fim de se testar visualmente o resultado do sistema. Com este vídeo de demonstração foi possível perceber que o sistema possui um resultado aparente bom para os primeiros quadros, mas foi possível perceber uma degradação da acurácia, devido a três grandes fatores. O primeiro se deve ao fato de que pessoas parecidas, sobretudo considerando-se as vestimentas, têm grandes chances de serem identificadas como uma mesma pessoa, aumentando assim o número de falsos positivos.

O segundo fator ocorre devido a forma no qual uma pessoa é detectada, podendo produzir um retângulo relativamente grande em relação à pessoa e que pode conter mais in-

formações que o necessário e isso impacta no desempenho da reidentificação. Um fenômeno decorrente disso que pode ser observado nos vídeos é a reidentificação de uma pessoa como outra muito próxima a ela, pois o retângulo de referência contém ambas as pessoas, e isso faz com que o algoritmo de reidentificação considere a pessoa errada em sua análise.

A terceira e última grade observação que se pôde obter com a análise visual foi o fato de de que ao sistema identificar uma pessoa como uma outra pessoa, nova ou que já tenha sido identificada, i.e., quando ocorre um falso negativo, este erro se propaga para as demais reidentificações desta pessoa. Isso ocorre porque, no geral, esses dois *embeddings* são muito próximos e faz com que as reidentificações seguintes classifiquem esta pessoa ora como a original, ora como a identificação original, ora como a nova identificação.

7.2 Avaliações posteriores

Com a identificação dos problemas apresentados, foram propostas técnicas que poderiam ser utilizadas como possíveis soluções, mas devido à restrição de tempo do projeto não foi possível testá-las e comprovar que de fato podem resolver os problemas citados, valendo portanto como proposta para trabalhos futuros.

A primeira proposta foi a utilização da técnica de *threshold adaptativo* em conjunto com os dados inferidos de rastreamento para serem utilizados na reidentificação. A premissa é de que pessoas que estavam em um local em um determinado quadro, estarão em uma região próxima no quadro seguinte. Assim, pode-se admitir um *threshold* menor para pessoas identificadas próximas a região demarcada pelo algoritmo de tracking no quadro anterior. Esta técnica pode reduzir o número de falsos negativos e portanto reduzir o erro residual que se propaga durante a execução do sistema.

A segunda proposta visa diminuir o erro quando mais de uma pessoa se encontra no retângulo identificado. Propõe-se que se utilize um algoritmo de extração de contorno da pessoa, para que as demais informações do retângulo não sejam consideradas. Esta proposta também visa reduzir o número de falsos negativos.

8 CONSIDERAÇÕES FINAIS

*I did surveillance a lot, which sounds exciting,
but it never was.*

-- Miranda Lambert

8.1 Conclusões do projeto de formatura

O sistema descrito foi capaz de cumprir o objetivo de rastrear diferentes pessoas em múltiplas câmeras. No entanto, seu desempenho em termos de acurácia não é capaz de se manter conforme o requisito de 80% de acerto, principalmente com o decorrer do tempo, devido ao fenômeno de propagação de erro na re-identificação de pessoas. Durante a implementação, também foram detectados alguns problemas inerentes às técnicas apresentadas e apesar de não da falta de viabilidade de se solucioná-los, foram discutidas algumas possíveis soluções, que podem servir de base para pesquisas futuros.

8.2 Contribuições

Este projeto foi importante para apresentar uma forma de rastrear pessoas de forma autônoma utilizando apenas imagens, combinando-se três técnicas que abordam uma parte do problema (*tracking*, detecção de pessoas e identificação de pessoas), para se solucionar o problema como um todo. Assim, este trabalho serve de base para projetos semelhantes, que podem se utilizar desta solução para testar o desempenho do sistema utilizando-se outras tecnologias para cada uma destas partes, ou tentando-se substituir uma dessas técnicas por outra.

O sistema final por sua vez também pode ser encorporado em outros sistemas com aplicações específicas, como um sistema de vigilância ou um sistema de monitoramento de um estabelecimento comercial. Assim seria possível através do rastreamento do sistema aqui desenvolvido o levantamento de dados que seriam úteis para outras aplicações.

8.3 Perspectivas de continuidade

A partir deste trabalho pode-se realizar outros projetos que se utilizam do sistema para operar em uma aplicação específica, como um sistema de monitoramento e segurança, valendo-se do rastreamento e identificação que foi implementado.

Ademais, este projeto pode servir de base para outros projetos acadêmicos, a fim de melhorar seu desempenho e testar hipóteses aqui levantadas, como a integração do algoritmo de rastreamento na rotina de reidentificação para obtenção de melhor acurácia, e pesquisa na literatura de técnicas que possam ser utilizadas para amenizar ou solucionar os pontos negativos encontrados no sistema, como a propagação do erro ao longo do passar do tempo de execução.

REFERÊNCIAS

- 1 HERMANS, A.; BEYER, L.; LEIBE, B. *In Defense of the Triplet Loss for Person Re-Identification*. 2017.
- 2 WU, Y.; LIM, J.; YANG, M.-H. Online object tracking: A benchmark. *IEEE Conference on*, 2013.
- 3 PAREKH, H. S.; THAKORE, D. G.; JALIYA, U. K. A survey on object detection and tracking methods. *International Journal of Innovative Research in Computer and Communication Engineering*, v. 2, n. 2, 2014.
- 4 REDDY, K. R.; PRIYA, K. H.; NEELIMA, N. Object detection and tracking – a survey. *International Conference on Computational Intelligence and Communication Networks*, 2015.
- 5 LI, M. et al. A survey of video object tracking. *International Journal of Control and Automation*, v. 8, n. 9, p. 303–312, 2015.
- 6 FAN, L. et al. A survey on multiple object tracking algorithm. *IEEE International Conference on Information and Automation*, 2016.
- 7 LUO, W. et al. Multiple object tracking: A literature review. 2017.
- 8 VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: IEEE. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. [S.l.], 2001. v. 1, p. I–I.
- 9 FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, Elsevier, v. 55, n. 1, p. 119–139, 1997.
- 10 DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: IEEE. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. [S.l.], 2005. v. 1, p. 886–893.
- 11 JOACHIMS, T. *Making large-scale SVM learning practical*. [S.l.], 1998.
- 12 GRABNER, H.; GRABNER, M.; BISCHOF, H. Real-time tracking via on-line boosting. In: *Bmvc*. [S.l.: s.n.], 2006. v. 1, n. 5, p. 6.
- 13 BABENKO, B.; YANG, M.-H.; BELONGIE, S. Visual tracking with online multiple instance learning. In: IEEE. *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. [S.l.], 2009. p. 983–990.
- 14 HENRIQUES, J. F. et al. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, v. 37, n. 3, p. 583–596, 2015.

- 15 KALAL, Z. et al. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, v. 34, n. 7, p. 1409, 2012.
- 16 KALAL, Z.; MIKOLAJCZYK, K.; MATAS, J. Forward-backward error: Automatic detection of tracking failures. In: IEEE. *Pattern recognition (ICPR), 2010 20th international conference on*. [S.l.], 2010. p. 2756–2759.
- 17 HELD, D.; THRUN, S.; SAVARESE, S. Learning to track at 100 fps with deep regression networks. In: SPRINGER. *European Conference on Computer Vision*. [S.l.], 2016. p. 749–765.
- 18 LUKEZIC, A. et al. Discriminative correlation filter with channel and spatial reliability. In: *CVPR*. [S.l.: s.n.], 2017. v. 6, p. 8.
- 19 ZHENG, L.; YANG, Y.; HAUPTMANN, A. G. *Person Re-identification: Past, Present and Future*. 2016.
- 20 ZHANG, X. et al. *AlignedReID: Surpassing Human-Level Performance in Person Re-Identification*. 2017.
- 21 IODICE, S.; MIKOLAJCZYK, K. *Partial Person Re-identification with Alignment and Hallucination*. 2018.
- 22 LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998.
- 23 NAGI, J. et al. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In: IEEE. *Signal and Image Processing Applications (ICSIPA), 2011 IEEE International Conference on*. [S.l.], 2011. p. 342–347.
- 24 SRIVASTAVA, R. K.; GREFF, K.; SCHMIDHUBER, J. *Highway Networks*. 2015.
- 25 HE, K. et al. *Deep Residual Learning for Image Recognition*. 2015.
- 26 SCHROFF, F.; KALENICHENKO, D.; PHILBIN, J. Facenet: A unified embedding for face recognition and clustering. 2015.
- 27 HE, K. et al. Spatial pyramid pooling in deep convolutional networks for visual recognition. 2014.
- 28 OPENCV API Reference. Disponível em: <<https://docs.opencv.org/3.0-beta/modules/refman.html>>.
- 29 REDMON, J. *Darknet: Open Source Neural Networks in C*. 2013–2016. <<http://pjreddie.com/darknet/>>.
- 30 REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv*, 2018.
- 31 EVERINGHAM, M. et al. The pascal visual object classes (voc) challenge. *International journal of computer vision*, Springer, v. 88, n. 2, p. 303–338, 2010.
- 32 TRIPLET ReID. Disponível em: <<https://github.com/gustavotorresm/triplet-reid>>.
- 33 CVLAB. *Multi-camera pedestrian video*. 2016. Disponível em: <<https://cvlab.epfl.ch/data/pom/>>.

- 34 OXFORD, A. V. L. D. of Engineering Science University of. *Coarse Gaze Estimation in Visual Surveillance*. 2016. Disponível em: <<http://www.robots.ox.ac.uk/~lav/Research/Projects/2009bbenfold\headpose/project.html>>.
- 35 STORYBLOCKS. *City Creek Center*. Disponível em: <<https://www.videoblocks.com/video/urban-mall-and-shopping-center-christmas-decoration-day-dci-4k-901-city-creek-center-across-from>>.
- 36 SHARMA, G.; WU, W.; DALAL, E. N. The ciede2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. 2004.
- 37 DUCKER, M. *What Color is the Hottest Flame Lovely What is the Color Of Fire*. Disponível em: <<https://motherducker.us/what-color-is-the-hottest-flame/what-color-is-the-hottest-flame-lovely-what-is-the-color-of-fire/>>.
- 38 LIU, W.; CAMPS, O.; SZNAIER, M. Multi-camera multi-object tracking. 2017.
- 39 RISTANI, E. et al. Performance measures and a data set for multi-target, multi-camera tracking. In: *European Conference on Computer Vision workshop on Benchmarking Multi-Target Tracking*. [S.l.: s.n.], 2016.
- 40 KAEWTRAKULPONG, P.; BOWDEN, R. An improved adaptive background mixture model for real-time tracking with shadow detection. In: *Video-based surveillance systems*. [S.l.]: Springer, 2002. p. 135–144.
- 41 SOLERA, F. et al. Tracking social groups within and across cameras. *IEEE Transactions on Circuits and Systems for Video Technology*, 2016.
- 42 RISTANI, E.; TOMASI, C. Tracking multiple people online and in real time. In: SPRINGER. *Asian Conference on Computer Vision*. [S.l.], 2014. p. 444–459.