

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

**TRACKING DE OBJETOS EM MOVIMENTO EM
FONTES INDEPENDENTES**

São Paulo
5 de dezembro de 2018

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

**TRACKING DE OBJETOS EM MOVIMENTO EM
FONTES INDEPENDENTES**

Trabalho apresentado à Escola Politécnica
da Universidade de São Paulo para ob-
tenção do Título de Engenheiro Engenharia
de Computação.

São Paulo
5 de dezembro de 2018

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

**TRACKING DE OBJETOS EM MOVIMENTO EM
FONTES INDEPENDENTES**

Trabalho apresentado à Escola Politécnica
da Universidade de São Paulo para ob-
tenção do Título de Engenheiro Engenharia
de Computação.

Área de Concentração:
Visão Computacional

Orientador:
Prof. Dr. Edson Satoshi Gomi

São Paulo
5 de dezembro de 2018

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

Desde o chamado *Renascimento do Deep Learning*, o campo da visão computacional tem uma crescente importância na sociedade e como tanto vem se tornando objeto de estudo de um sem-número de pesquisas acadêmicas. Inobstante, apesar destas infinitas técnicas no “estado-da-arte”, capazes de superarem humanos nas tarefas de reconhecer e interpretar imagens, são poucas as aplicações no qual estas tecnologias são deveras empregadas. Este trabalho visa mudar este cenário e aproximar modelos altamente performáticos de sistemas do mundo real, em um contexto, não menos relevante que as tecnologias mencionadas, de rastreamento e monitoramento de múltiplas pessoas em múltiplas câmeras.

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

ABSTRACT

Since the *Renaissance of Deep Learning*, the field of computer vision has got an ever-growing importance in society and thus is a subject of a myriad of academic researches lately. Yet, although we've got an infinite number of cutting-edge techniques, capable of surpassing human capabilities in distinguishing and reasoning about images, few are the application in which such technologies are *de facto* employed. This work aims at changing this scenario and decreasing the distance between high-performing models and real-world systems, in a context that itself is not less relevant than the former mentioned technologies, of tracking and surveillance of multiple people in multiple cameras.

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

LISTA DE FIGURAS

1	Modelo de rede neural convolucional de 7 camadas	20
2	Camada de <i>spatial pyramid pooling</i>	23
3	Exemplo de funcionamento do <i>YOLO</i>	27
4	Arquitetura da rede neural do <i>YOLO</i>	27
5	Diagrama dos módulos do sistema	31
6	Exemplo do funcionamento do sistema.	32
8	Exemplo do funcionamento do algoritmo de rastreamento.	37
9	Representação visual das cores no espaço de cores CIELAB	39
10	Diagrama com as partes do sistema.	41
11	Resultados dos testes de [28] com 364 imagens do <i>dataset</i> de <i>Duke</i>	43

LISTA DE TABELAS

1	Benchmark de algoritmo de re-identificação de pessoas	40
2	Benchmark de algoritmo de re-identificação de pessoas com remoção de <i>background</i>	42

SUMÁRIO

Parte I: INTRODUÇÃO	11	
1	Introdução	12
1.1	Objetivo	12
1.2	Motivação	12
1.3	Justificativa	13
1.4	Organização do Trabalho	13
2	Aspectos Conceituais	14
2.1	Identificação de Pessoas	14
2.2	<i>Tracking</i> em Vídeo	15
2.2.1	<i>Boosting Tracker</i>	15
2.2.2	<i>MIL Tracker</i>	15
2.2.3	<i>KCF Tracker</i>	16
2.2.4	<i>TLD Tracker</i>	16
2.2.5	<i>MedianFlow Tracker</i>	16
2.2.6	<i>GOTURN Tracker</i>	16
2.2.7	<i>MOSSE Tracker</i>	17
2.3	Correspondência de Imagens	18
2.3.1	Redes neurais convolucionais e deep-learning	19
2.3.2	Rede residual e <i>ResNet</i>	20
2.3.3	Triplet loss	21
2.3.4	<i>Pooling</i> espacial	22
3	Tecnologias utilizadas	24

3.1	Python	24
3.2	<i>Frameworks de machine-learning</i>	24
3.3	OpenCV	25
3.3.1	Leitura de imagens	25
3.3.2	Remoção de <i>background</i>	25
3.3.3	Identificação de pessoas	25
3.3.4	<i>Tracking</i> de pessoas	26
3.4	<i>Darknet</i>	26
3.5	YOLO	26
3.6	Trinet	28
3.7	Google Cloud Platform	29
4	Metodologia de trabalho	30
4.1	Concepção	30
4.2	Projeto	30
4.2.1	Readequação de escopo	31
4.3	Implementação	31
4.4	Testes	32
5	Requisitos do sistema	34
5.1	Requisitos	34
5.1.1	Entrada e saída de vídeo	34
5.1.2	Cliente	35
6	Projeto e implementação	36
6.1	Testes e Provas de conceito	36
6.1.1	Identificação de pessoas	36
6.1.2	<i>Tracking</i>	36

6.1.3	Re-identificação de pessoas	37
6.1.3.1	Algoritmos tradicionais	37
6.1.3.2	Algoritmos com redes neurais	42
6.2	Projeto e implementação	43
6.2.1	Sistema de <i>tracking</i>	43
6.2.2	Sistema de detecção de pessoas	44
6.2.3	Sistema de re-identificação	44
6.2.4	Implementação	44
7	Testes e avaliação	46
8	Considerações finais	47
8.1	Conclusões do projeto de formatura	47
8.2	Contribuições	47
8.3	Perspectivas de continuidade	48
Bibliografia		49

PARTE I

INTRODUÇÃO

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 de conclusão de curso é realizar o projeto de um sistema computacional inteligente que seja capaz de identificar e rastrear de forma autônoma pessoas deslocando-se em um ambiente monitorado por múltiplas câmeras.

Além disso, este sistema é capaz de inferir informações relevantes sobre o rastreamento, como o trajeto que uma pessoa percorreu, em quais das câmeras esta pessoa esteve presente e regiões de interesse no qual essa pessoa, e.g., um local no qual ela esteve parada por um tempo superior a um valor predeterminado.

Este sistema tem por objetivo servir como objeto de estudo para a agração, maturação e criação de conhecimentos na área de visão computacional e aprendizagem de máquina, verificando seu desempenho para aplicações de interesse a outras áreas do conhecimento.

1.2 Motivação

É incontestável que o uso de computadores para executar tarefas complexas para um ser humano no âmbito de visão computacional está se tornando cada vez mais importante. É, portanto, de grande interesse que uma máquina seja capaz de realizar uma função fundamental, como rastrear uma pessoa. Apesar de existirem algoritmos satisfatórios neste campo [1], eles ainda são restritos e não são capazes de, por exemplo, rastrear um mesmo objeto em diferentes capturas.

Com o *Renascimento* das redes neurais e o advento do *Deep Learning*, surgem novas técnicas capazes de impulsionar o campo da visão computacional, criando modelos mais robustos e que sejam bem difundidos na literatura e na comunidade. Essa miríade de mo-

delos pré-treinados possibilita sua fácil adaptação para outras aplicações, como o *tracking* e a *re-identificação* de pessoas.

1.3 Justificativa

O reconhecimento e rastreamento de objetos tem diversas aplicações na vida atual. Podemos citar o uso na área de segurança para a vigilância de pessoas, na área acadêmica para o acompanhamento de animais em estudos comportamentais, ou até no comércio com a análise do comportamento dos grupos de pessoas em um centro de lojas.

Esse último servirá de base para o nosso projeto. Vamos aplicar o sistema desenvolvido para analisar o comportamento de compradores em um centro comercial, buscando encontrar padrões comuns entre os clientes, hábitos repetidos entre os consumidores que mais compram, ou qualquer outra característica que possa ser útil para as lojas entenderem seus fregueses.

1.4 Organização do Trabalho

Na *Parte I* (pág. 12) apresentamos uma introdução ao trabalho, na qual descrevemos o objetivo, a motivação e a justificativa para o trabalho. Na *Parte II* (pág. 14) discorremos sobre os aspectos conceituais abordados no projeto. Na *Parte III* (pág. 24) falamos das tecnologias utilizadas no projeto. Na *Parte IV* (pág. 30) mostramos a metodologia de trabalho, com as fases de desenvolvimento. *Parte V* (pág. 34) temos as especificações e requisitos do sistema desenvolvido. Na *Parte VI* (pág. 36) expomos os resultados obtidos com o projeto e implementação do sistema. Na *Parte VII* (pág. 46) exibimos os procedimentos de teste do sistema. Por fim, na *Parte VIII* (pág. 47) fazemos as considerações finais do projeto.

2 ASPECTOS CONCEITUAIS

O rastreamento de pessoas em um vídeo passa por três principais atividades, como descrito em [2]:

- 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: aproximar 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 [2]–[6]. Como o intuito desse trabalho é o rastreamento de pessoas, juntaremos as atividades de detecção e classificação de objetos em uma só, chamadas aqui de identificação de pessoas. A seguir, abordaremos 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 abaixo, e também o framework *YOLO*, descrito na seção 3.5.

O algoritmo *Haar Cascade* [7] 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* [8], 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* [9] 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* [10], 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* [11] 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* [12], 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* [13] 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* [14], 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* [15] 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* [16] 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* [17], 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.

Podemos 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*[18][19][20], 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 à algoritmos de visão computacional e que também será 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 **lecun**. 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*.

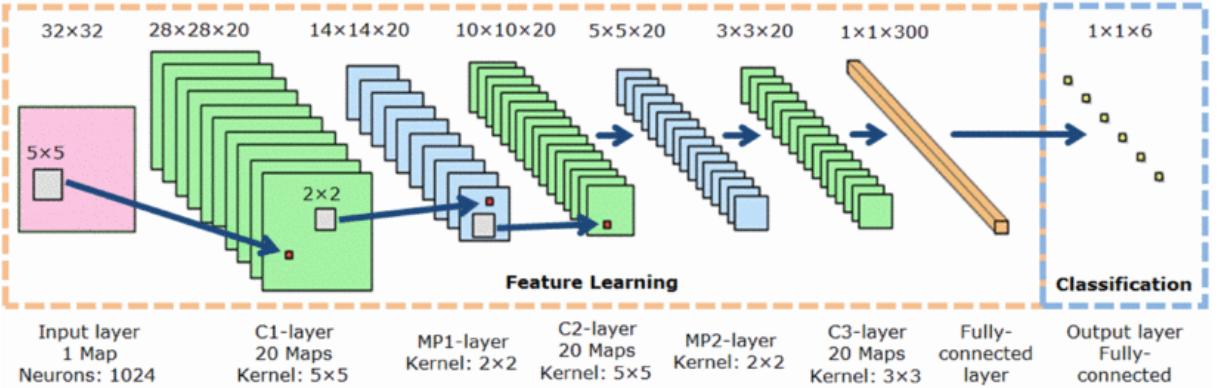


Figura 1: Modelo de rede neural convolucional de 7 camadas

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*[21].

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*[22]. 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 **facenet** 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 [23].

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.[23] A figura 2 ilustra a ação deste tipo de camada.

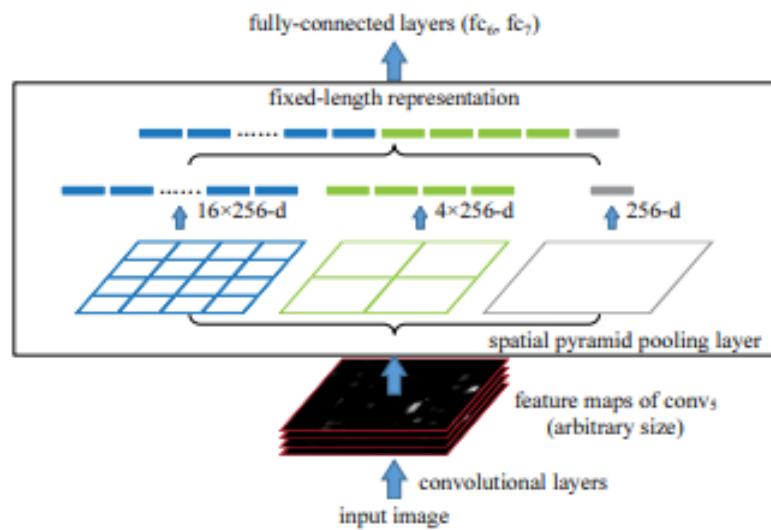


Figura 2: Camada de *spatial pyramid pooling*

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 e provas de conceito, 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 [24].

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 *provas de conceito*, 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 [25] 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* [26] 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* [27]. 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.

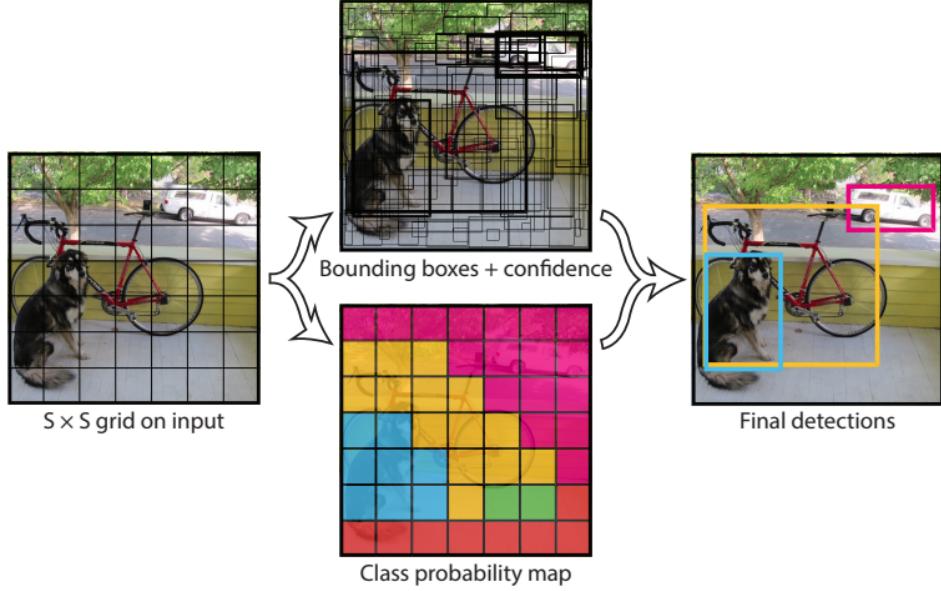


Figura 3: Exemplo de funcionamento do (YOLO), retirado de [26]. Na esquerda, a imagem inicial dividida em uma grade de $S \times S$. No meio, os $7 \times 7 \times 2$ retângulos de demarcação em cima e o mapa de probabilidade de classes abaixo. Na direita, as detecções finais.

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.

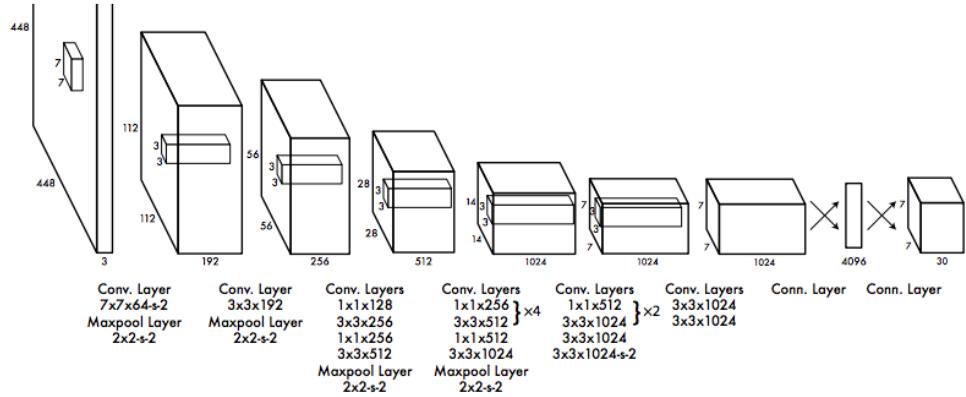


Figura 4: Arquitetura da rede neural do *YOLO*, retirada de [26]. São 24 camadas convolucionais seguidas por duas camadas totalmente conectadas.

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* [28] é uma arquitetura de rede neural proposta para solucionar o problema de re-identificação de pessoas em câmeras de segurança, de código aberto [29] e, portanto, amplamente testado e avaliado. A arquitetura da rede segue o modelo *ResNet50*, possuindo, portanto 50 camadas treináveis, como mostrado em ??.

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 [30], [31] 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.

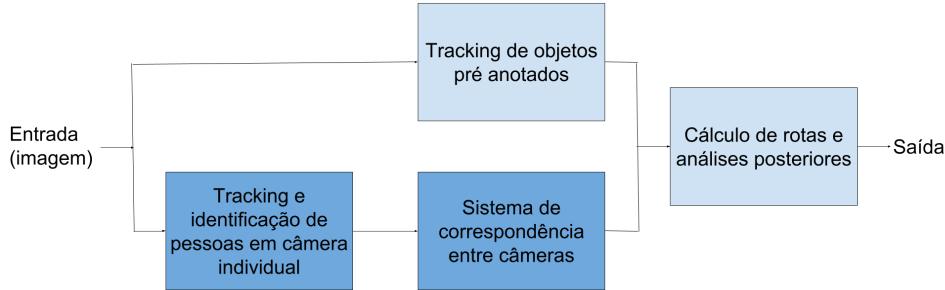


Figura 5: Diagrama dos módulos do sistema. Em azul escuro, as partes essenciais. Em azul claro, possíveis partes extras para a aplicação em análise de comportamento de cliente.

- 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.

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

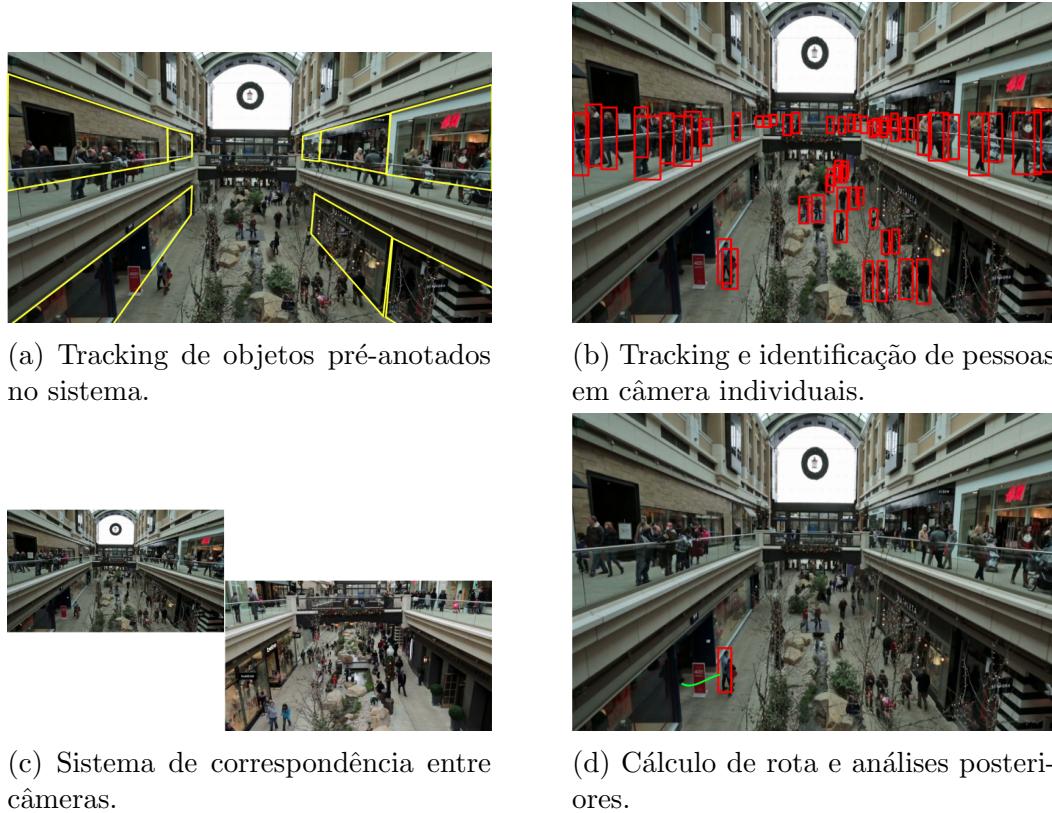


Figura 6: Exemplo do funcionamento do sistema.

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.

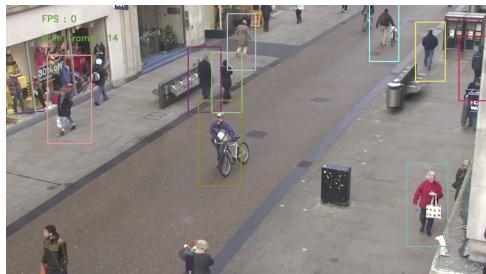
6 PROJETO E IMPLEMENTAÇÃO

6.1 Testes e Provas de conceito

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. Os resultados podem ser visualizados na tabela ???. 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 ???. 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.



(a) Identificação de pessoas utilizando *OpenCV*.



(b) Identificação de pessoas utilizando *YOLOv3*.

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.**tracking-bm** Novamente, a preferência por precisão sobre performance teve impacto no modelo do sistema, como será apresentado em ??.

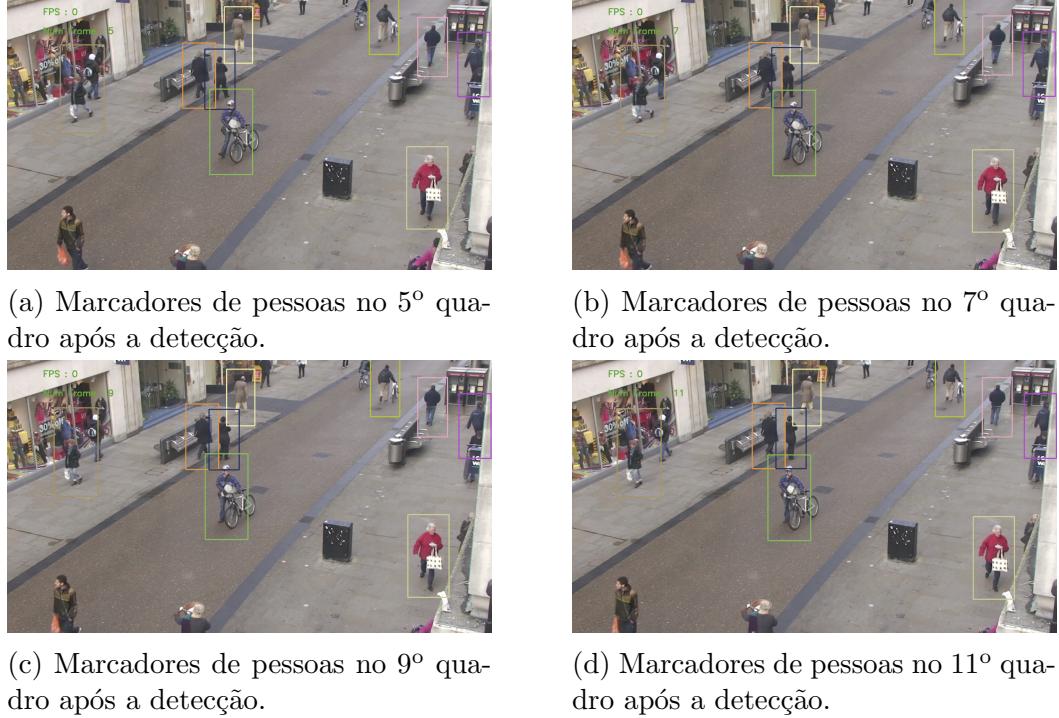


Figura 8: Exemplo do funcionamento do algoritmo de rastreamento.

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 [32].

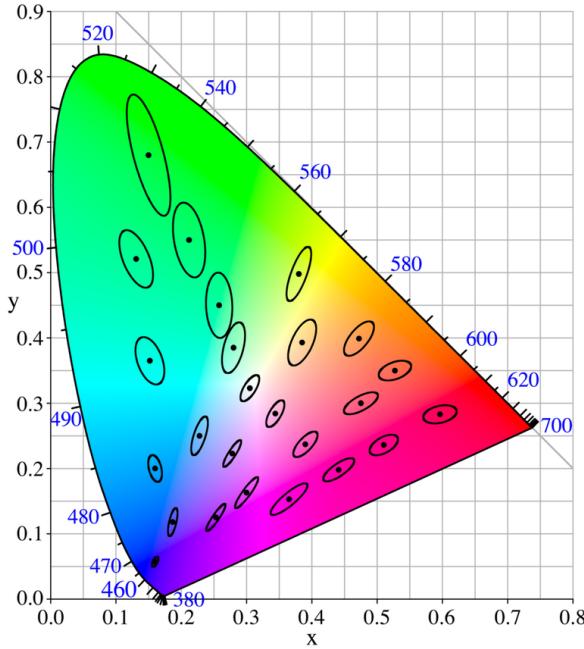


Figura 9: Representação visual das cores no espaço de cores CIELAB

Aplicando os conceitos já provados em [33], 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[34]. 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 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

	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.

OpenCV.

Realizando testes visuais de remoção de *background*, foi considerada como melhor função a "*BackgroundSubtractorMOG*", que implementa o algoritmo introduzido no artigo [35], 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 [28] disponível em [29]. O teste foi feito utilizando 364 imagens do *dataset* de *Duke* [36] 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

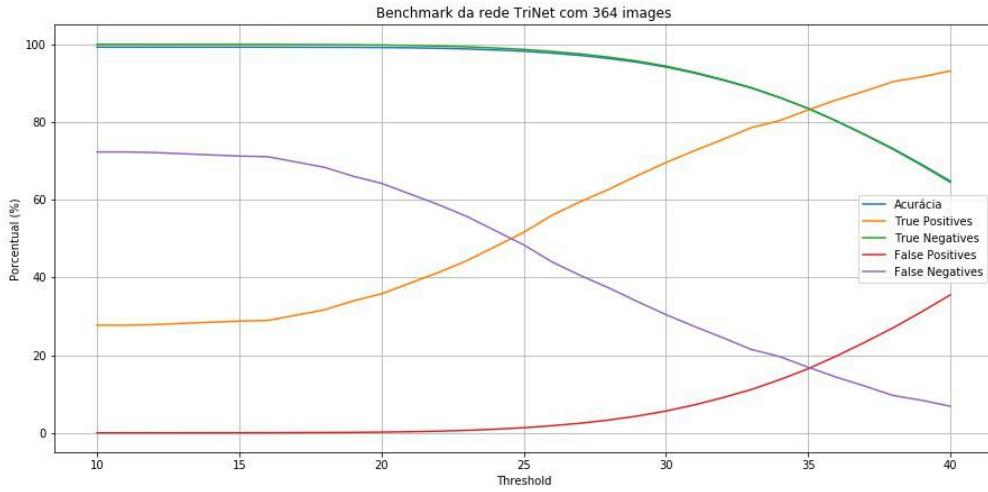


Figura 11: Resultados dos testes de [28] com 364 imagens do *dataset* de *Duke*.

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.

6.2.4 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

Os testes finais foram realizados após a montagem do sistema, e envolvem testes manuais realizados em um ambiente real, com potenciais usuários.

A partir dos testes foi possível dimensionar o escopo do projeto e suas limitação. Dentre as limitações encontradas, está a perda de acurácia para câmeras cuja saturação e temperatura de cores não estejam semelhantes, ocasionando um elevado número de falsos negativos. Portanto, adicionou-se aos pré-requisitos do sistema que as câmeras devem ser previamente ajustadas para manter as métricas acima.

8 CONSIDERAÇÕES FINAIS

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

-- Miranda Lambert

8.1 Conclusões do projeto de formatura

Este projeto cumpriu as expectativas como um todo, e foi capaz de cumprir seu objetivo principal de rastrear pessoas em múltiplas câmeras, combinando diferentes técnicas do “estado da arte” para tal. No entanto, foi possível perceber dificuldades e limitações nas tecnologias escolhidas, que acarretaram numa restrição do escopo do projeto. Entre elas, destacam-se a eficiência computacional dos algoritmos, que não permite um funcionamento *seamless* em *real time*, sendo os pontos de gargalo a infraestrutura disponível e os algoritmos de *tracking*, que não são paralelizáveis.

8.2 Contribuições

A realização do projeto possui como principal contribuição mostrar como se pode integrar diferentes tecnologias *cutting edge* de *visão computacional* para a realização de um sistema real, com objetivos reais, principalmente no Brasil, onde a presença destes algoritmos ainda não se faz tão presente como nos grandes centros tecnológicos, tanto do ponto de vista acadêmico quanto industrial.

Ademais, este sistema também serve como base para projetos semelhantes que necessitam do rastreamento pessoas, como na área de segurança. O sistema é flexível o suficiente para permitir adaptações para outras aplicações, como o rastreamento de animais ao invés de pessoas, bastando para isso utilizar um *dataset* adequado para o treinamento da rede. Ele também é robusto o suficiente para que seja utilizado até mesmo para aplicações de uma só câmera, por possuir maior acurácia que as técnicas clássicas utilizadas.

Finalmente, no âmbito social, este projeto é importante para trazer à tona discussões acerca de pontos sensíveis que este projeto engloba e que também são muito abordados em outras aplicações de visão computacional. Um destes pontos é a utilização de sistemas

inteligentes para o *tracking* de pessoas, cuja prática deve ser ponderada especificamente para as diferentes aplicações, já que muitas vezes pode não ser considerada ética ou até mesmo legal. Outro ponto é como manter a privacidade das pessoas com tal tipo de sistema, o que se torna algo ainda mais crítico com a técnica de *re-identificação*.

8.3 Perspectivas de continuidade

Para trabalhos futuros, pode-se atacar os pontos deste projeto que impactaram negativamente na concepção do sistema, para se obter um sistema mais performático sem a necessidade de se escalar verticalmente a máquina no qual ele se hospeda. Os principais pontos a serem explorados são os algoritmos de *multi-tracking* e a comunicação das câmeras com o servidor. Em termos de acurácia dos resultados obtidos, assim como todo modelo de *machine-learning*, podem ser aplicadas novas técnicas e realizados experimentos de modificação para se tentar obter melhores resultados. Também, se disponíveis mais recursos, poder-se-ia criar um *dataset* com maior qualidade, no qual tenta-se sanar os problemas mais comuns exibidos pelo sistema, como a sensibilidade à qualidade da imagem.

BIBLIOGRAFIA

- [1] Y. Wu, J. Lim e M.-H. Yang, “Online object tracking: A benchmark”, *IEEE Conference on*, 2013.
- [2] H. S. Parekh, D. G. Thakore e U. K. Jaliya, “A survey on object detection and tracking methods”, *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 2, nº 2, 2014.
- [3] K. R. Reddy, K. H. Priya e N. Neelima, “Object detection and tracking – a survey”, *International Conference on Computational Intelligence and Communication Networks*, 2015.
- [4] M. Li, Z. Cai, C. Wei e Y. Yuan, “A survey of video object tracking”, *International Journal of Control and Automation*, vol. 8, nº 9, pp. 303–312, 2015.
- [5] L. Fan, Z. Wang, B. Cail, C. Tao, Z. Zhang, Y. Wang, S. Li, F. Huang, S. Fu e F. Zhang, “A survey on multiple object tracking algorithm”, *IEEE International Conference on Information and Automation*, 2016.
- [6] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, X. Zhao e T.-K. Kim, “Multiple object tracking: A literature review”, 2017.
- [7] P. Viola e M. Jones, “Rapid object detection using a boosted cascade of simple features”, em *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, IEEE, vol. 1, 2001, pp. I–I.
- [8] Y. Freund e R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting”, *Journal of computer and system sciences*, vol. 55, nº 1, pp. 119–139, 1997.
- [9] N. Dalal e B. Triggs, “Histograms of oriented gradients for human detection”, em *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, IEEE, vol. 1, 2005, pp. 886–893.
- [10] T. Joachims, “Making large-scale svm learning practical”, Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund, rel. téc., 1998.

- [11] H. Grabner, M. Grabner e H. Bischof, “Real-time tracking via on-line boosting.”, em *Bmvc*, vol. 1, 2006, p. 6.
- [12] B. Babenko, M.-H. Yang e S. Belongie, “Visual tracking with online multiple instance learning”, em *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, IEEE, 2009, pp. 983–990.
- [13] J. F. Henriques, R. Caseiro, P. Martins e J. Batista, “High-speed tracking with kernelized correlation filters”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, nº 3, pp. 583–596, 2015.
- [14] Z. Kalal, K. Mikolajczyk, J. Matas et al., “Tracking-learning-detection”, *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, nº 7, p. 1409, 2012.
- [15] Z. Kalal, K. Mikolajczyk e J. Matas, “Forward-backward error: Automatic detection of tracking failures”, em *Pattern recognition (ICPR), 2010 20th international conference on*, IEEE, 2010, pp. 2756–2759.
- [16] D. Held, S. Thrun e S. Savarese, “Learning to track at 100 fps with deep regression networks”, em *European Conference on Computer Vision*, Springer, 2016, pp. 749–765.
- [17] A. Lukezic, T. Vojir, L. C. Zajc, J. Matas e M. Kristan, “Discriminative correlation filter with channel and spatial reliability.”, em *CVPR*, vol. 6, 2017, p. 8.
- [18] L. Zheng, Y. Yang e A. G. Hauptmann, *Person re-identification: Past, present and future*, 2016. eprint: [arXiv:1610.02984](https://arxiv.org/abs/1610.02984).
- [19] X. Zhang, H. Luo, X. Fan, W. Xiang, Y. Sun, Q. Xiao, W. Jiang, C. Zhang e J. Sun, *Alignedreid: Surpassing human-level performance in person re-identification*, 2017. eprint: [arXiv:1711.08184](https://arxiv.org/abs/1711.08184).
- [20] S. Iodice e K. Mikolajczyk, *Partial person re-identification with alignment and hallucination*, 2018. eprint: [arXiv:1807.09162](https://arxiv.org/abs/1807.09162).
- [21] R. K. Srivastava, K. Greff e J. Schmidhuber, *Highway networks*, 2015. eprint: [arXiv:1505.00387](https://arxiv.org/abs/1505.00387).
- [22] K. He, X. Zhang, S. Ren e J. Sun, *Deep residual learning for image recognition*, 2015. eprint: [arXiv:1512.03385](https://arxiv.org/abs/1512.03385).
- [23] ——, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, 2014. DOI: [10.1007/978-3-319-10578-9_23](https://doi.org/10.1007/978-3-319-10578-9_23). eprint: [arXiv:1406.4729](https://arxiv.org/abs/1406.4729).
- [24] *Opencv api reference*. endereço: <https://docs.opencv.org/3.0-beta/modules/refman.html> (acesso em 14/10/2018).

- [25] J. Redmon, *Darknet: Open source neural networks in c*, <http://pjreddie.com/darknet/>, 2013–2016.
- [26] J. Redmon e A. Farhadi, “Yolov3: An incremental improvement”, *ArXiv*, 2018.
- [27] M. Everingham, L. Van Gool, C. K. Williams, J. Winn e A. Zisserman, “The pascal visual object classes (voc) challenge”, *International journal of computer vision*, vol. 88, nº 2, pp. 303–338, 2010.
- [28] A. Hermans, L. Beyer e B. Leibe, *In defense of the triplet loss for person re-identification*, 2017. eprint: arXiv:1703.07737.
- [29] *Triplet reid*. endereço: <https://github.com/gustavotorresm/triplet-reid> (acesso em 03/12/2018).
- [30] CVLAB. (2016). Multi-camera pedestrian video, endereço: <https://cvlab.epfl.ch/data/pom/> (acesso em 19/03/2018).
- [31] A. V. L. .-. D. of Engineering Science - University of Oxford. (2016). Coarse gaze estimation in visual surveillance, endereço: http://www.robots.ox.ac.uk/~lav/Research/Projects/2009bbenfold_headpose/project.html (acesso em 18/10/2018).
- [32] G. Sharma, W. Wu e E. N. Dalal, “The ciede2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations”, 2004.
- [33] W. Liu, O. Camps e M. Sznajer, “Multi-camera multi-object tracking”, 2017.
- [34] E. Ristani, F. Solera, R. Zou, R. Cucchiara e C. Tomasi, “Performance measures and a data set for multi-target, multi-camera tracking”, em *European Conference on Computer Vision workshop on Benchmarking Multi-Target Tracking*, 2016.
- [35] P. KaewTraKulPong e R. Bowden, “An improved adaptive background mixture model for real-time tracking with shadow detection”, em *Video-based surveillance systems*, Springer, 2002, pp. 135–144.
- [36] E. Ristani, F. Solera, R. Zou, R. Cucchiara e C. Tomasi, “Performance measures and a data set for multi-target, multi-camera tracking”, em *European Conference on Computer Vision workshop on Benchmarking Multi-Target Tracking*, 2016.