UNIVERSIDADE FEDERAL DO PARANÁ

GUILHERME CORDEIRO VOGT
LUIS GUILHERME DIAS DE SOUZA

NEXTCAM - SISTEMA EMBARCADO DE DETECÇÃO DE PEDESTRES POR IMAGEM EM TEMPO REAL

CURITIBA
2018

GUILHERME CORDEIRO VOGT
LUIS GUILHERME DIAS DE SOUZA

NEXTCAM - SISTEMA EMBARCADO DE DETECÇÃO DE PEDESTRES POR IMAGEM EM TEMPO REAL

Projeto apresentado como requisito à obtenção de nota da disciplina de Trabalho de Conclusão de Curso B, do Curso de Engenharia Elétrica com Ênfase em Sistemas Eletrônicos Embarcados.
Orientador: Prof. Dr. Eduardo Parente Ribeiro

CURITIBA
2018

TERMO DE APROVAÇÃO

GUILHERME CORDEIRO VOGT
LUIS GUILHERME DIAS DE SOUZA

NEXTCAM – SISTEMA EMBARCADO DE DETECÇÃO DE PEDESTRES POR IMAGEM EM TEMPO REAL

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica da Universidade Federal do Paraná como requisito à obtenção do grau Engenheiro Eletricista, pela seguinte banca examinadora:

_____

Orientador: Prof. Dr. Eduardo Parente Ribeiro
Departamento de Engenharia Elétrica, UFPR

_____

Prof. Dr. José Carlos da Cunha
Departamento de Engenharia Elétrica, UFPR

_____

Profa. M. Sc. Theoma Muriel Sanches Otobo
Departamento de Engenharia Elétrica, UFPR

Curitiba, 04 de dezembro de 2018

AGRADECIMENTOS

A Equipe agradece a universidade, professores, técnicos administrativos e todos os colegas de graduação, pelo ambiente prolífico e aberto ao debate científico. Sem a criteriosidade com a qual o método fora apresentado e ensinado este trabalho não seria possível. Aos colegas, agradecemos por todos esses anos de convívio e amizades, os levaremos por toda vida.

A Equipe agradece as nossas famílias e amigos pelo apoio e atenção dedicadas ao longo da trajetória. Sem o comprometimento e paciência apresentados a execução deste projeto seria muito mais desgastante.

Eu, Luis Guilherme Dias de Souza, agradeço à toda minha família, em especial aos meus pais, Heraldo J. L. de Souza e Denise Dias de Souza, meus irmãos Luis Gustavo Dias de Souza e Francine L. Dias de Souza pela dedicação, amor, motivação e apoio em toda minha jornada acadêmica.

Eu, Guilherme Cordeiro Vogt, agradeço à toda minha família, em especial aos meus pais, Ana Maria Cordeiro Vogt e Sergio Luis Vogt, meu irmão Vinicius Cordeiro Vogt e minha companheira Renata Letícia Castro Santos pela dedicação, amor, atenção, carinho e suporte através de tantos desafios.

“It never gets any easier. The nature of the problems change.”

Paul Graham

RESUMO

Segurança é um mercado muito importante e amplo na nossa sociedade, onde um dos métodos de vigilância passivo mais utilizado é o por meio de câmeras digitais. Com o constante aumento na violência e as deficiências e vulnerabilidades dos sistemas de vigilância por câmeras disponíveis, o sistema de câmeras inteligentes abordado neste documento apresenta-se como uma solução mais viável ao problema. O objetivo deste projeto consiste em desenvolver um dispositivo de monitoramento embarcado inteligente, com um algoritmo de reconhecimento de pedestres singular. Aliando a melhor relação precisão/desempenho, com um sistema de baixo custo e com funcionamento em tempo real. Desenvolvendo um *software* embarcado, em conjunto com um *hardware* formado por um microcomputador e o módulo de uma câmera para captação de imagens, onde o processamento do detector é local e viável em aplicações de segurança. A maior parte dos detectores de pedestres hoje propostos possuem foco em precisão, porém são poucos que empregam técnicas de optimização ou importam-se com desempenho. O conjunto de dados de imagens que será usado no treinamento dos algoritmos do sistema é o *INRIA Person Dataset*, que contém 614 imagens de treinamento e 288 imagens de testes. Os resultados desse artigo indicam uma evolução dos algoritmos de inteligência artificial embarcados, com foco em miniaturização de modelos aprendizagem, que possibilitam o uso de técnicas mais complexas em plataformas cada vez mais simples.

Palavras-chave: Câmera. Embarcado. Pedestre. Detector. CNN. INRIA. *Caltech Pedestrian Dataset*

ABSTRACT

Private safety is a wide and very important market in our society, where one of the most common vigilance methods is via digital cameras. With the ongoing rise in violence and the vulnerabilities of the camera monitoring systems on the present market, the smart cameras system discussed at this paper proposes a more viable solution to the problem. The main goal of this project is to develop an embedded monitoring device with its own pedestrian detection algorithm, focused on finding the best precision/performance relation, low cost and real-time operation. This project consists of the development of an embedded software suited for a hardware composed of a microcomputer and a camera module, therefore having locally processed images, allowing for more complex safety applications. Most part of today's pedestrian detectors have their main focus on enhancing their precision, very few of them strike the performance issue, nor employ optimization techniques. The train and test dataset used for developing the detection algorithm of this paper is the INRIA Person Dataset, containing around 900 images total. This article results indicate an evolution of the embedded artificial intelligence algorithms, with focus on learning model miniaturization, which allow the usage of more complex techniques on increasingly simple platforms.

Key-words: *Camera*. *Embedded*. *Pedestrian*. *Detector*. CNN. INRIA. *Caltech Pedestrian Dataset*

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AJAX	*Asynchronous Javascript and XML*
API	*Application Programming Interface*
BMP	*Bitmap*
BSI	*Backside Ilumination*
CCD	*Charge Coupled Device*
CFTV	Circuito Fechado de Televisão
CIS	*CMOS Image Sensor*
CMOS	*Complementary Metal-Oxide-Semiconductor*
CPU	*Central Processing Unity*
CSI	*Camera Serial Interface*
CV	*Computer Vision*
DETER	*Detection of events for threat evaluation and recognition*
DPCM	*Differential Pulse Code Modulation*
DSI	*Display Serial Interface*
DSP	*Digital Signal Processor*
FPS	*Frames Per Second*
FSI	*Frontside Ilumination*
GIF	*Graphics Interchange Format*
GPIO	*General Purpose Input/Output*
GPU	*Graphical Processing Unity*
HDMI	*High-Definition Multimedia Interface*

HDR	*High Dynamic Range*
HL	*Honeywell Laboratories*
HTTP	*Hypertext Transfer Protocol*
INRIA	*Institut National de Recherche en Informatique et en Automatique*
IR	*Infrared*
JPEG	*Joint Photographic Experts Group*
JSON	*JavaScript Object Notation*
LE	*Low Energy*
LED	*Light-emitting Diode*
LRTE	*Latency Reduction and Transport Efficiency*
MIPI CSI	*Mobile Industry Processor Interface - Camera Serial Interface*
OR	*Object Recognition*
PDI	Processamento Digital de Imagens
PNG	*Portable Network Graphic*
PSD	*Power Spectral Density*
RAM	*Random Access Memory*
RGB	*Red, Green and Blue*
SCCB	*Serial Camera Control Bus*
SESP	Secretária de Estado da Segurança Pública
SMS	*Short Message Service*
SNR	*Signal to Noise Ratio*
SoC	*System on Chip*
URL	*Uniform Resource Locator*

USB	*Universal Serial Bus*
UVC	*USB Video Class*
V4L	*Video4Linux*
VGA	*Video Graphics Array*

SUMÁRIO

1 INTRODUÇÃO

Observando os dados dos relatórios de segurança (SESP, 2016) é evidente o crescimento da violência nos meios urbanos. O mercado atualmente é carente de um sistema inteligente de monitoramento capaz de prover detecção confiável e com bom tempo de resposta. A maioria dos sistemas atuais possuem um nó centralizador das imagens, armazenamento e processamento em algum tipo de computador ou servidor e poucos apresentam soluções com processamento das imagens na câmera (INTELBRAS, 2016). Existem sistemas de redes de câmeras em desenvolvimento em amplo caráter científico. Kyrkou et al. apresentam uma solução de optimização de sistemas de detecção baseados em múltiplas câmeras (KYRKOU et al., 2017). Outro projeto, da "*University of California - Riverside*", propõem o rastreamento de pessoas por meio de trajetórias e câmeras em rede (ZHANG et al., 2015). Ainda, há o projeto DETER (*Detection of events for threat evaluation and recognition*), sendo um projeto de pesquisa e desenvolvimento de monitoramento de pedestres e veículos (BRUNO, 2012).

Além destes, há *softwares* inteligentes de videomonitoramento já desenvolvidos que estão começando a serem aplicados no Brasil. Introduzindo um novo conceito de segurança, mudando a forma como empresas de diferentes tamanhos atuam na área de segurança e inteligência em videomonitoramento, aplicando um *software* de vigilância em redes de câmeras (DIGIFORT, 2017).

Um sistema embarcado, capaz de realizar a detecção de pedestres por imagem, se tornaria numa solução muito mais viável, ainda mais por se tornar um elemento ativo no sistema de monitoramento. Unindo um sistema de baixo custo, com algoritmos de inteligência artificial embarcados, além de níveis adequados de precisão e desempenho.

Portanto, o desenvolvimento de projetos de sistemas inteligentes para detecção de pedestres é interessante sob vários aspectos. Sendo capaz de gerar um monitoramento robusto, com alarmes acionados por meio de imagem para prover melhor segurança. Além de ser um tema relevante, com um mercado amplo, um problema bem definido e carente de soluções.

1.1 PROBLEMA

Segurança é um dos temas mais discutidos e abordados, seja no setor público ou privado, e presente no cotidiano de todos. Segundo a Secretaria de Estado da Segurança Pública, no primeiro trimestre de 2016, o número de crimes e furtos em ambientes comerciais subiu 9,33% em relação ao mesmo período do ano anterior em Curitiba. O levantamento da Secretaria de Estado da Segurança Pública também apresentou um aumento de 5,25% no número de roubos e furtos, entre 2015 e 2016, em Curitiba (SESP, 2016).

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Propor um algoritmo de reconhecimento de pedestres por imagem, que estabeleça a melhor relação precisão/desempenho e opere em tempo real num sistema embarcado de baixo custo.

1.2.2 Objetivos Específicos

Dada a missão proposta pelo objetivo geral, foram definidos os principais objetivos específicos:

- Realizar uma revisão teórica do tema com as palavras chave "*camera*","*embedded*", "*pedestrian*", "*detector*", "*convolutional neural networks*"e "*INRIA*";
- Realizar uma análise comparativa de adequação dos sensores de imagem, quanto à plataforma embarcada Raspberry Pi;
- Aplicar conceitos de redes neurais convolucionais para determinar o algoritmo com a melhor precisão;
- Aplicar conceitos de otimização de software para elaborar uma solução com o melhor desempenho;
- Comparar a precisão do detector proposto quanto a diferentes algoritmos de detecção, aplicados no INRIA *Person Dataset*;

- Comparar o desempenho do detector quando aplicado a um vídeo de amostra, com pedestres em ambiente controlado.

1.3 JUSTIFICATIVA

Buscar um sistema de detecção de pedestres é importante para conseguir desenvolver sistemas de segurança e monitoramento mais complexos e confiáveis, visto dados da própria Secretária de Estado da Segurança Pública (SESP). Aplicar esse sistema na estrutura de segurança pública proporcionaria um maior conforto para a sociedade, permitindo que algoritmos inteligentes de detecção identifiquem possíveis adversidades de forma rápida e precisa. Aliado a isso, buscar um algoritmo próprio para reconhecimento proprocionaria uma independência ao sistema, tornando-o singular. Isto posto, ainda se acrescenta o fato da busca por um sistema de baixo custo, o qual concederia uma grande aplicação e utilização ao projeto, assim alcançando diversos níveis da sociedade por meio de uma plataforma flexível e com possibilidade de escalonamento.

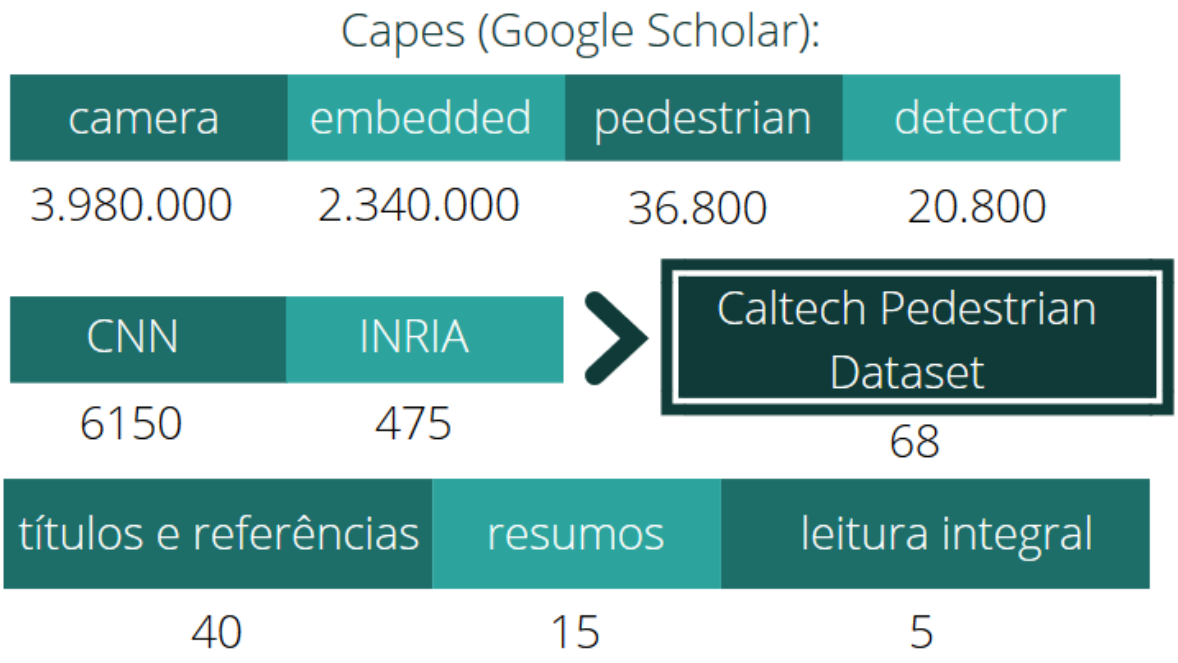
2 REVISÃO BIBLIOGRÁFICA

2.1 BIBLIOMETRIA

Para se atender um dos objetivos específicos deste trabalho e viabilizar a escrita do mesmo, a equipe realizou uma análise bibliométrica de periódicos na base integrada de dados da CAPES/CNPq (CNPQ, 2016) seguindo a metodologia clássica de análise de artigos proposta por Chueke et al. (CHUEKE; AMATUCCI, 2015), fazendo uso de *hot topics*, buscando a análise por impacto e relevância da produção cientifica. A Figura 2.1 apresenta os indicadores da avaliação de produção intelectual sobre o tema deste trabalho.

Ao adicionar os tópicos ilustrados na busca, o número de periódicos reduziu a cada etapa. Após a busca pelo conjunto de dados INRIA (DALAL; TRIGGS, 2005), as pesquisas foram apontadas para do conjunto de dados de detecção de pedestres *Caltech Pedestrian Dataset* (CALTECH, 2009), onde terminam-se as buscas. Ao término desta análise, foram selecionados cinco artigos como referências principais do presente trabalho de conclusão de curso, tais periódicos são apresentados no Capítulo 2.2.

FIGURA 2.1 – Análise bibliométrica



Fonte: Dos Autores

2.2 ESTADO DA ARTE

2.2.1 *To cctv or not to cctv. A review of current research into the effectiveness of CCTV systems in reducing crime* (ARMITAGE, 2002) e Contra-manual para câmeras inteligentes: vigilância, tecnologia e percepção (BRUNO, 2012)

A utilização de CFTV (Circuito Fechado de Televisão) como uma medida preventiva contra crimes é amplamente utilizada, os mecanismos que visam a utilização deste sistema são baseados segundo algumas premissas como: dissuasão, eficiência de desdobramento de segurança, autodisciplina de possíveis agressores por saberem que estão sendo monitorados, monitoramento em tempo real e a necessidade da presença de segurança (ARMITAGE, 2002).

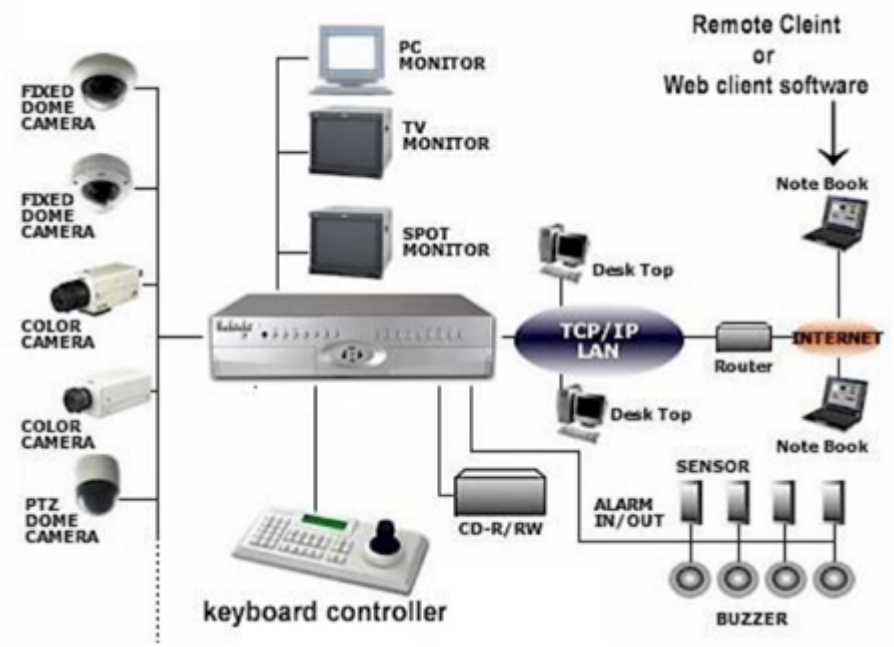
Os sistemas de monitoramento atuais que são encontrados no mercado consistem em dispositivos concentrados em uma unidade de processamento, a qual necessita de um operador para realizar o controle de câmeras, pois o sistema é baseado em imagens remotas, de acordo com o conceito de CFTV. Esse tipo de sistema consiste num certo conjunto de câmeras que captam imagens e as exibem, em tempo real, para algum monitor ou monitores. Pelo fato de ser necessário um operador, muitos desses sistemas não podem ser utilizados em residências, por conta do alto custo da contratação de um profissional para realizar apenas o monitoramento. Mesmo em grandes empresas isso pode se tornar um alto custo. Para ocorrer uma redução no custo é necessário minimizar o fator humano, dessa forma criando um sistema de segurança inteligente e autônomo (ELETRICA, 2014). Uma topologia clássica de um sistema de monitoramento CFTV, constituído por câmeras, monitores, meios de transmissão e uma única central de processamento é apresentado na Figura 2.2 (BAZOTE, 2012).

As vantagens de um sistema inteligente vão desde a redução do custo até fatores ergonômicos, com relação aos operadores. Pelo fato de não possuir um operador, é eliminado o risco de desatenção ou cansaço do mesmo, além disso, um sistema autônomo pode trabalhar ininterruptamente cedendo imagens em tempo real (BRUNO, 2012).

Estas imagens são cedidas via *internet*, apresentando ao cliente algum tipo de aviso de irregularidade por meio de SMS e contato por *e-mail*. Contribuindo a isso, mesmo em locais que hajam responsáveis por monitorar as câmeras, o sistema inteligente pode contribuir com a segurança simplificando o trabalho do operador

(ELETRICA, 2014).

FIGURA 2.2 – Topologia clássica de um CFTV



Fonte: (BAZOTE, 2012)

Contudo, pelo fato de ser autônomo o sistema é muito dependente da tecnologia utilizada, ademais a partir do momento que um aviso é gerado para o proprietário, a empresa de segurança responsável ou as autoridades devem tomar as medidas necessárias (ELETRICA, 2014).

Os conceitos básicos de um sistema inteligente consistem em captação de imagem, rastreio por técnicas de visão computacional, auto ajuste da câmera e transferência de dados. A captação de imagem é padrão, sendo feita a partir da câmera utilizada. O rastreio por técnicas de visão computacional é feito a partir de algum algoritmo de detecção que será utilizado. O auto ajuste da câmera irá depender da detecção feita anteriormente. Após essas etapas, a transferência de dados com o proprietário é feita, a partir de algum módulo de comunicação em rede (ELETRICA, 2014).

Um sistema inteligente pode ser definido a partir de conceitos básicos, como pode ser visto na Figura 2.3. Além disso, foi provado que de acordo com diferentes tipos de casos um operador treinado e um operador novato podem apresentar uma percepção diferenciada para cenas suspeitas. Isso ocasiona mais um problema para um sistema de monitoramento CFTV, pois o treinamento de um profissional é estritamente

necessário para obter resultados mais satisfatórios (HOWARD et al., 2013).

FIGURA 2.3 – Conceitos básicos de um sistema inteligente

Fonte: (ELETRICA, 2014)

Um sistema inteligente é capaz de reconhecer e diferenciar diferentes padrões de conduta, podendo classificá-los como inseguros ou suspeitos. De acordo com alguns parâmetros definidos no sistema, as câmeras operam segundo alguns algoritmos que analisam e identificam sujeitos, objetos ou atitudes que sejam temerárias. Esse sistema pode ressaltar, por exemplo, indivíduos que estejam numa mesma área por muito tempo, residencial ou empresarial, de forma a impedir um possível assalto. Esse dispositivo pode, ainda, evidenciar um objeto suspeito que tenha sido abandonado em algum local com grande fluxo de pessoas, um conjunto de pessoas suspeitas ou qualquer outro tipo de circunstância que tenha sido antecipadamente qualificada como necessária para o sistema destacar, um exemplo pode ser visto na Figura 2.4 (BRUNO, 2012).

Um sistema de monitoramento de câmeras inteligentes supera limitações perceptivas do monitoramento humano, já que a concentração humana é muito limitada para tratar com imagens que em muitos casos são monótonas. A atenção da grande parte dos cidadãos cai a um nível muito abaixo do satisfatório, depois de apenas 20 minutos de uma atividade. Assim, fica impraticável que um operador possa monitorar nove ou mais câmeras por muito tempo. Uma camada inteligente num sistema de monitoramento serve para destacar os índices de ameaças ou qualquer outra situação necessária que demande uma atenção maior. O sistema além de capturar, enviar e salvar imagens iria “interpretá-las” (BRUNO, 2012).

FIGURA 2.4 – Sistema de um monitoramento inteligente

Fonte: (BRUNO, 2012)

Um projeto de pesquisa e desenvolvimento de monitoramento de pedestres e veículos interessante é o DETER (*Detection of events for threat evaluation and recognition*), o qual consiste em identificar movimentos suspeitos de pedestres e veículos numa área específica. O sistema inicialmente foi testado num estacionamento do *Honeywell Laboratories* (HL) em Minneapolis (MORELLAS; PAVLIDIS; TSIAMYRTZIS, 2003). Esse sistema teve sucesso em detectar diferentes padrões de movimento, que segundo seus parâmetros seriam ameaçadores, como: invasão de calçada por veículos, veículos em velocidade não permitida, pedestres correndo, pedestres caminhando em zigue-zague e diversos automóveis entrando ao mesmo tempo em vias distintas. No momento que identifica essas potenciais ameaças, o DETER soa um sinal de alarme para o vigilante, permitindo uma intervenção rápida do incidente (BRUNO, 2012).

Uma característica marcante de um sistema inteligente está em sua capacidade de reação instantânea, ou seja, além de ser um dispositivo com um tempo real de observação ele seria um dispositivo com um tempo real de reação. Isso é possível devido à memória algorítmica, que se baseia nos padrões que o dispositivo deve identificar e alertar. Assim, a maior expectativa desse sistema inteligente está em antecipar e conter eventos indesejados, relativos a segurança, no futuro. Dessa forma,

a memória dos padrões iria servir como um índice futuro, permitindo um monitoramento competente e que se antecipa à ação (BRUNO, 2012).

No Brasil ainda não há sistemas de monitoramento inteligentes como os aqui citados, sendo que o mercado atual é constituído, em sua grande maioria, pelos equipamentos de CFTV comuns ou CFTV em rede. Contudo, são sistemas ainda muito simples em que são necessários operadores para monitorar as câmeras, sendo que a interpretação dos mesmos é primordial para avaliar se há algum indício de ocorrência. Dessa forma, um sistema inteligente seria primordial para aumentar a segurança de residências, empresas e até mesmo cidades.

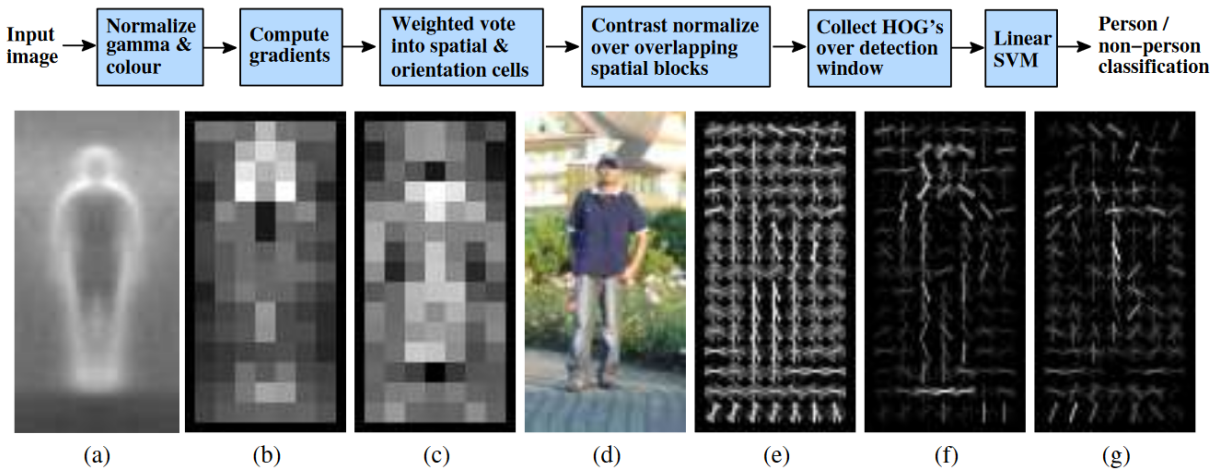
2.2.2 *Histograms of oriented gradients for human detection* (DALAL; TRIGGS, 2005)

O periódico de Dalal e Triggs é um dos pioneiros nos métodos de detecção de pedestres. Atualmente é um dos mais citados artigos no tema e tem parte de seu impacto associado a abertura de seu código fonte de detecção, hoje incorporado à biblioteca OpenCV (OCVTEAM, 2017).

O método, apresentado na Figura 2.5, parte da computação dos gradientes de cores na imagem (a), separação em células (b; c) e distinção da orientação destes gradientes (e), assim formando histogramas de gradientes orientados. Os historiogramas são alimentados à uma máquina de vetor de suporte de janela gaussiana (f; g) de busca para classificação das orientações relevantes como pedestre ou não-pedestre (HEARST et al., 1998).

O grande impedimento deste algoritmo são suas variações a escala (pedestres com diferentes dimensões nas imagens), pois a máquina de vetor de suporte é treinada para um tamanho fixo de características, assim forçando uma busca por janela deslizante de tamanho variável, o que acarreta na presença de diversos candidatos positivos propostos para um único candidato verdadeiramente positivo e demandando técnicas de triagem de saídas (supressor de não-máxima) para uma maior confiabilidade.

FIGURA 2.5 – Fluxo de funcionamento do algoritmo e demonstração passos do detector



Fonte: (DALAL; TRIGGS, 2005)

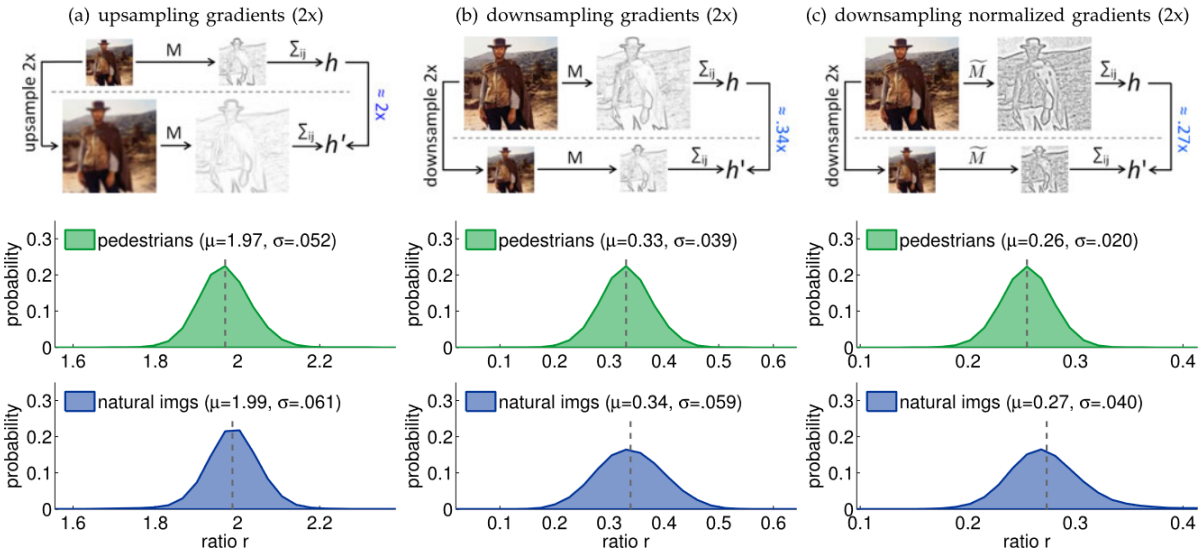
2.2.3 *The fastest pedestrian detector in the west.* (DOLLÁR; BELONGIE; PERONA, 2010)

O método de detecção apresentado no artigo é feito para optimização de detectores baseados em características múltiplas, podendo até ser de diagramas de histogramas, com baixo impacto na precisão do algoritmo. O código do algoritmo de detecção, bem como das demais soluções apresentadas no estado da arte, são abertos e com licenças de uso livre acadêmico.

O artigo original, de 2010, é usado de base para o código disponibilizado e segue a mesma filosofia do artigo mais recente (DOLLÁR et al., 2014). A mais recente publicação melhor ilustra a técnica de classificação do detector usando o método de extração de características já estabelecido.

A Figura 2.6 apresenta a base de criação do algoritmo, que diferentemente da solução apresentada por Hoang et al. (HOANG; LE; JO, 2014), não faz uso da orientação dos gradientes para determinação de saliência, mas sim de sua distribuição estatística extraída de amostra em variações de escala. Ainda na Figura 2.6 vê-se as variações de média e desvio padrão, características muito comuns no meio da aprendizagem supervisionada, para cada estratégia de amostragem e como esses dados se diferem de imagens não contendo candidatos a pedestres.

FIGURA 2.6 – Comportamento dos histogramas de gradientes de acordo com o fator de escala



Fonte: (DOLLÁR et al., 2014)

O periódico de Dóllar et al. se destaca pelo método de extração de informações, mas quanto ao classificador e detector em si é utilizado uma técnica mais difundida no estado da arte, a de árvores de decisão acentuadas (ROE et al., 2005). O fundamento de tais classificadores é o reforço (acentuação) de características que apontem a presença do candidato alimentado à árvore. Tais classificadores, junto às máquina de vetores de suporte, apresentaram-se como alternativas importantes às redes neurais convolucionais durante anos graças ao tamanho reduzido de modelos e velocidade de treinamento.

2.2.4 *MobileNets: Efficient CNN's for Mobile Vision Applications* (HOWARD et al., 2017)

A *MobileNet* (HOWARD et al., 2017) foi proposta pela *Google Inc.* com o intuito de reduzir o número de parâmetros de redes neurais atuais como a AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) e manter o mesmo nível de precisão. Tal tecnologia é capaz de atingir tais resultados usando convoluções separáveis de profundidade, que mesclam os valores de diferentes canais usando pequenas matrizes. Uma analogia possível, ilustrada pela Figura 2.7, ao seu método de análise é o de uma broca extratora de amostras de gelo: não se é necessário extrair porções grandes do terreno (filtros inteiros) para se classificar a natureza daquele clima, mas sim leituras

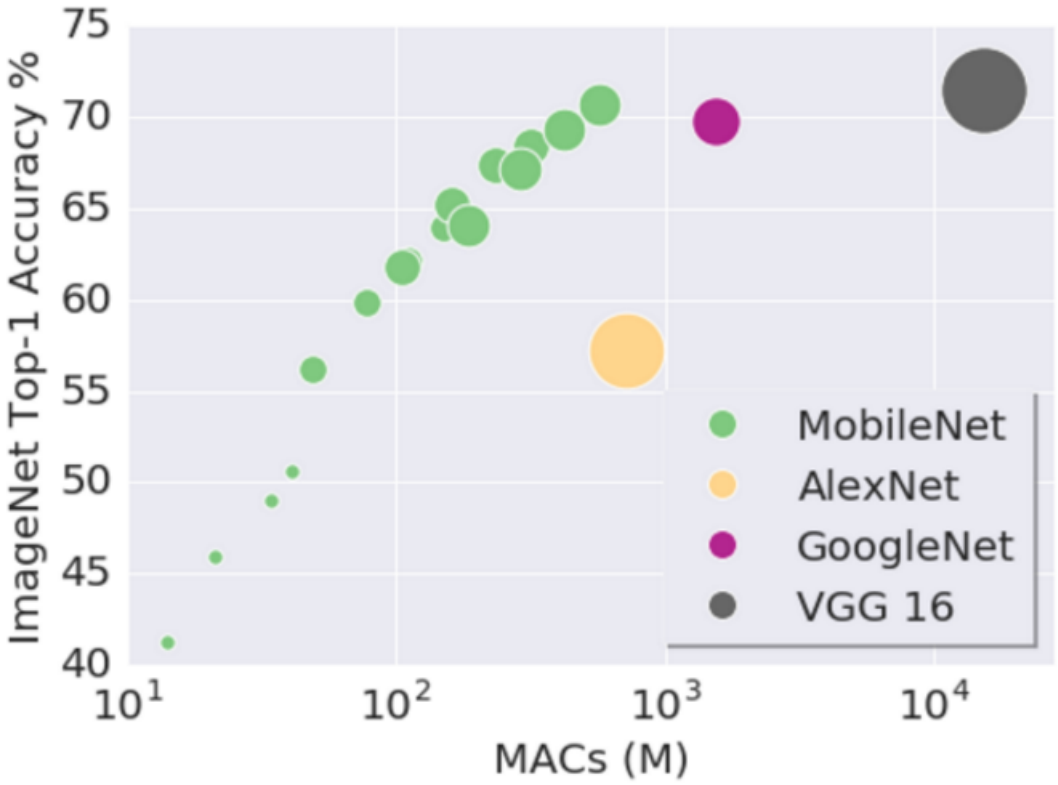
profundas (convoluções profundas) do gelo dos locais. O volume de informação se reduz, porém a relevância (saliência) se mantém.

A Figura 2.8 demonstra que tal topologia é capaz de atingir precisões equivalentes a modelos conhecidos do estado da arte usando cerca de sessenta vezes menos parâmetros.

O número de parâmetros de uma rede neural (pesos e tendências) é inversamente proporcional ao custo computacional de aplicação da mesma, assim tornando a *MobileNets* uma solução viável ao projeto proposto.

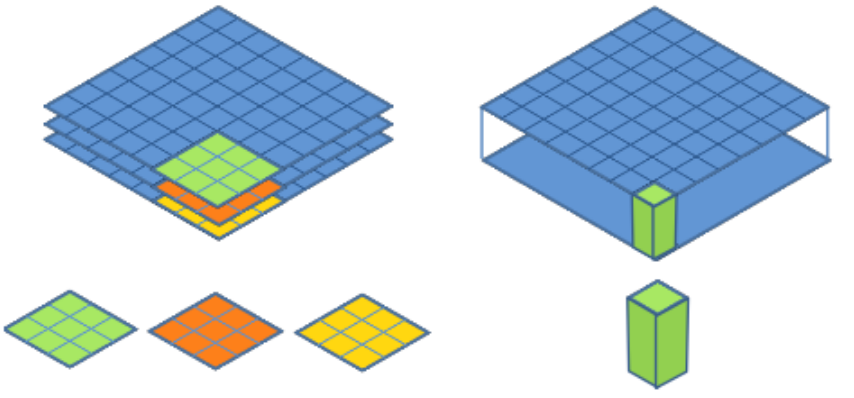
O gráfico da Figura 2.8 trás em seu eixo vertical a precisão média do classificador topo de lista, ou seja, dentre a lista de respostas proposta pela rede neural a resposta correta estava em primeira posição. Já no eixo horizontal vem o número de parâmetros associados a cada modelo, neste casa usa-se a sigla MAC (Multiplications and Additions Calculations) na ordem dos milhões.

FIGURA 2.8 – Comparação de precisão da *MobileNet* com outras topologias

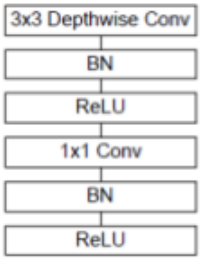


Fonte: (BHAT, 2017)

FIGURA 2.7 – Paleoclimatologia e convolução de profundidade



Depthwise Convolutional Filters Pointwise Convolutional Filters



Depthwise Separable Convolution

Fontes: (RIEBEEK, 2005) e (HOWARD et al., 2017)

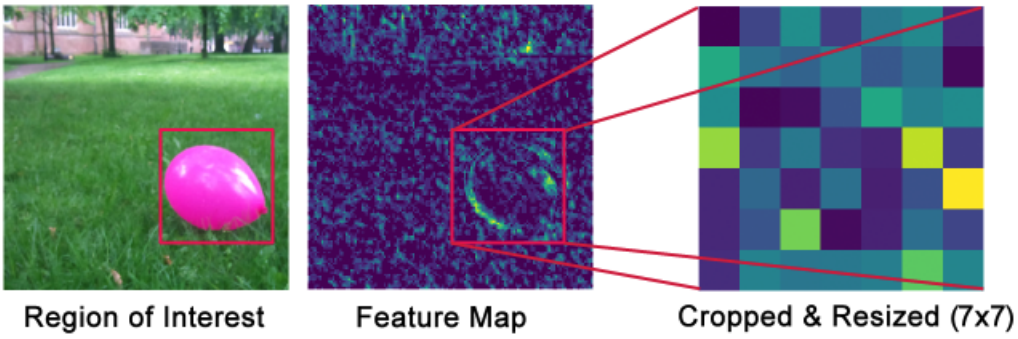
2.2.5 *Mask R-CNN* (HE et al., 2017)

Abordando-se o critério precisão de algoritmos de detecção, é apresentado o artigo *Mask R-CNN* (HE et al., 2017). Esta publicação fora feita pela divisão de inteligência artificial de outra grande corporação norte-americana, o Facebook.

O algoritmo em questão é abordado pela comunidade como o pináculo da detecção de objetos, pois além de detecta-los, a metodologia é capaz de segmenta-los, estipulando um contorno específico para cada item da cena. Tal abordagem se torna relevante ao problema proposto do projeto, pois com variações de contorno de pedestres pode-se não apenas realizar a localização na cena, mas também inferir comportamentos e ações.

A *Mask R-CNN* tem seu funcionamento completo dividido em três etapas: localização, detecção e segmentação. O passo de localização envolve a busca por saliências nos mapas de características e tem como resultado apenas a proposição de uma região retangular candidata de interesse. A Figura 2.9 demonstra o processo de localização, onde o aglomerado de características (*features*) é definido.

FIGURA 2.9 – Tarefa de localização



Fonte: (ABDULLA, 2018)

Os passos a seguir são os de classificação e segmentação. A tarefa de classificação já é muito abordada (LEE et al., 2017) e já vem tendo avanço ao longo dos últimos quatro anos, mas segmentação não. A ideia de segmentação surge com o aumento da complexidade e generalização dos novos modelos de redes neurais convolucionais e a introdução de conjuntos de dados mais refinados, grande e públicos, como é o caso do *Microsoft COCO Dataset* (LIN et al., 2014). Observa-se na Figura 2.10 um exemplo de saída de um detector do tipo *Mask R-CNN*.

FIGURA 2.10 – Exemplo de segmentação em cenário de trafego urbano

Fonte: (HE et al., 2017)

2.3 RASPBERRY PI

O *Raspberry Pi* é um microcomputador de baixo custo e bom desempenho, primariamente desenvolvido para ensinar programação e computação à crianças da Inglaterra. A ideia da "*Raspberry Pi Foundation*" parte do princípio que no futuro cada vez mais serão requisitados programadores mundo afora e os conhecimentos de computação deveriam partir do ensino de base para se suprir esta necessidade. O projeto se desenvolveu de tal forma, que este microcomputador do tamanho de um cartão de crédito, se tornou muito popular no meio científico e de engenharia. Hoje é possível observar os mais diversos projetos sendo desenvolvidos na plataforma e com uma comunidade muito ativa, principalmente por ser de *software* e *hardware* livres (RASPBERRY, 2016).

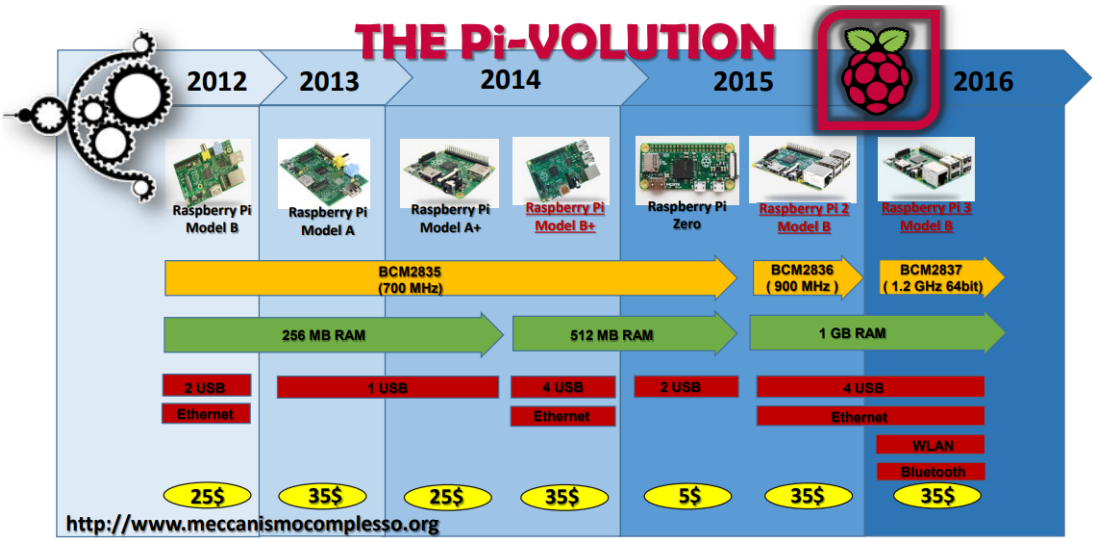
Ainda sobre a comunidade, o projeto promove diversas conferências e encontros para que os desenvolvedores, professores e alunos sejam capazes de aprimorar e trocar conhecimentos. Estas conferências são chamadas de *Raspberry Jam* ou em português, "geleias de framboesa" e são parte fundamental da fundação e do crescimento

do projeto (RASPBERRY, 2016).

Vale ressaltar que o microcomputador em questão ainda é uma plataforma ativa e possui novas versões de hardware sendo lançadas periodicamente. Cada nova versão traz evoluções significativas à plataforma em questão de desempenho, versatilidade e periféricos (RASPBERRY, 2016). Uma evolução sucinta deste microcomputador pode ser vista na Figura 2.11, a qual ilustra algumas diferenças das diversas versões (PI, 2015).

Os principais avanços entre as versões ocorreram, principalmente, em 2014, entre o modelo *Raspberry Pi A+* e o seu sucessor, o *Raspberry Pi B+*. Pois, como pode ser analisado na Figura 2.11, a versão do modelo B+ foi construída com 4 entradas USB 2.0, placa de rede *Ethernet* e o dobro de memória RAM. Nos anos seguintes novos *hardwares* foram desenvolvidos, sendo significativo salientar o modelo mais novo, o *Raspberry Pi 3 B*. O qual apresentou expressivas melhoras e alternativas para o desenvolvimento de projetos, sendo esse o *hardware* que será utilizado para o desenvolvimento da proposta em questão.

FIGURA 2.11 – *Evolução hardware Raspberry*



Fonte: (PI, 2015)

2.3.1 O Hardware

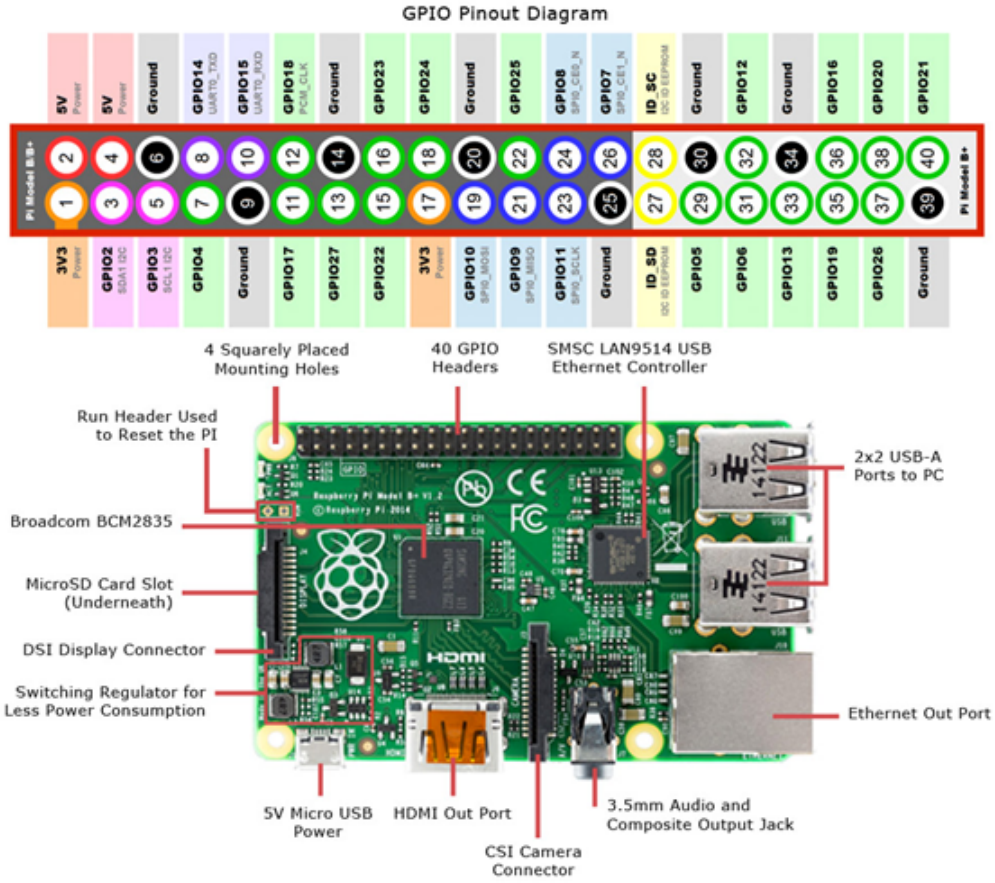
O microcomputador é composto de uma arquitetura Von Neumann, onde a memória de dados e de programa são armazenadas num mesmo dispositivo e compartilham um mesmo barramento (CASTRO; CASTRO, 2015). O primeiro modelo que

será descrito se trata do *Raspberry Pi B+*, ou seja, é o *hardware* que foi utilizado no início do projeto aqui desenvolvido. O segundo modelo se trata do novo *hardware* adquirido, o *Raspberry Pi 3 B*. As especificações de ambos serão apresentadas, a fim de comparação entre os dois modelos.

2.3.1.1 *Raspberry Pi B+*

Ele faz uso de uma CPU ARM1176JZF-S de instruções ARMv6 com frequência nominal de 700MHz, com até mesmo um DSP no chip-set, no entanto, ainda, com API reservada ao fabricante. Possui 512Mb de memória RAM disponíveis que ficam dispostos fisicamente sobre o processador. No *Raspberry Pi B+* encontra-se uma GPU *Broadcom VideoCore* IV de 250MHz que dispõem de OpenGL ES 1.1, OpenGL ES 2.0, OpenVG 1.1 acelerado via hardware, Open EGL, OpenMAX e 1080p30 H.264 com decodificação de alto perfil. O armazenamento não-volátil é feito por meio de cartão SD. A Figura 2.12 apresenta o diagrama de componentes do *Raspberry Pi B+*.

FIGURA 2.12 – Diagrama de componentes do *Raspberry Pi B+*



Fonte: (CIRCUIT, 2016)

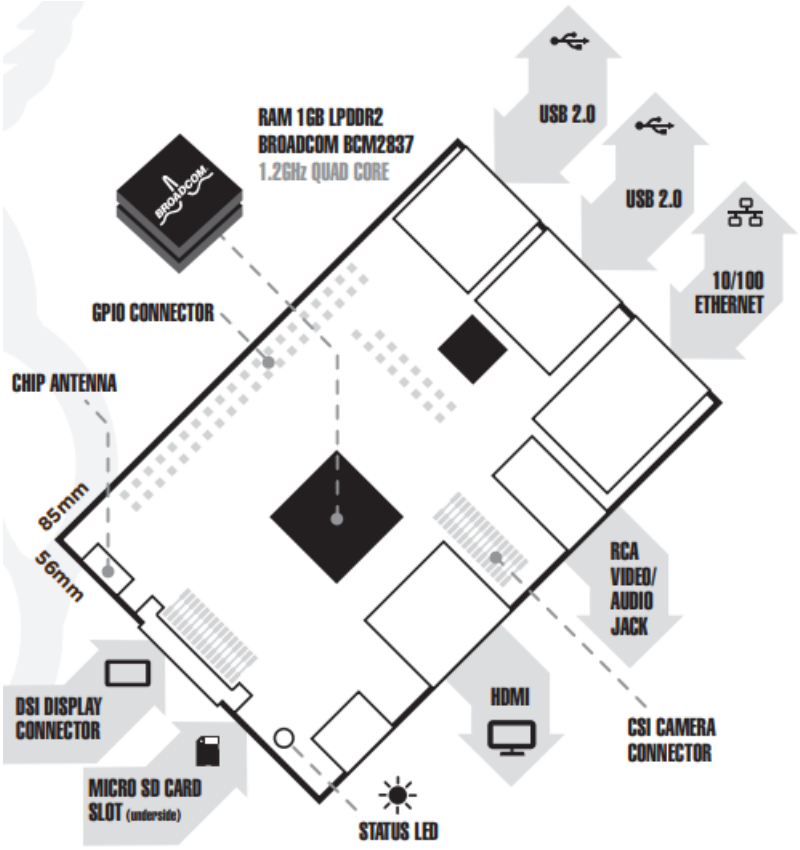
Pontos importantes a se destacar desta versão do modelo é seu Hub de 4 entradas USB 2.0, placa de rede *Ethernet*, saída HDMI e uma fonte chaveada eficiente.

Outra característica importante é o barramento de entradas de baixo nível, de 40 pinos, localizado na parte lateral do dispositivo. Entre os 40 pinos, que podem ser vistos na Figura 2.12, há barramentos de comunicação I²C, SPI e UART, GPIOs, alimentação (5V, 3,3V e GND) e JTAG ARM (ELINUX, 2016).

2.3.1.2 *Raspberry Pi 3 B*

O *Raspberry Pi 3* possui capacidade de processamento até 6 vezes maior que os modelos anteriores. Esta última geração do *Raspberry Pi* está equipado com um processador *Broadcom* BCM2837 atualizado, memória RAM de 1GB e 900MHz de velocidade e faz uso de uma CPU ARM Cortex-A53 quad-core de 64 bits, que funciona com frequência nominal de 1,2 GHz e usando o conjunto de instruções ARMv8. A Figura 2.13 apresenta o diagrama de componentes do *Raspberry Pi 3 B*.

FIGURA 2.13 – Diagrama de componentes do *Raspberry Pi 3 B*



Fonte: (RSCOMPONENTS, 2017)

Todas essas melhorias levam a um aumento de 50 a 60% no desempenho do *hardware*, no modo de 32 bits, quando comparado ao Pi 2 e um aumento de aproximadamente 10 vezes sobre o Raspberry Pi original. Além disso, possui uma GPU *Dual core VideoCore IV® Multimedia Co-Processor* de 300MHz que dispõem de OpenGL ES 2.0, OpenVG 1.1 acelerado via hardware e 1080p30 H.264 com decodificação de alto perfil (JAMECO, 2017).

Outra mudança a destacar é a adição de suporte a redes sem fio sem necessitar de adaptadores, ou seja, foi incorporado uma integração *Wireless LAN* e Bluetooth na placa. Agora há antena Bluetooth 4.1 LE (*Low Energy*) integrada e comunicação wi-fi nos padrões B, G e N, protocolo 802.11n. Isso desafoga as portas USB, aliviando as mesmas, além de ser muito mais aproveitável e conveniente. Permitindo ocupar as portas USB com sensores, atuadores ou outros equipamentos, já que agora é possível utilizar teclado e mouse Bluetooth e se conectar a uma rede WLAN sem utilizar adaptadores (CIPOLI, 2016).

Outros componentes da placa, como os ports de entrada e saída e o barramento de baixo nível, seguem possuindo as mesmas características já descritas na página 31. Com 40 pinos na GPIO, 4 entradas USB 2.0, slot para armazenamento não-volátil via cartão micro SD, placa de rede *Ethernet*, interface CSI para câmera e DSI para display (PI, 2015).

Mesmo com todas essas mudanças, outro fator que permaneceu inalterado foi o preço da placa. Custando, ainda, os mesmos 35 dólares. Permitindo que os objetivos e os propósitos desta série de *hardwares* permaneçam inalterados: buscando sempre a educação, facilidade de uso e o baixo custo (PI, 2015).

2.3.2 O Software

Sendo o *Raspberry Pi* um microcomputador, ele faz uso de um sistema operacional para abstração de recursos, prover interfaces para acesso a recursos e gerência de recursos: processador, memória, dispositivos, etc. Por ser uma plataforma de software livre, ela possui diversas distribuições com chamadas ao sistema Unix-compatíveis. Dentre estas distribuições, a mais popular e que será usada no projeto é a Raspian (ROSEBROCK, 2016).

O Raspian é um Debian otimizado para o hardware do *Raspberry Pi*, com as

aplicações sobre ele ganhando desempenho devido ao refinamento de instruções tipo ARMv6. Essa plataforma possui configurações de compilação ajustadas para produzir um código otimizado, que será executado no Raspberry Pi. Isso proporciona um desempenho significativamente mais rápido para aplicações que fazem uso intensivo de operações aritméticas de ponto flutuante, manipulações de imagens e processamento de algoritmos. A versão base do sistema operacional também já traz mais de trinta e cinco mil bibliotecas compiladas e de fácil instalação na plataforma (RASPIAN, 2016). Mesmo sendo um sistema operacional já difundido, o mesmo continua com um desenvolvimento ativo com ênfase em melhorar a estabilidade e o desempenho do maior número de pacotes com base no Debian

O sistema ainda tem como filosofia manter-se o mais próximo do Debian possível, como pode se ver na Figura 2.14, por ser um sistema operacional bastante difundido, assim mantendo uma base sólida e vastamente documentada à distribuição para o microcomputador (RASPIAN, 2016).

FIGURA 2.14 – Área de trabalho do Raspian

Fonte: (RASPBERRY, 2016)

2.4 LINGUAGEM C++

Uma das linguagens de mais ampla utilização, possui características fundamentais para o desenvolvimento embarcado. Surgido na década de 80, o C++ veio da necessidade de uma melhor estruturação no código fonte de distribuições padrão Unix. Sendo ela uma linguagem compilada, o C++ é conhecida por ter alto desempenho e bastante difusão entre aplicações de elevado custo computacional, como jogos, sistemas operacionais ou visão computacional (CPLUSPLUS, 2017).

Por sua eficiência, o C++ é referência em sistemas de visão computacional em tempo real, assim tornando-se linguagem chave em nossa aplicação. O método de provar algoritmos usando *Python* e então escrever a aplicação real em C++ é largamente utilizado na indústria, no desenvolvimento em processamento de imagens.

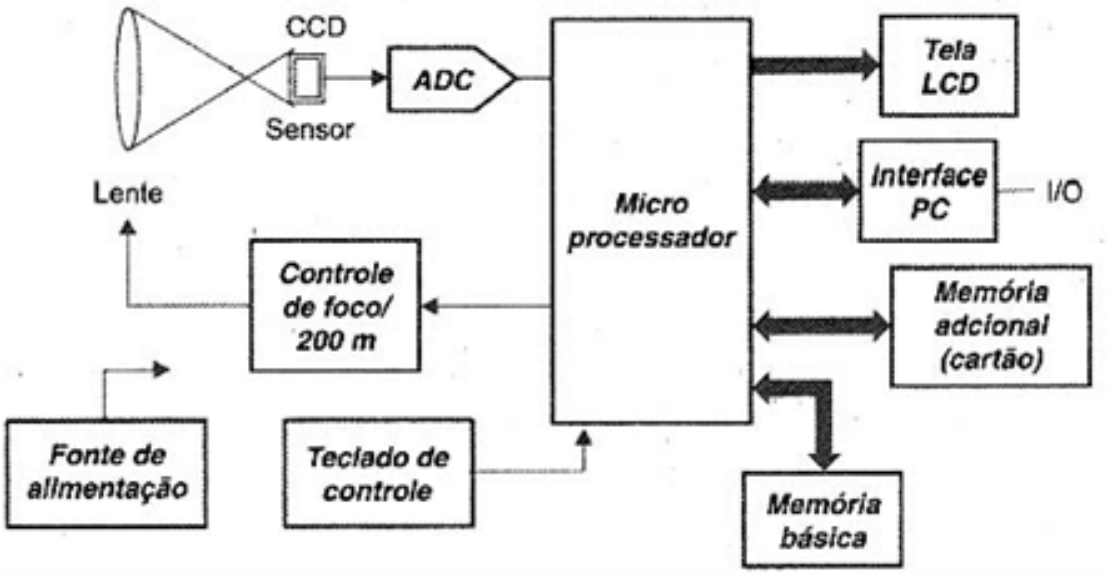
2.5 CÂMERAS

Uma câmera digital consiste em apanhar a imagem a partir de um sensor especial e alterar cada ponto dessa imagem em bits, que em seguida são armazenados numa memória. Um diagrama de blocos básico de uma câmera digital pode ser visto na Figura 2.15 (BRAGA, 2014). O principal elemento de uma câmera digital é o sensor CCD (*Charge Coupled Device*), ou seja, um dispositivo de cargas acopladas, que consiste em um chip semicondutor organizado em matrizes lineares ou de duas dimensões (MENESES; ALMEIDA, 2012).

Este *chip* trabalha com uma técnica de varredura que identifica a intensidade de luz que incide sobre cada componente da matriz, que em seguida codifica essa intensidade em valores digitais. A digitalização do sinal do CCD ocorre a partir de um conversor analógico para digital e o processamento da imagem é feita por um microcontrolador, que reuni as diversas funções da câmera digital (BRAGA, 2014).

Uma análise comparativa entre três tipos de câmeras será feita, a fim de determinar a melhor para se realizar um sistema de monitoramento inteligente que possua baixo custo, mas sem dispensar um bom desempenho de processamento.

FIGURA 2.15 – Diagrama de blocos básico de uma câmera digital



Fonte: (BRAGA, 2014)

2.5.1 OV5647

A câmera OV5647 é um módulo próprio para *Raspberry Pi*. Trata-se de um sensor CMOS de imagem de baixa tensão e alto desempenho que fornece imagens de 5 megapixels usando tecnologia *OmniBSI™*. Ela fornece múltiplas imagens de diversas resoluções, a partir do controle do barramento serial da câmera (SPARKFUN, 2009). O Módulo que contém a câmera pode ser visto na Figura 2.16.

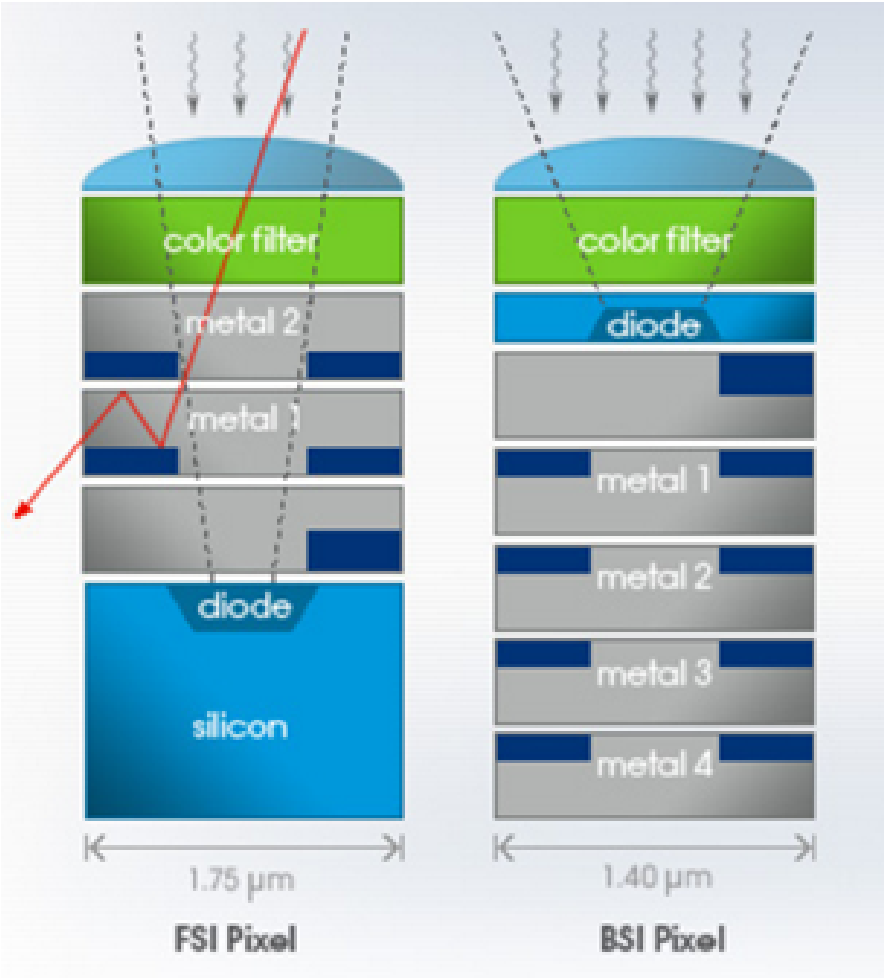
FIGURA 2.16 – Módulo câmera OV5647

Fonte: (MASOOD, 2016)

O módulo produzido pela *Omni Vision* possui uma resolução de 2592 x 1944 pixels, ou seja, 5M pixels para imagens sem movimento. Além de a câmera possuir um tamanho muito reduzido, de apenas 8,5 x 8,5 x 5 mm (WINSCH; SANTOS, 2013).

A tecnologia *OmniBSI™* representa uma revolução para a produção em massa de sensores de imagem CMOS (CIS), adotando uma abordagem diferenciada com relação as arquiteturas tradicionais. A tecnologia BSI (*Backside Ilumination*), proporciona melhorias contínuas na sensibilidade, reprodução de cor e qualidade de imagem. Essa tecnologia envolve em inverter o sensor de imagem, fazendo com que o mesmo fique de “cabeça para baixo”, aplicando os filtros de cor e micro lentes na parte traseira dos pixels. Assim, a luz captada é recolhida através da parte traseira do sensor (OVT, 2011). A Figura 2.17 apresenta uma comparação entre uma tecnologia FSI e BSI (OVT, 2011).

FIGURA 2.17 – Tecnologia FSI e BSI



Fonte: (OVT, 2011)

Ao se alterar a construção da câmera, as camadas de metal e dielétricas residem abaixo do conjunto de sensores, proporcionando um caminho mais direto para a luz viajar até o pixel, otimizando o fator de preenchimento. Esta abordagem difere das arquiteturas FSI (*Front Side Ilumination*) convencionais, as quais a luz se dirige até a área fotossensível pela parte frontal do pixel. Aliado a isso, a tecnologia BSI proporciona a criação de módulos de câmeras ainda menores que os já existentes (OVT, 2011).

Esse módulo suporta formatos como BMP, GIF, JPEG e PNG, além de modos não reduzidos em RGB ou YUV. A câmera também permite realizar vídeos em 1080p30, ou seja, 1920x1080x30fps. Como o *Raspberry Pi* possui uma interface própria para este módulo, a utilização do mesmo se torna uma vantagem, pois possibilita uma boa transferência de imagem e controle (WINSCH; SANTOS, 2013). Assim, por conta da resolução, tamanho e preço acessível (cerca de 15 dólares) torna-se uma boa escolha de câmera.

2.5.2 Logitech C270

A Logitech C270 trata-se de uma webcam de baixo custo, cerca de 90 reais, que oferece uma boa flexibilidade e qualidade se comparada à outras câmeras de custo reduzido. Ela fornece uma imagem de boa definição, com resolução de 1280x720 pixels e taxa de captura de 30 fps (OLEARI; RIZZINI; CASELLI, 2013).

Possui conexão USB 2.0 de alta velocidade, controles de panorâmica, inclinação e zoom. Além disso, é possível realizar busca de rosto e identificação de movimento, de acordo com a fabricante (LOGITECH, 2016). A Figura 2.18 apresenta a *webcam* C270.

Este sensor de imagem é compatível com o padrão UVC (USB *Video Class*), permitindo a configuração de parâmetros da imagem como contraste, brilho, ganho, exposição, balanço de branco e outros. Essa câmera necessita ser configurada para oferecer a melhor taxa de frame, sem que ocorra uma saturação na banda da porta USB 2.0 (OLEARI; RIZZINI; CASELLI, 2013).

Levando em conta apenas o mercado nacional, esta câmera se torna uma escolha razoável, pois oferece uma resolução de boa qualidade e um preço, até certo ponto, acessível. Contudo, sua implementação necessita de um maior estudo pelo fato

de não existir uma compatibilidade direta com o *Raspberry Pi*.

FIGURA 2.18 – Câmera C270

Fonte: (LOGITECH, 2016)

2.5.3 *Haiworld New Version 5*

Este módulo consiste em uma câmera OV5647, a qual já foi descrita e apresentada na seção 2.5.1, e duas lentes com LEDs infravermelhos. Este novo módulo permite que a câmera opere com visão noturna, devido aos dois focos de luz infravermelha que estão anexos ao módulo (COUTINHO, 2016). Assim, a câmera opera com as mesmas características do módulo sozinho, possuindo auto-foco, resolução de 5M pixels, tecnologia *OmniBSI™*, total compatibilidade com o *Raspberry Pi* e tamanho, ainda, muito reduzido.

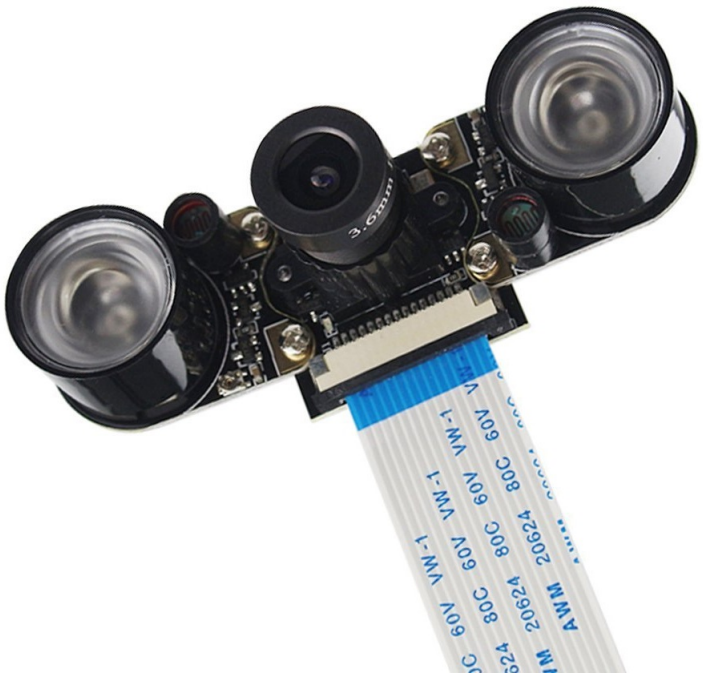
Sendo assim, esta câmera é ideal para operar em ambientes com alta ou baixa luminosidade. Tanto para operar com imagens estáticas ou gravação de vídeo. A Figura 2.19 apresenta o modelo em questão, sendo possível verificar as duas lentes de LEDs infravermelhos.

A visão noturna é viável graças a uma combinação de dois parâmetros, alcance espectral e faixa de intensidade, ambos em valores suficientes. A faixa de intensidade

se refere à capacidade de enxergar objetos em locais em que há condições de pouca luminosidade. As técnicas de alcance espectral permitem detectar a radiação que é invisível para um observador humano. A visão do ser humano é restrita a uma certa variedade do espectro eletromagnético, comumente chamado de luz visível. Assim, os dois LEDs do módulo permitem que ocorra essa melhora na visualização do espectro, ou seja, que se possa observar espectros mais amplos, pois os LEDs operam como dispositivos de iluminação IR (*Infrared)* (BLAKE, 2015).

Os LEDs agem em conjunto com o módulo, caso haja iluminação muito baixa eles podem ser usados como foco e auxiliar a câmera. Oferecendo melhor qualidade de imagem e permitindo realizar captura de vídeo de alta velocidade, através da escuridão, névoa e chuva (BLAKE, 2015).

FIGURA 2.19 – Módulo câmera *Haiworld New Version 5*



Fonte: (AMAZON, 2017)

2.6 PROCESSAMENTO DIGITAL DE IMAGENS

As câmeras e sensores já foram abordados, assim é possível começar a falar sobre o processamento digital de imagens (PDI). O PDI é um ramo do processamento digital de sinais, onde os sinais de entrada são matrizes discretas de duas dimensões.

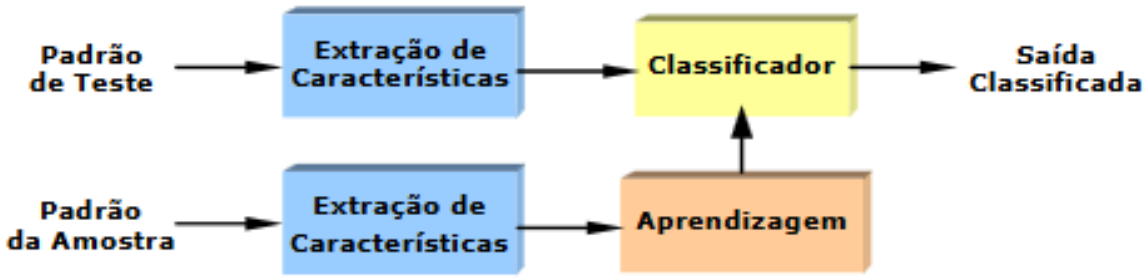
Os algoritmos de processamento tendem a seguir certos passos fixos antes das aplicações específicas. Geralmente após a captura da imagem, são feitos pré-

processamentos que filtram ruídos e corrigem distorções na imagem. Depois são normalmente avaliadas características, histogramas, formas e regiões dentro da imagem. Após levantadas as informações, se disponível e necessário, observa-se se a entrada de dados é um vetor de imagens, assim podendo analisar movimentação, fluxo óptico e planos de fundo (QUEIROZ; GOMES, 2006).

Depois de feitas as extrações de informações principais, o processamento de imagens torna-se visão computacional, onde compreende-se a inteligência e tomada de decisões baseadas nas informações extraídas das imagens. Muitas vezes, a solidez de certo sistema de visão computacional está mais relacionada a qualidade da filtragem e extração de informações das imagens do que o refinamento do aprendizado de máquina. Dentro da visão computacional são abordados algoritmos de classificação, como na Figura 2.20, redes neurais, cascatas de aprendizagem e outros algoritmos de inteligência artificial (ROSEBROCK, 2016).

Uma análise dos passos de processamento empregados será feita, a fim de fundamentar os conceitos do algoritmo e disponibizar critério à priori para comparação com futuros novos métodos. Destacam-se no algoritmo empregado as etapas de captura, pré-processamento e visão computacional.

FIGURA 2.20 – Diagrama básico de um classificador



Fonte: (QUEIROZ; GOMES, 2006)

2.6.1 Captura

A etapa de captura pode ser definida em três camadas principais:

- Camada de *Software*, a qual abrange a biblioteca RaspiCam e o *software* de CV (*Computer Vision*);
- Camada de Driver, a qual é composta pela interface *Video4Linux*;
- Camada de *Hardware*, na qual está inserida a câmera e o seu protocolo de

comunicação com a Camada de Driver.

2.6.1.1 Camada de *Hardware*

É nessa camada que está situada a câmera que realiza o monitoramento, ou seja, o dispositivo de captura das imagens. Nesta camada ocorre o processo de aquisição das imagens, as quais são interpretadas nos sistemas de visão computacional. Esses sistemas buscam elaborar definições adequadas dos dados contidos nas imagens capturadas, sendo que as informações geradas podem ser empregadas, por exemplo, na classificação e reconhecimento de padrões de imagens ou na classificação de objetos (CORSO; ALMEIDA, 2007).

Contudo, para que esse monitoramento, através das imagens, chegue até o *software* de visão computacional é necessário um protocolo de comunicação entre essa camada de *hardware* e a camada superior. O protocolo responsável por essa intermediação é o protocolo de interface serial MIPI CSI (*Mobile Industry Processor Interface - Camera Serial Interface*).

O protocolo MIPI CSI é amplamente utilizado, simples e de alta velocidade, destinado principalmente à transmissão de imagem e vídeo entre câmeras e dispositivos hospedeiros. Trata-se de uma interface de câmera, sendo a mais utilizada na indústria. Isso ocorreu devida a sua facilidade de uso e capacidade de suportar uma ampla gama de aplicativos de alto desempenho, incluindo 1080p, 4K, 8K, além de vídeo e imagens em alta resolução (ALLIANCE, 2017).

Permite realizar qualquer implementação com uma única câmera ou com múltiplos dispositivos. A interface também pode ser usada para interligar câmeras em dispositivos de realidade virtual, aplicações de veículos inteligentes automotivos, aparelhos de IoT (*Internet of Things*), sistemas de segurança ou vigilância de reconhecimento facial (ALLIANCE, 2017).

As principais características desse protocolo são as seguintes (ALLIANCE, 2017):

- Profundidade de cor RAW-16 e RAW-20, o que melhora as imagens do tipo HDR (*High Dynamic Range*) e relação sinal/ruído SNR (*Signal to Noise Ratio*);
- Redução da latência e eficiência de transporte LRTE (*Latency Reduction and Transport Efficiency*), facilitando a percepção, processamento e tomada de deci-

sões em tempo real e otimizando o transporte para reduzir o número de fios, taxa de alternância e consumo de energia;

- Modulação por Codificação de Pulso Diferencial DPCM (*Differential Pulse Code Modulation*) para compressão de imagens, reduzindo a largura de banda enquanto fornece imagens com taxas SNR superiores;
- Redução da Densidade Espectral PSD (*Power Spectral Density*), minimizando a interferência à rádio e permitindo maior alcance para canais mais longos.

2.6.1.2 Camada de Driver

É nessa camada que está situado o *device driver* do sistema. Ou seja, é nela que ocorre a comunicação entre o *hardware* e o *software*. O *device driver* é um código de computador que realiza a comunicação entre o *software* e os seus periféricos, nesse caso a câmera(*hardware*). A disponibilidade desse código é importante, pois sem o mesmo não seria possível acessar o *hardware* e, consequentemente, impossibilitando acessar as informações e dados enviados. A interface utilizada para realizar a comunicação entre as outras duas camadas é a *Video4Linux*.

O V4L (*Video4Linux*) é uma interface kernel para rádios analógicos, captura de vídeo e drivers de saída. Os drivers do V4L são implementados como módulos do kernel, carregados manualmente pelo administrador do sistema ou automaticamente quando um dispositivo é aberto pela primeira vez, fornecendo funções auxiliares e uma interface de aplicação comum (SCHIMEK et al., 1999).

Sendo assim, o *Video4Linux* possui as características comuns de uma Interface de Programação de Aplicativos API (*Application Programming Interface*), como (SCHIMEK et al., 1999):

- Realizar a abertura e fechamento de dispositivos;
- Permitir a consulta das capacidades do *hardware*;
- Alterar parâmetros de captura, como exposição e gama;
- Verificar a prioridade da aplicação;
- Manipular entrada/saída de vídeo e audio;
- Aplicar sintonizadores e moduladores no sinal;
- Permitir definir um conjunto de padrões para a entrada/saída, para que o usuário tenha um controle sobre a customização da imagem;

- Modularização e portabilidade de funções
- Viabilizar a realização de recortes e definição de escalas na imagem, além de diversos outros recursos;

Dessa forma, essa interface possibilita que os dados do periférico (*hardware*) possam ser repassados, adequadamente, ao sistema ou *software* que irá utilizar estas informações para um determinado fim. No caso deste trabalho, para que a visão computacional seja aplicada nas imagens.

2.6.1.3 Camada de *Software*

É nesta camada que ocorre a parte de visão computacional. Após o *frame* ter sido capturado pelo sensor de imagem e transferido à camada de Driver, o *software* irá atuar realizando o processamento digital de imagem. As bibliotecas *RaspiCam* e *OpenCV* são utilizadas, pois ambas permitem controle, manipulação e implementação das câmeras e, consequentemente, das imagens.

A biblioteca *RaspiCam* possui total compatibilidade com o microcomputador *Raspberry Pi* e apresenta como principais características (CORDOBA., 2017):

- Fornece uma classe *RaspiCam* para controle fácil e total da câmera;
- Fornece a classe *RaspiCam_Still* e *RaspiCam_Still_Cv* para controlar a câmera no modo estático;
- Fornece a classe *RaspiCam_Cv* para controle fácil da câmera com o OpenCV;
- Fácil compilação e instalação usando o cmake.

O OpenCV é uma biblioteca *open-source*, projetada para ter eficiência computacional e com foco em aplicações de tempo real. Um dos objetivos desta interface é fornecer uma infraestrutura de visão computacional simples, a qual ajude os usuários a desenvolver aplicativos sofisticados (BRADSKI; KAEHLER, 2008).

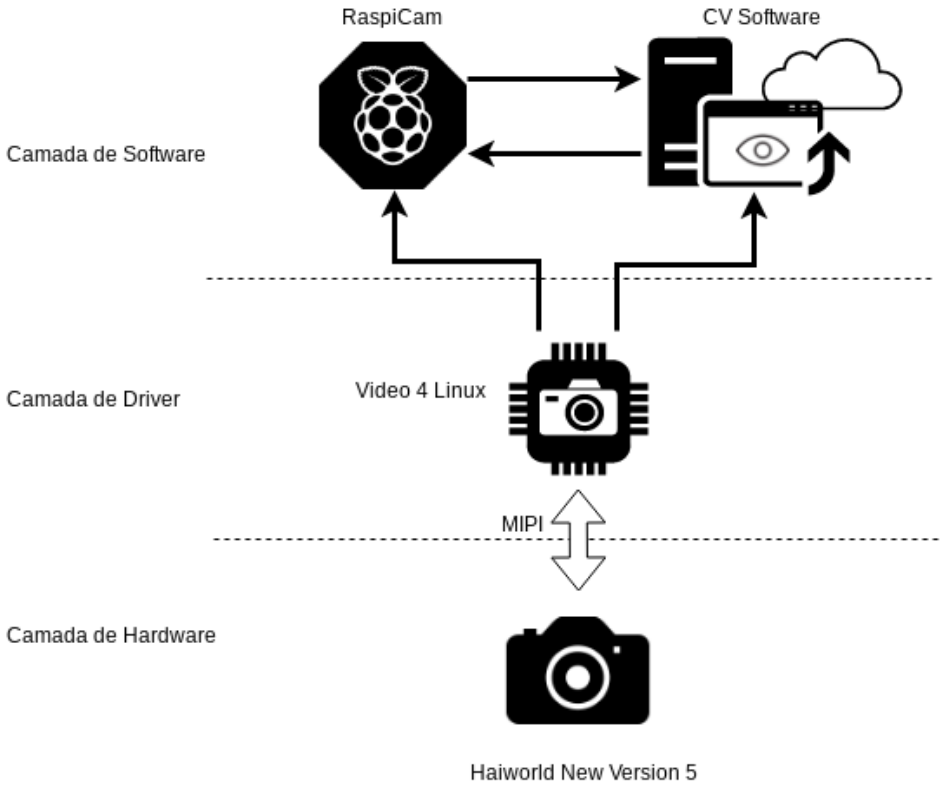
A biblioteca contém mais de 500 funções, que abrangem muitas áreas da visão computacional, permitindo realizar inspeção de produtos de fábricas, algoritmos de segurança, interface com o usuário, calibração da câmera e aplicações em robótica. Além disso, o OpenCV também contém uma Biblioteca de Aprendizado de Máquinas MLL (*Machine Learning Library*) de uso geral. Sendo focada no reconhecimento de

padrões e muito útil paras as tarefas de visão computacional (BRADSKI; KAEHLER, 2008).

Essas bibliotecas são utilizadas no *software*, para realizar o processamento digital de imagens e aplicar a visão computacional. A visão computacional é a modificação dos dados de uma imagem ou de um vídeo, capturados por uma câmera, em uma decisão ou uma nova representação. Todas essas modificações são feitas para alcançar algum objetivo particular (BRADSKI; KAEHLER, 2008).

A Figura 2.21 apresenta o diagrama de topologia da etapa de captura, nela é possível notar como é realizada a divisão e a comunicação entre as camadas. Sendo que as setas escuras representam a comunicação remota, ou à cabo, entre duas camadas ou dois itens da mesma camada. Além da comunicação via *Web* para servidor em nuvem. E a seta clara e bidirecional representa que a comunicação entre as respectivas camadas ocorre via barramento de dados.

FIGURA 2.21 – Diagrama de topologia da captura



Fonte: Dos autores

2.6.2 Pré-processamento

Dentro do processamento de imagens, o pré-processamento é a etapa onde os quadros já obtidos pela captura são filtrados, homogeneizados e adequados. O custo computacional de um algoritmo está intimamente ligado a qualidade de seu pré-processamento (BRADSKI; KAEHLER, 2008). Nesta fase do sistema são extraídos parâmetros globais dos quadros e feitas adequações aos mesmos, como no Apêndice F, onde são ajustados os tamanhos dos quadros e extraídos valores de iluminação média.

O conceito de filtragem é presente em qualquer variante do processamento digital de sinais. Sendo uma imagem um sinal discreto bidimensional, não podemos descartar o uso de filtros. Sendo os sensores ópticos sensores analógios, uma imagem produzida por uma câmera está sujeita aos mais variados tipos de ruídos (quantização, captura, térmicos, ópticos e elétricos), logo a filtragem se faz necessária. Os mais comuns tipos de filtros por imagem são o gaussiano e de mediana (passa-baixas), o laplaciano (passa-faixa) e o Sobel (passa-alta). As figuras 2.22 e 2.23 demonstram a aplicação de um filtro de mediana de janela 3x3 (VANDEVENNE, 2004).

FIGURA 2.22 – Imagem exemplo antes do filtro de mediana

Fonte: (VANDEVENNE, 2004)

FIGURA 2.23 – Imagem exemplo após o filtro de mediana

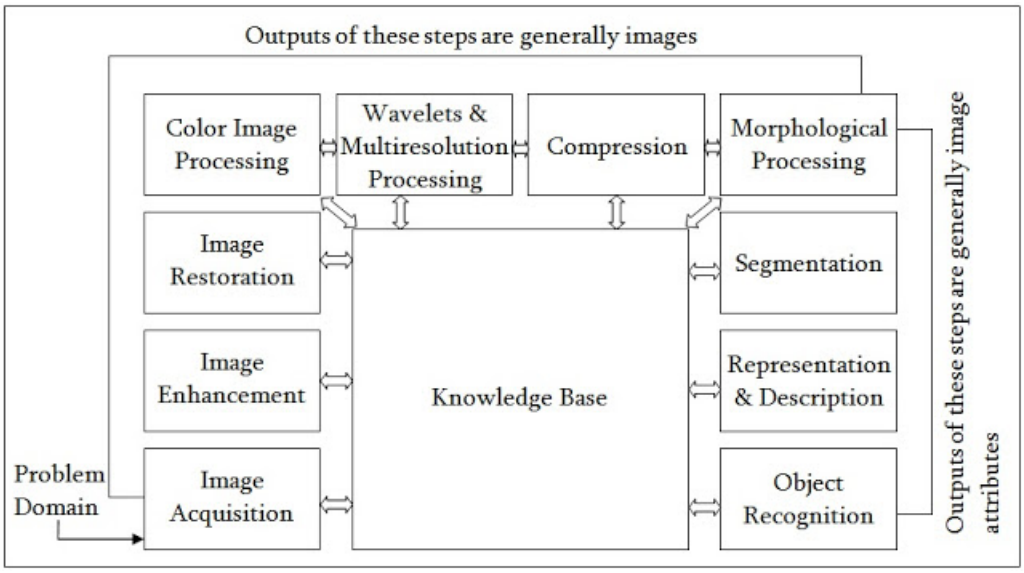
Fonte: (VANDEVENNE, 2004)

Um software em visão computacional geralmente terá um núcleo de inteligência inerente ao mesmo e muitas funções de visão envolvem algoritmos sensíveis aos tipos de dados de entrada (BRADSKI; KAEHLER, 2008). No classificador de gradientes orientados a histograma do Apêndice F é sabido que todas as imagens de entrada devem possuir a mesma dimensão, quantização e espaço de cor para o funciomento correto do algoritmo. Sendo assim, etapas de homogeneinização e adequação, como padronização de tamanho, cor, níveis de ruído e entropia de imagens são fundamentais no bom desempenho de qualquer software de processamento digital de imagens (ROSEBROCK, 2016).

2.6.3 Visão computacional

Esta etapa compreende o núcleo de qualquer algoritmo de PDI a ser aplicado. Existem as mais diversas técnicas e categorias de visão computacional, sendo todas interligadas a vastas áreas de pesquisa (CLASSNOTES, 2013). A Figura 2.24 exemplifica algumas áreas e como podem ser distribuidas. Das áreas ilustradas, a mais importante para o projeto em questão é a de reconhecimento de objetos.

FIGURA 2.24 – Diagrama exemplo de um sistema de visão computacional

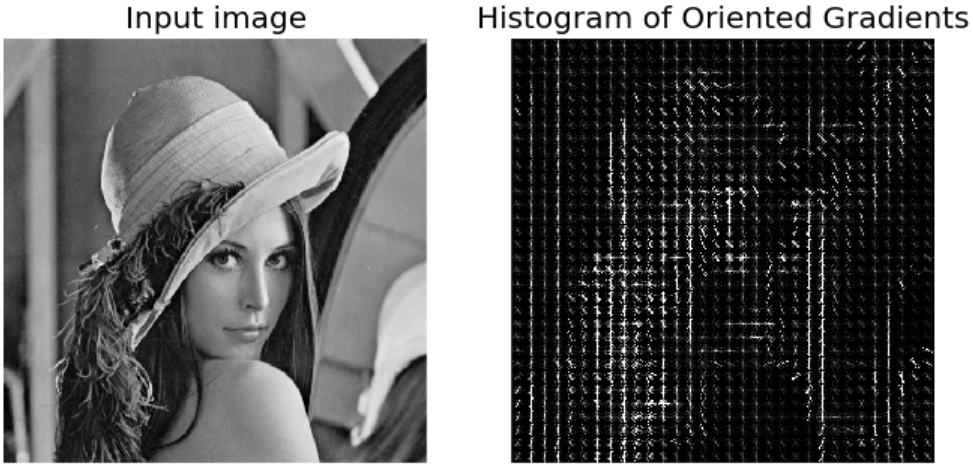


Fonte: (CLASSNOTES, 2013)

O reconhecimento de objetos ou OR (*Object Recognition*) aborda diferentes técnicas de reconhecimento de padrões estatísticos para determinar com certo nível de confiabilidade a presença de um objeto em um quadro (CYGANEK, 2013). Em PDI um objeto é tudo aquilo que pode ser tratado como portador de padrão. Uma pessoa, uma caixa de fósforos e um código de barras são todos objetos pela perspectiva da visão computacional.

A técnica de análise estatística empregada nos algoritmos dos Apêndices B e F são de histogramas de gradientes orientados, onde para uma deteminada célula de pixels, são determinados os gradientes da região e, assim, representados por histogramas que carregam o valor dos gradientes, como intensidade da coluna, e a direção dos gradientes, como posição da coluna dentro da janela normalizada. Na Figura 2.25 se pode observar o comportamento da técnica, para cada célula de pixels são computados e armazenados padrões de histogramas. Para uma futura detecção do objeto em questão, basta uma comparação de proximidade entre os histrogramas, seja por produto ou subtração em escalas múltiplas (como no Apêndide F). Tal padrão sendo encontrado, encontra-se o objeto (CYGANEK, 2013).

FIGURA 2.25 – Exemplo de histogramas de gradientes orientados



Fonte: (MONTANEZ, 2016)

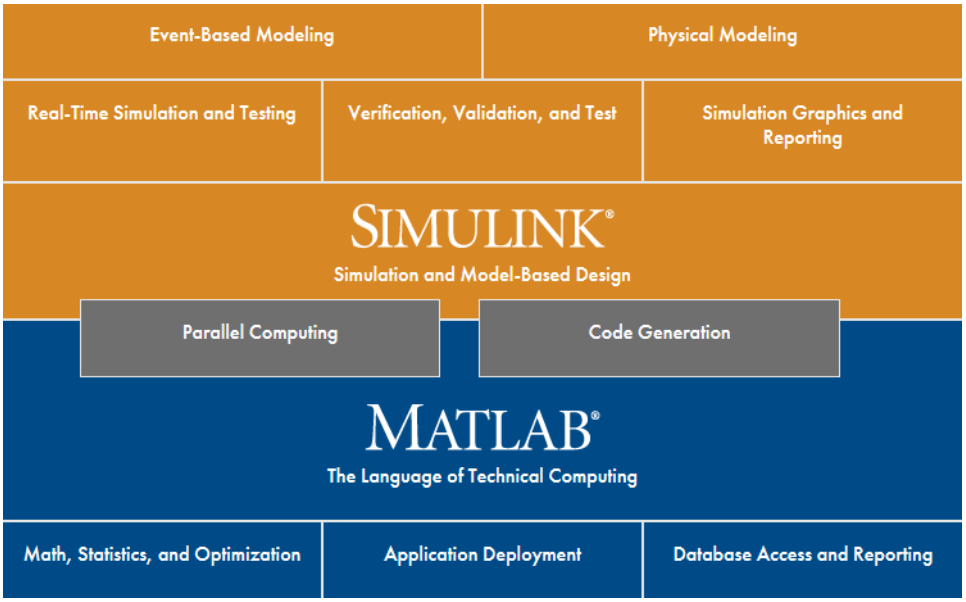
2.7 MATLAB

Matlab pode ser definido como uma linguagem técnica de computação, sendo um ambiente de programação para desenvolvimento de algoritmos, visualização e análise da dados e computação numérica. Desenvolvido pela *MathWorks*, a qual se trata da empresa líder no desenvolvimento de *software* para computação matemática. Ainda, é um ambiente de desenvolvimento buscado por engenheiros e cientistas, a fim de acelerar o ritmo de descoberta e inovação. Além do *software* Matlab, a *MathWorks* desenvolveu o *Simulink*, um ambiente gráfico para simulação e modelagem de sistemas dinâmicos e embarcados (MATHWORKS, 2018).

Estes programas são utilizados em diversos âmbitos da indústria, como na área automotiva, eletrônica, automação industrial, aeroespacial, comunicações, além de outras inúmeras áreas. Permitindo implementar ferramentas fundamentais para pesquisa e desenvolvimento. Além disso, os dois também são usados para modelagem e simulação em campos cada vez mais técnicos e específicos, como em aplicações de biologia computacional e de serviços financeiros (MATHWORKS, 2018).

Atualmente mais de 5000 instituições de ensino, em todo o mundo, utilizam as ferramentas (Matlab e Simulink) para ensino e pesquisa. Sendo aplicados em uma vasta gama de disciplinas técnicas (MATHWORKS, 2018). As diversas aplicações podem ser vistas na Figura 2.26, a qual mostra as categorias para as duas ferramentas.

FIGURA 2.26 – Gama de aplicações das ferramentas *Simulink* e Matlab



Fonte: (MATHWORKS, 2018)

Para o projeto em questão foi utlizada a *Neural Network Toolbox* do Matlab, a fim de aplicar funções de *Deep Learning* para diferentes redes neurais desenvolvidas. Com o intuito de utilizar este tipo de aprendizagem para classificação de imagens.

2.7.1 *Neural Network Toolbox*

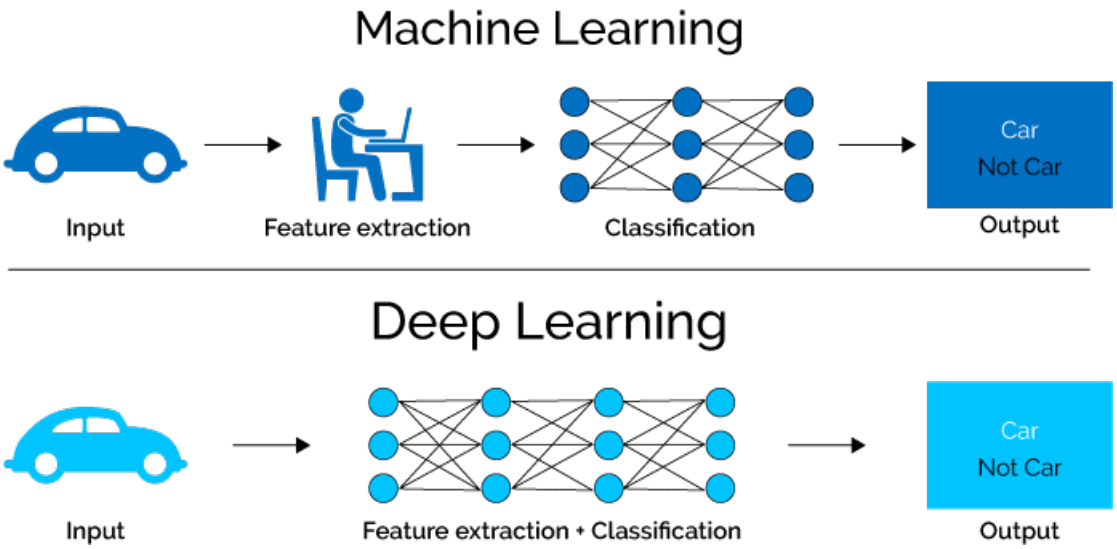
Trata-se de um conjunto de funções que criam, treinam, visualizam e simulam aprendizagens de redes neurais via *deep learning* e aprendizagem superficial. Fornecendo algoritmos, modelos pré-treinados e aplicativos para executar classificação, agrupamento, regressão, previsão de séries de tempo, redução de dimensionalidade e modelagem e controle de sistemas dinâmicos (MATLAB, 2018).

As redes de aprendizagem de máquina incluem redes neurais convulocionais (CNNs), topologias de rede de gráficos acíclicos direcionados (DAG) e autoencodificadores para classificação e regressão de imagens, e aprendizagem de recursos. A aprendizagem de máquinas utiliza de redes neurais para aprender representações úteis de recursos, a partir dos dados, para classificação e regressão. Isto posto, é possível criar e treinar novas redes ou usar redes pré-treinadas disponibilizadas pelo Matlab (MATLAB, 2018).

O *deep learning* é um ramo da aprendizagem de máquina, pois ensina os computadores a realizarem uma função natural do ser humano: aprender com a experiência

de eventos já ocorridos. Os algoritmos responsáveis por realizarem a aprendizagem de máquina utilizam métodos computacionais para "colher" e "aprender" informações, diretamente dos dados disponibilizados, independentemente se há uma equação para determinar o modelo. Assim sendo, o *deep learning* é especialmente adequado para o reconhecimento de imagens, sendo um método relevante para aplicar em problemas de reconhecimento facial e detecção de movimento, além de ser aplicável em muitas tecnologias como direção autônoma, detecção de pista, detecção de pedestre e estacionamento autônomo (MATLAB, 2018). A Figura 2.27 apresenta as diferenças já relatadas, entre a aprendizagem de máquina e o *deep learning*.

FIGURA 2.27 – Diferenças entre aprendizagem de máquina e *deep learning*



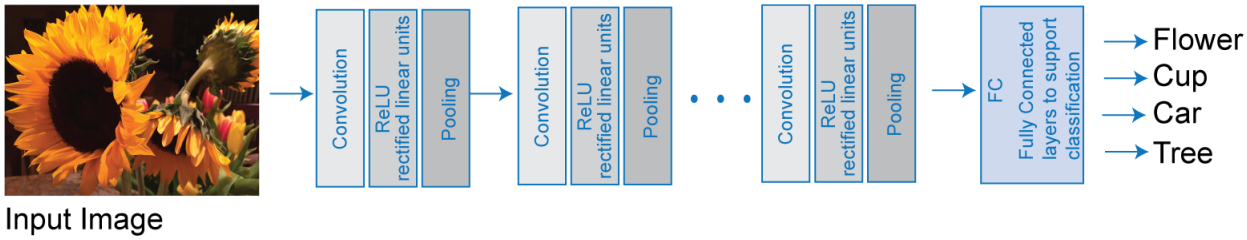
Fonte: (PHPHR, 2018)

As redes neurais utilizadas pelo *deep learning* combinam múltiplas camadas de processamento não-lineares, usando elementos simples para operar em paralelo, inspirados nos sistemas nervosos biológicos (neurônios). Os modelos de *deep learning* podem alcançar níveis altíssimos de precisão na classificação de objetos, sendo que, em alguns casos, esses níveis podem exceder o desempenho em nível humano (MATLAB, 2018).

Para aperfeiçoar o desempenho das redes neurais é preciso treinar modelos, utilizando um amplo conjunto de dados tabelados e arquiteturas de redes neurais que contêm diversas camadas, geralmente incluindo algumas camadas convolucionais. O treinamento destes modelos demanda um intensivo custo computacional, sendo que, regularmente, há a necessidade de aplicar métodos de aceleração de GPU de alto

desempenho para finalizá-los rapidamente. A Figura 2.28 apresenta como as redes neurais convolucionais combinam camadas que aprendem automaticamente, utilizando recursos de diversas imagens, a fim de classificar novos padrões (MATLAB, 2018).

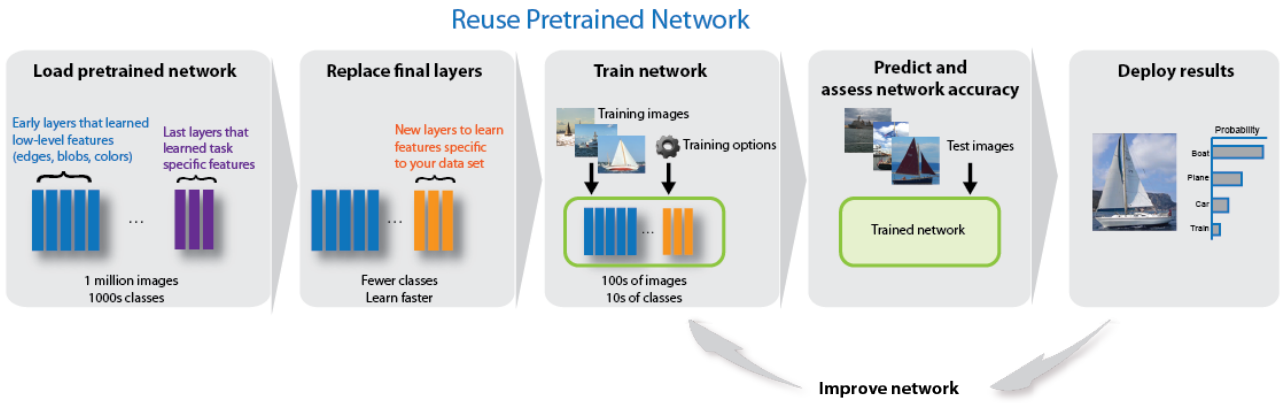
FIGURA 2.28 – Diagrama para classificação de novas imagens



Fonte: (MATLAB, 2018)

Outro exemplo pode ser visto na Figura 2.29, onde o conceito de rede pré-treinada é exemplificado. O propósito inicial é realizar a classificação das imagens, aplicando redes que foram pré-treinadas diretamente a problemas de classificação. Em seguida, ocorre a etapa de transferência de aprendizagem, a qual utiliza camadas de uma rede treinada com um grande conjunto de dados e realiza um ajuste fino, a fim de obter um novo conjunto de dados mais preciso. Por fim, a partir de uma rede pré-treinada como um extrator, ocorre a extração dos recursos e classificação daquele padrão.

FIGURA 2.29 – Uso de redes pré-treinadas



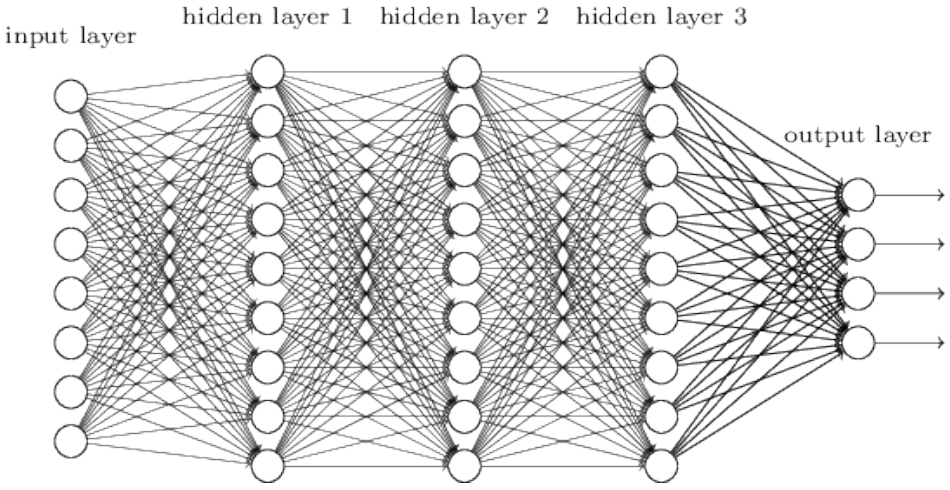
Fonte: (MATLAB, 2018)

2.8 REDES NEURAIS CONVOLUCIONAIS

Os primeiros conceitos de redes neurais surgem na década de 50 (ROSENBLATT, 1958), com as primeiras proposições de neurônios digitais, o *Perceptron*, e suas

funções de ativação. Em suas proposições inicias as redes apresentavam soluções para detecção de padrões, porém ainda pouco se sabia sobre como treiná-las e o quão computacionalmente complexo isso seria. Apenas no final de década de 70 os primeiros algoritmos de treinamento foram propostos (WERBOS, 1990), o *backpropagation*, e no final da década de 80 foi provado que uma rede neural multicamadas é um método universal de aproximação de funções matemáticas (HORNIK; STINCHCOMBE; WHITE, 1989). Com a gradativa evolução do poder computacional dos computadores, redes neurais foram tornando-se soluções cada vez mais viáveis no reconhecimento de padrões, já sendo capazes de realizar detecções com precisão superior ao cérebro humano (DENG, 2012). Cada problema ou aproximação que pode ser resolvido por redes neurais, implica na proposição de uma topologia única, que melhor generaliza a resolução do problema. A Figura 2.30 apresenta a topologia clássica de uma rede neural, com 8 neurônios de entrada, 3 camadas com 9 neurônios cada e 4 neurônios de saída. Tais redes, dadas as devidas alterações de hiperparâmetros são comuns em algoritmos de detecção de tendencias em conjuntos de dados e mineração de informações (LU; SETIONO; LIU, 1996).

FIGURA 2.30 – Rede Neural Clássica

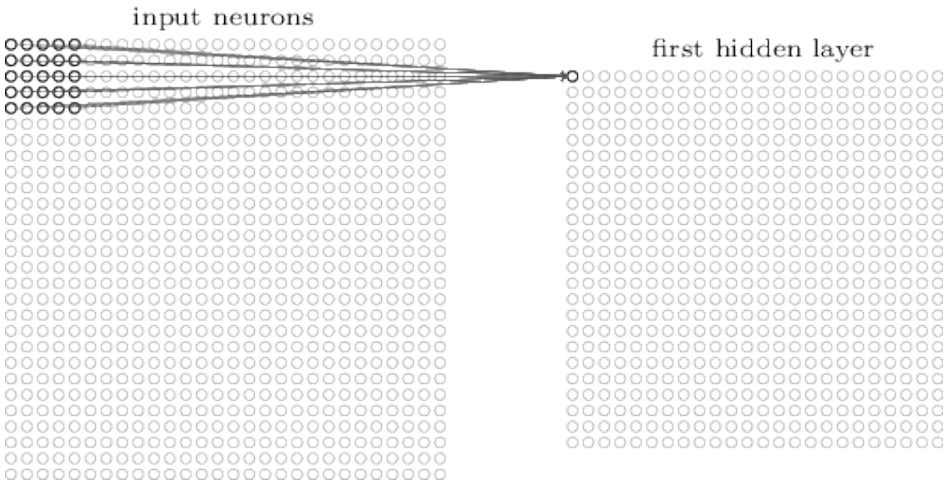


Fonte: (NIELSEN, 2015)

Contrariamente às topologias clássicas, o estado da arte aponta que quando se trata da detecção de padrões por imagem, redes neurais convolucionais apresentam melhores resultados (NIELSEN, 2015). Redes neurais convolucionais fazem uso dos mesmos algoritmos de aprendizagem das redes clássicas, porém são construídas de maneira diferente. Cada camada da rede é composta por n mapas de características

extraídas das camadas anteriores por meio de uma operação de convolução. A Figura 2.31 exemplifica a ligação entre camadas, onde o valor a função de ativação de uma camada posterior remete à um *canvas* das ativações da camada anterior. Dessa forma, cada neurônio das camadas seguintes, carrega o peso das características apresentadas em determinada região da imagem.

FIGURA 2.31 – Exemplo de convolução



Fonte: (NIELSEN, 2015)

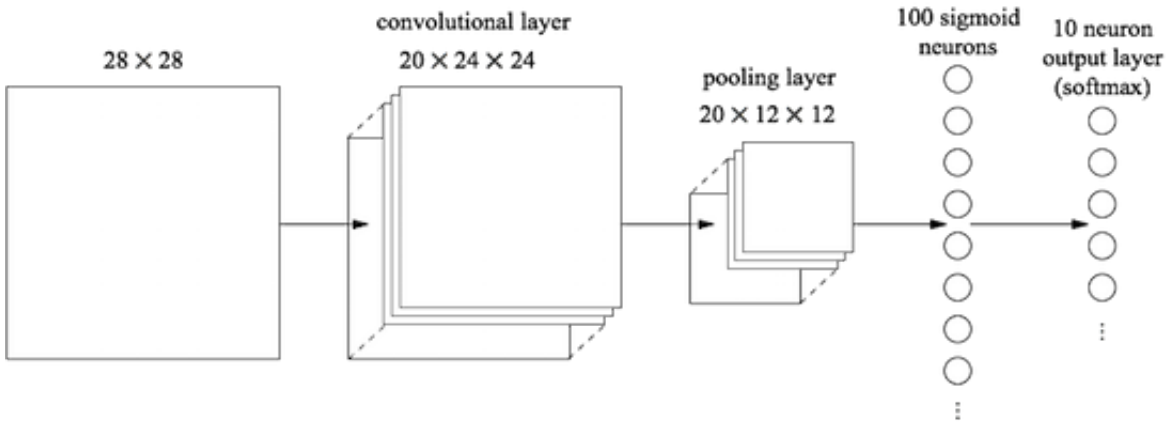
Chamam-se então redes neurais convolucionais pela similaridade de operação executada pela função de ativação dos neurônios, Equação 2.1 (NIELSEN, 2015), à função de convolução discreta multidimensional, Equação 2.2 (OPPENHEIM, 2010).

$$y(a,b,w) = \sigma\left(b + \sum_{l=0}^{4}\sum_{m=0}^{4} w_{l,m}a_{j+l,k+m}\right) \tag{2.1}$$

$$Z(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} X(m,n)Y(i-m,j-n) \tag{2.2}$$

Desta forma, uma representação mais fiel da natureza das CNNs usadas nos detectores propostos neste projeto encontra-se na Figura 2.32. As camadas de entrada performam operações de convolução, as camadas de *pooling* executam a aglomeração dos dados provenientes dos mapas de características, para redução de ruídos e do volume de dados, e as camadas seguintes unidimensionais tratam as saídas dos detectores e executam a separação e classificação das saídas.

FIGURA 2.32 – Exemplo de CNN



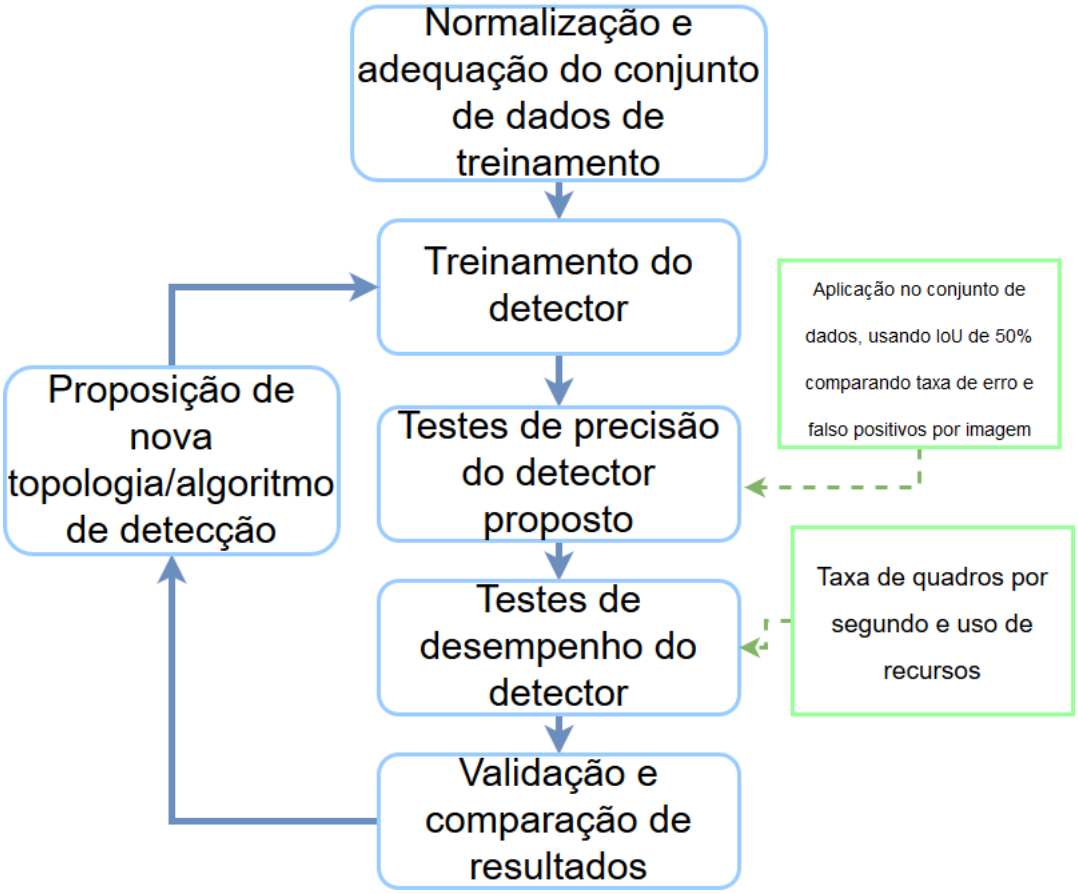
Fonte: (NIELSEN, 2015)

3 DESENVOLVIMENTO

3.1 METODOLOGIA

A Figura 3.1 apresenta um fluxograma que define a metodologia que será utilizada neste projeto.

FIGURA 3.1 – Fluxograma com a metodologia do projeto



Fonte: Dos autores

Analisando a Figura 3.1 é possível verificar que cada caixa possui uma finalidade para o desenvolvimento do sistema de detecção de pedestres. Salienta-se o uso de visão computacional, processamento digital de imagens e redes neurais, a fim de obter um equipamento que possua alto desempenho e um algoritmo de detecção inteligente capaz de ser autônomo. As caixas auxiliares dizem quais métricas serão observadas bem como alguns parâmetros de avaliação, como a medida de precisão IoU (*Intersection over Union*) (RAHMAN; WANG, 2016).

A precisão dos algoritmos propostos será avaliada usando o conjunto de dados de imagens de pedestres INRIA (*Institut National de Recherche en Informatique et en Automatique*) (DALAL; TRIGGS, 2005). O banco de imagens é composto de 614 imagens de treinamento e 288 imagens de teste. Mesmo sendo de 2005 o conjunto de dados ainda é relevante em diversos outros artigos do estado da arte (CALTECH, 2009). A Figura 3.2 traz algumas das imagens presentes no conjunto.

FIGURA 3.2 – Exemplo de imagens do INRIA *dataset*

Fonte: (DALAL; TRIGGS, 2005)

O *hardware* atual encontra-se adequado e funcional perante o problema proposto, porém a busca por um melhor método de detecção de pedestres por imagem segue sendo prioridade do projeto. A metodologia aplicada trata em comparar o desempenho dos algoritmos propostos sobre um vídeo de amostra, presente no conjunto padrão de amostras do OpenCV (OCVTEAM, 2017). O ambiente controlado permite avaliar o desempenho levando em consideração um número limitado de variáveis e melhor explorando indicadores de melhorias.

3.2 ELABORAÇÃO

3.2.1 Análise comparativa de adequação dos sensores de imagem

Ponderando-se as informações fornecidas na fundamentação teórica, o módulo OV5647 consiste na escolha mais segura para implementar no projeto, já que possui total compatibilidade com o *hardware*. Consequentemente este módulo não deve apresentar problemas para os primeiros conjuntos de dados, uso da CPU, uso de memória e velocidade de resposta e latência, sendo que mesmo com sua compatibilidade com o *Raspberry*, os conjuntos de dados que representam as respostas que o módulo fornece necessitam ser comparadas com o conjunto de dados da câmera Logitech C270. A fim de determinar, experimentalmente, a câmera que irá responder melhor ao código de detecção que será implementado.

O uso da câmera OV5647 se torna muito específico. Por ser um periférico dedicado ao *Raspberry Pi*, o módulo é conectado via cabo *flat* à plataforma, diferentemente das convencionais câmeras USB. A configuração e uso desse dispositivo precedem algumas configurações na plataforma. Como em qualquer alteração, deve-se atualizar a lista de aplicações do Linux e então se atualizar a plataforma antes. Ao acessar a configuração do computador via comando "*sudoraspi-config*" habilita-se o dispositivo e então é necessária uma reinicialização do sistema.

Já com a câmera instalada, pode-se usar funções das bibliotecas nativas do *Raspberry*. Para se manter um padrão de amostragem, foi escrito um breve código de gravação de vídeo para processamento. O código escrito em Python do Apêndice A faz uso das funções e objetos nativos da biblioteca "*picamera*", e seu funcionamento envolve um bloco de estanciamento do objeto, seguido do início da captura, a espera que garante a alocação do *hardware* pelo *kernel* do sistema operacional para a colheita de imagens e por fim um fechamento da captura. Os comentários do código do Apêndice A ilustram bem a funcionalidade da estrutura.

Para a avaliação individual de cada câmera, um vídeo foi gravado com cada, em formato "*.h264*" que envolve pouca compressão ou codificação, sendo assim um resultado de uma captura quase pura "*near-RAW capture*". O padrão utilizado foi de vídeos curtos de 4 segundos, sendo gravados à 30 quadros por segundo. A escolha de vídeos curtos parte do princípio que nosso próximo algoritmo, o de detecção, é mais minucioso e avalia quadro a quadro do filme e por questões práticas, foi julgado 120

quadros como uma base de dados suficiente.

Seguindo a linha de desenvolvimento do projeto, foi testada em sequência a abordagem com *webcam* USB. Para o uso de tal, foi necessária a elaboração de um novo código de captura de vídeo. o Apêndice C representa o código empregado. Por não utilizar bibliotecas exclusivas do *Raspberry*, o método de captura é levemente diferente e faz uso das funções padrão do OpenCV para captura, funções essas que por sua vez empregam métodos da Video4Linux em segundo plano (SCHIMEK et al., 1999).

O código do Apêndice E por sua vez é escrito em outra linguem de programação, o C++, e é utilizado no teste da *Haiworld New Version 5*, vista na Figura 2.19. A mudança de linguagem obrigou o emprego de outras bibliotecas para comunicação com o sensor de imagem, de protocolo MIPI/CSI. É utilizado um pacote de código aberto fornecido pelo núcleo de pesquisas em visão computacional da Universidade de Córdoba, na Espanha (CORDOBA., 2017), e tal migração se faz necessária para a facilidade de incorporação do algoritmo de captura em futuros detectores. As capturas de imagens obtidas e comportamento de cada uma das câmeras são discutidos no Capítulo 4 e nas Figuras 4.1, 4.2 e 4.3. A Figura 3.3 apresenta o protótipo montado pela equipe, nela vê-se o gabinete contendo o *Raspberry Pi 3 B* na parte de baixo e a câmera *Haiworld New Version 5* montada na parte superior.

FIGURA 3.3 – Protótipo

Fonte: Dos Autores

3.2.2 Experimentos e construção dos algoritmos de detecção

Outra parte do desenvolvimento aborda a detecção de seres humanos nos vídeos. Os parâmetros de detecção ditam que as pessoas devem estar com todo o seu corpo na cena e com o mínimo de oclusão possível. O algoritmo do detector envolve analisar os quadros do vídeo e buscar em cada imagem a presença de um indivíduo. Esta parte da elaboração será dividida entre os métodos de detecção apresentados no estado da arte, para melhor atrelar o desenvolvimento aos passos metodológicos.

3.2.2.1 Histogramas de gradientes orientados e máquinas de vetores de suporte

Tal abordagem é apresentada pelo Apêndice B. No bloco inicial deste algoritmo são importadas as bibliotecas que serão usadas e instanciado o objeto "*hog*" que usa o descritor de gradientes orientados da biblioteca *OpenCV* (cv2). Então é atribuído ao objeto o método de classificação baseado em máquinas de vetor de suporte, com detecção de pessoas como parâmetro. Por fim, abre-se o arquivo de vídeo gerado pelo código do Apêndice C e inicia-se a captura de quadros.

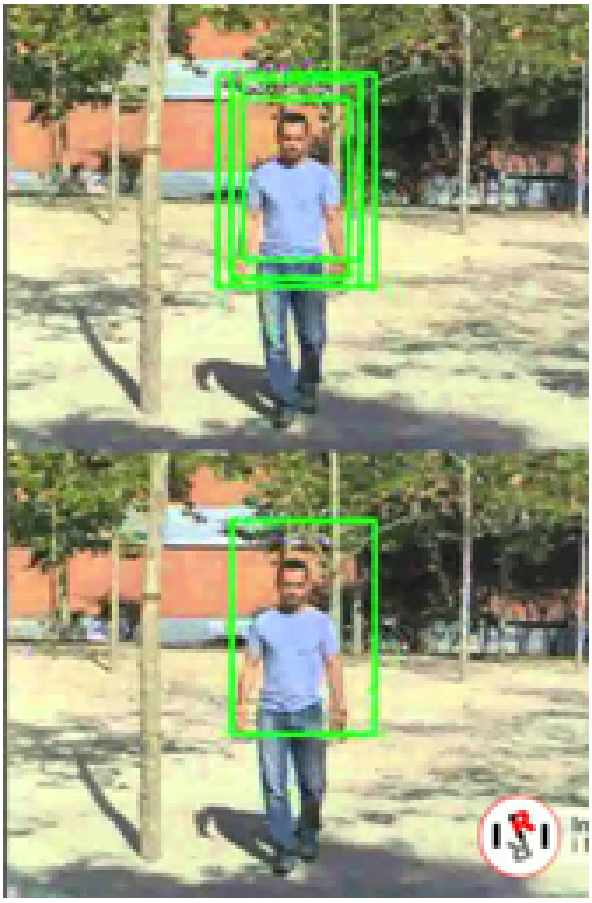
Ainda no Apêndice B, é feito o processamento das imagens armazenadas e enquanto há quadros a serem coletados, o laço se mantém. Cada imagem é re-dimensionada para um limite de 400 pixels de largura, pois imagens muito grandes elevam o custo computacional do algoritmo. Então se aplica a função de detecção baseada no objeto "*hog*" do bloco de inicialização. A função de detecção se utiliza de alguns parâmetros, como tamanho da janela de busca e escala, sendo a função "*hog.detectMultiscale()*" o trecho de maior custo computacional do código.

Esta função retorna uma matriz contendo o valor das coordenadas dos *pixels*, do canto superior esquerdo e inferior direito de cada caixa eleita como um candidato positivo, ou seja, demarca a posição na imagem onde o algoritmo julgou que há um ser humano presente.

Como a função "*hog.detectMultiscale()*" tende a buscar variando a escala, há uma certa superposição nas detecções, como já antecipado no Capítulo 2, logo uma rotina de supressão é aplicada para se eliminar tais redundâncias. Tal função geralmente leva em consideração a região candidata que tenha apresentado o maior índice de confiança do detector. Um exemplo da superposição encontra-se na Figura 3.4. Por fim, é usada uma função de desenho simples da biblioteca *OpenCV* para se

demarcar a saída do detector e são mostrados os quadros de saída em uma janela com aguardo de entrada de tecla para quebra da execução.

FIGURA 3.4 – Saída do detector sem e com supressão de superposição



Fonte: (VILLAMIZAR et al., 2009)

Os comentários dos blocos dos códigos adicionam informações à estrutura e sequência de execução do algoritmo. Os códigos desenvolvidos estão apresentados a partir do Apêndice C.

O código utiliza um codificador de vídeo mais complexo que a captura anterior, porém com melhor qualidade de captação. As próprias bibliotecas do *OpenCV* foram responsáveis pelo gerenciamento do hardware, sendo assim traduzido em funções do tipo "*cap = cv2.VideoCapture(0)*" que inicializa a captura de vídeo no dispositivo '0' ou padrão, e "*ret, frame = cap.read()*" que retorna a verificação de captura "*ret*" e o quadro capturado pela *webcam*.

Diferentemente do código de captura implementado em Python, este não trabalha com uma variável de tempo em segundos, mas nos permite configurar a taxa de captura de quadros do codificador de vídeo. Com a taxa configurada para

30 quadros por segundo e um contador de imagens que limita o número máximo de capturas em 120, são novamente amostrados vídeos de 4 segundos.

O algoritmo detector de pessoas é o mesmo para ambos os vídeos, logo se manteve o mesmo programa de detecção para o vídeo vindo da *Webcam*, alterando-se apenas o parâmetro do arquivo de entrada da rotina, pois fora-se utilizado o detector do tipo HOG na comparação qualitativa entre os sensores de imagem.

Agora com o foco do projeto em elevar o desempenho da aplicação, os passos seguintes envolveram migrar os principais algoritmos para C++ e testar as capacidades do hardware. Sendo o *OpenCV* uma biblioteca aberta e em desenvolvimento, novas versões da mesma foram lançadas desde a implementação do Apêndice B, assim ainda mais otimizações foram trazidas, onde uma série de *bugs* foram corrigidos e um novo núcleo de codificação de vídeo acelerado por hardware foi adicionado, assim reduzindo o tempo de codificação dos quadros capturados (OCVTEAM, 2017). Ao longo da elaboração, houve a substituição do *Raspberry Pi B+* pelo *Raspberry Pi 3 B*, por questões de desempenho e disponibilidade, e a adequação a um novo hardware sempre trás desafios e desta vez não foi diferente. Outras funcionalidades implementadas ao novo *Raspberry* e diferentes compilações de bibliotecas padrões do sistema operacional trouxeram imprevistos. A depreciação de bibliotecas do Debian antes utilizadas e um maior consumo de corrente do dispositivo em relação a versão anterior, são problemas que apresentaram alto custo de tempo para serem contornados, com trocas de fontes de alimentação e recompilações do *kernel* do dispositivo.

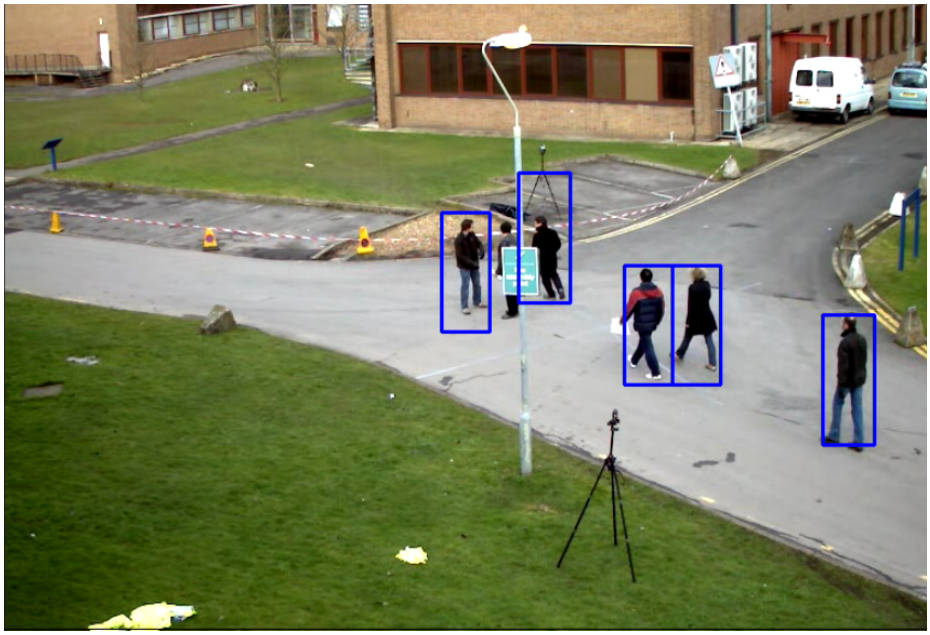
Os algoritmos migrados são documentados nos Apêndices E e F deste relatório. O processo de adequação a uma linguagem de menor nível consome tempo, mas, segundo a literatura (ROSEBROCK, 2016), é mais eficiente e tais diferenças serão discutidas no Capítulo 4.

A principal diferença notada no processo de migração foi a necessidade de uso de uma biblioteca de terceiros, para a ponte de integração entre câmera e *OpenCV* em C++ (CORDOBA., 2017), já que a implementação anterior já era integrada. Outra diferença que aumentou a complexidade da migração original é o fato do supressor de superposição ser implementado em um biblioteca exclusiva em *Python*, logo foi preciso readequar o supressor usando uma função similar do *OpenCV* chamada *cv::groupRectangles()*.

Esse código, Apêndice G, tem seu desempenho comparado com o código do

do Apêndice B. Os resultados se encontram na Tabela 1 do Capítulo 4. Quadros da amostra realizada podem ser vistos na Figura 3.5.

FIGURA 3.5 – Imagem da amostra de vídeo usada nos testes de desempenho



Fonte: Dos autores

3.2.2.2 Fastest Pedestrian Detector in the West

Partindo para o critério do objetivo específico que determina a evolução a nível de desempenho, foi desenvolvida uma solução à partir do artigo *Fastest Pedestrian Detector in the West* (FPDW) (DOLLÁR; BELONGIE; PERONA, 2010). Tal algoritmo é desenvolvido usando classificadores de características de múltiplas escalas por meio de árvores de decisão e tem o alto desempenho apontado pelo estado da arte (CALTECH, 2009).

A implementação deste método na plataforma embarcada foi um verdadeiro desafio. Não se encontram publicações que demonstrem a adequação do mesmo, e as que ao menos o citam comentam o insucesso da elaboração (URZEDO, 2017). Por ser um algoritmo compilado em C++ e apenas para processadores de arquitetura Intel x86 (CHM, 2018) com 64 bits, a migração dos algoritmos para o hardware embarcado *Raspberry Pi* foi necessária. Um novo sistema operacional teve de ser compilado e configurado para o *hardware*, o pi64 (AMARNI, 2018).

Como a FPDW dispõe de repositório em código aberto (https://github.com/apennisi/fastestpedestriandetectorinthewest), foram feitas as alterações necessárias nos

cabeçalhos da biblioteca para a compatibilidade entre arquiteturas Intel x86 e ARM-v8, que envolvem questões de compatibilidade de instruções e interpretação de ponto flutuante, para que a compilação fosse bem sucedida. O Apêndice J trás o código de utilização do detector e seus resultados são discutidos no Capítulo 4.

3.2.2.3 MobileNets

Abordando o quesito precisão do sistema, foram propostos algoritmos de detecção de padrões fazendo uso de *deep learning*, como pode ser visto no Tópico 2.8 do Capítulo 2. Sendo um ramo de conhecimento novo para os integrantes do projeto, uma série de algoritmos de testes foram desenvolvidos em MATLAB com o intuito de melhor desenvolver a compreensão da equipe quanto ao novo conceito.

Foi implementado uma solução usando redes neurais convolucionais, com treinamento de ajuste de pesos e limiares pelo método de descida estocástica de gradiente (NIELSEN, 2015). As taxas de aprendizado e números de *epochs* encontram-se descritas nas *Training Options* do Apêndice I. Inicialmente duas topologias de rede foram propostas, levando em consideração topologias propostas por Nielsen no Capítulo 6 de seu livro de 2015, *Neural networks and deep learning*. Então, uma terceira proposta foi adicionada, fazendo uso de características retangulares. As estruturas de redes propostas no livro são desenvolvidas para o reconhecimento de caracteres, mas foram ponto de partida para a elaboração da equipe.

As Figuras 3.6, 3.7 e 3.8 descrevem a sequência de camadas que compõem as redes propostas, bem como suas dimensões e propriedades.

Mesmo estas implementações preliminares foram capazes de produzir resultados na detecção de pedestres e tais resultados podem ser vistos no Capítulo 4. Mesmo sendo interessante do ponto de vista de prototipagem, estes avanços são de difícil implementação embarcada, porém com os conhecimentos adquiridos foi possível a implementação da solução em *MobileNets*.

O Apêndice K demonstra a aplicação desenvolvida. A *MobileNet* por si só é treinada para detectar 1000 classes de objetos diferentes, porém a aplicação discutida neste trabalho visa apenas detectar pedestres, desta forma, o modelo já treinado da *MobileNet-128* foi recalibrado e teve suas camadas finais retreinadas para melhor se adequar a aplicação. Tal método é conhecido como *fine tuning* (PAN; YANG et

al., 2010) e foi responsável pela melhora dos resultados obtidos pelo detector aqui utilizado em relação aos modelos *out of the box*. O retreino foi executado em um computador com placa de vídeo GeForce GTX1050ti com 4 Gb de memória de vídeo (NVIDIA, 2017), mesmo com tal *hardware* o retreinamento levou cerca de vinte e uma horas. Algo que ressalta a versatilidade do método baseado em redes neurais convolucionais é a capacidade de treinamento dos detectores em um ambiente diferente do de aplicação, pois os custos computacionais de treinamento são fundamentalmente diferentes (NIELSEN, 2015).

FIGURA 3.6 – Camadas da rede versão 1

```
layers =

  12x1 Layer array with layers:

     1   ''   Image Input             32x32x3 images with 'zerocenter' normalization
     2   ''   Convolution             20 3x3 convolutions with stride [1  1] and padding [0  0]
     3   ''   ReLU                    ReLU
     4   ''   Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0]
     5   ''   Convolution             40 3x3 convolutions with stride [1  1] and padding [0  0]
     6   ''   ReLU                    ReLU
     7   ''   Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0]
     8   ''   Fully Connected         1000 fully connected layer
     9   ''   ReLU                    ReLU
    10   ''   Fully Connected         2 fully connected layer
    11   ''   Softmax                 softmax
    12   ''   Classification Output   crossentropyex
```

Fonte: Dos autores

FIGURA 3.7 – Camadas da rede versão 2

```
layers =

  15x1 Layer array with layers:

     1   ''   Image Input             64x64x3 images with 'zerocenter' normalization
     2   ''   Convolution             80 7x7 convolutions with stride [1  1] and padding [0  0]
     3   ''   ReLU                    ReLU
     4   ''   Max Pooling             3x3 max pooling with stride [2  2] and padding [0  0]
     5   ''   Convolution             40 3x3 convolutions with stride [1  1] and padding [0  0]
     6   ''   ReLU                    ReLU
     7   ''   Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0]
     8   ''   Convolution             20 3x3 convolutions with stride [1  1] and padding [0  0]
     9   ''   ReLU                    ReLU
    10   ''   Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0]
    11   ''   Fully Connected         64 fully connected layer
    12   ''   ReLU                    ReLU
    13   ''   Fully Connected         2 fully connected layer
    14   ''   Softmax                 softmax
    15   ''   Classification Output   crossentropyex
```

Fonte: Dos autores

FIGURA 3.8 – Camadas da rede versão 3

```
layers =

  15x1 Layer array with layers:

     1   ''   Image Input             64x64x3 images with 'zerocenter' normalization
     2   ''   Convolution             128 7x5 convolutions with stride [1  1] and padding [0  0  0  0]
     3   ''   ReLU                    ReLU
     4   ''   Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
     5   ''   Convolution             64 5x3 convolutions with stride [1  1] and padding [0  0  0  0]
     6   ''   ReLU                    ReLU
     7   ''   Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
     8   ''   Convolution             32 5x3 convolutions with stride [1  1] and padding [0  0  0  0]
     9   ''   ReLU                    ReLU
    10   ''   Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
    11   ''   Fully Connected         64 fully connected layer
    12   ''   ReLU                    ReLU
    13   ''   Fully Connected         2 fully connected layer
    14   ''   Softmax                 softmax
    15   ''   Classification Output   crossentropyex
```

Fonte: Dos autores

Como padrão da elaboração, o algoritmo foi depois migrado para C++ para um aumento em seu desempenho, ou seja, o Apêndice K foi convertido no Apêndice L. O Capítulo 4 apresenta os avanços provenientes da adoção deste método.

3.2.2.4 Mask R-CNN

Mesmo possuindo grande apelo do estado da arte e alta precisão média, foi inviável a adaptação deste método à plataforma embarcada. Os modelos treinados desta biblioteca são grandes, mais de 380 Mb comprimidos enquanto os modelos de MobileNets possuem menos de 30 Mb, e quando separados em tensores, não são suportados pela memória RAM do microcomputador (ERICKSON et al., 2017).

Vários testes foram elaborados, usando instruções do terminal, para avaliações preliminares, porém sem sucesso. A grande demanda por memória e capacidade de processamento desta solução acabam tornando-se excessivas para o *Raspberry Pi*.

4 RESULTADOS

Um código, Apêndice D, para detecção da comutação máxima da GPIO do *Raspberry Pi* foi implementado, a fim de verificar o melhor sensor de imagem, dentre os aqui apresentados, que pode ser utilizado a fim de respeitar os requisitos impostos na metodologia. Visto isso, verificou-se que a melhor taxa de transferência que o hardware tolera são 25 MHz para a comutação da sua GPIO.

Essa taxa propicia a utilização da câmera C270, do módulo OV5647 e da câmera *Haiworld New Version 5*, já que todos possuem uma taxa de transferência que se enquadra à essa frequência máxima suportada pelo *Raspberry*.

4.1 MÓDULO OV5647

O módulo OV5647 foi implementado, se mostrando apto a ser utilizado em conjunto com o *Raspberry*. Um código de gravação de vídeo foi desenvolvido, a fim de realizar a detecção de indivíduos no quadro. Por falta de espaço no local de testes, se fez necessário “simular” a presença de pessoas, pois assim foi possível enquadrar os indivíduos no quadro de reconhecimento da câmera. Nos testes iniciais o algoritmo de reconhecimento não conseguia realizar a detecção desejada, pois como o local dos testes era muito reduzido o quadro de detecção do código não conseguia enquadrar um “corpo inteiro”, ocasionando detecções errôneas e, consequentemente, não demonstrando o resultado que a equipe desejava.

Contudo, ao utilizar uma imagem para “simular” a presença de pessoas, ou seja, realizar o enquadramento correto, a detecção foi efetuada corretamente. É possível notar que na figura há apenas duas pessoas em que uma detecção de “corpo inteiro” pode ser feita, algo que o algoritmo conseguiu realizar. Além disso, o código de supressão desenvolvido também apresentou bons resultados, já que não houve superposição nos quadros detectores, diminuindo a chance de falsos positivos. A Figura 4.1 apresenta um quadro do vídeo gravado, no qual foi realizada a detecção de duas pessoas na imagem utilizada para teste.

FIGURA 4.1 – Saída do algoritmo de detecção módulo OV5647

Fonte: Dos autores

4.2 LOGITECH C270

Após o teste do módulo OV5647 foi implementado o mesmo algoritmo de detecção com a câmera Logitech C270, a fim de comparar o desempenho do software, taxa de resposta do sistema, precisão na detecção e qualidade geral do vídeo. Por isso o mesmo método de detecção foi efetuado, a fim de preservar o teste inicial e, dessa forma, uma comparação efetiva ser realizada.

Isto posto, um código de detecção de vídeo para a *webcam* foi desenvolvido, considerando que o código anterior não poderia ser utilizado pelo fato do módulo OV5647 possuir compatibilidade com o *hardware*, via comunicação MIPI CSI e uma comunicação via USB ser necessária para a *webcam*, tendo em vista que a mesma não possui essa correlação.

De início foi possível notar uma qualidade de vídeo um pouco inferior ao módulo OV5647, o que acarretou na tentativa de melhora da forma de captura de vídeo, ou seja, alterar o algoritmo. Contudo foi verificado que mesmo com essa alteração a qualidade continuava a mesma, sendo assim foi possível averiguar que a câmera não permitia alterar a sua resolução fora da sua interface de configuração padrão. Essa interface não possui compatibilidade com o *Raspberry Pi* e como o objetivo deste projeto é

realizar testes com esse *hardware* isso ocasionou que os testes fossem feitos com uma resolução *standard* da câmera.

A forma de detecção foi a mesma utilizada no caso anterior, dessa forma foi feita a mesma “simulação” para identificar a presença de pessoas, enquadrando os indivíduos no quadro de reconhecimento da câmera, permitindo a gravação do vídeo. Como o mesmo algoritmo de identificação foi utilizado, é possível notar as mesmas características do teste anterior com o código de supressão evitando superposição de quadros e a detecção sendo realizada corretamente. Permitindo avaliar que não houve diferença quanto ao desempenho do *software*. Entretanto, com a utilização da Logitech o desempenho geral do sistema apresentou menor destreza se comparada com o módulo OV5647. Além disso, pelo problema de incompatibilidade com a interface da própria câmera a resolução implementada foi a padrão. A Figura 4.2 apresenta um quadro do vídeo gravado, no qual foi realizada a detecção de uma pessoa na imagem tomada para teste.

FIGURA 4.2 – Saída do algoritmo de detecção Logitech C270

Fonte: Dos autores

4.3 MIGRAÇÃO DE PLATAFORMA/ALGORITMO E *HAIWORLD NEW VERSION 5*

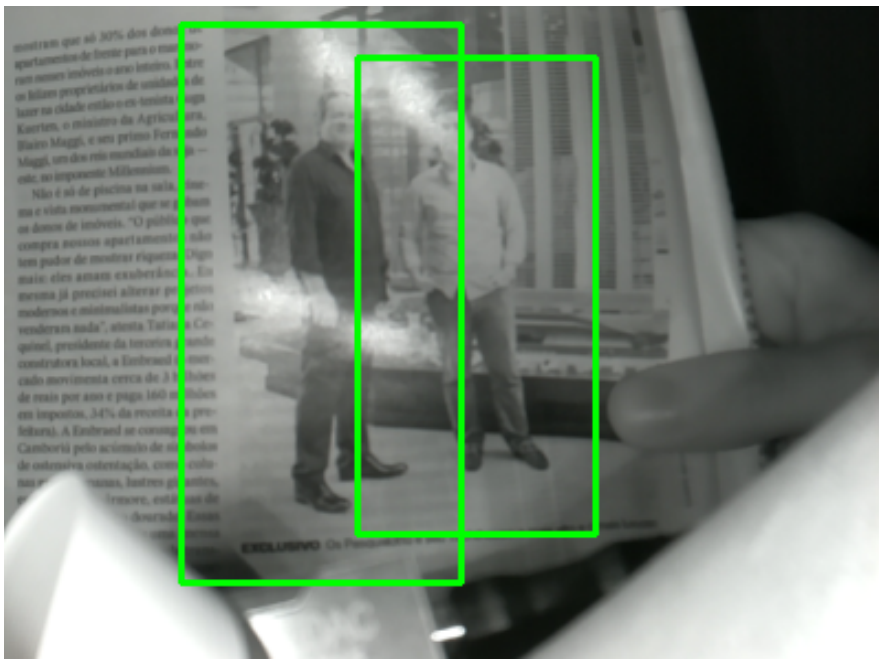
Após os ensaios preliminares, a equipe obteve uma oportunidade de atualizar o *hardware* em questão. Deixando de utilizar o dispositivo *Raspberry Pi B+*, para empregar uma versão superior, o microcomputador *Raspberry Pi 3 B*, sendo que a

diferença entre ambos já foi apresentada nos Capítulos 2.3.1.1 e 2.3.1.2. Ainda, os algoritmos de detecção sofreram alterações em sua linguagem, sendo migrados da linguagem Python para C++. Os códigos migrados foram implementados na nova plataforma e a melhora de desempenho foi notável. Não só pelo fato do novo hardware possuir um processador de 4 núcleos, com um tempo de relógio menor, como também a migração da linguagem de programação trouxe melhorias.

Os arquivos compilados em C++ demonstram um maior desempenho quando comparados com os interpretados em *Python*, pois possuem instruções nativas do processador ao invés de rodar sobre máquina virtual, assim viabilizando o uso da aplicação em um sistema em tempo real. Melhorando o tempo de resposta e o desempenho geral do algoritmo, ainda mais quando se visa embarcar o mesmo num *hardware* de baixo custo.

A nova câmera também trouxe benefícios, por ainda possuir a interface de comunicação MIPI da OV5647, tem um bom tempo de resposta, e seu sensor atuando dentro do espectro infravermelho próximo permitiu que o algoritmo fizesse detecções em ambientes com baixa luminosidade relativa, já que o dispositivo dispõe de resistores ajustáveis que controlam o limiar de atuação dos faróis infravermelhos como pode ser visto na Figura 4.3.

FIGURA 4.3 – Saída do algoritmo de detecção Haiworld New Version 5



Fonte: Dos autores

Do mesmo modo como nas detecções precedentes, foi utilizado o mesmo procedimento, por conseguinte foi executada a mesma “simulação” para identificar a presença de pessoas, enquadrando os indivíduos no quadro de reconhecimento da câmera, permitindo a gravação do vídeo.

Após a migração para a nova linguagem de programação, foi realizado um teste para uma detecção efetiva de uma pessoa. Além disso, houve uma mudança nos locais de testes, permitindo que o dispositivo estivesse numa posição adequada e fosse capaz de realizar a detecção efetivamente. Além de apresentar maior fluidez, o algoritmo realizou uma detecção eficaz, enquadrando corretamente o indivíduo e sem supressão de detecção. A Figura 4.4 apresenta um frame do vídeo gravado, no qual foi realizada a detecção do indivíduo.

FIGURA 4.4 – Detecção Haiworld New Version 5, novo teste

Fonte: Dos autores

Como o sensor de imagem utilizado foi o *Hairworld New Version 5*, a qualidade da imagem foi mantida e o código se manteve estável. Permitindo avaliar que o desempenho do *software* migrado se manteve e que não houve diferenças no tempo de resposta ou desempenho do sistema, mesmo com as novas funcionalidades incluídas ao *pipeline*.

Agora com métricas envolvidas, uma comparação quantitativa pode ser feita entre os algoritmos já descritos. A Tabela 1 traça uma comparação entre as duas

linguagens propostas, Python e C++, em funcionamento no novo *hardware*. Fazendo jus a metodologia proposta, foram avaliadas ocupação de memória e CPU das duas aplicações, porém as mesmas apresentaram comportarmentos similares. O comportamento foi um uso médio de 30% da memória RAM disponível e uso máximo de um núcleo do processador, ou seja, 25% da capacidade total do dispositivo.

Ainda assim, é observado um ganho médio de aproximadamente de 84% entre as linguagens, muito devido a natureza de ambas, onde uma é nativa e compilada (C++) e a outra é executada em máquina virtual e é interpretada (Python). Tal ganho possibilitou um maior vislumbre da aplicabilidade do dispositivo, visto que o ganho de desempenho foi notável.

TABELA 1 – Comparação de desempenho

Teste	FPS médio (Python)	FPS médio (C++)	Ganho de desempenho	Ganho médio
1	0,734	1,331	81,335%	
2	0,732	1,362	86,065%	83,6%
3	0,741	1,359	83,401%	

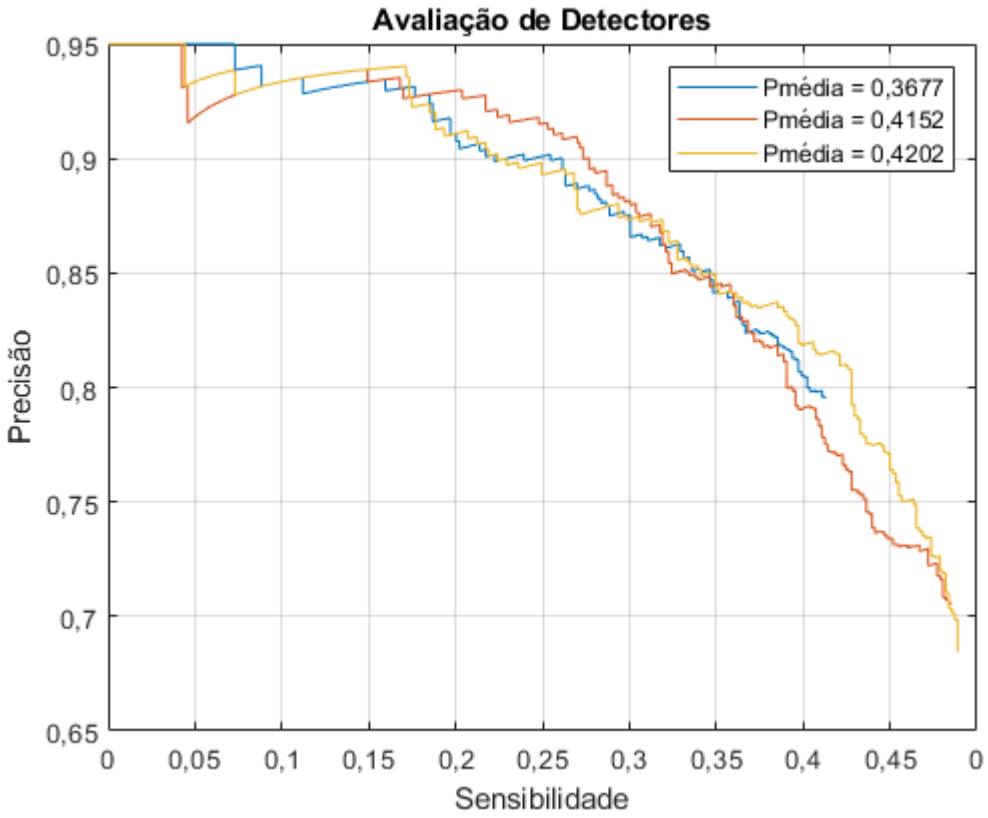
Fonte: Dos autores

4.4 DETECTORES PROPOSTOS

A partir das três camadas de redes propostas no Capítulo 3.2.2.3, foi possível obter os resultados da Figura 4.5. Onde é possível avaliar a precisão média inicial para os modelos desenvolvidos através de algoritmos de *deep learning* em MATLAB. Permitindo ponderar qual das camadas preliminares mais se adequava com o detector que a equipe pretendia elaborar. A implementação dessas três tipos de redes visam produzir resultados preliminares, com o intuito de avaliá-las para prosseguir com a evolução do detector.

Isto posto, através dos resultados propedêuticos, foi possível verificar que dentre as três camadas de redes neurais propostas, a melhor foi a camada de versão 3, a qual se utilizou de características retangulares para aplicar as convoluções. As Figuras 4.6, 4.7 e 4.8 apresentam exemplos de detecção de validação desempenhadas pelo detector versão 3. Os valores presentes acima das caixas de detecção nas imagens, representam as taxas de confiança atribuídas à região pela rede neural, permitindo ainda uma camada adicional de filtragem por confiança, que no caso do detector em questão empregou-se o limiar de 70% de confiança.

FIGURA 4.5 – Curva Precisão x Sensibilidade dos detectores 1, 2 e 3

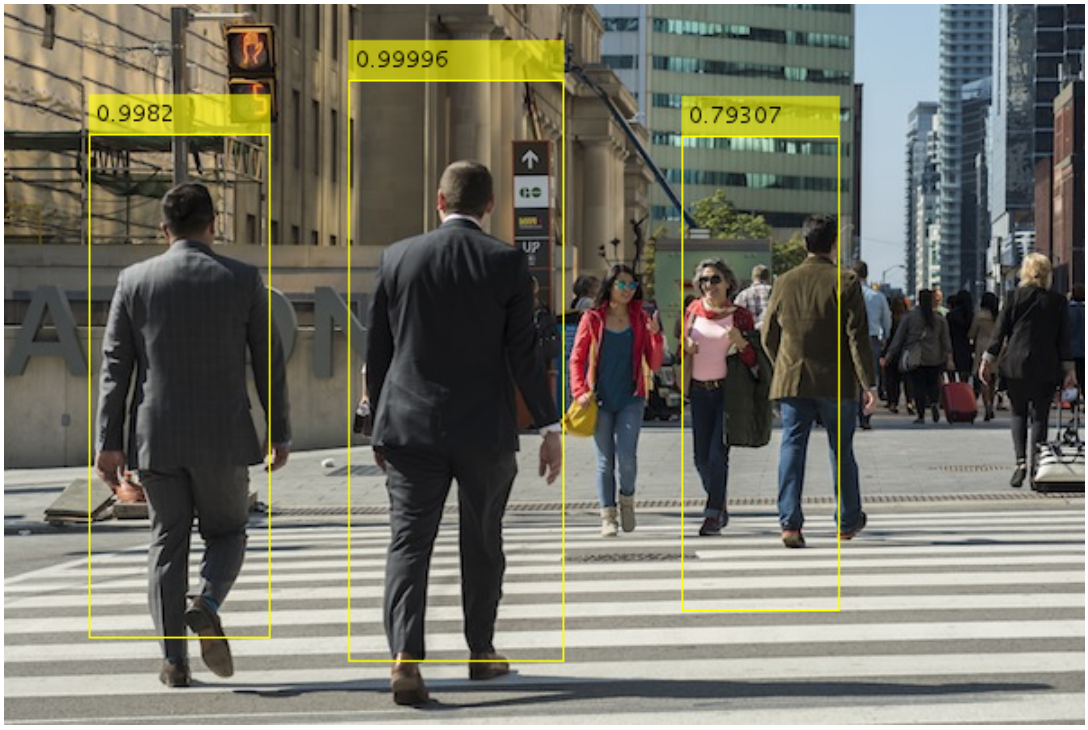


Fonte: Dos autores

FIGURA 4.6 – Exemplo de detecção de validação 1

Fonte: Dos autores

FIGURA 4.7 – Exemplo de detecção de validação 2



Fonte: Dos autores

FIGURA 4.8 – Exemplo de detecção de validação 3

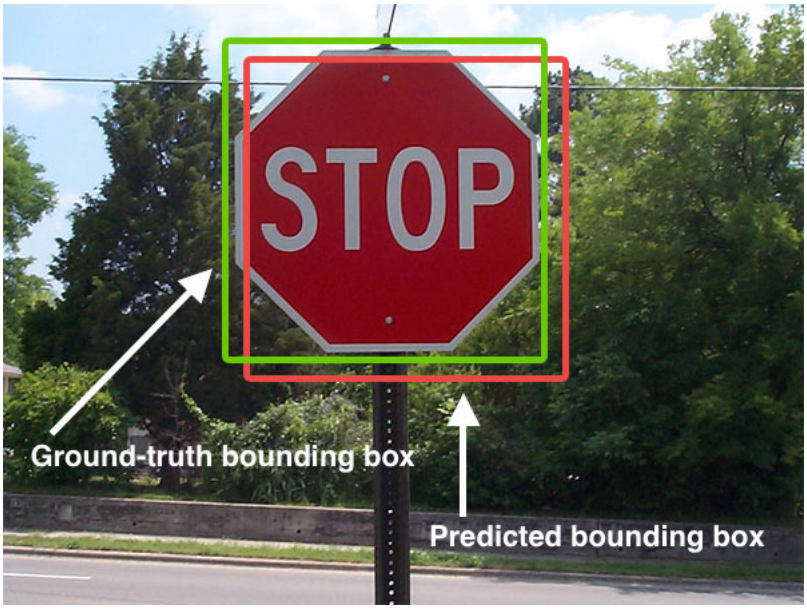
Fonte: Dos autores

Os algoritmos desenvolvidos e embarcados ao longo do Capítulo 3 são comparados no gráfico da Figura 4.14 e as Figuras 4.11, 4.12 e 4.13 demonstram qualitativamente os resultados obtidos numa mesma amostra da base de dados *INRIA* utilizada. Como os resultados apresentados pelos detectores são áreas destacadas,

a quantificação do erro torna-se complexa. O método abordado para inferência de resultados falsos positivos é o de intersecção sobre união, como já comentado na Metodologia e ilustrado na Figura 4.9 e 4.10, e considerando uma taxa superior a 50% de IoU para validação do resultado. Essa taxa foi utilizada, pois segundo Rosebrock (2016), é considerada como uma taxa de predição satisfatória.

Esse método consiste na avaliação a partir de duas caixas delimitadoras ou *bounding boxes*. A primeira caixa se trata da *ground-truth bounding boxes* ou, simplesmente, a caixa delimitadora que especifica onde está a imagem do nosso objeto. A segunda caixa é definida como *predicted bounding boxes*, sendo a caixa delimitadora prevista do modelo. A Figura 4.9 apresenta um exemplo de detecção para essa métrica, a associação dessas duas caixas permitem aplicar o método IoU.

FIGURA 4.9 – Exemplo de detecção métrica IoU



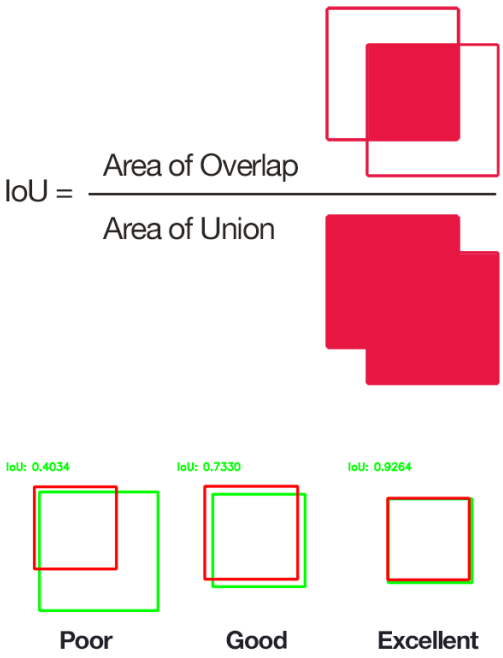
Fonte:(ROSEBROCK, 2016)

Assim sendo, a Figura 4.10 apresenta graficamente a razão requisitada para se encontrar o valor do IoU. No numerador, é dada a área de intersecção entre as duas caixas, enquanto no denominador é dada a área que engloba ambas as caixas.

Isto posto, esse método se trata de uma métrica a qual permite obter a exatidão do detector proposto. Quando se propõem modelos para detecção de objetos, devido à diversos parâmetros inerentes, as caixas delimitadoras dificilmente estarão totalmente correspondentes. Dessa forma, se torna necessário avaliar se as *predicted bounding boxes* estão sobrepostas às *ground-truth bounding boxes*, permitindo avaliar se as

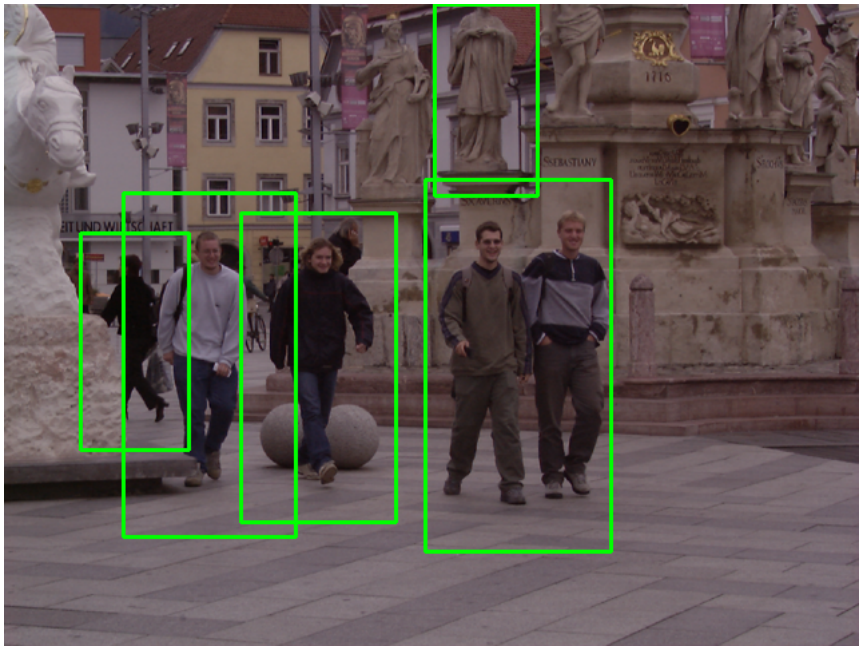
caixas delimitadoras do modelo estão correspondendo o mais próximo possível com o objeto.

FIGURA 4.10 – Representação gráfica da intersecção sobre união



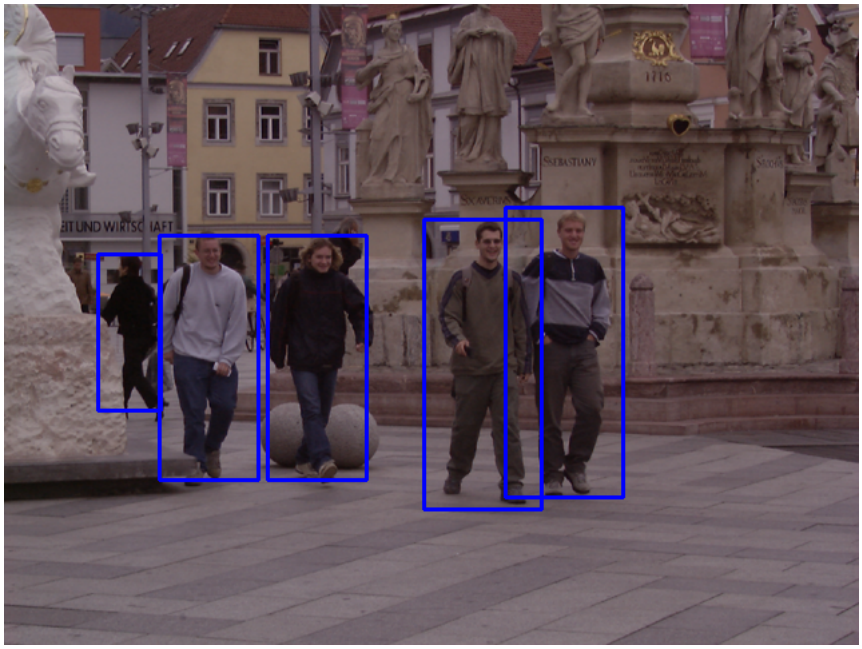
Fonte:(ROSEBROCK, 2016)

FIGURA 4.11 – Exemplo de detecção de validação comparativa - HOG



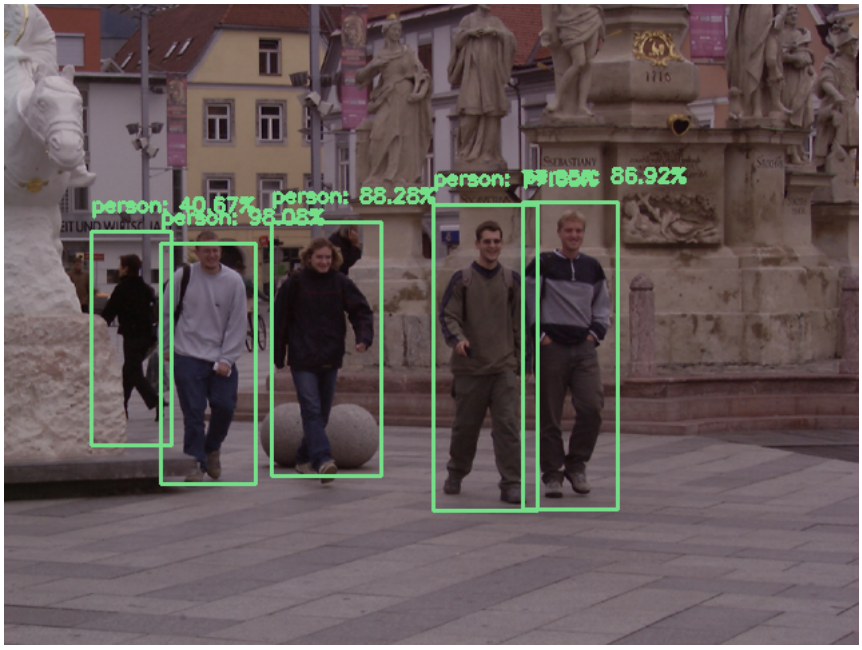
Fonte: Dos autores

FIGURA 4.12 – Exemplo de detecção de validação - FPDW



Fonte: Dos autores

FIGURA 4.13 – Exemplo de detecção de validação - *MobileNet*

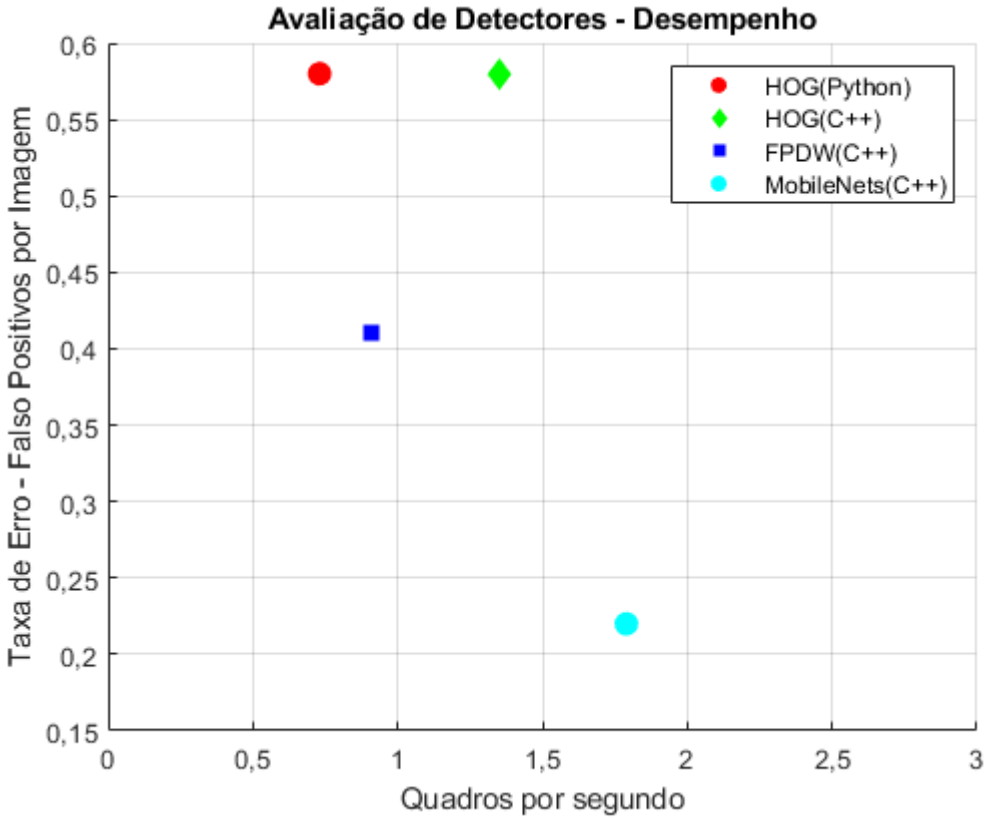


Fonte: Dos autores

O gráfico de comparação estabelecido, engloba os três tipos de detectores propostos nessa tese. Sendo por meio de algoritmos de histogramas de gradientes orientados, em linguagem de programação Python e C++, algoritmo FPDW e utilizando algoritmos de detecção de padrões fazendo uso de *deep learning* e redes neurais

convolucionais, a *MobileNet*. Assim sendo, os resultados podem ser vistos na Figura 4.14

FIGURA 4.14 – Gráfico de Comparação de Desempenho



Fonte: Dos autores

Com base nos resultados, é possível verificar, a partir das Tabelas 2 e 3, os índices de forma mais clara. Ainda, é nítido como o detector proposto via *MobileNet* possui desempenho e precisão muito superiores aos detectores restantes.

TABELA 2 – Desempenho dos algoritmos

	HOG (Python)	**HOG (C++)**	**FPDW (C++)**	**MobileNet**
FPS médio	0.73	1.35	0.91	1.79
Taxa de Erro	0.58	0.58	0.41	0.22

Fonte: Dos autores

TABELA 3 – Comparação do ganho de desempenho/precisão

	HOG (Python)	**HOG (C++)**	**FPDW (C++)**
Ganho de Desempenho MobileNet	145.21%	32.59%	96.7%
Ganho de Precisão MobileNet	62.07%	62.07%	46.34%

Fonte: Dos autores

Desta maneira, os resultados permitem averiguar que, a partir do uso e implementação de uma solução usando redes neurais convolucionais, o detector proposto pôde obter resultados consistentes. Ademais, por se tratar de uma aplicação treinada para detectar 1000 classes de objetos diferentes, foi possível, a partir da recalibração e retreinamento de algumas camadas do modelo já treinado, adequar a *MobileNet* para a aplicação que a equipe desejou.

5 CONCLUSÃO E TRABALHOS FUTUROS

5.1 CONCLUSÃO

Segurança é um mercado muito importante e amplo na nossa sociedade, onde um dos métodos de vigilância passivo mais utilizado é o por meio de câmeras digitais. Com o constatado aumento na violência e as deficiências e vulnerabilidades dos sistemas de vigilância por câmeras disponíveis, o sistema de câmeras inteligentes abordado neste documento apresenta-se como uma solução mais viável ao problema. O principal ponto do sistema é o fato de dispensar de um nó principal (PC ou servidor) para a comunicação entre câmeras e processamento de imagens.

Como visto na revisão bibliográfica, os microcomputadores possuem uma evolução constante nos proporcionando a evolução do projeto das câmeras (ELINUX, 2016). O módulo OV5647 se mostrou muito apropriado para o projeto, o qual consegue conciliar a utilização do *Raspberry Pi* com os requisitos estipulados, como facilidade de implementação, velocidade do *hardware* em conjunto com a câmera, taxas de resposta e etc. A implementação desse sensor de imagem possibilitou o avanço do projeto, sendo possível ser feita a detecção correta de pessoas em gravações de vídeo. A compatibilidade desse dispositivo com o *hardware* permitiu o seu uso a fim de demonstrar os resultados que a equipe esperava.

A câmera Logitech se mostrou uma boa alternativa para o desenvolvimento do projeto, respeitando os requisitos estabelecidos. Contudo, essa câmera apresentou alguns problemas relacionados à sua implementação, já discutidos neste relatório. Mesmo assim, a câmera foi utilizada apresentando resultados ainda razoáveis em comparação com o módulo OV5647.

Com a evolução do hardware foi notada uma melhoria no desempenho em relação a aplicação anterior, mesmo com os ônus de migração de bibliotecas e adaptação ao novo sensor de imagens. A equipe foi capaz de adequar os algoritmos já provados para a nova linguagem C++ e foi constatado o ganho em performance referenciado no Capítulo 2. Visto que o ganho de processamento com a nova plataforma é indiscutível, considerando que o aprimoramento do *hardware* causou uma evolução nos resultados da proposta. Além do uso da nova câmera *Haiworld New Version 5*, a qual apresentou

melhor qualidade de imagem e a importante característica de visão noturna, muito relevante quando considerado que o projeto tem como objetivo geral ser um sistema de segurança em tempo real.

Os detectores baseados em redes neurais convolucionais propostos no Capítulo 3 apresentam precisão média superior às soluções de detecção de pedestres anteriores. As simulações foram executadas em ambiente MATLAB e uma migração dos detectores se fez necessária para o uso na plataforma embarcada Raspberry Pi 3 B. Sendo assim, as redes neurais convolucionais demonstram ser uma solução possível para o aperfeiçoamento do detector empregado pelo ponto de vista da qualidade de detecção. Já no ponto de vista de aprimoramento de desempenho, foi embarcado o algoritmo FPDW (DOLLÁR; BELONGIE; PERONA, 2010), presente no Apêndice J, onde tal código apresentou desempenho semelhante às soluções embarcadas anteriormente, porém com precisão superior, dado o revés da adequação das funções nativas utilizadas para o *hardware* embarcado. Por fim, os detectores baseados em *MobileNets* demonstraram-se superiores aos demais em ambos os quesitos avaliados ao longo deste trabalho, então conclui-se que é o mais adequado algoritmo para a satisfação do objetivo geral deste documento, bem como a complitude integral dos objetivos específicos apresentados.

O projeto desenvolvido ao longo das disciplinas de TCC foi aceito em maio de 2018 no edital de inovação e financiamento de TCCs *Academic Working Capital* (AWC) dos Institutos TIM. Tal aceite trouxe à equipe grande senso de realização e preparo às próximas etapas que este projeto terá, porém agora como produto, trabalho de conclusão de curso, negócio e quaisquer outras barreiras que esta disciplinas incentivaram a equipe a buscar.

Em tempo, o projeto passou para a terceira etapa do desafio Paraná na categoria ideia de negócio e, além disso, a equipe teve o projeto aceito para participar da fase Maker e participarmos do Hack Brazil, um programa de aceleração de iniciativa da Brazil Conference at Harvard & MIT, tendo acesso a mentoria de ambas MIT & Harvard.

5.2 TRABALHOS FUTUROS

O projeto desenvolvido pela equipe foi aceito no programa de inovação TIM AWC 2018, aprovado para a terceira etapa do Desafio Paraná 2018 e, por fim, recebeu

um "Passe Hacker" para participar da fase Maker do Hack Brazil, onde os finalistas escolhidos vão à Boston nos E.U.A defender seu projeto. Isto posto, a equipe vislumbra empreender e aplicar o dispositivo no mercado atual, via fundação de uma empresa emergente, como uma startup, por exemplo. O projeto sendo aceito em notáveis programas de aceleração e inovação, mostra como o trabalho, resiliência e dedicação à inovação podem mudar a vida dos estudantes.

Os integrantes se organizam para no futuro viabilizarem um empreendimento na área de câmeras inteligentes, que se fundamenta em inovação e anseia o desenvolvimento tecnológico, apostando no conceito de *Smart Cities* para garantir um modelo de negócios próspero, capaz de atender o mercado à nível de produto e de serviço, juntamente à responsabilidade envolvida na construção de uma sociedade mais segura, fundamentada numa tecnologia ética que contempla o bem maior.

Por se tratar de um desafio singular para a equipe, podemos passar inúmeras dificuldades, ter de batalhar muito, perder noites de sono, mas nada nos impedirá de buscar nossos sonhos.

REFERÊNCIAS

ABDULLA, W. *Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow*. Matterport Engineering Tech-blog, 2018. Disponível em: <https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46>. Citado na página 28.

ALLIANCE, M. *MIPI Camera Serial Interface 2 (MIPI CSI-2)*. 2017. <https://mipi.org/specifications/csi-2>. Acesso em 15 out. 2017. Citado na página 42.

AMARNI, B. *pi64*. 2018. Disponível em: <https://github.com/bamarni/pi64>. Citado na página 63.

AMAZON. *Haiworld New Version 5*. 2017. <https://images-na.ssl-images-amazon.com/images/I/61Ht8nIZgFL._SL1000_.jpg>. Acesso em 11 set. 2017. Citado na página 40.

ARMITAGE, R. To cctv or not to cctv. *A review of current research into the effectiveness of CCTV systems in reducing crime*, p. 8, 2002. Citado 2 vezes nas páginas 12 e 19.

BAZOTE, M. *Princípios básicos e fundamentais de seguranća patrimonial*. 2012. <http://senhoraseguranca.com.br/wp-content/uploads/2012/03/equipamentos-eletronicos-apostila.pdf>. Acesso em 11 set. 2016. Citado 2 vezes nas páginas 19 e 20.

BHAT, A. *Deep Learning on smartphones (or at the "edge")*. 2017. <https://medium.com/deepindepth/deep-learning-at-the-edge-or-on-mobile-63e3527173bc. Acesso em 16 jun. 2018. Citado na página 26.

BLAKE, A. *THE MYSTERY OF NIGHT VISION*. 2015. <https://irilluminators.wordpress.com/2015/01/21/the-mystery-of-night-vision/>. Acesso em 09 set. 2017. Citado na página 40.

BRADSKI, G.; KAEHLER, A. *Learning OpenCV: Computer vision with the OpenCV library*. Sebastopol, CA: "O'Reilly Media, Inc.", 2008. Citado 4 vezes nas páginas 44, 45, 46 e 47.

BRAGA, N. C. *Como funciona a câmera fotográfica digital*. 2014. <http://www.newtoncbraga.com.br/index.php/como-funciona/3723-art515>. Acesso em 11 set. 2016. Citado 2 vezes nas páginas 35 e 36.

BRUNO, F. Contramanual para câmeras inteligentes: vigilância, tecnologia e percepção. *Galáxia*, Pontifícia Universidade Católica de São Paulo, n. 24, 2012. Citado 6 vezes nas páginas 12, 15, 19, 21, 22 e 23.

CALTECH. *Caltech Pedestrian Detection Benchmark*. 2009. <http://www.vision.caltech.edu/ImageDatasets/CaltechPedestrians/ >. Acesso em 20 nov. 2017. Citado 3 vezes nas páginas 18, 57 e 63.

CASTRO, M. L. A.; CASTRO, R. de O. Autômatos celulares: implementações de von neumann, conway e wolfram. *Revista de Ciências Exatas e Tecnologia*, v. 3, n. 3, p. 89–106, 2015. Citado na página 30.

CHM, C. H. M. *x86 Architecture*. 2018. <http://www.computerhistory.org/revolution/digital-logic/12/330>. Acesso em 08 mai. 2018. Citado na página 63.

CHUEKE, G. V.; AMATUCCI, M. O que é bibliometria? uma introdução ao fórum. *Internext*, v. 10, n. 2, p. 1–5, 2015. Citado na página 18.

CIPOLI, P. *Saiba tudo sobre o Raspberry Pi 3 e o que ele representa para o mercado*. 2016. <https://canaltech.com.br/hardware/saiba-tudo-sobre-o-raspberry-pi-3-59065/>. Acesso em 09 set. 2017. Citado na página 33.

CIRCUIT. *Raspberry Pi circuit note*. 2016. <http://www.jameco.com/Jameco/workshop/circuitnotes/raspberry_pi_circuit_note_fig2a.jpg>. Acesso em 09 set. 2016. Citado na página 31.

CLASSNOTES, O. *Describe the fundamental steps of digital image processing with a neat block diagram*. 2013. <http://www.onlineclassnotes.com/2011/10/describe-fundamental-steps-of-digital.html>. Acesso em 15 out. 2017. Citado 2 vezes nas páginas 47 e 48.

CNPQ. *Periodicos Capes*. 2016. <http://www.periodicos.capes.gov.br/. Acesso em 16 jun. 2016. Citado na página 18.

CORDOBA., G. A. of Artificial Vision"(A.V.A) of the University of. *RaspiCam RaspiCam: C++ API for using Raspberry camera with/without OpenCV*. 2017. <http://www.uco.es/investiga/grupos/ava/node/40 >. Acesso em 10 set. 2017. Citado 3 vezes nas páginas 44, 59 e 62.

CORSO, D. A.; ALMEIDA, R. H. D. Extração de características baseadas em forma para o reconhecimento de padrões em um sistema de visão computacional. *Encontro de Engenharia e Tecnologia em Computação*, 2007. Citado na página 42.

COUTINHO, M. P. Sistema de monitoramento residencial. 2016. Citado na página 39.

CPLUSPLUS. *CPlusplus History of C++*. 2017. <http://www.cplusplus.com/info/history/>. Acesso em 09 set. 2017. Citado na página 35.

CYGANEK, B. *Object Detection and Recognition in Digital Images: Theory and Practice*. Hoboken, NJ: "John Wiley & Sons, Ltd", 2013. Citado na página 48.

DALAL, N.; TRIGGS, B. Inria person dataset. *Online: http://pascal. inrialpes. fr/data/human*, 2005. Citado 5 vezes nas páginas 12, 18, 23, 24 e 57.

DENG, L. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, IEEE, v. 29, n. 6, p. 141–142, 2012. Citado na página 53.

DIGIFORT. *Sistemas de vigilância Digifort*. 2017. <http://www.digifort.com.br/digifort.php>. Acesso em 14 out. 2017. Citado na página 15.

DOLLÁR, P. et al. Fast feature pyramids for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, v. 36, n. 8, p. 1532–1545, 2014. Citado 2 vezes nas páginas 24 e 25.

DOLLÁR, P.; BELONGIE, S. J.; PERONA, P. The fastest pedestrian detector in the west. In: CITESEER. *Bmvc*. [S.l.], 2010. v. 2, n. 3, p. 7. Citado 6 vezes nas páginas 12, 14, 24, 63, 81 e 107.

ELETRICA, S. *Sistema de segurança inteligentes: Tranquilidade em tempo real*. 2014. <http://www.sabereletrica.com.br/sistemas-de-seguranca>. Acesso em 09 set. 2016. Citado 3 vezes nas páginas 19, 20 e 21.

ELINUX. *RPi Hardware*. 2016. <http://elinux.org/RPi_Hardware>. Acesso em 17 set. 2016. Citado 2 vezes nas páginas 32 e 80.

ERICKSON, B. J. et al. Toolkits and libraries for deep learning. *Journal of digital imaging*, Springer, v. 30, n. 4, p. 400–405, 2017. Citado na página 66.

HE, K. et al. Mask r-cnn. In: IEEE. *Computer Vision (ICCV), 2017 IEEE International Conference on*. [S.l.], 2017. p. 2980–2988. Citado 3 vezes nas páginas 12, 28 e 29.

HEARST, M. A. et al. Support vector machines. *IEEE Intelligent Systems and their applications*, IEEE, v. 13, n. 4, p. 18–28, 1998. Citado na página 23.

HOANG, V.-D.; LE, M.-H.; JO, K.-H. Hybrid cascade boosting machine using variant scale blocks based hog features for pedestrian detection. *Neurocomputing*, Elsevier, v. 135, p. 357–366, 2014. Citado na página 24.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks*, Elsevier, v. 2, n. 5, p. 359–366, 1989. Citado na página 53.

HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. Citado 3 vezes nas páginas 12, 25 e 27.

HOWARD, C. J. et al. Suspiciousness perception in dynamic scenes: a comparison of cctv operators and novices. *Frontiers in human neuroscience*, Frontiers Media SA, v. 7, 2013. Citado na página 21.

INTELBRAS. *Manual do usuário SIM Plus*. 2016. <http://www.intelbras.com.br/sites/default/files/downloads/manual_sim_plus_portugues_02-15_site.pdf>. Acesso em 09 set. 2016. Citado na página 15.

JAMECO. *Raspberry Pi 3 Model B*. 2017. <http://www.jameco.com/z/83-17300-Raspberry-Pi-Raspberry-Pi-3-Model-B-BCM2837_2237790.html>. Acesso em 09 set. 2017. Citado na página 33.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2012. p. 1097–1105. Citado na página 25.

KYRKOU, C. et al. Optimizing the detection performance of smart camera networks through a probabilistic image-based model. *IEEE Transactions on Circuits and Systems for Video Technology*, IEEE, 2017. Citado na página 15.

LEE, H. S. et al. Can deep neural networks match the related objects?: A survey on imagenet-trained classification models. *arXiv preprint arXiv:1709.03806*, 2017. Citado na página 28.

LIN, T.-Y. et al. Microsoft coco: Common objects in context. In: SPRINGER. *European conference on computer vision*. [S.l.], 2014. p. 740–755. Citado na página 28.

LOGITECH. *HD Webcam C270*. 2016. <http://www.logitech.com/pt-br/product/hd-webcam-c270?crid=34>. Acesso em 15 set. 2016. Citado 2 vezes nas páginas 38 e 39.

LU, H.; SETIONO, R.; LIU, H. Effective data mining using neural networks. *IEEE transactions on knowledge and data engineering*, IEEE, v. 8, n. 6, p. 957–961, 1996. Citado na página 53.

MASOOD, A. Stereo pi: Portable digital stereo camera. 2016. Citado na página 36.

MATHWORKS. *MathWorks - Company Overview*. 2018. <https://www.mathworks.com/company/aboutus.html >. Acesso em 24 mar. 2018. Citado 2 vezes nas páginas 49 e 50.

MATLAB. *Neural Network Toolbox*. 2018. <https://www.mathworks.com/help/nnet/index.html?s_tid=srchtitle >. Acesso em 24 mar. 2018. Citado 3 vezes nas páginas 50, 51 e 52.

MENESES, P. R.; ALMEIDA, T. d. Introdução ao processamento de imagens de sensoriamento remoto. *Embrapa Cerrados-Livros técnicos (INFOTECA-E)*, Brasília, DF: UnB, 2012., 2012. Citado na página 35.

MONTANEZ, E. R. *Deep Learning: Cuando las IAs se comportan como niños de cinco años*. 2016. <https://ingenieriada.wordpress.com/2016/11/11/deep-learning-cuando-las-ias-se-comportan-como-ninos-de-cinco-anos/>. Acesso em 16 out. 2017. Citado na página 49.

MORELLAS, V.; PAVLIDIS, I.; TSIAMYRTZIS, P. Deter: Detection of events for threat evaluation and recognition. *Machine Vision and Applications*, Springer, v. 15, n. 1, p. 29–45, 2003. Citado na página 22.

NIELSEN, M. A. *Neural networks and deep learning*. [S.l.]: Determination Press, 2015. Citado 5 vezes nas páginas 53, 54, 55, 64 e 65.

NVIDIA. *NVIDIA Product Families, Graphics Cards, and Technologies*. 2017. Disponível em: <https://www.nvidia.com/page/products.html>. Citado na página 65.

OCVTEAM. *OpenCV 3.3 release notes*. 2017. <https://opencv.org/opencv-3-3.html>. Acesso em 13 out. 2017. Citado 3 vezes nas páginas 23, 57 e 62.

OLEARI, F.; RIZZINI, D. L.; CASELLI, S. A low-cost stereo system for 3d object recognition. In: IEEE. *Intelligent Computer Communication and Processing (ICCP), 2013 IEEE International Conference on*. [S.l.], 2013. p. 127–132. Citado na página 38.

OPPENHEIM, A. V. *Sinais e sistemas*. [S.l.]: Prentice-Hall, 2010. Citado na página 54.

OVT. *Omni BSI*. 2011. <http://www.ovt.com/technologies/technology.php?TID=2>. Acesso em 13 set. 2016. Citado 2 vezes nas páginas 37 e 38.

PAN, S. J.; YANG, Q. et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, Institute of Electrical and Electronics Engineers, Inc., 345 E. 47 th St. NY NY 10017-2394 USA, v. 22, n. 10, p. 1345–1359, 2010. Citado na página 65.

PHPHR. *Difference between Deep Learning and Machine Learning*. 2018. <linksoftvn.com/difference-between-deep-learning-and-machine-learning/ >. Acesso em 24 mar. 2018. Citado na página 51.

PI, R. *Raspberry Pi Model B*. 2015. Citado 2 vezes nas páginas 30 e 33.

QUEIROZ, J. E. R. de; GOMES, H. M. Introdução ao processamento digital de imagens. *RITA*, v. 13, n. 2, p. 11–42, 2006. Citado na página 41.

RAHMAN, M. A.; WANG, Y. Optimizing intersection-over-union in deep neural networks for image segmentation. In: SPRINGER. *International Symposium on Visual Computing*. [S.l.], 2016. p. 234–244. Citado na página 56.

RASPBERRY. *Raspberry Pi - About us*. 2016. <https://www.raspberrypi.org/about/>. Acesso em 09 set. 2016. Citado 3 vezes nas páginas 29, 30 e 34.

RASPIAN. *Raspian - About Raspian*. 2016. <https://www.raspbian.org/RaspbianAbout>. Acesso em 19 set. 2016. Citado na página 34.

RIEBEEK, H. *Paleoclimatology*. 2005. <https://earthobservatory.nasa.gov/Features/Paleoclimatology_IceCores/. Acesso em 16 jun. 2018. Citado na página 27.

ROE, B. P. et al. Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, Elsevier, v. 543, n. 2-3, p. 577–584, 2005. Citado na página 25.

ROSEBROCK, A. *Practical Python and OpenCV*. Miami, FL: pyimageseach, 2016. Citado 6 vezes nas páginas 33, 41, 47, 62, 75 e 76.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citado na página 52.

RSCOMPONENTS. *Raspberry Pi - Raspberry Pi 3 Model B*. 2017. <http://docs-europe.electrocomponents.com/webdocs/14ba/0900766b814ba5fd.pdf>. Acesso em 09 set. 2017. Citado na página 32.

SCHIMEK, M. H. et al. Video for linux two api specification. *History*, v. 6, p. 11, 1999. Citado 2 vezes nas páginas 43 e 59.

SESP. *Secretária da Seguranća Pública e Administraćão Penitenciária - Coordenadoria de Análise e Planejamento Estratégico - Relatório Estatístico Criminal*. 2016. <http://www.seguranca.pr.gov.br/arquivos/File/Relatorio_Estatistico_1Trimestre_2016.pdf>. Acesso em 09 set. 2016. Citado 2 vezes nas páginas 15 e 16.

SPARKFUN. *Datasheet OV5647*. 2009. <http://cdn.sparkfun.com/datasheets/Dev/RaspberryPi/ov5647_full.pdf>. Acesso em 13 set. 2016. Citado na página 36.

URZEDO, T. *Detecção de Pedestres usando o OpenCV*. 2017. Disponível em: <http://www.decom.ufop.br/imobilis/deteccao-de-pedestres-usando-o-opencv/>. Citado na página 63.

VANDEVENNE, L. *Image Filtering*. 2004. <http://lodev.org/cgtutor/filtering.html>. Acesso em 15 out. 2017. Citado 2 vezes nas páginas 46 e 47.

VILLAMIZAR, M. et al. Combining color-based invariant gradient detector with hog descriptors for robust image detection in scenes under cast shadows. In: IEEE. *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. [S.l.], 2009. p. 1997–2002. Citado na página 61.

WERBOS, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, IEEE, v. 78, n. 10, p. 1550–1560, 1990. Citado na página 53.

WINSCH, G. A.; SANTOS, P. H. S. *Desenvolvimento de um escâner tridimensional baseado em sistema ótico e feixe de laser*. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2013. Citado 2 vezes nas páginas 37 e 38.

ZHANG, S. et al. A camera network tracking (camnet) dataset and performance baseline. In: IEEE. *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*. [S.l.], 2015. p. 365–372. Citado na página 15.

APÊNDICE A – CÓDIGO DE GRAVAÇÃO DO VÍDEO

```
# -*- coding: utf-8 -*-
# importa as bibliotecas necessarias
importpicamera
from time import sleep

# inicializacao da camera
camera = picamera.PiCamera()
camera.start_preview()

# define o nome do arquivo de saida
camera.start_recording("recorded.h264")

#grava por 5 segundos
camera.wait_recording(5)

#finaliza a gravacao e a exibicao
camera.stop_recording()
camera.stop_preview()
```

APÊNDICE B – CÓDIGO DE DETECÇÃO

```
# importa as bibliotecas necessarias
from imutils.object_detection import non_max_suppression
import imutils
import numpy as np
import time
import cv2

# inicializa a variavel do amostrador
i = 0
# inicializa o descritor de histogramas orientados a gradiente
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
cap= cv2.VideoCapture('recorded.h264')
# retira os quadros do video ja gravado
while(cap.isOpened()):
    # armazena o quadro do video em uma variavel matricial
    ret, frame = cap.read()

    # redimensionamento da imagem
    image = frame
    image = imutils.resize(image, width=min(400, image.shape[1]))

    # detectcao com o uso dos padroes
    (rects, weights) = hog.detectMultiScale(image, winStride=(4, 4),
    padding=(8, 8), scale=1.05)

    # aplica um supressor de sobreposicao de
    # deteccao para reduzir falsos verdadeiros
    rects = np.array([[x, y, x + w, y + h] for (x, y, w, h) in rects])
    pick = non_max_suppression(rects, probs=None, overlapThresh=0.65)
```

```
# desenha as caixas de deteccao
for (xA, yA, xB, yB) in pick:
    cv2.rectangle(image, (xA, yA), (xB, yB), (0, 255, 0), 2)

# janela de saida dos quadros
cv2.imshow("After NMS", image)
key = cv2.waitKey(3)

# sair do programa ao apertar a tecla q
if key == ord("q"):
break
```

APÊNDICE C – CÓDIGO DE GRAVAÇAO DA *WEBCAM*

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

fc = 0

fourcc = cv2.cv.CV_FOURCC(*'XVID')
out = cv2.VideoWriter("drop.avi", fourcc, 30.0, (640,480))

# Armazena 120 quadros
while(cap.isOpened()) and fc < 120:
ret, frame = cap.read()
if ret==True:

# Grava os quadros
out.write(frame)
fc += 1
cv2.imshow('frame',frame)
cv2.waitKey(30)


# Desaloca recursos
cap.release()
out.release()
cv2.destroyAllWindows()
```

APÊNDICE D – CÓDIGO PARA TESTE DA GPIO

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <errno.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>

static volatile uint32_t *gpio;

int main(intargc, char **argv)
{
        intfd;

        // Obtain handle to physical memory
        if ((fd = open ("/dev/mem", O_RDWR | O_SYNC) ) < 0) {
                printf("Unable to open /dev/mem: %s\n", strerror(errno));
                return -1;
        }

        // map a page of memory to gpio at offset
        // 0x20200000 which is where GPIO goodnessstarts
        gpio = (uint32_t *)mmap(0,
                                getpagesize(),
                                PROT_READ|PROT_WRITE,
                                MAP_SHARED,
                                fd,
                                0x20200000);
```

```
        if (gpio == 0){
                printf("Mmap failed: %s\n", strerror(errno));
                return -1;
        }
         // set gpio17 as an output
        // increment the pointer to 0x20200004
        // set the value through a little bit twiddling where we
        //only modify the bits 21-23 in the register
        *(gpio + 1) = (*(gpio + 1) & ~(7 << 21)) | (1 << 21);

  // Now do a loop as fast as possible in assembler
                __asm__("loop:\n\t"

            "mov r1,#1\n\t"                         //PIN ON
            "lsl r1,#17\n\t"
            "strr1,[%0,#28]\n\t"
            "mov r1,#1\n\t"                         //PIN OFF
            "lsl r1,#17\n\t"
            "strr1,[%0,#40]\n\t"
             "b loop\n\t"
            "r"((uint32_t)gpio)
            "r0", "r1"
            );
}
```

APÊNDICE E – CÓDIGO DE TESTE DA NOVA CÂMERA

```
#include <string>
#include <iostream>
#include <raspicam/raspicam_cv.h>
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

int main ( int argc,char **argv ) {

    raspicam::RaspiCam_Cv Camera;
    Mat image;

    Camera.set( CV_CAP_PROP_FORMAT, CV_8UC1 );
    if (!Camera.open())
    {
        {cerr<<"Falha ao abrir a camera"<<endl;return -1;}
    }

    Camera.grab();
    Camera.retrieve ( image);
    cout<<"Captura feita."<<endl;
    Camera.release();

    imshow("Imagem da camera",image);
    waitKey(27);

    return 0;
}
```

APÊNDICE F – CÓDIGO DE DETECÇÃO MIGRADO - PRODUTOR

```
#include <string>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <raspicam/raspicam_cv.h>
#include <opencv2/opencv.hpp>
using namespace cv;
using namespace std;

int main ( int argc,char **argv ) {

    raspicam::RaspiCam_Cv Camera;
    Mat image, lux_frame;
    double old_lux;

    HOGDescriptor hog;
    hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());

    Camera.set( CV_CAP_PROP_FORMAT, CV_8UC1 );
    Camera.set(CV_CAP_PROP_FRAME_WIDTH,480);
    Camera.set(CV_CAP_PROP_FRAME_HEIGHT,360);

  ofstream myfile ("log.txt");

    if (!Camera.open())
    {
        {cerr<<"Falha ao abrir a camera"<<endl;return -1;}
    }
    for(;;)
    {
```

```
        Camera.grab();
        Camera.retrieve (image);
        image.copyTo(lux_frame);

        Scalar m = mean(lux_frame);

    printf("%f\n", m[0]);

    ofstream myfile ("log.txt", std::ios_base::app);

    if (myfile.is_open())
    {
      myfile.fill('0');
      myfile << std::setw(7) << m[0];
      myfile << '\n';
      myfile.close();
    }

        vector<Rect> detected;
        hog.detectMultiScale(image, detected, 0, Size(8, 8), Size(8,8), 1.2);

        groupRectangles(detected, 1, 0.33);

        for(int i=0; i < detected.size(); i++)
        {
            rectangle(image, detected[i], cv::Scalar(0,255,0), 2);
        }
        imshow("Imagem da camera",image);
        old_lux = m[0];
        waitKey(27);
    }
    return 0;
}
```

APÊNDICE G – CÓDIGO DE DETECÇÃO COM AMOSTRA - C++

```
#include <string>
#include <iostream>
#include <fstream>
#include <iomanip>
//#include <raspicam/raspicam_cv.h>
#include <opencv2/opencv.hpp>
#include <time.h>

using namespace cv;
using namespace std;


int main ( int argc,char **argv ) {

    //raspicam::RaspiCam_Cv Camera;
    Mat image, lux_frame, frame;
    //double old_lux;
  int frames = 0;
  float total = 0, start_time, tdiff;
  struct timespec gettime_now;
  VideoCapture cap;


    HOGDescriptor hog;
    hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());

    //Camera.set( CV_CAP_PROP_FORMAT, CV_8UC1 );
    //Camera.set(CV_CAP_PROP_FRAME_WIDTH,480);
  //Camera.set(CV_CAP_PROP_FRAME_HEIGHT,360);
```

```
//ofstream myfile ("log.txt");

  //if (!Camera.open())
 // {
  //    {cerr<<"Falha ao abrir a camera"<<endl;return -1;}
 // }

cap.open("sample.h264");

clock_gettime(CLOCK_REALTIME, &gettime_now);


  for(;;)
  {
      //Camera.grab();
      //Camera.retrieve (image);

      //clock_t begin_time = clock();
      cap >> frame;
      start_time = gettime_now.tv_nsec;
      resize(frame, image, Size(360, 240));

      image.copyTo(lux_frame);

      Scalar m = mean(lux_frame);

  printf("%f\n", m[0]);

  ofstream myfile ("log.txt", std::ios_base::app);

  if (myfile.is_open())
  {
    myfile.fill('0');
```

```
      myfile << std::setw(7) << m[0];
      myfile << '\n';
      myfile.close();
    }

        vector<Rect> detected;
        hog.detectMultiScale(image, detected, 0, Size(8, 8), Size(8,8), 1.06);

       //groupRectangles(detected, 1, 0.33);

        printf ("%d\n", frames);

        for(int i=0; i < detected.size(); i++)
        {
            rectangle(image, detected[i], cv::Scalar(0,255,0), 2);
        }

        imshow("Imagem da camera",image);
        // old_lux = m[0];
        waitKey(1);
        clock_gettime(CLOCK_REALTIME, &gettime_now);
        tdiff = gettime_now.tv_nsec - start_time;
        if (tdiff < 0)
      tdiff += 1000000000;

    total = total + tdiff/1000000000;
        printf("Avg FPS: %f\n", frames/total);
        frames++;
    }
    return 0;
}
```

APÊNDICE H – CÓDIGO DE CONSUMO E ADEQUAÇÃO DO BANCO DE MARCADORES DAS IMAGENS - MATLAB

```
numfotos = 614;

for k = 1:numfotos

    fname = ['C:\\inria_train_gt\\inria_train_gt (' num2str(k) ').txt'];
    data = PASreadrecord(fname);
    Objboxes = [];

    for n = 1:length(data.objects)
            Objboxes(:, n) = data.objects(n).bbox;
    end

    objsize = size(Objboxes);

            for n = 1:objsize(2)
                Objboxes(3, n) =  Objboxes(3, n) - Objboxes(1, n);
                Objboxes(4, n) =  Objboxes(4, n) - Objboxes(2, n);
            end

    Objboxes = Objboxes';
    fullData{k} = Objboxes;
    fullName{k} =  ['C:\\inria_train\\inria_train (' num2str(k) ').png'];

end
fullName = fullName';
fullData = fullData';

T = table(fullName, fullData);
save pedestrian_train_inria.mat T
```

APÊNDICE I – CÓDIGO DE TREINAMENTO COM REDES NEURAIS CONVOLUCIONAIS POR REGIÃO - MATLAB

```
data_train = load('pedestrian_train_inria.mat');
data_test = load('pedestrian_test_inria.mat');
personDataset_train = data_train.T;
personDataset_test = data_test.T;

doTraining = true;
doEval = true;

personDataset_train(1:4,:)

testData = personDataset_test(1:end,:);
trainingData = personDataset_train(1:end,:);

%R-CNN Concept
inputLayer = imageInputLayer([64 64 3]);

middleLayers = [

    convolution2dLayer(7, 80, 'Stride', 1)
    reluLayer()
    maxPooling2dLayer(3, 'Stride',2)
    convolution2dLayer(3, 40, 'Stride', 1)
    reluLayer()
    maxPooling2dLayer(2, 'Stride',2)
    convolution2dLayer(3, 20, 'Stride', 1)
    reluLayer()
    maxPooling2dLayer(2, 'Stride',2)

    ];
```

```
finalLayers = [

    fullyConnectedLayer(64)

    reluLayer()

    fullyConnectedLayer(width(personDataset_train))

    softmaxLayer()
    classificationLayer()
];
layers = [
    inputLayer
    middleLayers
    finalLayers
    ]

% Options for step 1.
optionsStage1 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'InitialLearnRate', 1e-5, ...
    'CheckpointPath', tempdir);

% Options for step 2.
optionsStage2 = trainingOptions('sgdm', ...
    'MaxEpochs', 15, ...
    'InitialLearnRate', 1e-5, ...
    'CheckpointPath', tempdir);

% Options for step 3.
optionsStage3 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'InitialLearnRate', 1e-6, ...
```

```
    'CheckpointPath', tempdir);

% Options for step 4
optionsStage4 = trainingOptions('sgdm', ...
    'MaxEpochs', 15, ...
    'InitialLearnRate', 1e-6, ...
    'CheckpointPath', tempdir);

options = [
    optionsStage1
    optionsStage2
    optionsStage3
    optionsStage4
    ];

if doTraining

    rng(0);
    detector = trainFasterRCNNObjectDetector(trainingData, layers, options, ...
        'NegativeOverlapRange', [0 0.3], ...
        'PositiveOverlapRange', [0.6 1], ...
        'BoxPyramidScale', 1.2);
end

% Ex.1
I = imread('C:\p1.jpg');
% Usar amostras.
[bboxes, scores] = detect(detector, I);

 for k = 1:length(scores)

     if scores(k) < 0.7
         scores(k) = 0;
```

```
        bboxes(k, :) = 0;
    end

end
I = insertObjectAnnotation(I, 'rectangle', bboxes, scores);
figure
imshow(I)

% Ex.2
I = imread('C:\p2.jpg');
% Usar amostras.
[bboxes, scores] = detect(detector, I);

for k = 1:length(scores)
    if scores(k) < 0.7
        scores(k) = 0;
        bboxes(k, :) = 0;
    end
end
I = insertObjectAnnotation(I, 'rectangle', bboxes, scores);
figure
imshow(I)

% Ex.3
I = imread('C:\p3.jpg');
% Usar amostras.
[bboxes, scores] = detect(detector, I);

for k = 1:length(scores)
    if scores(k) < 0.7
        scores(k) = 0;
        bboxes(k, :) = 0;
    end
```

```
 end
 I = insertObjectAnnotation(I, 'rectangle', bboxes, scores);
 figure
 imshow(I)

if doEval

    resultsStruct = struct([]);
    for i = 1:height(testData)
        I = imread(testData.fullName{i});
        [bboxes, scores, labels] = detect(detector, I);
        resultsStruct(i).Boxes = bboxes;
        resultsStruct(i).Scores = scores;
        resultsStruct(i).Labels = labels;
    end

    results = struct2table(resultsStruct);
end

expectedResults = testData(:, 2:end);

% Avaliador
[ap, recall, precision] = ...
evaluateDetectionPrecision(results, expectedResults, 0.3);
% Plotagem
figure
plot(recall, precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.3f', ap))

save detector_v3_inria.mat detector
```

APÊNDICE J – CÓDIGO DE TESTE DO ALGORITMO DE DETECÇÃO DO FPDW (DOLLÁR; BELONGIE; PERONA, 2010) - C++

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include "fpdw_detector.h"
#include <time.h>

int main(int argc, char **argv)
{
    //cv::Mat image = cv::imread(argv[2]);

    cv::VideoCapture cap;
    cv::Mat frame, image;
    int frames = 0;
    float total = 0, start_time, tdiff;
  struct timespec gettime_now;

    cap.open("vtest.avi");

  clock_gettime(CLOCK_REALTIME, &gettime_now);
    fpdw::detector::FPDWDetector detector(argv[1], 50.);


    for(;;){

        cap >> image;
        start_time = gettime_now.tv_nsec;
        //cv::resize(frame, image, cv::Size(360, 240));

        detector.process(image);
        std::vector<cv::Rect> rect = detector.getBBoxes();
```

```
        for(const auto &i : rect)
        {
        cv::rectangle(image, i, cv::Scalar(255, 0, 0), 2);
        }

        //cv::resize(image, image, cv::Size(image.cols*0.5, image.rows*0.5));
        cv::imshow("image", image);
        cv::waitKey(1);

        clock_gettime(CLOCK_REALTIME, &gettime_now);
        tdiff = gettime_now.tv_nsec - start_time;
        if (tdiff < 0)
      tdiff += 1000000000;

    total = total + tdiff/1000000000;
        printf("Avg FPS: %f\n", frames/total);
        frames++;

    }
    return 0;
}
```

APÊNDICE K – CÓDIGO DE APLICAÇÃO DO MODELO MOBILENETS CALIBRADO PARA PEDESTRES - PYTHON

```
import numpy as np
import cv2
import sys

COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# load
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

for x in range(1, 289):
    input_str = "../Pictures/B_inria_test/inria_test (" + str(x) + ").png"
    out_str = "../Pictures/inria_test_r/out (" + str(x) + ").png"
    #f_str = "outer/mob (" + str(x) + ").txt"
    f_str = "outer/score (" + str(x) + ").txt"
    file = open(f_str,"w")
    print input_str
    image = cv2.imread(input_str)
    (h, w) = image.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(image,
    (300, 300)), 0.007843, (300, 300), 127.5)


    print("[INFO] computing object detections...")
    net.setInput(blob)
    detections = net.forward()


    for i in np.arange(0, detections.shape[2]):
```

```
  confidence = detections[0, 0, i, 2]


  if confidence > args["confidence"]:

    idx = int(detections[0, 0, i, 1])
    box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
    (startX, startY, endX, endY) = box.astype("int")


    label = "{}: {:.2f}%".format("person", confidence * 100)
    print("[INFO] {}".format(label))
    cv2.rectangle(image, (startX, startY), (endX, endY),
      COLORS[idx], 2)
    y = startY - 15 if startY - 15 > 15 else startY + 15
    cv2.putText(image, label, (startX, y),
      cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)

        if idx == 15:
            file.write(str(confidence) + "\n")

cv2.imwrite(out_str, image)
            #file.write(str(startX) + " " + str(startY) +
            " " + str(endX - startX) + " " + str(endY - startY) +"\n")
file.close()
# show the output image
#cv2.waitKey(0)
```

APÊNDICE L – CÓDIGO DE APLICAÇÃO DO MODELO MOBILENETS CALIBRADO PARA PEDESTRES - C++

```
#include <opencv2/dnn.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/core/utils/trace.hpp>
using namespace cv;
using namespace cv::dnn;
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <raspicam/raspicam_cv.h>
#include <chrono>
#include <thread>
#include <time.h>

using namespace std;


int main(int argc, char **argv)
{
    CV_TRACE_FUNCTION();
    String modelTxt = "MobileNetSSD_deploy.prototxt";
    String modelBin = "MobileNetSSD_deploy.caffemodel";
    raspicam::RaspiCam_Cv Camera;
    Net net = dnn::readNetFromCaffe(modelTxt, modelBin);

    if (net.empty())
    {
        std::cerr
```

```
        << "Can't load network by using the following files: "
        << std::endl;

        std::cerr << "prototxt:   " << modelTxt << std::endl;
        std::cerr << "caffemodel: " << modelBin << std::endl;
        exit(-1);
    }

    Mat img;
    Mat img2;
    int frames = 0;
struct timespec gettime_now;
float total = 0, start_time, tdiff;

    Camera.set( CV_CAP_PROP_FORMAT, CV_8UC3 );
    Camera.set(CV_CAP_PROP_FRAME_WIDTH,480);
Camera.set(CV_CAP_PROP_FRAME_HEIGHT,360);
std::this_thread::sleep_for(std::chrono::milliseconds(3000));


    if (!Camera.open())
    {
        {cerr<<"Falha ao abrir a camera"<<endl;return -1;}
    }
    clock_gettime(CLOCK_REALTIME, &gettime_now);

for(;;){

    Camera.grab();
    Camera.retrieve(img);
    start_time = gettime_now.tv_nsec;
```

```
if (img.data){
  Mat inputBlob = blobFromImage
  (img, 0.007843, Size(300,300), Scalar(127.5, 127.5, 127.5), false);

  net.setInput(inputBlob, "data");
  Mat detection = net.forward("detection_out");
  Mat detectionMat
  (detection.size[2], detection.size[3], CV_8U, detection.ptr<float>());

  ostringstream ss;
  float confidenceThreshold = 0.2;
  for (int i = 0; i < detectionMat.rows; i++)
  {
    float confidence = detectionMat.at<float>(i, 2);

  if (confidence > confidenceThreshold)
  {
  int idx = static_cast<int>(detectionMat.at<float>(i, 1));
  int xLeftBottom = static_cast<int>(detectionMat.at<float>(i, 3) * img.cols);
  int yLeftBottom = static_cast<int>(detectionMat.at<float>(i, 4) * img.rows);
  int xRightTop = static_cast<int>(detectionMat.at<float>(i, 5) * img.cols);
  int yRightTop = static_cast<int>(detectionMat.at<float>(i, 6) * img.rows);

      Rect object((int)xLeftBottom, (int)yLeftBottom,
            (int)(xRightTop - xLeftBottom),
            (int)(yRightTop - yLeftBottom));

      rectangle(img, object, Scalar(0, 255, 0), 2);
    }
  }
}
//resize(img, img2, Size(300,300));
clock_gettime(CLOCK_REALTIME, &gettime_now);
```

```
        tdiff = gettime_now.tv_nsec - start_time;
        if (tdiff < 0)
      tdiff += 1000000000;

    total = total + tdiff/1000000000;
        printf("Avg FPS: %f\n", 1/(frames/total));
        frames++;
    imshow("detections", img);
    waitKey(1);
  }
    return 0;
}
```