UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Rastreamento de Múltiplos Objetos Utilizando Modelos de Aprendizado Profundo em Hardware Limitado

Trabalho de Conclusão de Curso

Arianne Santos da Macena

São Cristóvão – Sergipe

2021

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Arianne Santos da Macena

Rastreamento de Múltiplos Objetos Utilizando Modelos de Aprendizado Profundo em Hardware Limitado

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): Prof. Dr. Leonardo Nogueira Matos
Coorientador(a): Me. Thiago Dias Bispo

São Cristóvão – Sergipe

2021

Lista de ilustrações

Lista de tabelas

Este trabalho é dedicado àquelas pessoas que nos fazem continuar,
mesmo quando queremos desistir.

waves crashing
on distant shores of time

Resumo

É notável o avanço recente da Inteligência Artificial na vida cotidiana das pessoas que, de uma forma ou de outra, usam soluções de tecnologia em suas atividades. Em grande parte este avanço se deve ao emprego de Redes Neurais Convolucionais, que são particularmente úteis na solução de problemas de localização, detecção e classificação de imagens. Estas redes também são usadas no rastreamento de múltiplos objetos. Elas podem localizar e classificar um objeto, mantendo sua identidade única ao longo do tempo. Esta é uma das razões que as tornam atraentes em aplicações de computação na borda, tendo em vista o potencial emprego em áreas como vigilância eletrônica, controle de tráfego, contagem de pedestres, dentre outras. Por outro lado, as arquiteturas consideradas estado-da-arte demandam grande capacidade computacional, em termos de processamento, consumo de memória e, consequentemente, energia. Essas exigências dificultam o emprego de modelos complexos em equipamentos com restrições de recursos computacionais, como Raspberry Pi. Apesar de ser possível realizar tarefas mais complexas do que em outras plataformas, utilizar Redes Neurais Convolucionais em Raspberry Pi é ainda desafiador. Este trabalho propõe a exploração de uma arquitetura estado-da-arte, a YOLOv4-tiny, simultaneamente ao rastreador de objetos do algoritmo SmartSORT, em Raspberry Pi 4. A solução proposta foi experimentada no rastreamento de múltiplos pedestres do benchmark MOT Challenge 2016 e em um vídeo de um ambiente comercial controlado. Foi obtido uma precisão média de 69% e uma taxa de processamento de quadros de 1,2 FPS.

Palavras-chave: Visão Computacional, Redes Neurais Convolucionais, YOLO, Rastreamento de Múltiplos Objetos, Raspberry Pi.

Abstract

It is worth paying attention to the recent progress of artificial intelligence in the everyday life of people who, one way or another, use technological solutions in their activities. Much of this advancement is due to the use of convolutional neural networks, which are particularly useful in solving problems related to locating, detecting and classifying images. These networks are also used to track multiple objects. They can locate and classify an object while maintaining its unique identity over time. This is one of the reasons that make them attractive for edge computing applications, given the potential employment in areas such as electronic surveillance, traffic control, pedestrian counting, among others. On the other hand, architectures considered state-of-the-art require a lot of computing power in terms of processing, memory consumption, and thus energy. These requirements make it difficult to use complex models in hardware with limited computing resources, such as the Raspberry Pi. While it is possible to perform more complex tasks than on other platforms, using convolutional neural networks on the Raspberry Pi still presents a challenge. This monography proposes to explore the state-of-the-art YOLOv4-tiny architecture, simultaneously with the SmartSORT algorithm object tracker, in the Raspberry Pi 4. The proposed solution was tested in tracking multiple pedestrians from the MOT Challenge 2016 benchmark and in a video showing a controlled commercial environment. An average accuracy of 69% and a frame processing speed of 1.2 fps were achieved.

Keywords: Computer Vision, Convolutional Neural Networks, YOLO, Multiple Object Tracking, Raspberry Pi.

Lista de abreviaturas e siglas

MOT	*Multiple Object Tracking* (do inglês, Rastreamento de Múltiplos Objetos)
MTT	*Multiple Target Tracking* (do inglês, Rastreamento de Multi-alvos)
HoG	*Histogram of Oriented Gradients* (do inglês, Histograma de Gradientes Orientados)
LBP	*Local Binary Pattern* (do inglês, Padrão Binário local)
CNN	*Convolutional Neural Network* (do inglês, Redes Neurais Convolucionais)
MLP	*Multilayer Perceptron* (do inglês, Perceptron Multicamadas)
SOT	*Single Object Tracking* (do inglês, Rastreamento de um Único Objeto)
FC	*Fully Connected* (do inglês, Totalmente Conectada)
YOLO	*You Only Look Once* (do inglês, Você só olha uma vez)
SAT	*Self-AdversarialTraining* (do inglês, Treinamento de Autodefesa)
NAS	*Neural Architecture Search* (do inglês, Pesquisa de Arquitetura Neural)
SBC	*Single Board Computers* (do inglês, Computador de Placa Única)
USB	*Universal Serial Bus* (do inglês, Porta Serial Universal)
HDMI	*High-Definition Multimedia Interface* (do inglês, Interface Condutiva Totalmente Digital de Áudio e Vídeo)
RAM	*Random Access Memory* (do inglês, Memória de Acesso Aleatório)
SoC	*System On Chip* (do inglês, Sistema Em Um Chip)
IoT	*Internet of Things* (do inglês, Internet das coisas)
CUDA	*Compute Unified Device Architecture* (do inglês, Arquitetura de Dispositivo de Computação Unificada)

Lista de símbolos

α	Letra grega Gama
$\in$	Pertence
$\cup$	União
$\sum$	Somatório

Sumário

1 Introdução

Para a área de Visão Computacional, o paradigma de rastreamento de múltiplos objetos (do inglês, *Multiple Object Tracking* ou MOT), ou rastreamento de multi-alvos (do inglês, *Multiple Target Tracking* ou MTT), possui uma grande importância devido à variadade de aplicações que o envolvem. Dentre essas aplicações podemos destacar: o rastreamento de diversos alvos como pedestres, veículos, atletas praticando esporte, grupos de animais (pássaros, morcegos, formigas, peixes, células, etc); desenvolvimento de carros autônomos e vigilância de tráfego; e aplicações na área de robótica e análise de dados biomédicos (LUO et al., 2014; EMAMI et al., 2018; YOON et al., 2019).

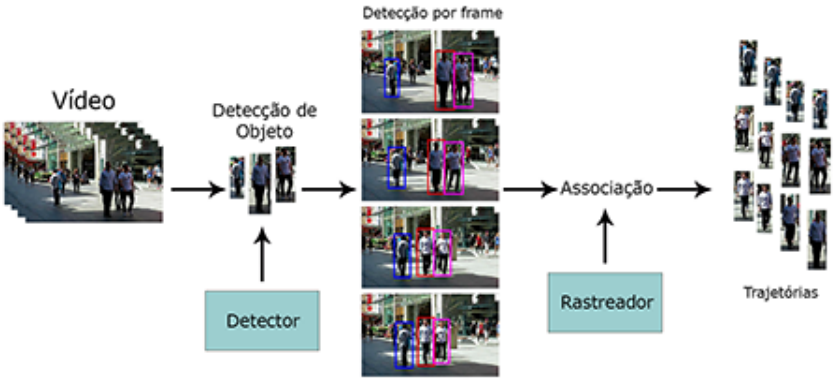
O objetivo principal desse paradigma é localizar e classificar um objeto, enquanto mantém sua identidade única ao longo do tempo. Essa atividade é considerada complexa quando aplicada em vídeos pois requer a localização de objetos em *frames* individuais e a associação deles em múltiplos frames. A possível entrada e saída de um alvo na cena pode gerar situações frequentes de oclusão parcial ou completa (sobreposição do objeto rastreado por um outro objeto), como também interações entre objetos que possuem aparências similares, ocasionando em troca de identidade (LUO et al., 2014; MAHMOUDI et al., 2019; SUN et al., 2019b; CIAPARRONE et al., 2020).

O progresso na área de detecção de objetos tornou popular uma estratégia chamada rastreamento por detecção (no inglês, *tracking-by-detection*). Ela se propõe a resolver o problema de associação das detecções de um objeto sugerindo a descoberta da localização do alvo rastreado, ao usar *bounding boxes* que identificam o alvo individualmente em cada *frame* (CHEN et al., 2017; HE et al., 2019; MAHMOUDI et al., 2019; CIAPARRONE et al., 2020).

Antes de resolver o problema citado anteriormente, faz-se necessário definir uma espécie de matriz de custos, em que cada elemento desta matriz é uma medida de similaridade (ou afinidade). A matriz é gerada por modelos construídos a partir de algoritmos de rastreamento por detecção. Para descrever cada um dos objetos detectados pode-se mencionar diversas das suas

características, como a aparência, movimentação ou a combinação destas e outras particularidades (YOON et al., 2018; MENESES, 2019; SUN et al., 2019b).

Figura 1 – Ilustração da representação da estratéria *tracking-by-detection*.



Fonte: A própria autora.

Diversas técnicas são usadas para a construção de modelos de aparência, desde as características baseadas em histograma, como HOG e texturas, ou também o LBP (características criadas manualmente), e todas elas podem ser consideradas importantes caso nenhum dado de treinamento seja fornecido. Devido ao desenvolvimento em larga escala de banco de dados visuais e o aumento do poder computacional, as Redes Neurais Convolucionais (do inglês, *Convolutional Neural Network* ou CNN) vêm demonstrando ótimo desempenho e sendo usadas para a extração de características visuais por causa da sua capacidade de aprendizado e poder discriminativo (LIU et al., 2018; HE et al., 2019; WANG et al., 2019).

É notável ressaltar que o treinamento das CNNs de forma *online* (que geram trajetórias *frame* por *frame* usando o *frame* anterior e o atual como base) geram um alto custo computacional e de tempo de processamento, e por essa razão muitos algoritmos de rastreamento por detecção se mantêm apenas com características criadas manualmente. Como solução, a CNN pode ser treinada de forma *offline* ou pode-se usar de modelos de CNN pré-treinados (CHEN et al., 2017; LIU et al., 2018).

Modelos pré-treinados são redes que aprenderam a extrair características e classificar objetos em diferentes classes, que geralmente são treinadas usando grandes *datasets* que possuem milhões de imagens. Muitos trabalhos utilizam essa estratégia aplicando modelos como AlexNet, VGG, ResNet, NAS, Inception, MobileNet, YOLO entre outros modelos (e.g *EfficientNet* (CUAN et al., 2018; FANG et al., 2019; HE et al., 2019; HU et al., 2019; MAQSOOD et al., 2019; MENESES, 2019; SINGH et al., 2020; SUN et al., 2019b; TAN; LE, 2019; XU et al., 2019a; WONG et al., 2019).

Uma nova rede neural é capaz de classificar conjuntos de certas categorias, para isso ela deve passar pelo processo de treinamento a partir de pesos inicializados de forma randômica,

porém esse processo seria otimizado se o conhecimento já adquirido por um modelo pré-treinado pudesse ser aproveitado. Isso pode ser feito através de estratégias de aprendizagem de máquina, como a *fine tuning* (ajuste fino), que "congela"certos pesos ou valores dos parâmetros treinados, para que eles não sejam atualizados durante o treinamento da nova rede e não causem perda do "conhecimento"já adquirido.

A eficiência das CNN advém de seus múltiplos níveis de abstração e muitas camadas de processamento, que alcançam alta acurácia e velocidade. A família de modelos YOLO possui alguns dos tipos mais rápidos para detecção de objetos em tempo real, porém o tamanho dos modelos desses algoritmos é volumoso, demandando uma infraestrutura custosa (REDMON et al., 2016; REDMON; FARHADI, 2017; REDMON; FARHADI, 2018). Soluções para sistemas com *hardware* limitado acabam se tornando mais acessíveis por serem mais baratas e utilizarem menos consumo de energia e recursos.

A combinação de modelos de aprendizagem profunda e sistemas com recursos limitados aceleraria a adoção de tecnologias de visão computacional em diversos campos, como na computação em borda (do inglês, *edge computing*), agricultura ou no mercado de massa (VELASCO-MONTERO et al., 2018; KRISTIANI et al., 2020; GONZALEZ-HUITRON et al., 2021). Algumas arquiteturas foram criadas para a utilização de modelos reduzidos e com menos parâmetros, como o Tiny-YOLO , o MobileNet e o SqueezeNet , que favorecem a criação de soluções para sistemas limitados em *hardware*, mantêm a eficiência de algoritmos tradicionais e a acurácia de inferência.(IANDOLA et al., 2016a; HOWARD et al., 2017; REDMON; FARHADI, 2018)

Através dessas informações, pode-se afirmar que o uso de métodos e estruturas que utilizam aprendizagem de máquina voltadas à área de visão computacional oferece oportunidades para explorar diversas combinações de estratégias, dentro domínio do rastreamento de múltiplos objetos. Por exemplo, em Meneses (2019) foi criado um algoritmo chamado SmartSORT que utiliza a YOLO como extrator de características profundas, mas explora um modelo de regressão que calcula os custos de associações entre detecções e trajetórias, utilizando algoritmo de aprendizagem de máquina que demanda menos custo computacional. Comparado com outros sete algoritmos, o SmartSORT conquista ganho de velocidade de até 125% e os resultados obtidos são competitivos. Portanto, se faz evidente a chance de utilização de algoritmos que combinem modelos e técnicas de aprendizagem de máquina para que o seu uso seja possível também em dispositivos de *hardware* limitado.

1.1 Motivação

O interesse para a realização desta pesquisa é a oportunidade de explorar estratégias e modelos atuais que utilizam técnicas de aprendizagem profunda, validando que o aumento da qualidade da associação entre os objetos pode viabilizar o uso de modelos em soluções com

hardware limitado. Sendo o encontro dessas soluções de interesse do meio acadêmico e também da indústria, já que elas possuem baixo custo de aquisição, bom gerenciamento no consumo de energia e poder de processamento relativamente alto. (VELASCO-MONTERO et al., 2018)

Ao utilizar técnicas de aprendizagem profunda em sistemas embarcados é possível construir aplicações que ajudam no desenvolvimento para carros autônomos (BECHTEL et al., 2018), na contagem automática de passageiros de ônibus (MENESES, 2019), na classificação de doenças em plantas (GONZALEZ-HUITRON et al., 2021), no monitoramento de câmeras de segurança (KHUDHAIR; GHANI, 2020), dentre outras aplicações. Portanto, ao explorar estas técnicas em modelos estado da arte, para que sua utilização possa ser feita em ambientes computacionais restritos, sem perca de acurácia, possibilita-se a transformação dessas aplicações em produtos.

1.2 Objetivos

A pesquisa tem como objetivo geral utilizar o algoritmo SmartSORT, proposto por Meneses (2019), explorando o uso de um modelo de extração de características baseado em Aprendizado Profundo (do inglês, *Deep Learning*) em um sistema *hardware* limitado.

1.2.1 Objetivos Específicos

- Realizar uma revisão bibliográfica sobre arquiteturas de redes neurais profundas, utilizadas para extração de características e detecção de objetos que possuam um desempenho siginificativo, acompanhando o estado da arte;
- Criar um dataset para auxiliar no treinamento da solução proposta neste trabalho;
- Explorar o uso da família de redes YOLO no SmartSORT como modelo de extração de características de objetos através do seu detector;
- Embarcar a solução em dispositivo com recursos limitados;
- Comparar resultados obtidos com os do SmartSORT.

1.3 Estrutura do documento

A pesquisa está organizada da seguinte maneira, o Capítulo 2 discute a fundamentação teórica, que descreve conceitos básicos para entendimento deste estudo. Também apresenta tecnologias em *hardware* comumente utilizadas, aplicando redes neurais em funcionalidades do mundo real. O Capítulo 3 descreve estudos que fazem uso de técnicas que envolvem estratégias de rastreamento de múltiplos objetos, e outras pesquisas que desenvolvem essas estratégias para sistemas limitados em recursos. No Capítulo 4 é apresentada a solução proposta para a presente

pesquisa e no Capítulo 5 e Capítulo 6, respectivamente, os resultados da aplicação dessa solução e as conclusões finais.

2

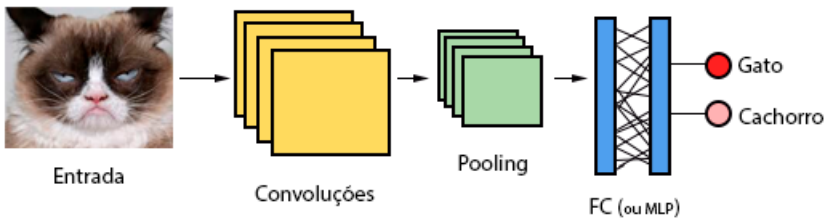
Fundamentação Teórica

2.1 Redes neurais convolucionais

Redes convolucionais, ou redes neurais convolucionais (do inglês, *convolutional neural networks* ou CNN), são um tipo especializado de rede neural profunda, elas possuem este nome pelo emprego que fazem de uma operação matemática chamada convolução (GOODFELLOW et al., 2016). Estas redes usam convolução no lugar de matrizes de multiplicação gerais em pelo menos uma de suas camadas.

As CNNs se tornaram populares depois que uma de suas arquiteturas ganhou a edição de 2012 de uma competição da ImageNet[1]. A solução vencedora se chama AlexNet (KRIZHEVSKY et al., 2012). Uma CNN é estruturada por uma série de camadas convolucionais que atuam como extratores de características, seguidas por um classificador (geralmente uma MLP), também conhecido como camada totalmente conectada (do inglês, *Fully Connected Layers*, ou FC), como mostra a Figura 2.

Figura 2 – Ilustração da arquitetura básica de uma CNN



Fonte: A própria autora.

[1] *http://www.image-net.org/*

As primeiras camadas da CNN recebem a imagem de entrada e executam nela convoluções usando múltiplos *kernels* (ou filtros), que resultam em conjuntos de mapas de características. Cada um desses mapas determina a intensidade e localização de uma característica específica; eles podem ser submetidos a uma operação de sub amostragem chamada *Pooling*, que retorna outros mapas de características numa resolução menor (SOUSA, 2019).

Os seguintes tipos de *Pooling* podem ser citados: Pooling máximo (do inglês, *Max Pooling*) e Pooling médio (do inglês, *Average Pooling*). O *pooling* máximo retorna o valor máximo da parte da imagem coberta pelo *kernel*, enquanto o *pooling* médio retorna a média entre todos os valores da secção da entrada analisada. Essas camadas são importantes para suprimir ruídos e reduzir dimensionalidade (SAHA, 2018). As próximas camadas convolucionais usam os mapas de características resultantes das camadas anteriores para executar mais convoluções e gerar novos mapas. Os mapas da última camada são a entrada para o classificador (i.e a camada FC).

2.1.1 Aplicações de redes neurais convolucionais em visão computacional

A visão computacional possui uma variedade de problemas que envolvem por exemplo a classificação, detecção e rastreamento de objetos em imagens. A resolução destas questões alcançou um grande avanço com o emprego na visão computacional de algoritmos que utilizam redes neurais.

2.1.1.1 Classificação de imagens

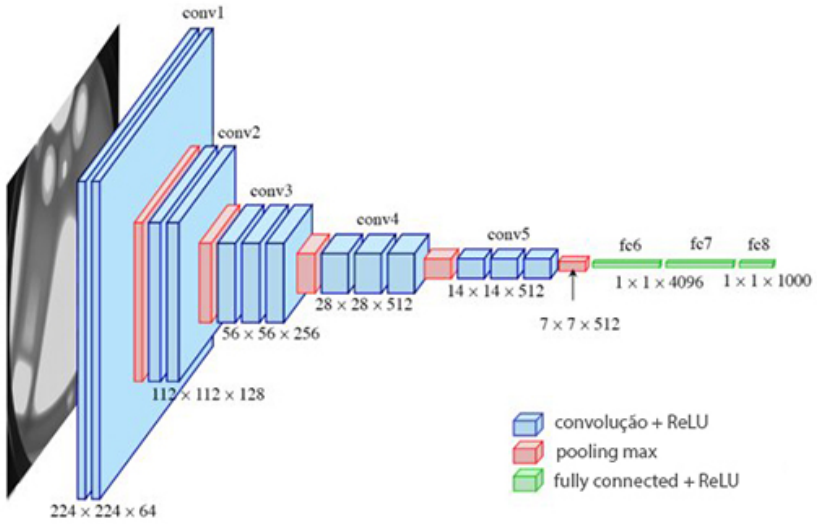
A classificação de imagens é um problema que abrange também a extração de características de uma imagem de entrada. Como dito na seção 2.1, as primeiras camadas da rede neural recebem uma imagem e executam funções de convolução, extraindo características e inferindo uma classe sobre essa imagem. Um clássico exemplo de rede concebida para abordar o problema de classificação de imagens é a AlexNet (KRIZHEVSKY et al., 2012).

Para a AlexNet foi proposto que dado uma imagem, ela poderia ser classificada em uma dentre 1000 diferentes classes (e.g gatos, cachorros, etc). Por outro lado, essa rede acaba sendo considerada grande por possuir cerca de 60 milhões de parâmetros e 650.000 neurônios (GOODFELLOW et al., 2016), limitando a inferência nessa rede em infraestruturas que exigem mais poder computacional. A partir da AlexNet, outras redes foram surgindo com modificações na profundidade e tamanho de entrada, mesclando técnicas de aprendizagem de máquina, oferecendo aumento de precisão na classificação de imagens. Algumas dessas redes serão apresentadas nas próximas subseções.

2.1.1.1.1 Rede VGG

A rede VGG foi apresentada por um grupo da Universidade de Oxford após ganhar o segundo lugar na competição ILSVRC-204 da ImageNet (PAK; KIM, 2017), superando outros modelos nas tarefas de classificação e localização. Existem muitas variações da rede VGG (VGG16, VGG19, etc), que diferem entre si pelo total de número de camadas na rede. Um ponto peculiar em relação a rede VGG é que ela possui um *kernel*/filtro pequeno, de apenas 3x3 em todo o processo de convolução (PATEL, 2020) e isso é considerado uma melhoria diante da AlexNet, que usa *kernels* de dimensões 11×11 e 5×5.

Figura 3 – Ilustração da estrutura básica da rede VGG16



Fonte: Adaptado de Ferguson et al. (2017)

A rede possui 16 camadas (do inglês, *layers*), incluindo 13 camadas convolucionais (que possuem diferentes profundidades, dependendo da arquitetura) e 3 camadas *fully connected* (para que seu poder discriminação entre as classes seja maior). Existem 5 camadas de *pooling* máximo que seguem algumas das camadas convolucionais. Para todas as 16 camadas são aplicadas a função de ativação ReLU. A ReLU é basicamente uma transformação não linear que ajuda a propagação dos dados de entrada de cada camada para a próxima, de forma que não se percam dados ao atualizar os pesos. A última camada é a camada *softmax* (SIMONYAN; ZISSERMAN, 2014).

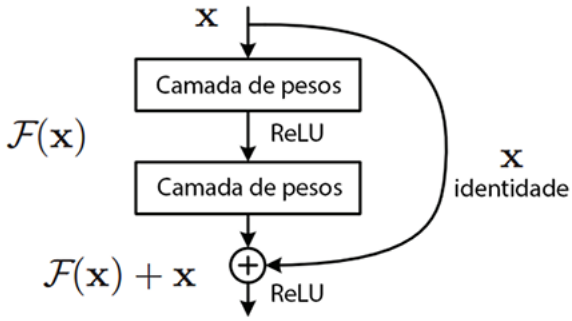
2.1.1.1.2 Rede ResNet

A rede da Microsoft, ResNet, ganhou o primeiro lugar da competição de 2015 (ILSVRC-2015) da ImageNet, com 3.6% de erro. (PAK; KIM, 2017) A ResNet propõe o conceito de aprendizagem residual profunda (do inglês, *deep residual learning*). A instiga para criação dessa

rede foi o fato de que: adicionar mais camadas a uma rede neural já existente pode causar perda de desempenho e precisão.

Considerando um cenário em que camadas adicionadas recentemente à rede podem ser treinadas em um mapeamento de identidade $f(x) = x$, o novo modelo será tão eficiente quanto o original. Para isso, é necessário parametrizar o desvio pela identidade, já que é retornado $x + f(x)$ da função de ativação. Caso haja alguma camada que não seja necessária, basta que $f(x) = 0$, e o valor da entrada x será mantido. Na ResNet, isso é feito por meio de blocos residuais (HE et al., 2016a; HE et al., 2016b), ou "conexões de salto", ilustrados na Figura 4. Essa estratégia permite que os dados se propaguem novamente para as camadas anteriores (EBRAHIMI; ABADI, 2018).

Figura 4 – Bloco residual usado na rede ResNet



Fonte: adaptado de He et al. (2016a)

A ResNet se inspirou no projeto da VGG e a maioria das camadas convolucionais possuem filtros de 3x3, que termina com uma camada *pooling* de média global e uma camada *fully connected* de 1000 saídas com *softmax*.

2.1.1.2 Detecção de objetos

Em um cenário onde existe apenas um objeto em uma imagem, a identificação deste objeto poderia se resumir na aplicação de um classificador de imagem já treinado. Porém, supondo que neste cenário existam múltiplos objetos, o classificador estaria limitado, pois necessitaria identificar a localização de cada objeto presente na imagem.

Nesse caso, o conceito de detecção de objetos se faz útil, já que esse processo vai tentar descobrir a localização e a classe para cada objeto. Grande parte dos algoritmos de detecção de objetos usam esse mesmo esquema e aplicam um mesmo classificador sobre diferentes regiões de uma imagem (MENESES, 2019).

Redes como a R-CNN (GIRSHICK et al., 2014), *Fast* R-CNN (GIRSHICK, 2015) e *Faster* R-CNN (REN et al., 2015) são as chamadas redes baseadas em região (do inglês, *region based*). Elas selecionam regiões na imagem onde existe maior probabilidade de estar os objetos

de interesse. Diferentemente das redes que utilizam dois estágios (um para gerar regiões de interesse e outro para detectar objetos para cada região), as abordagens da SSD (LIU et al., 2016) e da YOLO (REDMON et al., 2016) utilizam um único estágio para detecção e classificação, diminuindo a complexidade.

Os modelos citados anteriormente e na subseção 2.1.1.1 são considerados estado da arte, mas ainda são modelos grandes que limitam a aplicação em sistemas com recursos limitados. Diante desse problema, surgiram modelos reduzidos que utilizam o conhecimento e técnicas dos modelos descritos e possibilitam seu uso no contexto de sistemas com recursos limitados. Algumas dessas redes serão apresentadas nas próximas subseções.

Após a classificação e a detecção dos objetos, ainda existe o problema de rastreamento destes objetos em vídeo, que consiste de uma sequência de imagens. Esse problema abarca os mapas de características dos classificadores, a localização dos objetos, e o cálculo das associações de detecção e discriminação. A questão é analisada e testada usando o paradigma de rastreamento de múltiplos objetos, que será abordado na seção seção 2.2.

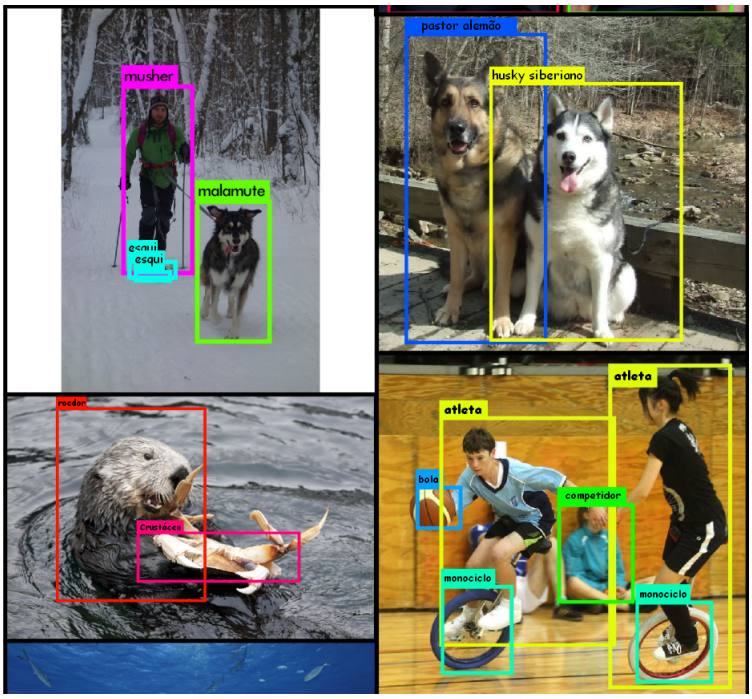
2.1.1.2.1 YOLO

Proposta por Redmon et al. (2016), a YOLO (Você só olha uma vez, do inglês, *You Only Look Once*) é uma rede neural inovadora que trata o desafio de detecção e classificação de objetos como um problema de regressão, pois ao invés de selecionar na imagem regiões de interesse para serem classificadas posteriormente, na YOLO, as previsões das coordenadas dos objetos (usando *bounding boxes*) e da probabilidade de qual classe de objetos eles são para a imagem inteira, são calculadas em apenas um processo.

O modelo divide a imagem em um grid de SxS células, cada célula no grid contém B *bounding boxes* e um valor que representa o quanto de certeza a rede tem de que aquela *bounding box* possui um objeto. Cada *bounding box* tenta prever coordenadas que representam seu centro, sua largura e altura em relação a imagem inteira e também o valor de confiança representado pelo IOU (*Intersection over Union*, ou do inglês, Interseção sobre União) entre ela mesma e qualquer outra *bounding box* em *ground truth*[2]. (REDMON et al., 2016)

[2] Conceito que signfica "verdade básica"ou "verdade absoluta", ou seja, é a saída esperada. No caso de detecção de objetos, seria quando a *bounding box* fosse detectada corretamente de acordo com a base em questão.

Figura 5 – Objetos detectados pela YOLO



Fonte: Adaptado de Redmon e Farhadi (2017)

Atualmente existem algumas diferentes versões da YOLO, a primeira delas foi engendrada por Redmon et al. (2016). Redmon e Farhadi (2017) apresentaram a YOLOv2, uma versão mais precisa e rápida que a primeira e a YOLO9000, que se propõe a detectar mais de 9000 categorias de objetos. Posteriormente, a YOLOv3 foi idealizada por Redmon e Farhadi (2018), com melhoria da eficiência e reconhecimento de objetos pequenos na imagem, ao custo do aumento do tamanho e complexidade da rede (REDMON; FARHADI, 2018).

Algumas versões mais leves também foram produzidas, como a Tiny YOLO e YOLOv3-tiny. Atualmente a versão mais nova é a YOLOv4 que, apesar de não ser produzida pelos mesmos autores das versões anteriores, propõe melhorias na velocidade de inferência e acurácia, superando outros detectores e utilizando novos métodos de *data augmentation* chamados de *Mosaic* e SAT (*Self-Adversarial Training*, ou do inglês, Treinamento de Autodefesa) (BOCHKOVSKIY et al., 2020).

Todas as versões citadas utilizam para treinamento e inferência uma arquitetura de rede neural chamada *Darknet* (REDMON et al., 2016; REDMON; FARHADI, 2017; REDMON; FARHADI, 2018; BOCHKOVSKIY et al., 2020). Essa arquitetura criada por Joseph Redmon, um dos inventores da YOLO, é *open source* e escrita na linguagem de programação C e CUDA (REDMON, 2013–2016).

Tabela 1 – Comparação de resultados de performance entre versões YOLO na base COCO

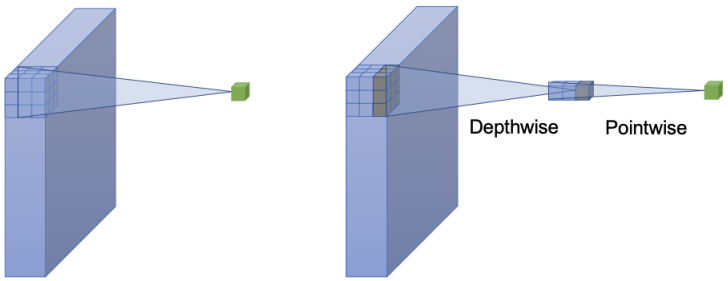
Modelo	**Acurácia média (mAP)**
YOLOv2	48.1%
Tiny YOLO	23.7%
YOLOv3	55.3%
YOLOv3-tiny	33.1%
YOLOv4	**62.8%**
YOLOv4-tiny	40.2%

Fonte: Adaptado de Redmon (2013–2018) e Bochkovskiy (2020).

2.1.1.2.2 MobileNet

Howard et al. (2017) apresentou a MobileNet como uma classe de modelos eficientes para dispositivos com recursos limitados. Ao invés do uso de operações convolucionais tradicionais, ela utiliza operações convolucionais separadas em profundidade (do inglês, *depthwise separable convolutions*). Esse tipo de operação consiste em dividir uma operação de convolução tradicional em duas operações distintas (ver Figura 6). A primeira operação (*dephwise*) aplica um filtro convolucional para cada canal da imagem de entrada, e a segunda operação (*pointwise*) cria uma combinação linear entre todos os canais no resultado recebido da primeira operação. (GUO et al., 2019)

Figura 6 – Convolução tradicional e convolução separada por profundidade

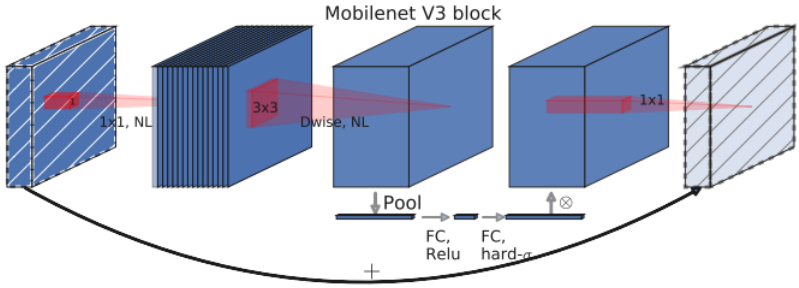


Fonte: Adaptado de Guo et al. (2019).

A segunda versão, MobileNetV2, introduz o uso de blocos residuais invertidos (do inglês, *inverted residuals*) e gargalos lineares (do inglês, *linear bottlenecks*). A rede começa com uma camada de expansão, que aumenta o número de canais de entrada para serem usados na convolução separada em profundidade. Depois disso, o resultado vindo dessa convolução vai para uma camada de projeção que diminui os dados recebidos, e é considerada uma camada gargalo (do inglês, *bottleneck layer*), porque reduz a quantidade de dados que fluem pela rede. Nesse bloco inteiro é aplicado uma conexão residual, que funciona de maneira similar à ResNet, ajudando no fluxo de valores de gradiente na rede (SANDLER et al., 2018).

Posteriormente, a MobileNetV3 proposta por Howard et al. (2019), faz a combinação de várias técnicas. Ela adiciona camadas de compressão (do inglês, *squeeze*) e de excitação (*excitation*) no bloco de camada gargalo construída na MobileNetV1 (ver Figura 7). Uma nova função de ativação foi empregada, a *swish*, que gera melhores resultados em dispositivos com recursos limitados do que a função *sigmoide* (HOWARD et al., 2019). Além disso, foi utilizada uma técnica chamada Busca de Arquitetura Neural (do inglês, *Neural Architecture Search* ou NAS) que produz uma *thread* de módulos que são colocados juntos para formar um único modelo que tenha a melhor acurácia possível.

Figura 7 – Bloco de camadas da MobileNetV3 contendo as camadas de *squeeze*, *dephwise*, *pointwise*, *excitation* e aplicação de uma conexão residual entre as camadas de *squeeze* e *excitation*.

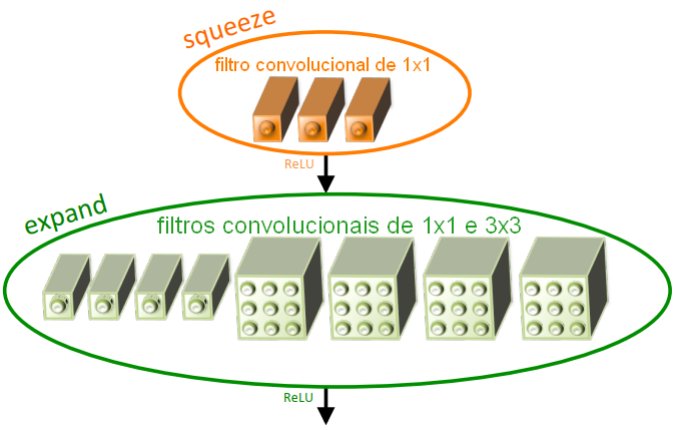


Fonte: Adaptado de Howard et al. (2019).

2.1.1.2.3 SqueezeNet

A SqueezeNet foi apresentada em Iandola et al. (2016b) como uma rede de mesma acurácia que a AlexNet, porém de menor tamanho, com quase 50 vezes menos parâmetros e 3 vezes mais rápida. A rede emprega três principais estratégias: a primeira é a substitiuição de filtros convolucionais de 3x3 por filtros de 1x1, diminuindo os parâmetros em até 9 vezes. A segunda é a diminuição do número de canais de entrada para filtros de 3x3, utilizando camadas de compressão (do inglês, *squeeze layers*). E a terceira é o uso de camadas de *pooling* de forma tardia, pois assim, muitas camadas da rede produziriam grandes mapas de ativação e isso ocasionaria em maior acurácia na classificação.

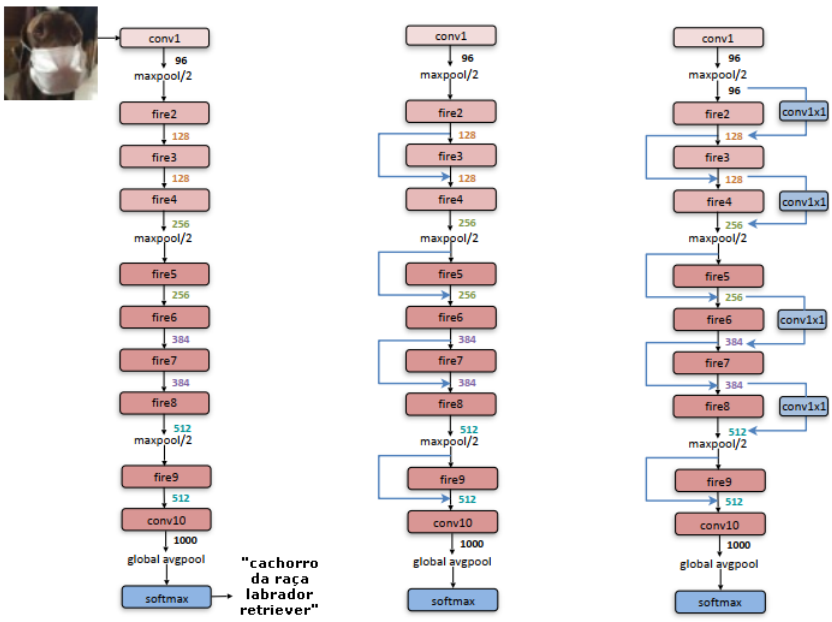
Figura 8 – Módulo de incêndio.



Fonte: Adaptado de Iandola et al. (2016b).

Para aplicar as estratégias, os autores conceberam um módulo chamado Incêncio (do inglês, *fire module*), conforme mostrado na Figura 8. Esse módulo é composto por duas camadas, uma de compressão (*squeeze*) e outra de expansão (*expand*). A camada de compressão utiliza um filtro de convolução de 1x1, reduzindo os canais de entrada, e alimenta a camada de expansão que utiliza filtros de convolução de 1x1 e 3x3. A partir daí, nota-se a aplicação das duas primeiras estratégias citadas. Na estrutura da rede (ver Figura 9), o uso de camadas *max pooling* são feitos relativamente tarde, aplicando a última estratégia mencionada.

Figura 9 – Estrutura da SqueezeNet



Fonte: Adaptado de Iandola et al. (2016b).

2.1.2 Técnicas de compressão de modelos

As redes neurais profundas (DNN) vêm ganhando popularidade pelos seus notáveis resultados em diversas áreas de aplicação. A princípio, modelos mais profundos têm maior capacidadade de reconhecimento de características complexas, ao custo do aumento da dificuldade de treino já que há o crescimento substancial da quantidade de parâmetros a serem aprendidos.

Atualmente muitas aplicações demandam recursos de processamento e execução em tempo real e essa superparametrização limita o uso de modelos profundos nesses tipos de aplicações (POKHREL, 2020). As redes alternativas, citadas na subseção 2.1.1.2, utilizam menos parâmetros e possuem arquiteturas com técnicas que ajudam a suprimir a falta de mais profundidade. Além da existência de técnicas para reduzir o tamanho do modelo, de modo que ele possa executar em dispositivos com recursos limitados.

2.1.2.1 Poda (*pruning*)

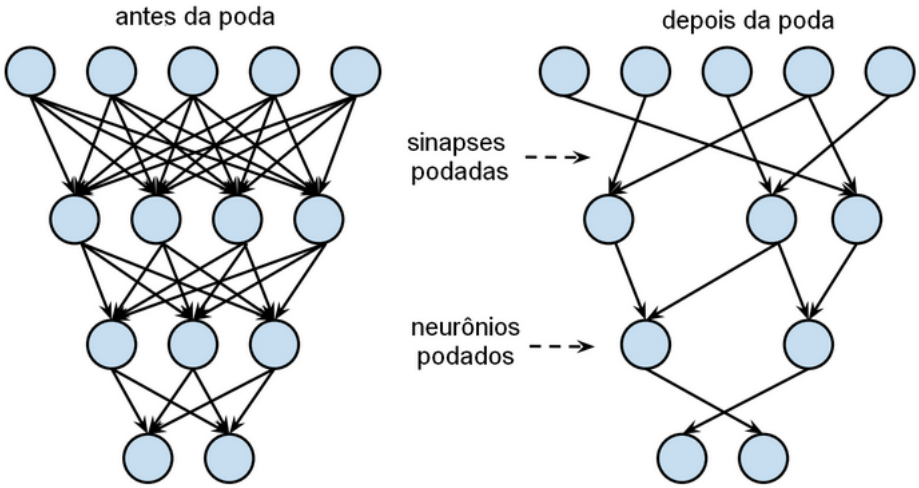
A técnica de *pruning* reduz o número de parâmetros de um modelo, removendo as conexões/neurônios redundantes baseados em gradientes ou outras heurísticas (POKHREL, 2020). Abordagens que utilizam a *pruning* acreditam no conceito de que é necessário treinar redes maiores para achar subredes menores, com desempenhos que tendem a ser o mesmo ou próximos ao do modelo original (SALVI; BARROS, 2020).

Existem diversos caminhos para se podar uma rede neural. Seja ao podar os pesos, configurando parâmetros para o valor zero, o que torna a rede espaçada e diminui a quantidade de parâmetros, ao passo que mantém a arquitetura original. E também pode-se remover um ou mais neurônios da rede, modificando sua arquitetura tornando-a menor, enquanto mantém a acurácia do modelo original (HAN et al., 2015).

O uso dessa técnica traz desafios sobre o conhecimento de o quê e quando podar. Existem diversas heurísticas e métodos que determinam qual nó é menos importante e causará menos impacto na acurácia da rede, algumas delas se baseiam nos pesos ou valores de ativação dos neurônios. É possível observar que alguns pesos possuem valores muito próximos a zero, ou alguns neurônios em uma camada possuem pesos ou valores de ativação com o mesmo dado, em ambos os casos é possível setar o valor do peso para zero ou simplesmente remover os neurônios da rede, causando o mínimo impacto no modelo.

Geralmente a aplicação da poda é feita após a fase de treinamento da rede neural. Fazendo o uso de redes pré-treinadas, também é comum que após a aplicação da poda, o desempenho da rede caia, problema que pode ser resolvido usando a estratégia de *fine tuning* para retreinar o modelo (BLALOCK et al., 2020). A quantidade de vezes da aplicação da poda ou da *fine tuning* é empírica e depende do contexto em que a rede neural vai ser aplicada.

Figura 10 – Conexões/sinapses e neurônios depois da aplicação da poda



Fonte: Adaptado de Han et al. (2015).

2.1.2.2 Destilação de conhecimento (*knowledge distillation*)

Destilação de conhecimento é uma técnica de compressão de modelo que permite que um modelo menor tenha o mesmo desempenho que um modelo pré-treinado e/ou maior. Esse processo se assemelha a relação professor-aluno, onde a rede pré-treinada é o professor e a rede menor é o aluno. Hinton et al. (2015) generaliza o método de compressão proposto por Buciluǎ et al. (2006) e propõe o conceito de destilação (do inglês, *distillation*), que usa uma versão simplificada dos valores de saída de uma rede "professor", para ensinar uma rede "aluno"(YIM et al., 2017).

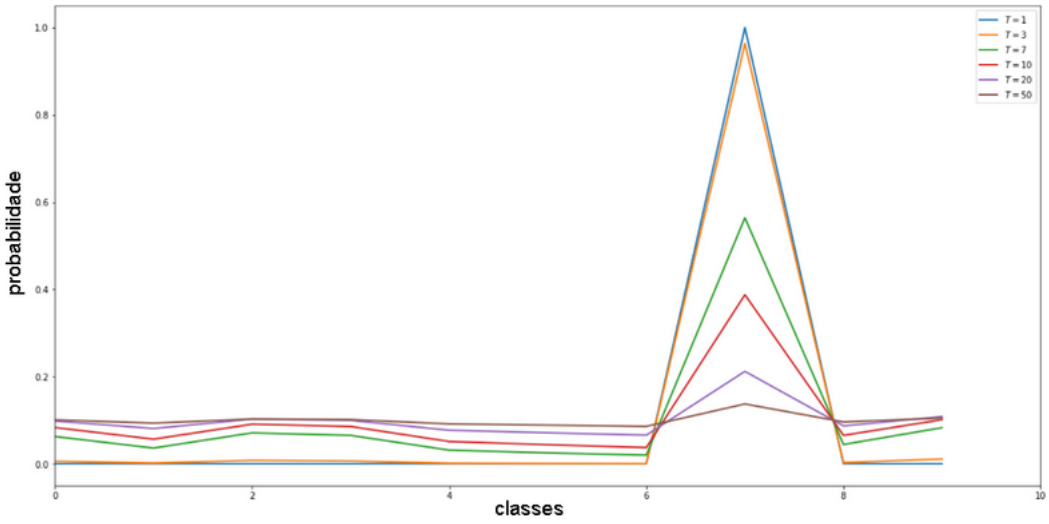
Redes neurais produzem probabilidades de classe usando a função de ativação *softmax*, que convertem os chamados *logit* (valores de saída da última camada) nas suas respectivas probabilidades. Para garantir que a rede "aluno"produza a mesma saída que a rede "professor", uma função de custo é aplicada para tentar fazer com que suas probabilidades de saída correspondam com as da rede "professor"(que são chamadas de *soft targets*).

Essas probabilidades são de 0 a 1, e a soma de todas é igual a 1. Cada probabilidade de saída está relacionada com cada classe possível a ser identificada. Em muitos casos, a probabilidade para a classe alvo pode estar muito alta e as probabilidades para as outras classes muito próximas a zero. Neste cenário, essas informações não agregam valor em relação a verdade absoluta na base de dados utilizada. Como resolução, Hinton et al. (2015) introduz o conceito de "temperatura *softmax*"que se utiliza da Equação 2.1:

$$p_i = \frac{exp(\frac{z_i}{T}))}{\sum_j exp(\frac{z_i}{T}))} \tag{2.1}$$

A probabilidade p_i de classes i são calculadas a partir dos *logits* z, onde T é o parâmetro de temperatura. Quando $T = 1$, obtém-se a função *softmax* padrão. Conforme T cresce, a probabilidade gerada pela *softmax* se torna mais suave (ver Figura 11), fornecendo mais informações sobre quais classes a rede "professor"considera mais semelhantes à classe alvo (INTELLABS, 2018).

Figura 11 – Probabilidade de predição do número "7". Conforme a temperatura (T) cresce, a probabilidade de identificar a classe alvo "7"suaviza.



Fonte: A de Kompella (2018).

O ser humano pode, por exemplo, correlacionar a representação do número "7"escrito à mão com o número "1", atitude que se assemelha ao comportamento de um modelo que retorna uma alta probabilidade da classe "1"enquanto está inferindo o número "7"(Figura 12).

Figura 12 – Representação visual dos números um e sete escritos à mão

Fonte: Adaptado de Kompella (2018).

A diferença entre os humanos e os modelos de redes neurais que usam o valor de temperatura na suas funções *softmax*, é que os humanos não podem quantificar o quanto o número "7"parece com o número "1", à medida que os modelos fazem isso com precisão. Hinton et al. (2015) nomeia esse conhecimento como "*dark knowledge*". Paralelamente a probabilidade de

identificar o número "7", os modelos também reconhecem a informação de o quanto o número "7"se parece com o número "1". É esse conhecimento que é transferido para a rede "aluno"com o processo de destilação.

2.2 Rastreamento de Múltiplos Objetos (MOT)

O paradigma de rastreamento de múltiplos objetos (MOT) visa localizar múltiplos objetos em *frames* de vídeos, preservando suas respectivas identidades. Enquanto no paradigma de rastreamento de um único objeto (SOT) a aparência de um alvo é conhecida a priori, em MOT é necessário que a etapa de detecção aconteça para identificar os alvos que entram e saem da cena (CIAPARRONE et al., 2020).

Na etapa de detecção, a medida de similaridade entre objetos nos *frames* de vídeos é calculada a partir de modelos de descrição de objetos, que podem ser baseados na aparência, movimentação, etc. Estes modelos facilitam a criação de uma identidade para o alvo no próximo *frame* para que a associação da detecção ao alvo seja feita, atualizando o seu estado. O estado de um objeto o descreve com algumas variáveis que o caracterizam, como suas coordenadas espaciais e sua área.

O rastreamento de múltiplos objetos pode ser visto como um problema de estimativa multivariável. Dada uma sequência de imagens, é empregada a variável s_t^i para denotar o estado do i-ésimo objeto no t-ésimo *frame*, onde $\boldsymbol{S}_t = (s_t^1, s_t^2, ..., s_t^{M_t})$ são os estados de todos os M_t objetos no t-ésimo *frame*. Dado $s_{i_s:i_e}^i = (s_{i_s}^i, ..., s_{i_e}^i)$ como a sequência de estados do i-ésimo objeto, onde i_s e i_e são, respectivamente, o primeiro e último *frame* em que o alvo i aparece, e $\boldsymbol{S}_{1:t} = (\boldsymbol{S}_1, \boldsymbol{S}_2, ..., S_t)$ denota a sequência de todos os estados de todos os objetos desde o primeiro ao t-ésimo *frame*.

Seguindo a estratégia mais usada, *tracking-by-detection*, denota-se d_t^i como o conjunto de detecções para o objeto i-th no t-ésimo *frame*, $\boldsymbol{T}_t = (d_t^1, d_t^2, ..., d_t^{M_t})$ o conjunto de detecções para todos os M_t objetos no t-ésimo *frame*, e $\boldsymbol{T}_{1:t} = (\boldsymbol{T}_1, \boldsymbol{T}_2, ..., T_t)$ o conjunto de todas as detecções seqüenciais coletadas de todos os objetos do primeiro ao t-ésimo *frame* (LUO et al., 2014). Logo, o MOT pode ser modelado através da estimativa de inferência bayesiana, a partir da distribuição condicional dos estados sequenciais, sendo:

$$\hat{\boldsymbol{S}}_{1:t} = \arg\max_{\boldsymbol{S}_{1:t}} P(\boldsymbol{S}_{1:t} \mid \boldsymbol{O}_{1:t}) \tag{2.2}$$

em que a Equação 2.2 pode ser resolvida tanto por inferência probabilística ou alguma abordagem de otimização determinística. Os modelos que utilizam filtro de Kalman, filtro de Kalman extendido e o filtro de partículas são de inferência probabilísticas. Já para abordagens de otimização determinística existem o Método Húngaro, Algoritmos gulosos para grafos bipartidos, Problema do caminho mínimo (*K-shortest paths*), entre outros (LUO et al., 2014).

A maioria dos algoritmos MOT são divididos em partes que são representadas da seguinte forma (CIAPARRONE et al., 2020):

a) Detecção: um algoritmo de detecção de objetos analisa cada *frame* para identificar quais objetos percentem às classes alvo usando *bounding boxes*, também conhecidas como detecções;

b) Extração de características/Previsão de movimento: um ou mais algoritmos de extração de características analisam as detecções e/ou as trajetórias para extrair características de aparência, movimento ou outros tipos;

c) Afinidade: características extraídas e previsões de movimento são mescladas para gerar um valor de similaridade entre detecções e trajetórias;

d) Associação: os valores de similaridade são usados para associar detecções e trajetórias que pertencem ao mesmo alvo, ao que são atribuídos identificadores para as detecções do mesmo objeto.

2.2.1 Representação de objetos

Para que os objetos sejam detectados em diversos *frames* e sejam reidentificados corretamente é necessária uma medida de similaridade entre objetos. Caso aconteça algum problema de detecção, a identidade do objeto é recuperada devido a esse valor de equivalência. Esta medida também é necessária para que o rastreador seja capaz de associar a detecção do objeto à sua trajetória (MENESES, 2019).

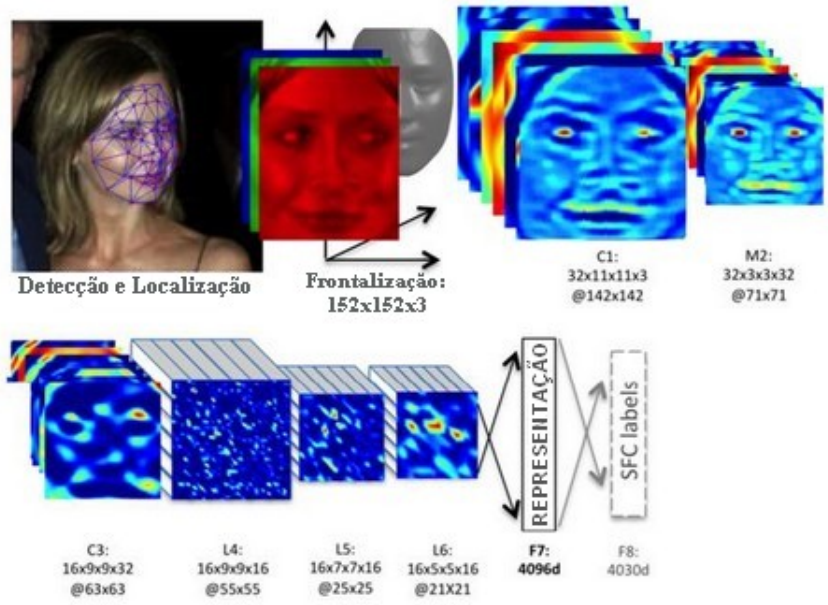
Desse modo, para que haja a descrição dos objetos e discriminação entre eles são criados modelos que correspondem a um conjunto de características do objeto. Dentre os modelos de descrição de objetos estão o modelo de aparência, de movimentação (ou dinâmico), ou modelos que combinem tipos (SUN et al., 2019b).

2.2.1.1 Modelos de aparência

Modelos de aparência descrevem o objeto a partir de suas características visuais. Essas características podem envolver cores, formas e texturas do objeto para que, em nível de detecção, um objeto possa ser diferenciado de outro. Existem muitas representações para o modelo de aparência, como as que fazem uso de histogramas (MITZEL; LEIBE, 2011), ou as que se baseiam na orientação de gradientes, como o HoG (YU et al., 2008).

Existem também métodos para construção de modelo de aparência através da extração de características por meio de redes neurais convolucionais, de modo que essas características podem ser chamadas de características profundas, pois possuem ricas informações semânticas e conseguem ser discriminatórias entre diferentes categorias de objetos (XU et al., 2019b).

Figura 13 – Pipeline de extração de características em uma CNN



Fonte: Adaptado de Alashkar (2016).

Para obter o valor de similaridade entre objetos utilizando modelos de aparência aplica-se o valor do complemento da distância entre o vetor de características de um objeto no instante $t + 1$ e o vetor de uma região delimitada pela detecção (i.e *bounding box*). Dentre essas distâncias estão: a Eucledіana, a do cosseno e a x^2 ponderada (MENESES, 2019).

2.2.1.2 Outros modelos

Também é possível descrever objetos no cenário de rastreamento através de modelos de movimentação. Eles se baseiam no posicionamento e no padrão de movimento do objeto rastreado até o quadro atual, para estimar seu posicionamento no quadro futuro. Dentre as ferramentas utilizadas para tal descrição, destaca-se o Filtro de Kalman (LI et al., 2019).

Para obter o valor de similaridade entre objetos utilizando modelos de movimento, pode-se usar o Índice de Jaccard (MENESES, 2019), que mensura a taxa de sobreposição entre os mesmos. Estudos como os realizados por Wojke et al. (2017), Zhu et al. (2018) e He et al. (2019) mesclam modelos de movimento com outros tipos de modelos, principalmente os que usam CNN. Para calcular a similaridade do objeto nesses casos, pode-se usar uma soma ponderada onde serão somadas as medidas computadas com base em cada modelo, que serão multiplicadas por pesos dados às medidas de similaridade (MENESES, 2019) (e.g Equação 2.3, s_a medida obtida com base no modelo de aparência e s_m como medida obtida com base no modelo de movimento).

$$s = \alpha \cdot s_a + \beta \cdot s_m \tag{2.3}$$

2.2.2 Métricas

Para fornecer uma configuração comum, na qual os algoritmos podem ser razoavelmente testados e comparados, um conjunto de métricas foi estabelecido como padrão (CIAPARRONE et al., 2020). As métricas mais relevantes são as clássicas, definidas por Wu e Nevatia (2006), as métricas CLEAR, estabelecidas por Bernardin e Stiefelhagen (2008), as métricas ID fixadas por Ristani et al. (2016) e outras métricas populares para detecção de objetos apresentadas por Padilla et al. (2020).

2.2.2.1 Métricas clássicas por Wu e Nevatia (2006)

a) Majoritariamente rastreados (MT): porcentagem de trajetórias cobertas pelo rastreador ao longo de pelo menos 80% do seu tempo de vida;

b) Fragmentação: total de vezes em que o registro de uma trajetória foi interrompido pelo rastreador;

c) Majoritariamente perdidos (ML): porcentagem de trajetórias cobertas pelo rastreador ao longo de no máximo 20% do seu tempo de vida;

d) Falsa trajetória: trajetórias que foram previstas que não correspondem a um objeto real;

e) troca de ID: número de vezes que um objeto é corretamente rastreado mas seu ID foi trocado.

2.2.2.2 Métricas CLEAR por Bernardin e Stiefelhagen (2008)

a) FP: quantidade de falsos positivos em todo o vídeo, ou seja, objetos que foram erroneamente detectados;

b) FN: quantidade de falsos negativos em todo o vídeo, ou seja, objetos que deveriam ter sido detectados e não foram;

c) Fragm: total de vezes em que o registro de uma trajetória foi interrompido pelo rastreador;

d) IDSW: número de vezes que um objeto é corretamente rastreado mas seu ID foi trocado;

e) A pontuação MOTA é definida por:

$$MOTA = 1 - \frac{(FN + FP + IDSW))}{GT} \tag{2.4}$$

onde GT é o total de objetos no *ground truth*;

f) A pontuação MOTP é definida por:

$$MOTP = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t} \tag{2.5}$$

onde c_t é o número de objetos associados no t-ésimo *frame* e $d_{t,i}$ é a distância euclidiana entre cada um dos pares de detecção e objeto associados.

2.2.2.3 Métricas ID por Ristani et al. (2016)

Essas métricas são calculadas a partir de um grafo bipartido, onde V_T é o primeiro conjunto de vértices, que contém um nó para cada trajetória verdadeira e um nó falso positivo para cada trajetória computada. O segundo conjunto, V_T, possui nós para cada trajetória computada e nós falso negativos para cada trajetória verdadeira.

Os custos das arestas são dados, logo após uma aresta é escolhida a partir do número de *frames* falso negativos e falso positivos. Existem quatro possíveis cálculos caso uma associação seja feita: (1) Se um nó de V_T for combinado com um nó de V_C (i.e trajetória verdadeira com uma trajetória computada), um ID positivo verdadeiro é contado; (2) Cada falso positivo de V_T combinado com um nó de V_C, conta como um ID falso positivo; (3) Cada nó de V_T combinado com um falso negativo de V_C, conta como um ID falso negativo e, (4) cada falso positivo combinado com um falso negativo, conta como um ID negativo verdadeiro (CIAPARRONE et al., 2020). Depois disso, são contabilizadas as seguintes pontuações:

a) IDTP: soma dos pesos das arestas que possuem combinações de IDs positivo verdadeiros (porcentagem das detecções assinaladas como corretas em todo o video);

b) IDFN: soma dos pesos das aretas que possuem combinações com IDs falso negativos;

c) IDFP: soma dos pesos das arestas que possuem combinações com IDs falso positivos;

d) Precisão: $IDP = \frac{IDTP}{IDTP+IDFP}$

e) *Recall*: $IDR = \frac{IDTP}{IDTP+IDFN}$

f) F1: $IDF1 = \frac{2}{\frac{1}{IDP}+\frac{1}{IDR}} = \frac{2IDTP}{2IDTP+IDFP+IDFN}$

2.2.2.4 Métricas para detecção de objetos por Padilla et al. (2020)

Muitos *benchmarks* usam seus próprios meios de pontuação para avaliar algoritmos de detecção de objetos, que geralmente derivam de uma métrica chamada precisão média (AP, ou do inglês, *average precision*), ou seja, é a métrica mais comumente usada entre *benckmarks* e competições. Para examinar as variações da AP, é importante revisar os seguintes conceitos:

a) Verdadeiro positivo (TP, ou do inglês, *true positive*): consiste em uma detecção correta a partir de uma *bounding box* de *ground truth*;

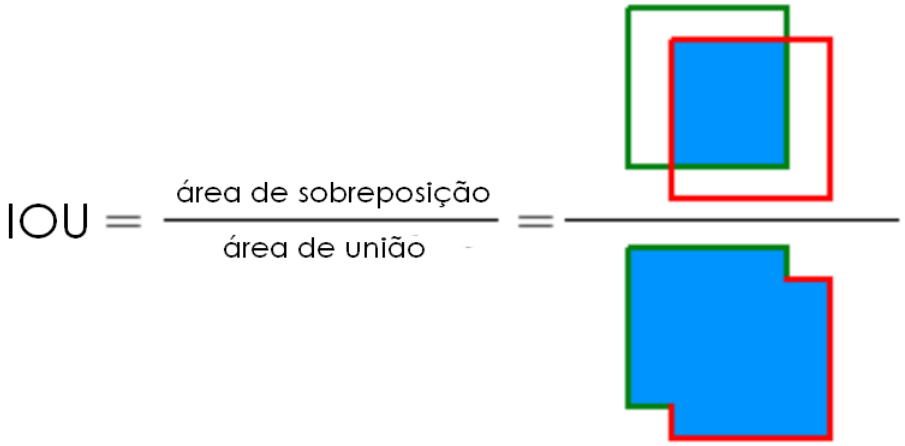
b) Falso positivo (FP, ou do inglês, *false positive*): consiste em uma detecção incorreta de um objeto que não existe ou uma detecção errada de um objeto existente;

c) Falso negativo (FN, ou do inglês, *false negative*): uma detecção de *ground truth* que não foi detectada;

Para usar os conceitos acima e determinar o que é uma "detecção correta"e uma "detecção incorreta", pode-se aplicar o cálculo da interseção sobre união (IOU). No escopo da detecção de objetos, a IOU mede a área de sobreposição entre a *bounding box* prevista B_p e a *bounding box* de *ground truth* B_{gt} divididas pela área da união entre elas, como mostra a Equação 2.6 e a Figura 14.

$$J\left(B_p, B_{gt}\right) = IOU = \frac{área\left(B_p \cap B_{gt}\right)}{área\left(B_p \cup B_{gt}\right)} \tag{2.6}$$

Figura 14 – Interseção sobre União (IOU)



Fonte: Adaptado de (Padilla et al., 2020).

Comparando o resultado da IOU dado um *threshold* (limite) t, é possível classificar uma deteção como correta ou incorreta. Se $IOU \geq t$ então a detecção está correta e se $IOU < t$ a detecção está incorreta.

Outras medidas são melhor consideradas em *frameworks* de detecção de objetos, como a precisão P e a *recall* R. Precisão é a habilidade do modelo para identificar apenas objetos relevantes e é definida pela porcentagem de predições positivas corretas. Já *recall* é a habilidade do modelo para achar todos os casos relevantes (todas as detecções de *ground truth*) e é definida pela porcentagem de predições positivas corretas, dado todas as detecções de *ground truth* (Padilla et al., 2020). Elas são calculadas usando as equações abaixo:

$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{todas as detecções}} \tag{2.7}$$

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{todas as detecções de } \textit{ground truth}} \tag{2.8}$$

Ao medir a relação entre precisão e *recall*, esses valores podem ser plotados em um gráfico para serem observados dadas algumas medidas como a de área sob a curva (AUC, ou do inglês, *area under the curve*), que se for alta, tende a indicar alta precisão e *recall*. Às vezes esses valores podem aparecer em zigzag num gráfico, dificultando a definição da AUC. Para remover esse comportamento, são aplicadas abordagens como a Interpolação de 11 pontos e a Interpolação de todos os pontos.

Na Interpolação de 11 pontos o formato do gráfico, sabida a relação entre precisão x *recall*, é sumarizado pela média dos valores máximos de precisão dado um conjunto de 11 valores de *recall* igualmente espaçados, como mostra a equação abaixo:

$$AP_{11} = \frac{1}{11} \sum_{R \in \{0, 0.1, \ldots 0.9, 1\}} P_{\text{interp}} \{R\}, \tag{2.9}$$

onde,

$$P_{\text{interp}} \{R\} = \max_{\widetilde{R}:\widetilde{R} \geq R} P\left(\widetilde{R}\right). \tag{2.10}$$

Na definição de AP_{11}, em vez de usar a precisão $P(R)$ observada em cada valor de *recall* R, o valor de AP é obtido considerando a precisão máxima $P_{interp}(R)$, cujo valor *recall* é maior que R. Em Interpolação de todos os pontos, em vez de interpolar apenas 11 pontos igualmente espaçados, pode-se interpolar por todos os pontos, dada a equação abaixo:

$$AP_{\text{todos}} = \sum \left(R_{n+1} - R_n\right) P_{\text{interp}} \left(R_{n+1}\right), \tag{2.11}$$

onde,

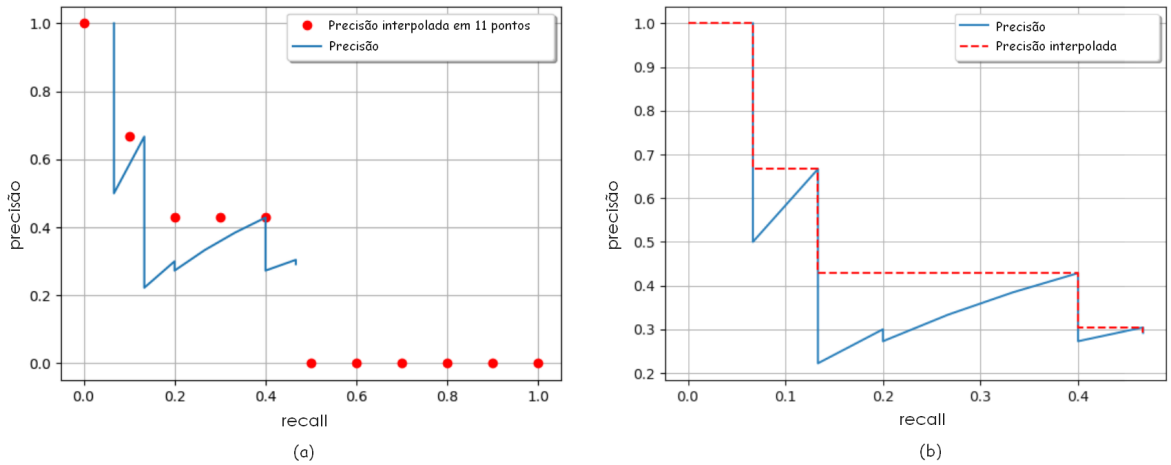
$$P_{\text{interp}} \{R_{n+1}\} = \max_{\widetilde{R}:\widetilde{R} \geq R_{n+1}} P\left(\widetilde{R}\right) \tag{2.12}$$

Ao invés de usar a precisão observada em apenas alguns pontos, como na interpolação de 11 pontos, a AP é obtida por interpolar a precisão para cada valor, obtendo a precisão máxima cujo valor de *recall* é maior ou igual a R_{n+1}.

Por fim, a média de AP (mAP) é uma métrica usada para medir a acurácia do detector de objeto sobre todas as classes em um dataset específico. A medida mAP é a média AP sobre todas as classes, definida pela equação Equação 2.13, sendo AP_i o valor de AP na i-ézima classe e N o total de classes sendo avaliados.

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{2.13}$$

Figura 15 – Representações gráficas das aborgadens de interpolação aplicadas sob valores de precisão e *recall*. Em (a) a representação da aplicação da interpolação em 11 pontos e em (b) a representação da aplicação de interpolação de todos os pontos.



Fonte: Adaptado de (Padilla et al., 2020).

2.2.3 Datasets e Benchmarks

Os possíveis alvos detectados em algoritmos MOT podem estar em condições difíceis (e.g alvos parcialmente visíveis, oclusos, reflexos em janelas, etc.), de maneira que a forma de avaliar esses algoritmos se torne custosa. Para ajudar neste e e em outros aspectos, foram criados *benchmarks* que fornecem grandes bases de dados visuais para análises de testes de algoritmos. Um dos *benchmarks* mais importantes para rastreamento de múltiplos objetos é o MOTChallenge[3].

O MOTChallenge fornece grandes bases para rastreamento de pedestres (e outros) que estão disponíveis ao público. Está à disposição, para todas as bases, os dados dos objetos e das detecções, tanto de treinamento quanto para testes. A razão pela qual estão disponíveis, é que a qualidade da detecção tem um grande impacto no desempenho final do rastreador. A avalição do algoritmo é feita através de uma submissão em um servidor teste, e o *ranking* dos algoritmos com melhor desempenho estão no *website* do *benchmark* (CIAPARRONE et al., 2020). As métricas utilizadas foram citadas na seção anterior.

MOT15 (2D MOT 2015)[4]: É a primeira base criada para o MOTChallenge, possui 22 vídeos, sendo 11 para treinamento e teste. O conjunto de teste contém mais de 10 minutos de gravação e 101345 *bounding boxes* com rótulo. Existem vídeos contendo imagens com alta resolução e outros filmados com câmeras em uma posição estática ou se movendo, com a visão na altura média dos pedestres. Encontram-se gravações feitas à noite com uma câmera em movimento e vídeos em posição de *outdoor* com grande quantidade de pedestres.

[3] $https://motchallenge.net/$
[4] $https://motchallenge.net/data/2D_MOT_2015$

MOT16[5]: Contém 14 vídeos com 292733 *bounding boxes* com rótulo. Foram acrescentados mais vídeos desafiadores contendo grandes quantidades de pessoas (LEAL-TAIXÉ et al., 2017).

MOT17[6]: Possui os mesmos vídeos do MOT16, mas com inclusão de 3 bases com dados de 3 detectores diferentes (Faster R-CNN, DPM e SDP) (CIAPARRONE et al., 2020).

MOT19[7]: Base de dados para o desafio de rastreamento CVPR (*Computer Vision and Pattern Recognition Conference*) 2019, dispõe 8 vídeos (4 pra treino e 4 para teste), com extremo fluxo de pedestres, chegando em uma média de 245 pedestres por *frame*. A base contém um total de 2259143 *bounding boxes*.

MOT20[8]: Propõe o mesmo desafio que o MOT19, com vídeos contendo extremo fluxo de pedestres, com uma média de 342.4 pedestres por *frame* . Possui 8 videos (5 pra treino e 3 para teste) e dessa vez com o total de 2102385 *bounding boxes*.

KITTI[9]: Diferente do MOT, nesse *benchmark* são avaliados rastreamento de veículos e pedestres (apesar de ter 8 classes rotuladas). Possui 21 vídeos para treino e 29 para teste. As métricas usadas são as CLEAR MOT e MT/PT/ML.

Outras bases: UA-DETRAC[10], focada em veículos e câmeras em tráfego, a PETS2009[11], focada em pedestres e bases como a PASCAL VOC[12] e a MS COCO[13], que possuem grande quantidade de imagens, labels de imagens e imagens segmentadas de objetos de diversas categorias.

2.3 Tecnologias para ambientes com recursos limitados

2.3.1 Raspberry Pi

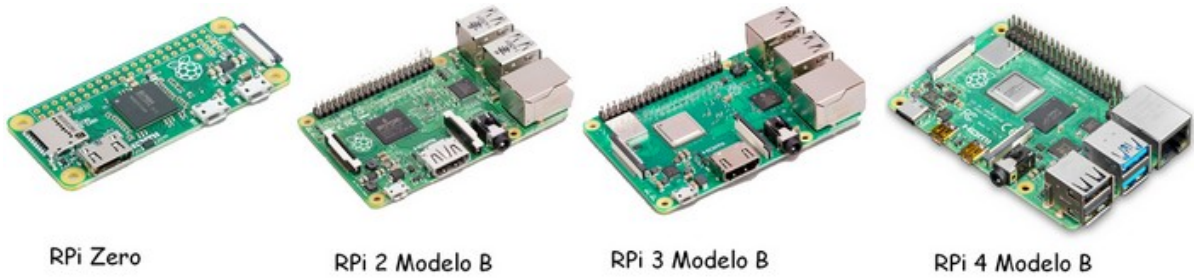
Raspberry Pi (RPi) é uma série de SBCs (em inglês, *Single Board Computers*) criados com a intenção inicial de prover um dispositivo barato para ajudar no ensino de computação nas escolas. Sua origem foi no laboratório de computação da Universidade de Cambridge, em 2006, e seu lançamento comercial foi em fevereiro de 2012. A primeira versão significante foi a Raspberry Pi 2 (Figura 16), pois seu processador de arquitetura ARM possuía mais núcleos e componentes que as primeiras versões (DENNIS, 2016).

Através de uma porta de microSD é possível gravar e executar diversos tipos de sistemas

[5] $https://motchallenge.net/data/MOT16/$
[6] $https://motchallenge.net/data/MOT17/$
[7] $https://motchallenge.net/data/CVPR_2019_Tracking_Challenge/$
[8] $https://motchallenge.net/data/MOT20/$
[9] $http://www.cvlibs.net/datasets/kitti/eval_tracking.php$
[10] $http://detrac-db.rit.albany.edu/$
[11] $http://www.cvg.reading.ac.uk/PETS2009/a.html$
[12] $http://host.robots.ox.ac.uk/pascal/VOC/$
[13] $https://cocodataset.org/$

operacionais na RPi, os mais populares são distribuições baseadas no sistema operacional Linux. Em destaque está a Raspbian, que é uma distribuição baseada no Debian (também baseado no Linux), otimizada para ser usada especialmente nas RPis e não possui relação com a organização responsável pela criação e venda das placas (RASPBIAN, 2021).

Figura 16 – Modelos Raspberry Pi



Fonte: Adaptado de Raspberry Pi Foundation (2021)

A série de modelos mais novos, Raspberry Pi 4 Modelo B, inclui novos recursos e melhorias em comparação aos seus modelos anteriores, como por exemplo: porta USB 3.0 e portas HDMI com suporte a telas com resolução 4K. A placa também possui melhorias no seu processador da Broadcom BCM2711, atualmente com 4 núcleos de 64 bits a 1,5GHz, oferecendo aumento de desempenho em 3 vezes mais que o modelo anterior[14] (Raspberry Pi Foundation, 2021).

As memórias RAM das placas desse modelo podem variar de tamanho, contendo de 1 a 8GiB. Portanto, com esses dados, é evidente que além das placas Raspberry Pi serem acessíveis, a melhoria contínua de seus componentes ao longo dos modelos incentiva seu uso na produção de soluções para diversos problemas em diferentes cenários. Fazendo inferência em redes neurais com algoritmos estado da arte, podemos encontrar sua aplicação na detecção de objetos em alguns trabalhos na literatura como Pena et al. (2017), Zhao et al. (2018), Manjari et al. (2019), Santos e Flôr (2019) e Gonzalez-Huitron et al. (2021). Por disponibilidade, no contexto desta pesquisa será usada a Raspberry Pi 4 Modelo B (Figura 16) de 2GiB de RAM.

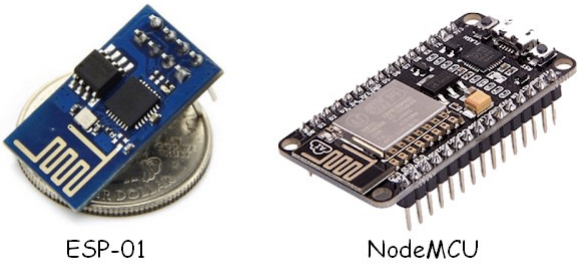
2.3.2 Espressif ESP

ESP é uma série de SoC (em inglês, *System On Chip*) que foca em IoT (em inglês, *Internet of Things*), e foi criada por uma empresa chinesa chamada *Espressif Systems*. As placas dessa série possuem baixos custo de aquisição e consumo de energia, além de incluir microcontroladores e Wi-Fi integrados. As primeiras variantes são módulos nomeados de ESP-X, sendo X o valor da sua respectiva versão (e.g ESP-01). Esses módulos diferem no tamanho e quantidade de

[14] Para comparações com informações detalhadas entre modelos RPi: *https* : *//cdn.shopify.com/s/files/1/0176/3274/files/Raspberry − Pi − Comparison*$_r$*4.pdf*?3484

pinagem para acesso externo. Uma variante muito conhecida é a ESP8266 NodeMCU, que é baseada no módulo ESP-12 e possui iniciativa em código aberto. Os módulos ESP8266 utilizam o microprocessador da linha Cadence Tensilica LX106 a 80MHz (Espressif, 2021).

Figura 17 – Módulos ESP8266.



Fonte: Adaptado de Curvello (2015) e K (2018)

Inicialmente as placas ESP foram pensadas para serem usadas como ponte, em protocolos de redes num ambiente de IoT, para outros dispositivos ou sistemas. Com a criação de um projeto que dá suporte as ESP8266 dentro do ambiente de desenvolvimento para Arduino (Arduino IDE), Ivan Grokhotkov, um dos engenheiros da *Espressif*, possibilitou a mudança na visão inicial em que as placas ESP foram criadas. Diversos projetos envolvendo eletrodomésticos, soluções para cidades inteligentes, *wearables* (dispositivos "vestíveis"como pulseiras inteligentes) e até matadores de mosquito elétrico (PEREZ, 2019) passaram a agregar placas ESP.

A série das ESP32 foi criada como sucessora da ESP8266, possuindo microprocessador Tensilia Xtensa LX6 com dois núcleos funcionando a 240MHz, e um coprocessador de ultra-baixo consumo (em inglês, *Ultra Low Power*[15] ou ULP) mais sofisticado, que permite executar algumas tarefas simples enquanto os processadores principais estão desligados. Ela tem suporte para comunicação nativa com *Bluetooth* 4.2 (BLE), diversos outros protocolos de comunicação e mais pinos GPIO (19 a mais que a ESP8266) (CURVELLO, 2019; Espressif, 2021).

Como foi dito anteriormente, as placas ESP são amplamente usadas como ponte em algum protocolo de rede e em trabalhos utilizando redes neurais, e funcionam com acerto da mesma forma. Diferentemente da Raspberry, nas ESP a inferência não é feita diretamente no dispositivo: ou a imagem é enviada para algum dispositivo com mais poder de processamento (computadores ou outros dispositivos com recursos limitados) (BRUSCHI et al., 2018) ou servidores em nuvem (SAGARIKA et al., 2020) com sugestões de uso de redes neurais distribuídas utilizando paralelismo de dados (DOKIC et al., 2020).

Um módulo relevante a ser citado é a ESP32-CAM, que utiliza o chip da variante ESP32-S, tem suporte para câmera OV2640 e OV7670 de 2MP e porta microSD. Assim como os outros

[15] *https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/ulp.html*

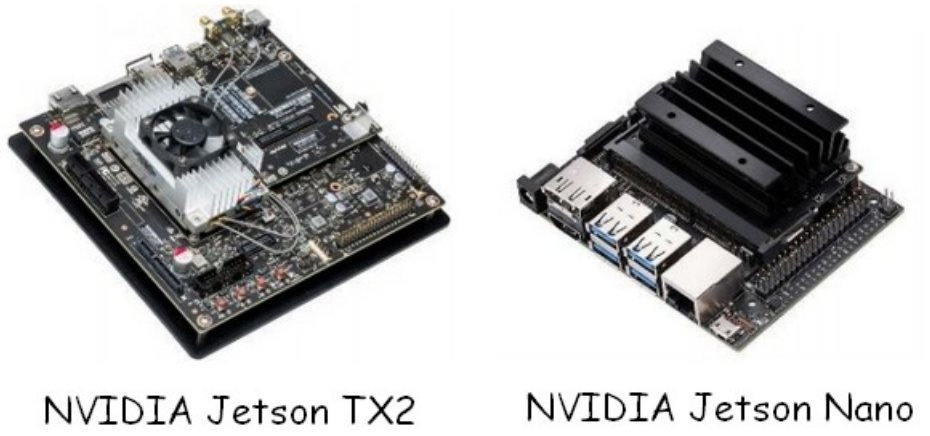
módulos, o ESP32-CAM é compacto, demanda baixo custo de energia e utiliza um cartão SD que possibilita a gravação de imagens obtidas da câmera e o envio delas pela rede. Em Kushnir et al. (2019) foi utilizada uma ESP32-CAM para coleta de imagens através de uma aplicação para dispositivos móveis usando a rede MobileNet. A classificação e detecção de objetos foram realizadas diretamente no dispositivo móvel utilizando as imagens coletadas pelo módulo ESP.

2.3.3 Jetson NVIDIA

Jetson foi o nome dado a família de módulos para Inteligência Artificial criada pela empresa NVIDIA[16]. A família é formada por módulos de alto desempenho, baixo consumo de processamento para aprendizagem profunda e visão computacional (NVIDIA, 2021). Eles dispõem de GPUs com múltiplos núcleos que dão suporte ao uso de computação paralela usando CUDA (ou *Compute Unified Device Architecture*, do inglês, Arquitetura de Dispositivo de Computação Unificada) e memória RAM de 4GB até 32GB.

A NVIDIA criou uma biblioteca CUDA para aceleração de GPU voltada para redes neurais profundas. A cuDNN (NVIDIA CUDA *Deep Neural Network*) pode ser utilizada nos módulos Jetson para aprimorar implementações de aprendizagem profunda como convoluções, *pooling*, normalização e camadas de ativação. (RAHMANIAR; HERNAWAN, 2021)

Figura 18 – Módulos NVIDIA Jetson.



Fonte: Adaptado de Rahmaniar e Hernawan (2021)

No contexto em que o presente trabalho envolve, de redes neurais profundas em sistemas com recursos limitados, os módulos Jetson são comumente usados na literatura em projetos relacionados com áreas em que esses módulos possuem como alvo: a robótica e a automação industrial, utilizando DNNs (ZHAO et al., 2020; NVIDIA, 2021).

[16] Para comparações com informações detalhadas entre módulos NVIDIA Jetson: *https://developer.nvidia.com/embedded/jetson-modules*

Estudos como os realizados por Fang et al. (2019), Wong et al. (2019) e Zhao et al. (2020) utilizaram infraestruturas maiores para o treinamento de DNNs e testes, em cenários executando tempo real, nos módulos da NVIDIA, sendo possível alcançar acurácia compatível diante a resultados estado da arte. A satisfação que os resultados com esses módulos podem alcançar, em desempenho e baixo consumo de energia, enfrenta problemas com a disparidade de custo aquisitivo em comparação às placas citadas anteriormente, como a Raspberry e a ESP. Em lojas credenciadas na *Amazon*[17], os valores das ESP giram em torno de 10 dólares, que distoa muito da RPi e Jetson, já que a Jetson TX2 chega a custar cerca de 400 dólares. O modelo Nano da Jetson possui um valor compatível com a RPi 4 Modelo B, ambas custam cerca de 40 dólares[18].

[17] https://www.amazon.com/

[18] Esses valores estão sendo comparados no ano da realização desse trabalho

3

Trabalhos Relacionados

O objetivo deste capítulo é citar estudos atuais que de alguma forma usem modelos que utilizam aprendizagem profunda para extrair características, e possuam resultados significativos. Deseja-se também apresentar pesquisas que exploram métodos para melhor execução da técnica de transferência de conhecimento.

Na seção 3.1 foi descrito o projeto definido como *baseline*. Ele não faz uso de modelo de aprendizagem profunda para extrair características, mas é usado na presente pesquisa para realizar o objetivo de extensão e comparação de resultados. Este estudo, e todos os outros descritos nesta seção, são modelos criados utilizando a estratégia de *tracking-by-detection* e são categorizados como *online* (já que todo os processos durante a detecção e rastreamento são voltados para execução em tempo real).

3.1 Trabalho *Baseline*

O algoritmo denominado SmartSORT, construído por Meneses (2019), tem como núcleo o modelo de regressão proposto, que recebe como entrada características relacionadas aos alvos, suas trajetórias e suas novas detecções, e retorna um valor de custo da associação $c_{i,j} \epsilon [-1, 1]$ entre cada trajetória com as novas detecções. O estado do objeto alvo é representado pelas posições horizontal e vertical em *pixels* a partir do seu centro, da altura e a razão de aspecto da detecção (*bounding box* retornada pelo detector) e uma variável correspondente ao descritor de aparência.

Quando um objeto entra e sai da imagem, seus identificadores precisam ser criados e destruídos, respectivamente. O SmartSORT trata disto incrementando um contador de perdas. Se o contador excede um limiar, a trajetória é descartada - presumindo que o objeto saiu da imagem -, enquanto que uma nova trajetória é criada para cada detecção que ainda não foi associada, a qual será descartada caso o contador de perda relacionado a ela seja incrementado. O estado do

alvo é atualizado quando uma trajetória é associada a uma nova detecção.

A inferência de conhecimento ao modelo de regressão é feita considerando um vetor de características $\boldsymbol{f}$ formado por atributos extraídos da última detecção associada ${d_i}^{kz}$ a uma trajetória T_i e a detecção no instante atual ${d_j}^{t+1}$. Neste vetor são considerados dados de posição e dimensões de ${d_i}^{kz}$, e valores normalizados entre a posição e as dimensões de ${d_i}^{kz}$ e ${d_j}^{t+1}$. Tais características foram consideradas de modo a fornecer ao modelo informações relacionadas ao posicionamento e à geometria do objeto avaliado, além de permitir a identificação de situações nas quais a associação entre a nova detecção e a trajetória atual do objeto é impraticável, dadas as suas distâncias.

Também são consideradas a diferença de aparência e variação temporal, como a distância do cosseno entre os descritores de suas regiões e também o intervalo de tempo $((t+1)+k_z)$ de quando foram realizadas. Além disso, para entender a movimentação e variação de aparência de um objeto implementou-se uma estratégia nomeada janela deslizante, que considera um novo vetor, o vetor $\boldsymbol{g}$, definido como:

$$\boldsymbol{g}_{T_i,d_j^{t+1}} = [\boldsymbol{f}_{d_i^{kZ},d_j^{t+1}}, \boldsymbol{f}_{d_i^{kZ-1},d_i^{kZ}}, ..., \boldsymbol{f}_{d_i^{kZ-W+1},d_i^{kZ-W+2}},] \tag{3.1}$$

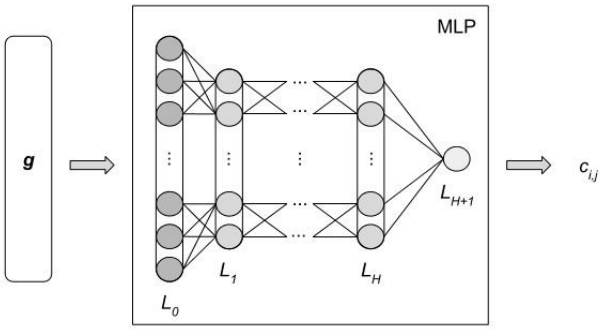
para cada detecção em T_i de comprimento Z, sendo W o tamanho da janela.

Um algoritmo de aprendizado supervisionado foi utilizado para induzir uma função de regressão ao modelo $r(\boldsymbol{g}_e)$, a partir de exemplos positivos E_p e negativos E_n já rotulados, em que $E = E_p \cup E_n$. Os exemplos positivos correspondiam à associação de uma trajetória T_i e uma detecção d_i^{t+1} relativa ao mesmo objeto, sendo rotulado como -1. Enquanto que os exemplos negativos correspondiam à associação de uma trajetória T_i e uma detecção d_j^{t+1}, relativa a objetos diferentes (o_i e o_j, respectivamente), sendo rotulados como 1.

Após o cálculo dos custos de associação para todas as combinações possíveis de detecções e trajetórias, o resultado é usado para a construção de um grafo bipartido entre dois conjuntos disjuntos de nós entre trajetórias e novas detecções, o qual será resolvido com a aplicação do Método Húngaro. Os pesos entre os nós são modelados como afinidades entre trajetórias e detecções (LUO et al., 2014). O hiper-parâmetro C_{max} é considerado para que associações com pesos maiores que ele sejam descartadas. O modelo de regressão foi induzido através de uma MLP (ilustrada na Figura 19) treinada com o algoritmo *Backpropagation*. A função de ativação utilizada após cada camada foi a Unidade Linear Retificada (do inglês, *Retctified Linear Unit* ou ReLU) e sobre a saída da última camada, foi aplicada a função Tangente Hiperbólica (tanh).

Meneses (2019) obtém resultados através de avaliações em cenários no contexto de rastreamento de pedestres e passageiros de ônibus. As avaliações foram dirigidas pelo *benchmark* MOT Challenge 2016 (ver subseção 2.2.3) e pela base de dados BUS Challenge 2018 (construída para realização do estudo de caso do trabalho), respectivamente. Para ambos os testes foi necessário criar bases de dados a partir de um arquivo que o *benchmark* disponibiliza, contendo as informações dos objetos ao longo dos vídeos (identificador, localização, dimensão, categoria e

Figura 19 – Ilustração da MLP construída para o SmartSORT



Fonte: Adaptado de Meneses (2019).

taxa de visibilidade).

A criação das bases se deu através de uma metodologia de amostragem estratificada uniforme de conjuntos (seleção de amostras aleatórias em subgrupos semelhantes). Além disso, precisou-se considerar um protocolo para preparação do método de rastreamento, onde haveria a seleção de hiper-parâmetros do modelo de regressão (um algoritmo de *grid seach* foi utilizado para esse passo) e do SmartSORT, e em consequência, seleção do melhor modelo. Com base nesse modelo, o SmartSORT foi avaliado a partir das métricas CLEAR (ver subseção 2.2.2) e comparado aos resultados de outros trabalhos (ver Tabela 2).

Apesar de obter um valor pouco mais elevado na taxa de troca de identificadores (IDS), em geral, os valores obtidos se equiparam aos outros algoritmos avalidados, demonstrando que o modelo de regressão induzido no SmartSORT é altamente competitivo em termos de acurácia e velocidade.

Tabela 2 – Resultados de métricas CLEAR para algoritmo SmartSORT

	↑ **MOTA**	↑ **MOTP**	↑ **MT**	↓ **ML**	↓ **IDS**	↓ **FM**	↑ **Velocidade**
RAN	63,0%	78,8%	33,9%	22,1%	**482**	1251	1,6 Hz
CNNMT	65,2%	78,4%	32,4%	21,3%	946	2283	11 Hz
EA-PHD-PF	52,5%	78,8%	19,0%	34,9%	910	1321	12 Hz
POI	**66,1%**	79,5%	**34,0%**	20,8%	805	3093	10 Hz
IOU	57,1%	77,1%	23,6%	32,9%	2167	3028	**3000 Hz**
SORT	59,8%	**79,6%**	25,4%	22,7%	1423	**1835**	60 Hz
DeepSORT	61,4%	79,1%	32,8%	18,2%	781	2008	17 Hz
SmartSORT	60,4%	78,9%	21,9%	**16,1%**	1135	2230	27 Hz

Fonte: Adaptado de Meneses (2019).

3.2 Trabalhos utilizando modelos de aprendizagem profunda

O método mais comum para empregar uma CNN como extrator de características é fazendo modificações em sua estrutura e/ou utilizando arquiteturas pré-treinadas e treinando-as para um fim específico. Zhu et al. (2018) usou a rede ResNet-50 (ilustrada na Figura 20) e aplicou a normalização euclidiana (L_2) às características retornadas ao longo da dimensão do canal. Liu et al. (2018) também usou a ResNet-50 pré-treinada, onde a camada Fully Connected (FC) foi descartada, sendo adicionadas duas novas FC. Khan et al. (2019) fez testes em vários modelos ResNet, mas escolheu a ResNet-30 por melhor tempo de execução para o trabalho proposto.

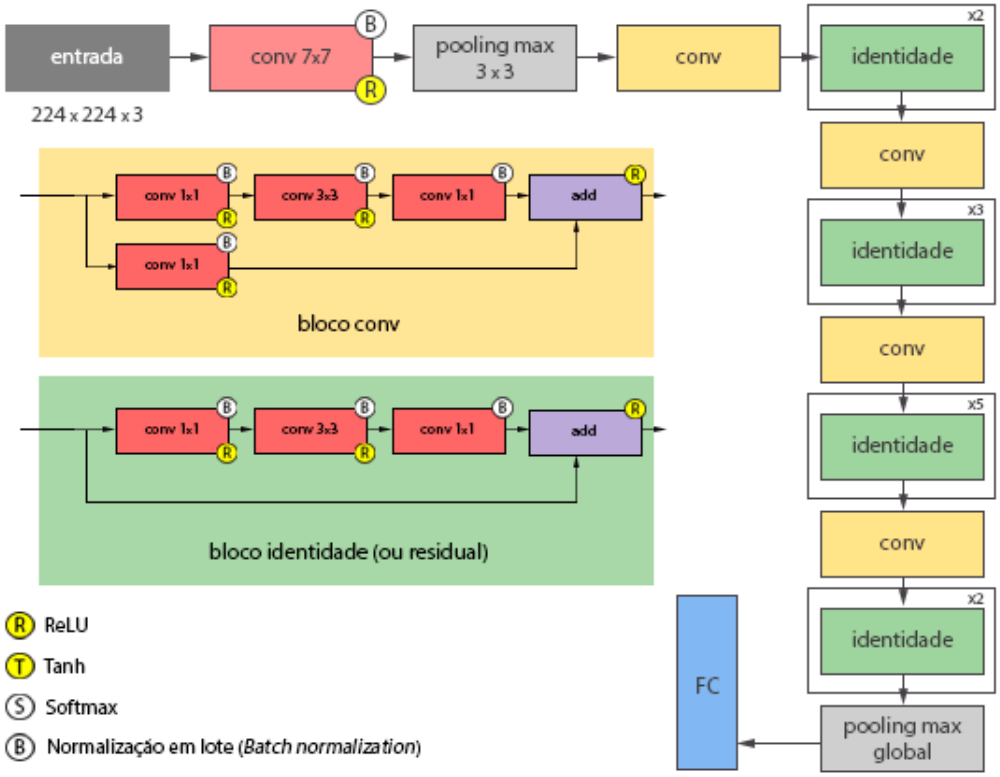
Apesar do uso de extração profunda de características, também foram consideradas outras características visuais a partir de funções manuais (e.g histogramas de cores). Xu et al. (2019a) usou-a para extração de características profundas. Chen et al. (2019) usou a rede GoogleNet em um modelo de aparência mesclado com mecanismos de atenção espaço-temporal. Já He et al. (2019) usou uma rede VGG otimizada (utilizando alguma técnica de poda, do inglês, *prunning*) e a treinou usando uma função de *Triple Loss*. Chen et al. (2017) usou a rede VGG-16 com seus parâmetros fixos para alimentar apenas os classificadores. Cuan et al. (2018) também usa a VGG-16 numa arquitetura proposta para uma rede neural siamesa com duas *branches*. Sun et al. (2019b) modifica a arquitetura VGG convertendo suas camadas FC e *softmax* em camadas convolucionais. Karunasekera et al. (2019) utiliza a última camada para extração de características da YOLOv3. E Wojke et al. (2017) usou a *Wide Residual Network* (WRN) para extração de características profundas.

Os estudos relacionados selecionados que utilizam alguma CNN na construção do modelo de aparência, ou usam a mesma arquitetura de rede neural em todas as etapas do rastreamento de múltiplos objetos (i.e detecção e rastreamento), ou utilizam arquiteturas que são focadas em localizar o melhor conjunto de características que distinguem um objeto de outro. Considerando que o objetivo da aprendizagem de características no rastreamento é avaliar a semelhança entre detecções e trajetórias, a arquitetura de redes neurais convolucionais siamesas é ideal. (LEAL-TAIXÉ et al., 2016; XU et al., 2019b).

Zhu et al. (2018) emprega esse modelo para lidar com detecções com ruído e oclusões. Também há mescla de características visuais manuais como HoG. Um mecanismo de atenção visual é adotado, ele permite que o modelo se concentre nas regiões mais relevantes da entrada para extrair as características mais discriminativas. Já Cuan et al. (2018) aplica uma arquitetura onde duas *branches* compartilham camadas CNN e possuem duas camadas FC com ReLu.

Todos os estudos citados nesta seção foram avaliados pelo conjunto de medidas CLEAR (BERNARDIN; STIEFELHAGEN, 2008) (ver subseção 2.2.2) e tiveram desempenho compatível com o estado da arte (ver Tabela 3).

Figura 20 – Ilustração da arquitetura da rede ResNet-50



Fonte: Adaptado de (KAIM, 2019).

Tabela 3 – Resultados de desempenho de trabalhos utilizando modelo de aparência profunda

	DATASET	↑MOTA	↑MOTP	↑MT	↓ML	↓IDS
Zhu et al. (2018)	MOT16	46,1%	73,8%	42,7%	-	**532**
Liu et al. (2018)	MOT16	43,70%	75,20%	10,50%	46,10%	983
Khan et al. (2019)	MOT16	**75, 2%**	**81, 3%**	**33, 2%**	**17, 5%**	825
Chen et al. (2019)	MOT16	49,8%	-	17,9%	37,7%	614
He et al. (2019)	MOT15	52,2%	74,9%	34,4%	16,1%	578
Chen et al. (2017)	MOT15	38,5%	72,6%	8,7%	37,4%	586
Cuan et al. (2018)	MOT15	43%	76,3%	15,3%	41,8%	876
Sun et al. (2019b)	MOT17	52,41%	75,9%	21,4%	30,7%	8431
Karunasekera et al. (2019)	KITTI	30,43%	72,73%	11,34%	43,30%	100
Xu et al. (2019a)	KITTI	48,5%	73,7%	17%	34,9%	747
Wojke et al. (2017)	KITTI	61,4%	79,1%	32,8%	18,2	781

3.3 Trabalhos utilizando modelos reduzidos de aprendizagem profunda

Os avanços em pesquisa na área de visão computacional voltadas para detecção e classificação de objetos vêm gerando resultados satisfatórios e novas estruturas de redes neurais, como foi mostrado na seção 2.1. Algumas dessas estruturas se aprimoraram em acurácia ao longo de suas variantes, mas ainda assim é possível perceber o desafio em usá-las em dispositivos com

hardware limitado, já que elas demandam muitos parâmetros e consequentemente espaço em memória, custo e tempo de processamento.

Muitas pesquisas como as de Apte et al. (2017), Huang et al. (2018), Mao et al. (2019), Wong et al. (2019), Fang et al. (2019), Zhao et al. (2020), Zhang et al. (2020) introduzem projetos que foram desenvolvidos usando um dos modelos de redes neurais que se popularizou por empregar estratégias que trazem boa acurácia num cenário em tempo real: a YOLO, uma notória estrutura abordada na subseção 2.1.1.2.1.

Em Huang et al. (2018) foi usada a Tiny YOLOv2 como ponto de partida para a criação da YOLO-LITE. Realizaram-se diversas alterações e testes como a adição e remoção de camadas, constatando que isto não trazia uma melhora notável na acurácia da rede. Também foi feito o uso de *batch normalization* (normalização em lote) — em resumo, padronização dos dados de uma camada para conter variância média em zero e desvio padrão em um, antes que esses dados sejam enviados para outra camada —, mas segundo Huang et al. (2018), como a rede é pequena, não sofreria muito por problemas causados por invariâncias.

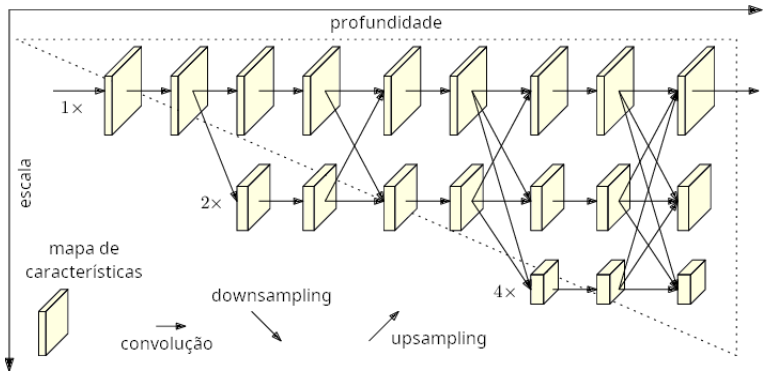
Uma outra técnica chamada *prunning* (poda) também foi testada, que seria a ação de remover alguns pesos da rede, baseados em sua importância, porém também foi constatado que para a YOLO-LITE essa técnica não gerava nenhuma melhoria em acurácia e velocidade. Essa rede trouxe, de forma empírica, mais uma possibilidade para detecções de objetos utilizando camadas rasas (do inglês, *shallow layers*) e redes neurais em dispositivos limitados, funcionando com velocidade compatível a outros modelos do estado da arte.

Usando a variante chamada Tiny YOLOv3, a Tinier-YOLO (FANG et al., 2019) foi proposta para reduzir o tamanho dessa variante enquanto aumenta sua acurácia em cenários em tempo real. A Tinier-YOLO utiliza a estrutura introduzida pela SqueezeNet (ver subseção 2.1.1.2.3) chamada *fire module*, em conjunto com o conceito de conexões densas (do inglês, *dense connection*), introduzido em Huang et al. (2017), para fortalecer a propagação de características entre camadas, pois uma camada em uma rede não depende apenas de características da camada adjacente, mas também das camadas anteriores. Além disso, assim como em Huang et al. (2018) não foi feito o uso de *batch normalization*, porém Fang et al. (2019) questiona o uso de camadas rasas usadas em Huang et al. (2018), pois sua eficácia não poderia ser comparada a de camadas profundas.

Por outro lado, Zhao et al. (2020) aproveita a estrutura que utiliza camadas rasas e profundas produzida por Huang et al. (2018), a YOLO-LITE, adicionando estruturas residuais da ResNet (ver subseção 2.1.1.1.2) e sub-redes paralelas de alta a baixa resolução (do inglês, *parallel hight-to-low resolution subnetworks*) introduzida em Sun et al. (2019a) numa arquiterura chamada *High-Resolution Net* (HRNet). Estas estruturas de sub-redes conservam representações em alta resolução durante o processo de estimação de pose. No caso da estrutura produzida por Zhao et al. (2020), a Mixed YOLOv3-LITE, as sub-redes são usadas para melhorar a precisão da detecção, pois elas conduzem a reconstrução multi-resolução de características profundas e rasas em múltiplas escalas, através de fusões multi-escalares entre camadas, de modo que os mapas de

características possuam características profundas e rasas ao mesmo tempo.

Figura 21 – Ilustração da arquitetura da rede HRNet



Fonte: Adaptado de Sun et al. (2019a).

A YOLO também inspira trabalhos como o de Wong et al. (2019), que usa a YOLO Nano, utilizando dos princípios da família de redes YOLO e de um conceito chamado síntese generativa (do inglês, *generative synthesis*). Introduzido em Wong et al. (2018) esse conceito explora a ideia de que máquinas geradoras podem automaticamente produzir redes neurais profundas, com arquiteturas eficientes que se enquadrem em requisitos e restrições especificadas por humanos.

Além disso, a YOLO Nano se utiliza de módulos com macroarquiteturas de projeção de expansão-projeção residual única (definidas como PEP), em conjunto com macroarquiteturas de expansão-projeção (definidas como EP). Essas macroarquiteturas permitem reduções em arquitetura e complexidade computacional. Elas possuem camadas de projeção com filtro convolucional de 1x1, que projeta canais de saída em tensores de saída com baixa dimensionalidade, camadas de expansão com filtro convolucional de 1x1, que expande o número de canais de alta dimensionalidade e camadas de convolução em profundidade que realizam convoluções espaciais com um filtro diferente em cada um dos canais de saída da camada de expansão.

A macroarquitetura de atenção leve totalmente conectada (FCA, ou do inglês, *light-weight fully-connected attention*) também é incorporada.Ela é formada por duas camadas totalmente conectadas que facilitam a recalibração de características e interdependências dinâmicas entre os canais, com base em informações globais, para prestar mais atenção às características de maior importância (WONG et al., 2019). Os resultados desse grupo de macroaquiteturas demonstraram que a YOLO Nano, em comparação com as redes Tiny YOLOv2 e Tiny YOLOv3, conseguiu reduzir o tamanho do modelo em 15.1x e 8.3x, aumentar a média de precisão média (mAP, ver subseção 2.2.2.4) de 12% e 10.7% e reduzir a quantidade de operações por inferência em 34% e 17%, respectivamente.

Outro estudo relevante para a presente pesquisa é o de Mao et al. (2019), a Mini-YOLOv3, que modifica a rede *backbone* de extração de características da YOLOv3, a *darknet-53*, para

aprimorar a sua velocidade. Visto que essa modificação traria impacto negativo na acurácia do modelo, é proposto um conjunto de módulos denomidado Rede Pirâmide de Característica em Várias escalas (MSFPN, ou do inglês, *Multi-Scale Feature Pyramid Network*). Neste conjunto estão implementados os seguintes módulos: módulo *concat*, módulo codificador-decodificador (do inglês, *encoder-decoder module*) e o módulo de fusão de característica (do inglês, *feature fusion module*).

O módulo concat funde três mapas de características recebidos do *darknet* para gerar um mapa base de características, já o módulo *encoder-decoder* gera um grupo de características multi-escalar e o módulo de fusão de características agrega os resultados dos módulos descritos anteriormente em uma pirâmide de características, mantendo a precisão na detecção de objetos mesmo após a redução do *backbone*. Além de propor um conjunto de módulos único, a Mini-YOLOv3 mescla métodos de redes como a aplicação a convolução separada por profundidade da MobileNet (ver subseção 2.1.1.2.2).

Tabela 4 – Resultados de desempenho de trabalhos utilizando modelo reduzido de aparência profunda usando o dataset PASCAL VOC 2007

Modelo	**mAP**	**Tamanho do modelo (megabytes)**
Mixed YOLOv3-LITE (ZHAO et al., 2020)	48,25%	20,5
YOLO Nano (WONG et al., 2019)	**69,1%**	4,0
Tinier-YOLO (FANG et al., 2019)	65,7%	8,9
YOLO-LITE (HUANG et al., 2018)	33,81%	**2,3**
Tiny YOLOv2 (REDMON; FARHADI, 2017)	57,1%	60,5
Tiny YOLOv3 (REDMON; FARHADI, 2018)	58,4%	33,4

Fonte: Adaptado de Zhao et al. (2020), Wong et al. (2019), Fang et al. (2019) e Huang et al. (2018).

4

Abordagem Desenvolvida

Neste capítulo, será mostrado o desenvolvimento do processo de análise de um modelo de aprendizagem profunda para ser usado com o método de rastreamento SmartSORT, criado por Meneses (2019). Este método foi escolhido pois ao ser avaliado no *benchmark* MOT Challenge 2016 mostrou-se ser superior em frequência de processamento, e sua acurácia foi similar em comparação a rastreadores estado da arte (ver Tabela 3). Seu nível de adaptabilidade também foi constatado, pois Meneses (2019) avalia o rastreador em três cenários de experimentação diferentes.

É importante ressaltar que o SmartSORT, diferente dos rastreadores aos quais foi comparado, não baseia-se em modelos de aprendizado profundo. Como visto nos capítulos anteriores, estes modelos demandam processamento e armazenamento, mas trazem ótimos resultados. Por isso, é interessante investigar a redução desses modelos e utilizá-los com o SmartSORT, para que seu uso seja possível em dispositivos limitados em *hardware*.

Para a presente pesquisa, o modelo a ser utilizado será uma das versões que fazem parte da família YOLO. A escolha do modelo YOLO se deu diante a sua popularidade por ser um detector de objetos que possui valores de acurácia e velocidade balanceados e a seus princípios ao criar um arquitetura que possui apenas um estágio ao detectar e classificar objetos.

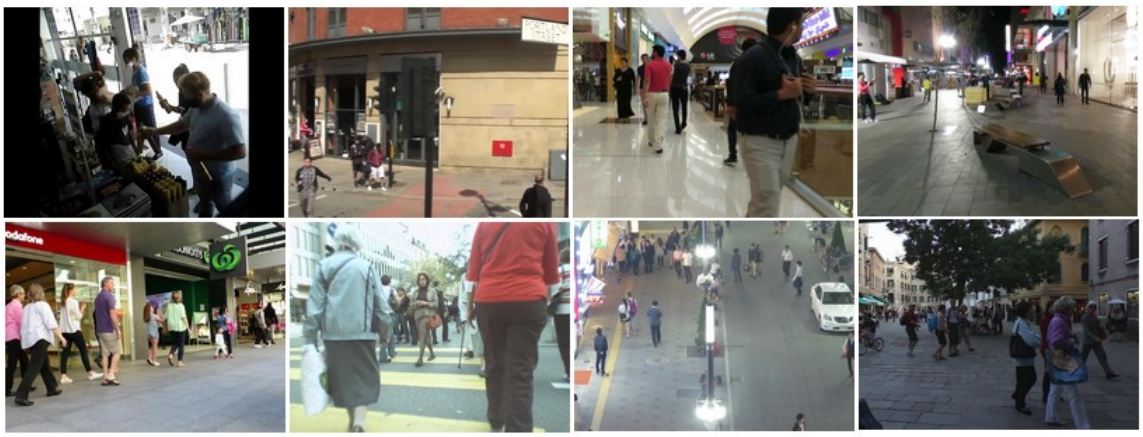
Os modelos SSD também possuem apenas um estágio de detecção e classificação assim como os modelos YOLO, porém possui deficiência em detectar objetos menores na imagem. Tanto modelos baseados na R-CNN quanto os modelos baseados na SSD, possuem uma estrutura para extração de características baseados na rede VGG que em comparação com a da YOLO, é mais complexa, tornando a velocidade da detecção mais lenta. Além disso, o YOLO utiliza esse processo durante o tempo de treinamento e teste, calculando implicitamente dados expecificamente de classes (MAO et al., 2019; WONG et al., 2019; ZHANG et al., 2020).

4.1 Treinamento do modelo

4.1.1 Dataset

Para treinar e validar o modelo, utilizou-se um *dataset* customizado com imagens dos vídeos da MOT Challenge 2016 (MOT16) e de um ambiente comercial controlado. A escolha de uso do *dataset* foi devido ao fato de que o SmartSORT foi avaliado a partir do MOT16, logo, existe a possibilidade de avaliar a hipótese de que se ao treinar o modelo na mesma base de imagens que o rastreador foi avaliado, suas métricas aumentariam. As imagens do ambiente comercial, além de enriquecerem o modelo, possibilitam sua transformação em um produto, já que seu treino foi realizado com imagens de cenário comercial real. Ambos os cenários possuem alto fluxo de pedestres e de constante oclusão, presença de distratores e variações de iluminação.

Figura 22 – Algumas imagens que compõe o *dataset* criado para este trabalho. Além de imagens de gravações próprias, são utilizadas imagens do *benchmark* MOT Challenge 2016.



Fonte: A própria autora.

No total o *dataset* possui 15466 imagens, sendo 5316 do MOT16 e 10150 do ambiente comercial. Ele foi dividido da seguinte forma: 80% das imagens para treinamento e 20% para teste. Essa divisão também aconteceu para cada uns dos cenários (ver Tabela 5).

Tabela 5 – Divisão do *dataset* a ser utilizado no treinamento e teste do modelo de aprendizagem profunda

Treinamento			**Teste**			**Total**
MOT16	Ambiente Comercial	Total	MOT16	Ambiente Comercial	Total	15.466
4.254	8.120	12.374	1.062	2.030	3.092	

4.1.1.1 MOT16

O MOT16 contém 14 sequências de vídeos que registram ambientes com deslocamento de múltiplos pedestres, filmados com câmeras estáticas e móveis. Para as análises feitas no presente trabalho, foram utilizadas apenas as imagens de treinamento do MOT16, correspondendo a 7 sequências de vídeos. No total, essas sequências possuem 110407 *bounding boxes*.

Cada sequência de vídeos desse *benchmark* possui arquivos de anotações com marcações da localização de cada pedestre com o seguinte formato:

$$M = [f, i, x, y, w, h, s, c, v], \tag{4.1}$$

onde f é o número correspondente *frame* na sequência de vídeo em questão, i é o identificador único da marcação, (x, y) são as coordenadas em duas dimensões do canto superior esquerdo da marcação, w e h representam, respectivamente, a sua largura e altura, s é uma variável de controle interno do *benchmark*. Para a construção do *dataset* usado neste trabalho, essa variável foi considerada pois ela indica que aquela marcação vai ser ignorada ou não na avaliação do *benchmark*. A variável c, que representa a classe a qual a marcação é identificada e para este trabalho, só foram utilizadas as marcações das classes: pedestre e pessoa estática. Por fim, v corresponde à taxa de visibilidade do pedestre, representada em valores de 0 a 1. Para essa variável, em casos específicos, ela foi considerada pois alguns objetos com visibilidade tinham visibilidade abaixo de 0.3 e esses objetos não estavam ao menos presente na imagem.

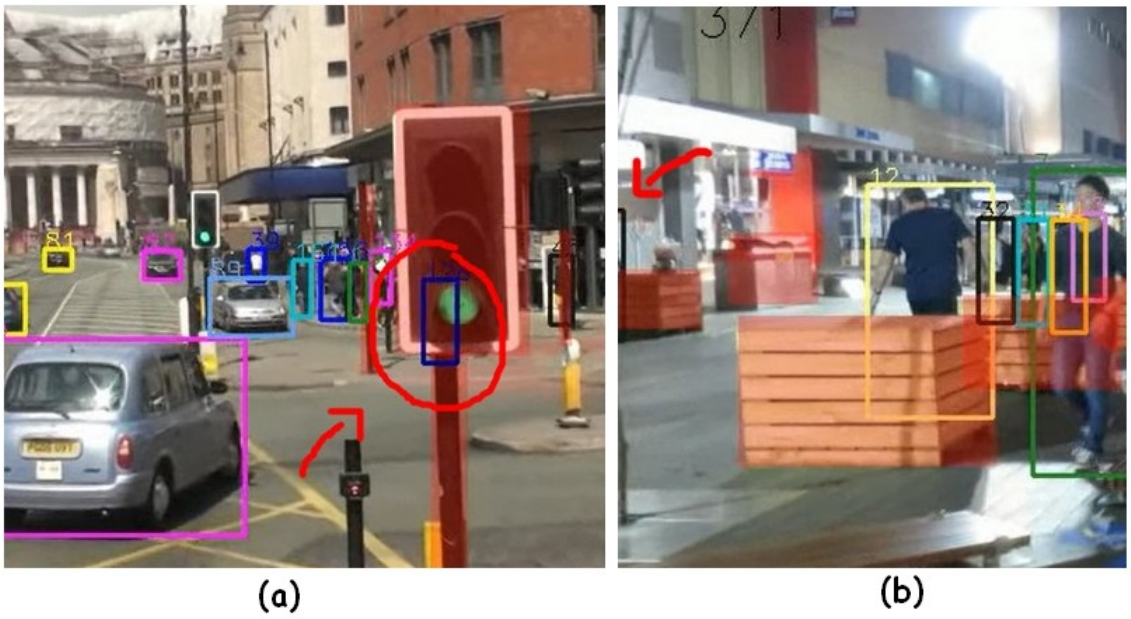
4.1.1.2 Ambiente comercial

As imagens coletadas para este cenário foram registradas numa loja de varejo no centro da cidade de Aracaju. Foi colocada uma placa ESP32-CAM, alimentada por um *powerbank*. A placa foi programada com um algoritmo[1] que tira fotos e as transforma em vídeos com duração e resolução variáveis. Para este trabalho foram gravados vídeos de 30 minutos com resolução de 800x600 e 24 FPS. As gravações possuem cerca de 7h de vídeo, gravadas no período entre 8h às 17h. Existem um total de 57216 *bounding boxes*.

Para marcação dessas imagens foi utilizado um *software* chamado ViTBAT[2]. O programa funciona apenas em sistema operacional *Windows* e executa em cima de outro *sofware* compilado com MATLAB [3]. Existem outras ferramentas multiplataformas disponíveis como a labelme[4] e LabelImgTool[5], e alternativas online[6].

[1] $https://github.com/jameszah/ESP32-CAM-Video-Recorder-junior$
[2] $https://vitbat.weebly.com/$
[3] *Software* de alta perfomance voltado para cálculo numérico. Site oficial: $https://www.mathworks.com/products/matlab$.
[4] $https://github.com/wkentaro/labelme$
[5] $https://github.com/lzx1413/LabelImgTool$
[6] $https://cvat.org/auth/login$, $https://github.com/microsoft/VoTT$, $https://labelbox.com/$

Figura 23 – Exemplos de oclusões totais em videos do MOT16. As marcações fornecidas pelo *benchmark* indicam que elas possuem visbilidade de 0.1 a 0.3 mas os objetos sequer aparecem na imagem. Esses casos foram desconsiderados no *dataset* criado. (a) imagem retirada da sequência MOT16-13, frame 275 e id 126. (b) imagem retirada da sequência MOT16-10, frame 364 e id 45.



Fonte: A própria autora.

O ViTBAT foi escolhido para este trabalho pois ele permite abrir vídeos ao invés de um conjunto de imagens. Após feita a marcação do objeto, esta marcação se mantém até o *frame* desejado, não se fazendo necessário marcar o objeto novamente. Também permite rotulação de objetos utilizando retângulos e pontos, bem como rotular comportamentos e grupos de objetos. Após salvar as marcações feitas, o programa exporta essas marcações em arquivos com seus dados e respectivo *frame* de vídeo. O formato das marcações do ViTBAT possuem o seguinte formato:

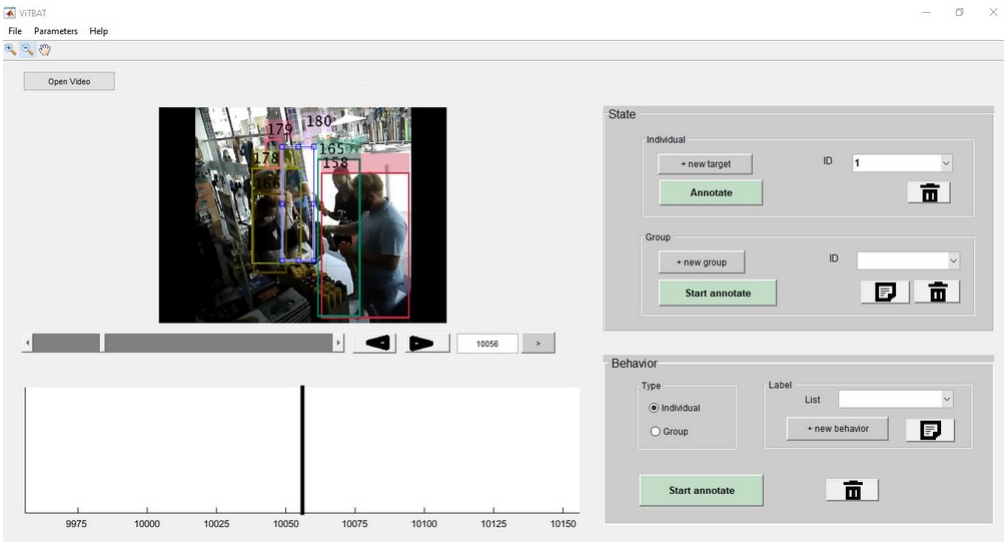
$$M = [f, i, x, y, w, h], \tag{4.2}$$

onde f é o número correspondente *frame* na sequência de vídeo em questão, i é o identificador único da marcação, (x, y) são as coordenadas em duas dimensões do canto superior esquerdo da marcação, w e h representam, respectivamente, a sua largura e altura.

4.1.1.3 Formatação do *dataset*

Os modelos da família YOLO possuem um formato específico para leitura das coordenadas de uma detecção numa imagem ao realizar o treinamento e teste utilizando o *framework Darknet*.

Figura 24 – Tela inicial do *software* ViTBAT



Fonte: A própria autora.

Sendo uma detecção D, que possui o formato:

$$D = [c, x, y, w, h], \tag{4.3}$$

onde c é índice da classe que identifica a detecção (e.g pessoa, carro, cachorro). Esse índice é um inteiro que corresponde ao número da linha num arquivo de classes, cada linha contém o nome da classe por extenso. O índice começa de 0, então por exemplo: se o índice é 0 e na primeira linha do arquivo possui a palavra "pessoa", significa que a detecção que possui $c = 0$ é uma pessoa. Por fim, (x, y) são as coordenadas em duas dimensões do centro da detecção na imagem, w e h representam, respectivamente, sua largura e altura.

As detecções YOLO e marcações do MOT16 e do ambiente comercial correspondem à objetos na imagem, mas as variáveis de seus respectivos formatos diferem. Para este trabalho, precisou-se transformar as anotações desses dois cenários para que o modelo fosse treinado usando suas imagens e validado usando seus dados de *ground truth*.

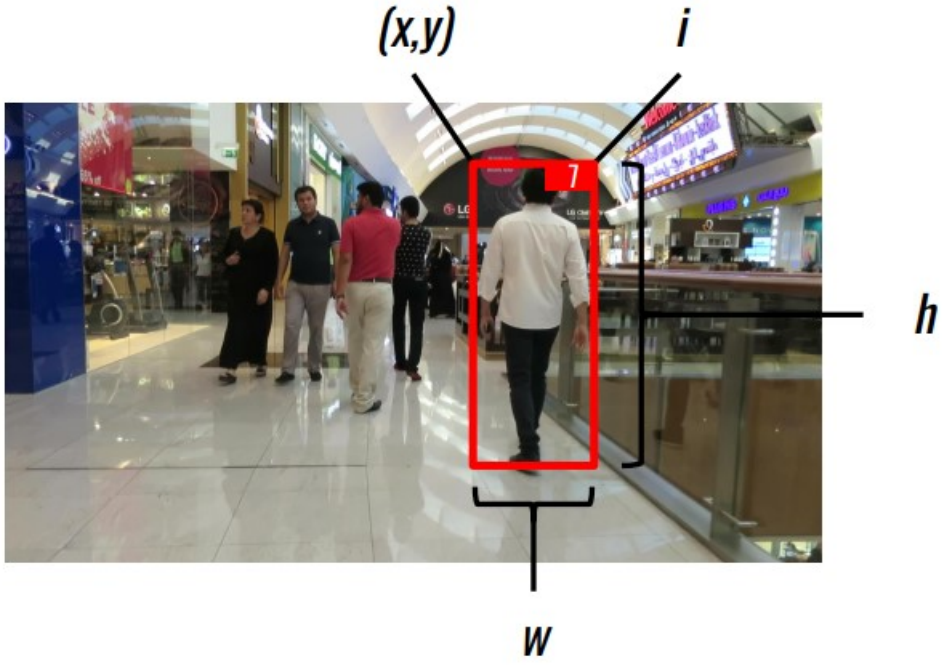
4.1.2 Resultados do treinamento

A versão do modelo da família YOLO a ser utilizada no presente trabalho, será a versão reduzida da YOLOv4, denominada YOLOv4-tiny. Esse modelo possui apenas 23,1MB. Ao invés de treinar do zero, foram utilizados pesos pré-treinados da YOLOv4-tiny que foram treinados até a 29a camada convolucional[7]. O treinamento do modelo foi feito usando GPU no Google Colab[8],

[7] *https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.conv.29*

[8] *https://colab.research.google.com*

Figura 25 – Representação de marcação fornecida pelo MOT Challenge e exportado do ViTBAT.



Fonte: Meneses (2019).

serviço de nuvem gratuito hospedado pelo Google, e o *framework Darknet*. Para configuração do modelo, foi alterado o número de classes (para apenas uma classe).

O modelo foi treinado em 4 etapas com o total de 10000 iterações. Os resultados de testes do modelo a cada 1000 iterações de treinamento são apresentados na Tabela 6. O modelo usado nos experimentos descritos nas seções a seguir foi o Trial 9.

Tabela 6 – Resultados de teste do modelo a cada 1000 iterações.

	Iterações	**Precisão**	***Recall***	**mAP**
Trial 1	0 - 2000	47%	41%	34.58%
Trial 2	0 - 2000	64%	51%	50.57%
Trial 3	2000 - 3000	64%	49%	47.85%
Trial 4	3000 - 4000	63%	51%	49.85%
Trial 5	4000 - 5000	68%	52%	52.27%
Trial 6	5000 - 6000	67%	52%	52.80%
Trial 7	6000 - 7000	68%	50%	51.16%
Trial 8	7000 - 8000	68%	52%	52.80%
Trial 9	8000 - 9000	69%	**55%**	**55.43**%
Trial 10	9000 - 10000	**73%**	53%	55.33%

4.2 Montagem dos experimentos

O SmartSORT foi avaliado no *benchmark* MOT Challenge 2016 seguindo as métricas CLEAR (ver subseção 2.2.2.2) e conseguiu resultados compatíveis com outros projetos estado da arte (Tabela 3). Infelizmente, não foi possível conseguir as anotações de marcação de *ground truth* que foram usadas para as sequências de vídeo de teste do MOT16 e comparadas em Meneses (2019). Sendo assim, a avaliação foi feita nas sequência de vídeos de teste do MOT16 usando o YOLOv4-tiny original e depois usando o YOLOv4-tiny treinado com *dataset* criado para este trabalho.

Inicialmente, foram colhidos dados de detecções por cada *frame* dos vídeos do MOT16 dos modelos YOLOv4-tiny original e YOLOv4-tiny treinado, já transformados no formato que o *benchmark* requer (ver Equação 4.1). As detecções foram coletadas direto da placa Raspberry Pi 4 (Figura 26) num ambiente virtual *python*. As detecções foram feitas carregando os modelos no *framework* OpenCV[9].

Figura 26 – Raspberry Pi 4 utilizada nos experimentos deste trabalho.

Fonte: A própria autora.

Após colhidos e formatados, os dados das detecções foram passados para o rastreador do SmartSORT, e a partir dos seus resultados, foram geradas as métricas. As métricas foram calculadas usando o TrackEval[10], algoritmo oficial que avalia resultados de rastreadores em

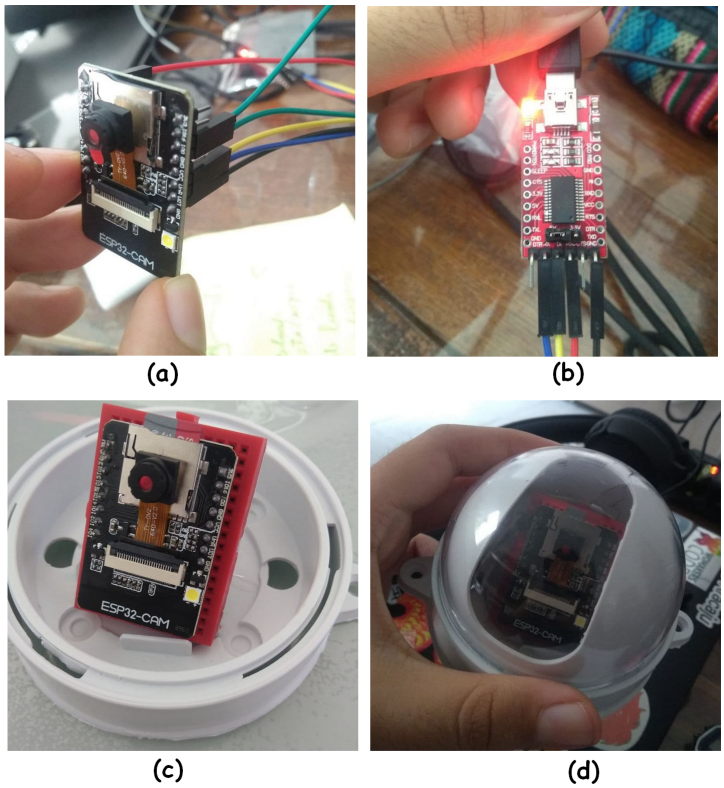
[9] *https://opencv.org/releases/*

[10] *https://github.com/JonathonLuiten/TrackEval/blob/master/docs/MOTChallenge-Official/Readme.md*

benchmarks como o MOT Challenge e outros.

Para o cáculo de velocidade de inferência, um valor *s* em segundos fracionados do momento atual é obtido antes de passar a imagem para o modelo para obter os dados das detecções, após esse processo, outro valor *e* em segundos fracionados do momento atual é obtido e é feita a subtração de *e* por *s*, resultando no tempo de inferência. O cáculo de *frames* por segundo é obtido de forma parecida, ao terminar a execução do modelo em um vídeo, os valores de FPS calculados em cada *frame* são somados e divididos pela quantidade total de *frames* no vídeo.

Figura 27 – *Setup* com ESP32-CAM utilizada para gravação em ambiente comercial controlado. O módulo ESP foi programado usando conversor FTDI através de suas portas GPIO (a, b). Após programar a ESP, ela foi fixada dentro de uma câmera falsa do tipo *bullet* (c, d) e colocada no ambiente comercial.



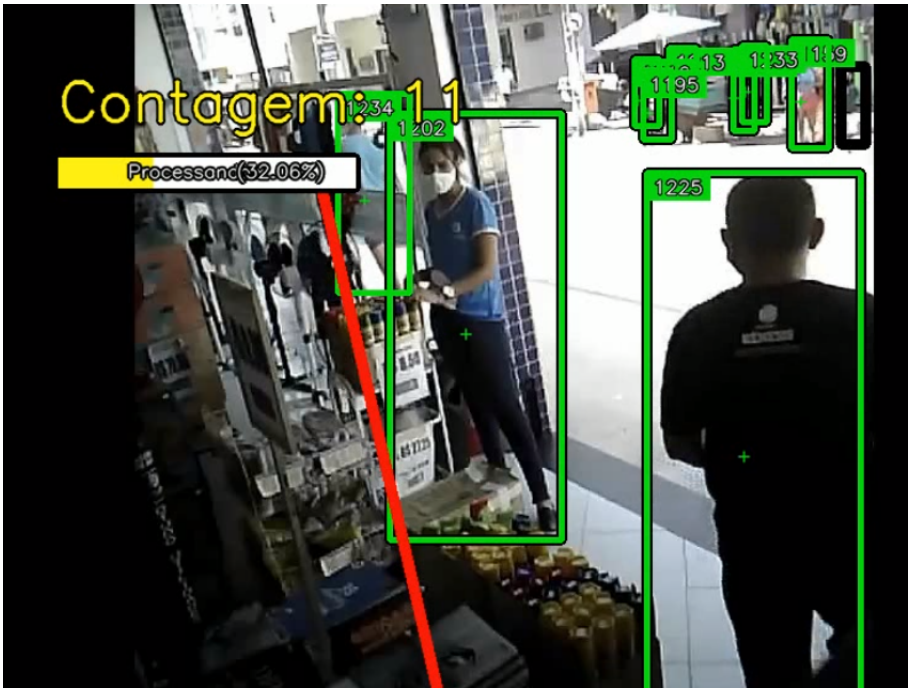
Fonte: A própria autora.

Também foi aplicada a experimentação de contagem de pessoas em um dos vídeos do ambiente comercial controlado usado no *dataset* customizado. Para a gravação dos vídeos foi utilizada uma ESP32-CAM. Ao programar este módulo, foram utilizadas portas GPIO ligadas a um conversor USB FTDI FT232RL (Figura 27). O algoritmo[11] executado na ESP32-CAM tirava fotos consecutivas e as salvava em formato de vídeo com resolução 800x600 a cada 30 minutos.

[11] *https://github.com/jameszah/ESP32-CAM-Video-Recorder-junior*

Este experimento de contagem foi realizado num *laptop* usando apenas CPU e na Raspberry Pi. A contagem é efetuada assim que um objeto cruza a fronteira delimitada na imagem (ver Figura 28). A verificação de cruzamento de fronteira é validada usando os centros geométricos das duas últimas detecções na trajetória de um objeto. Ao cruzar a fronteira, o contador é incrementado.

Figura 28 – Ilustração de resultado de contagem em cenário de ambiente comercial e marcações com detecções de pessoas.



Fonte: A própria autora.

5

Resultado e discussões

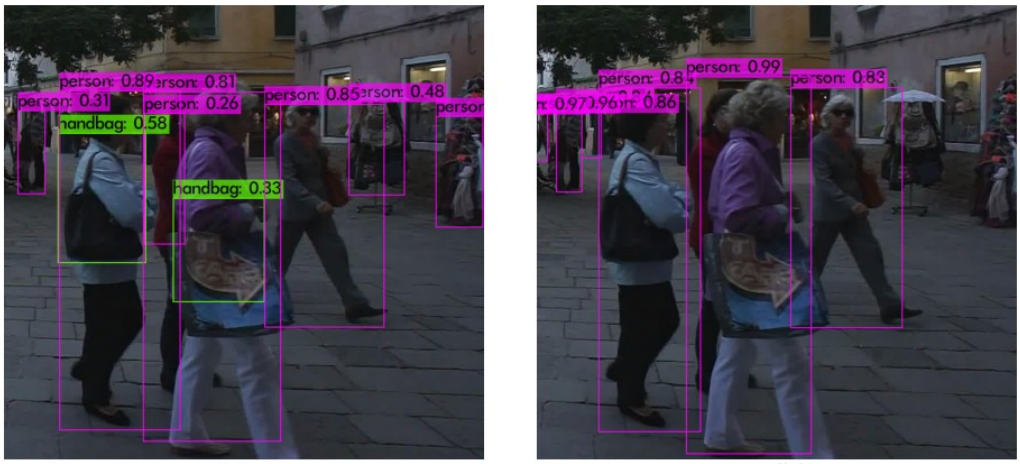
Após o treinamento do modelo, coleta de detecções e cálculos de tragetórias, o modelo original YOLOv4-tiny e sua versão após o treinamento do *dataset* customizado obtiveram os mesmos resultados nas métricas CLEAR para as sequências de vídeo de treinamento, demonstrando que o treinamento não modificou a acurácia no rastreador nem resolveu a alta taxa de troca de identificadores e fragmentação de trajetória.

Tabela 7 – Utilizando as detecções obtidas pelo YOLOv4-tiny antes e depois do treinamento, foram obtidos valores correspondentes às métricas CLEAR. Os valores obtidos com o uso desses dois modelos foram os mesmos. O algoritmo TrackEval disponibiliza os valores obtidos de um rastreador exemplo, o MPNTrack[1]

	↑ MOTA	↑ MOTP	↓ IDS	↓ FM
SmartSORT usando YOLOv4-tiny	11,40%	75%	225	447
MPNTrack	**59%**	**90%**	**63**	**115**

A execução do SmartSORT na Raspberry Pi 4 antes e depois do treinamento do modelo obtiveram tempo médio de inferência do modelo de cerca de 500ms executando à 0,77636 FPS.

Figura 29 – Comparação de detecções entre o modelo YOLOv4-tiny antes e depois do treinamento. Em (a), o modelo antes de ser treinado atribui menor porcentagem de confiança em objetos que possuem a classe avaliada (classe pessoa) e classifica objetos distrativos nessa mesma classe. Em (b), após o treinamento, é possível visualizar a diferença.



Fonte: A própria autora.

Apesar de não ter ajudado o rastreador a alcançar métricas melhores, o modelo treinado com *dataset* customizado aumentou cerca de 24% de mAP nas detecções (ver Figura 29) em apenas 10 mil iterações no treinamento. Esse dado foi calculado utilizando o *framework* darknet com a GPU do ambiente do Google Colab.

Para experimentação de contagem de pessoas, foi utilizado uma gravação do ambiente comercial usando para criação do *dataset* com duração de 6min. O total de pessoas contadas manualmente foi de 20. Ao aplicar o YOLOv4-tiny, o algoritmo soma um total de 6 pessoas. Após o treinamento, o modelo soma foi igual a esperada, 20 pessoas.

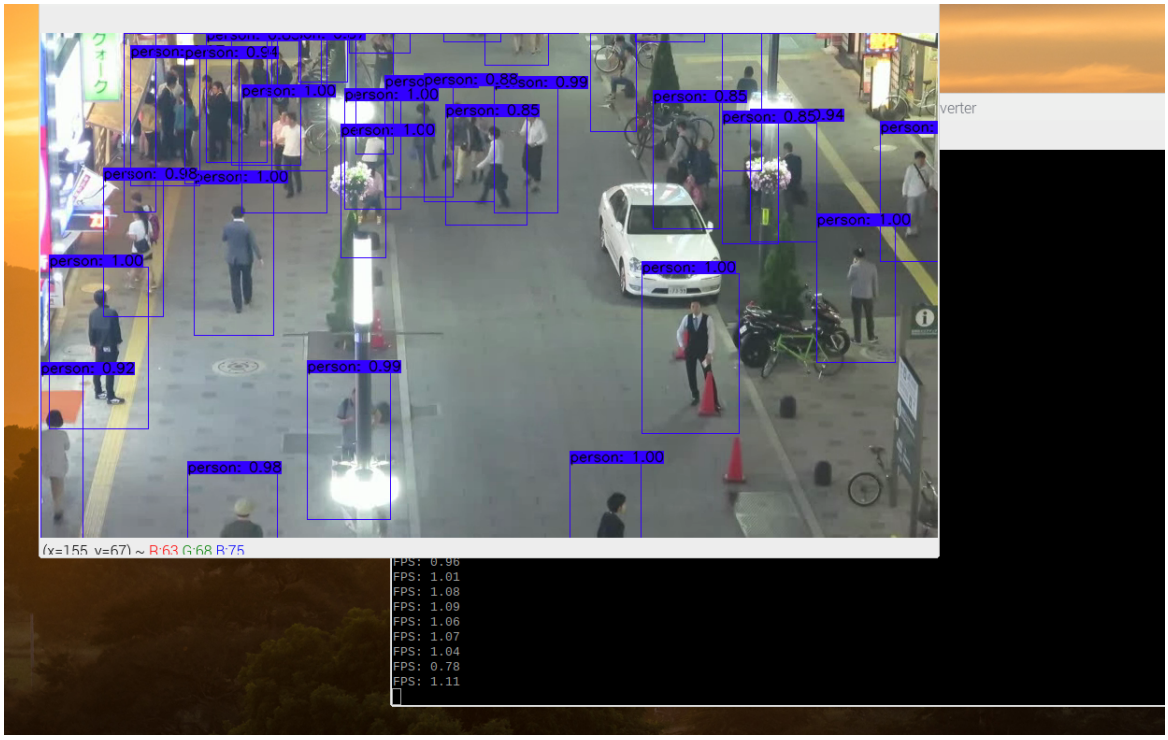
Para o YOLOv4-tiny antes do treinamento, o erro pode estar relacionado a sua falta de acurácia nesse cenário específico. Muitos objetos do tipo pedestre não são localizados ou outros objetos são classificados como pedestres, causando falsos negativos e positivos, consequentemente, sua trajetória não é rastreada. Nesse caso existe muita troca de identificadores de alvos. O tempo de inferência é de 50ms, melhor que na experimentação de métricas do MOT Challenge.

Ao utilizar o modelo após o treino, além de diminuir o tempo de inferência para 40ms o modelo conseguiu fazer a contagem correta de acordo com a contagem feita manualmente. Reproduzindo essa experimentação na Raspberry, o resultado com o modelo treinado também foi igual ao esperado porém para processar todo o video, com o total de 9000 *frames*, foi preciso cerca de 3h. O tempo de inferência foi de cerca de 800ms, executando a 0,7 FPS.

5.1 Experimentação usando Tensorflow Lite

Todas as execuções dos modelos na Raspberry obtiveram baixas taxas de FPS, sendo assim, foi elaborada uma experimentação fazendo a conversão de pesos do modelo YOLOv4-tiny para pesos no formato do *framework* Tensorflow Lite[2]. Utilizando ferramentas do Tensorflow Lite, é possível realizar diversos tipos de otimização como: redução de tamanho do modelo, menos uso de memória do dispositivo, redução do tempo de inferência e técnicas como a quantização[3]. Ao fazer a conversão e calcular o valor médio de FPS, foi obtido um pequeno aumento de 0,42, resultando em 1,2 FPS. Além disso o tamanho do modelo foi reduzido para 11,3MB.

Figura 30 – Medição de FPS ao executar o modelo YOLOv4-tiny treinado utilizando Tensorflow Lite.



Fonte: A própria autora.

[2] $https://www.tensorflow.org/lite/guide?hl=pt-br$
[3] $https://www.tensorflow.org/model_optimization/guide/quantization/post_training?hl=pt-br$

6

Conclusão e Trabalhos Futuros

Na execução deste trabalho foi possível investigar a utilização de modelos de aprendizagem profunda estado da arte como ferramentas para localização de objetos e extração de suas características Percebeu-se que a utilização de modelos pré-treinados facilita o uso desses modelos por não precisar treinar o modelo do zero. O tamanho desses modelos se torna um problema para sua utilização em dispositivos com *hardware* limitado, já que demandam tempo de processamento e em espaço em memória, limitando seu uso a infraestruturas mais caras e menos acessíveis.

Para que o uso desses modelos se tornem acessíveis, alem do uso de técnicas de compressão de dados existem modelos com estruturas menores mas que podem gerar resultados compatíveis com o estado da arte. Trabalhos que se utilizam desses modelos menores, mostrados no seção 3.3, mesclam técnicas para que as características dos objetos das imagens sejam alcançáveis na maioria das camadas (HUANG et al., 2017; FANG et al., 2019).

A utilização desses modelos em dispositivos limitados em *hardware* é uma área difícil que possui uma grande curva de aprendizagem. Isso foi percebido ao realizar esse trabalho. Apesar do bom desempenho em infraestruturas médias, realizar inferências e processar imagens em tempo real requer muito processamento para dispositivos limitados.

Neste trabalho, por motivos externos, não foi possível avançar na análise de modelos profundos em sistemas com recursos limitados e explorar técnicas de compressão desses modelos. Apesar do ganho em acurácia nos cenários específicos que o algoritmo SmartSORT foi avaliado, isso não ajudou a melhorar suas métricas como rastreador.

Segundo Huang et al. (2018), técnicas de compressão de dados como poda e o uso de normalização em lote não ajudam no aumento de acurácia para o cenário que envolve tecnologias com recursos limitados. Mas pode ser que a utilização dessas técnicas ajudem a comprimir o modelo de forma que seu tempo de inferência seja mais rápido e o modelo não precise de muitos parâmetros. Como foi mostrado na seção 5.1, ao otimizar o modelo usando Tensorflow Lite,

além de diminuir o tamanho do modelo, houve um ganho em FPS mesmo o executando em um ambiente limitado em recursos.

Como trabalhos futuros, existem diversas possibilidades. O modelo YOLOv4-tiny foi treinado em apenas 10 mil iterações. Seu treinamento em mais iterações e mais imagens, pode enriquecer seu conhecimento e ajudar no processo de cálculo de trajetórias e rastreamento de objetos. Além disso, o uso de técnicas de compressão de dados podem ajudar a equiparar a relação de sua parametrização e tamanho. Também é possível realizar o treinamento do YOLOv4-tiny em outros modelos como o MobileNet, SequeezeNet ou SSDLite, analisando a troca de informações dos filtros das camadas convolucionais.

Referências

ALASHKAR, T. Real-time lane and car detection on highway using vision system. 12 2016. Citado na página 32.

APTE, M.; MANGAT, S.; SEKHAR, P. Yolo net on ios. *CS231n. Stanford University*, 2017. Citado na página 48.

BECHTEL, M. G. et al. Deeppicar: A low-cost deep neural network-based autonomous car. In: IEEE. *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. [S.l.], 2018. p. 11–21. Citado na página 16.

BERNARDIN, K.; STIEFELHAGEN, R. Evaluating multiple object tracking performance: the clear mot metrics. *EURASIP Journal on Image and Video Processing*, Springer, v. 2008, p. 1–10, 2008. Citado 3 vezes nas páginas 11, 33 e 46.

BLALOCK, D. et al. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020. Citado na página 27.

BOCHKOVSKIY, A. *YOLOv4*. 2020. <https://github.com/AlexeyAB/darknet>. Citado na página 24.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. Citado na página 23.

BRUSCHI, P. et al. A novel integrated smart system for indoor air monitoring and gas recognition. In: IEEE. *2018 IEEE international conference on smart computing (SMARTCOMP)*. [S.l.], 2018. p. 470–475. Citado na página 40.

BUCILUă, C.; CARUANA, R.; NICULESCU-MIZIL, A. Model compression. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2006. p. 535–541. Citado na página 28.

CHEN, L. et al. Aggregate tracklet appearance features for multi-object tracking. *IEEE Signal Processing Letters*, IEEE, v. 26, n. 11, p. 1613–1617, 2019. Citado 2 vezes nas páginas 46 e 47.

CHEN, L. et al. Online multi-object tracking with convolutional neural networks. In: IEEE. *2017 IEEE International Conference on Image Processing (ICIP)*. [S.l.], 2017. p. 645–649. Citado 4 vezes nas páginas 13, 14, 46 e 47.

CIAPARRONE, G. et al. Deep learning in video multi-object tracking: A survey. *Neurocomputing*, Elsevier, v. 381, p. 61–88, 2020. Citado 7 vezes nas páginas 13, 30, 31, 33, 34, 37 e 38.

CUAN, B.; IDRISSI, K.; GARCIA, C. Deep siamese network for multiple object tracking. In: IEEE. *2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP)*. [S.l.], 2018. p. 1–6. Citado 3 vezes nas páginas 14, 46 e 47.

CURVELLO, A. *Apresentando o módulo ESP8266*. 2015. <https://www.embarcados.com.br/modulo-esp8266/>. Acesso em: 2021-03-02. Citado na página 40.

CURVELLO, A. *ESP32 – Um grande aliado para o Maker IoT*. 2019. <https://www.filipeflop.com/blog/esp32-um-grande-aliado-para-o-maker-iot/>. Acesso em: 2021-03-03. Citado na página 40.

DENNIS, A. K. *Raspberry Pi Computer Architecture Essentials*. [S.l.]: Packt Publishing, 2016. Citado na página 38.

DOKIC, K.; MARTINOVIC, M.; RADISIC, B. Neural networks with esp32-are two heads faster than one? In: IEEE. *2020 6th Conference on Data Science and Machine Learning Applications (CDMA)*. [S.l.], 2020. p. 141–145. Citado na página 40.

EBRAHIMI, M. S.; ABADI, H. K. Study of residual networks for image recognition. *arXiv preprint arXiv:1805.00325*, 2018. Citado na página 21.

EMAMI, P. et al. Machine learning methods for solving assignment problems in multi-target tracking. *arXiv preprint arXiv:1802.06897*, 2018. Citado na página 13.

Espressif. *About Espressif*. 2021. <https://www.espressif.com/en/company/about-us/who-we-are>. Acesso em: 2021-03-01. Citado na página 40.

FANG, W.; WANG, L.; REN, P. Tinier-yolo: A real-time object detection method for constrained environments. *IEEE Access*, IEEE, v. 8, p. 1935–1944, 2019. Citado 5 vezes nas páginas 14, 42, 48, 50 e 63.

FERGUSON, M. et al. Automatic localization of casting defects with convolutional neural networks. In: IEEE. *2017 IEEE international conference on big data (big data)*. [S.l.], 2017. p. 1726–1735. Citado na página 20.

GIRSHICK, R. Fast r-cnn. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 1440–1448. Citado na página 21.

GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2014. p. 580–587. Citado na página 21.

GONZALEZ-HUITRON, V. et al. Disease detection in tomato leaves via cnn with lightweight architectures implemented in raspberry pi 4. *Computers and Electronics in Agriculture*, Elsevier, v. 181, p. 105951, 2021. Citado 3 vezes nas páginas 15, 16 e 39.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado 2 vezes nas páginas 18 e 19.

GUO, Y. et al. Depthwise convolution is all you need for learning multiple visual domains. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2019. v. 33, p. 8368–8375. Citado na página 24.

HAN, S. et al. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015. Citado 2 vezes nas páginas 27 e 28.

HE, K. et al. Deep residual learning for image recognition. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016. Citado na página 21.

HE, K. et al. Identity mappings in deep residual networks. In: SPRINGER. *European conference on computer vision*. [S.l.], 2016. p. 630–645. Citado na página 21.

HE, M. et al. Fast online multi-pedestrian tracking via integrating motion model and deep appearance model. *IEEE Access*, IEEE, v. 7, p. 89475–89486, 2019. Citado 5 vezes nas páginas 13, 14, 32, 46 e 47.

HINTON, G.; VINYALS, O.; DEAN, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. Citado 2 vezes nas páginas 28 e 29.

HOWARD, A. et al. Searching for mobilenetv3. In: *Proceedings of the IEEE International Conference on Computer Vision*. [S.l.: s.n.], 2019. p. 1314–1324. Citado na página 25.

HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. Citado 2 vezes nas páginas 15 e 24.

HU, R. et al. Multiple cues association for multiple object tracking based on convolutional neural network. In: . [S.l.: s.n.], 2019. Citado na página 14.

HUANG, G. et al. Densely connected convolutional networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 4700–4708. Citado 2 vezes nas páginas 48 e 63.

HUANG, R.; PEDOEEM, J.; CHEN, C. Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers. In: IEEE. *2018 IEEE International Conference on Big Data (Big Data)*. [S.l.], 2018. p. 2503–2510. Citado 3 vezes nas páginas 48, 50 e 63.

IANDOLA, F. N. et al. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. Citado na página 15.

IANDOLA, F. N. et al. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. Disponível em: <http://arxiv.org/abs/1602.07360>. Citado 2 vezes nas páginas 25 e 26.

INTELLABS. *Knowledge Distillation*. 2018. <https://intellabs.github.io/distiller/knowledge_distillation.html>. Citado na página 29.

K, F. *Apresentando o módulo ESP8266*. 2018. <https://www.fernandok.com/2018/05/nodemcu-esp8266-detalhes-e-pinagem.html>. Acesso em: 2021-03-02. Citado na página 40.

KAIM, R. *Illustrated: 10 CNN Architectures*. 2019. Disponível em: <https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#e4b1>. Acesso em: 2020-03-09. Citado na página 47.

KARUNASEKERA, H.; ZHANG, H.; WANG, H. Real time multiple object tracking using deep features and localization information. In: IEEE. *2019 IEEE 15th International Conference on Control and Automation (ICCA)*. [S.l.], 2019. p. 332–337. Citado 2 vezes nas páginas 46 e 47.

KHAN, G.; TARIQ, Z.; KHAN, M. U. G. Multi-person tracking based on faster r-cnn and deep appearance features. In: *Visual Object Tracking in the Deep Neural Networks Era*. [S.l.]: IntechOpen, 2019. Citado 2 vezes nas páginas 46 e 47.

KHUDHAIR, A. B.; GHANI, R. F. Iot based smart video surveillance system using convolutional neural network. In: IEEE. *2020 6th International Engineering Conference "Sustainable Technology and Development"(IEC)*. [S.l.], 2020. p. 163–168. Citado na página 16.

KOMPELLA, R. *Tap into the dark knowledge using neural nets — Knowledge distillation*. 2018. <https://towardsdatascience.com/knowledge-distillation-and-the-concept-of-dark-knowledge-8b7aed8014ac>. Citado na página 29.

KRISTIANI, E.; YANG, C.-T.; HUANG, C.-Y. isec: an optimized deep learning model for image classification on edge computing. *IEEE Access*, IEEE, v. 8, p. 27267–27276, 2020. Citado na página 15.

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 2 vezes nas páginas 18 e 19.

KUSHNIR, V.; KOMAN, B.; YUZEVYCH, V. Iot image recognition system implementation for blind peoples using esp32, mobile phone and convolutional neural network. In: IEEE. *2019 XIth International Scientific and Practical Conference on Electronics and Information Technologies (ELIT)*. [S.l.], 2019. p. 183–187. Citado na página 41.

LEAL-TAIXÉ, L.; CANTON-FERRER, C.; SCHINDLER, K. Learning by tracking: Siamese cnn for robust target association. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. [S.l.: s.n.], 2016. p. 33–40. Citado na página 46.

LEAL-TAIXÉ, L. et al. Tracking the trackers: an analysis of the state of the art in multiple object tracking. *arXiv preprint arXiv:1704.02781*, 2017. Citado na página 38.

LI, W.; MU, J.; LIU, G. Multiple object tracking with motion and appearance cues. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. [S.l.: s.n.], 2019. p. 0–0. Citado na página 32.

LIU, M. et al. Online multiple object tracking using confidence score-based appearance model learning and hierarchical data association. *IET Computer Vision*, IET, v. 13, n. 3, p. 312–318, 2018. Citado 3 vezes nas páginas 14, 46 e 47.

LIU, W. et al. Ssd: Single shot multibox detector. In: SPRINGER. *European conference on computer vision*. [S.l.], 2016. p. 21–37. Citado na página 22.

LUO, W. et al. Multiple object tracking: A literature review. *arXiv preprint arXiv:1409.7618*, 2014. Citado 3 vezes nas páginas 13, 30 e 44.

MAHMOUDI, N.; AHADI, S. M.; RAHMATI, M. Multi-target tracking using cnn-based features: Cnnmtt. *Multimedia Tools and Applications*, Springer, v. 78, n. 6, p. 7077–7096, 2019. Citado na página 13.

MANJARI, K.; VERMA, M.; SINGAL, G. Creation: Computational constrained travel aid for object detection in outdoor environment. In: IEEE. *2019 15th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*. [S.l.], 2019. p. 247–254. Citado na página 39.

MAO, Q.-C. et al. Mini-yolov3: real-time object detector for embedded applications. *IEEE Access*, IEEE, v. 7, p. 133529–133538, 2019. Citado 3 vezes nas páginas 48, 49 e 51.

MAQSOOD, M. et al. Transfer learning assisted classification and detection of alzheimer's disease stages using 3d mri scans. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 19, n. 11, p. 2645, 2019. Citado na página 14.

MENESES, M. C. C. *Rastreamento em Tempo Real de Múltiplos Objetos por Associação de Detecções*. Dissertação (Mestrado) — Universidade Federal de Sergipe, 2019. Citado 12 vezes nas páginas 14, 15, 16, 21, 31, 32, 43, 44, 45, 51, 56 e 57.

MITZEL, D.; LEIBE, B. Real-time multi-person tracking with detector assisted structure propagation. In: IEEE. *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. [S.l.], 2011. p. 974–981. Citado na página 31.

NVIDIA. *Meet Jetson*. 2021. <https://developer.nvidia.com/embedded-computing>. Acesso em: 2021-03-03. Citado na página 41.

Padilla, R.; Netto, S. L.; da Silva, E. A. B. A survey on performance metrics for object-detection algorithms. In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. [S.l.: s.n.], 2020. p. 237–242. Citado 5 vezes nas páginas 11, 33, 34, 35 e 37.

PAK, M.; KIM, S. A review of deep learning in image recognition. In: IEEE. *2017 4th international conference on computer applications and information processing technology (CAIPT)*. [S.l.], 2017. p. 1–3. Citado na página 20.

PATEL, K. *Architecture comparison of AlexNet, VGGNet, ResNet, Inception, DenseNet*. 2020. Disponível em: <https://towardsdatascience.com/architecture-comparison-of-alexnet-vggnet-resnet-inception-densenet-beb8b116866d>. Acesso em: 2020-03-10. Citado na página 20.

PENA, D. et al. Benchmarking of cnns for low-cost, low-power robotics applications. In: *RSS 2017 Workshop: New Frontier for Deep Learning in Robotics*. [S.l.: s.n.], 2017. p. 1–5. Citado na página 39.

PEREZ, X. *A Comparison of the New ESP32-S2 to the ESP32*. 2019. <https://maker.pro/esp8266/tutorial/a-comparison-of-the-new-esp32-s2-to-the-esp32>. Acesso em: 2021-03-03. Citado na página 40.

POKHREL, S. *Model Compression: needs and importance*. 2020. <https://towardsdatascience.com/model-compression-needs-and-importance-6e5913996e1>. Citado na página 27.

RAHMANIAR, W.; HERNAWAN, A. Real-time human detection using deep learning on embedded platforms: A review. *Journal of Robotics and Control (JRC)*, v. 2, n. 6, p. 462–468, 2021. Citado na página 41.

Raspberry Pi Foundation. *Raspberry Products*. 2021. <https://www.raspbian.org/>. Acesso em: 2021-03-01. Citado na página 39.

RASPBIAN. *About Raspbian*. 2021. <https://www.raspbian.org/>. Acesso em: 2021-03-01. Citado na página 39.

REDMON, J. *Darknet: Open Source Neural Networks in C*. 2013–2016. <http://pjreddie.com/darknet/>. Citado na página 23.

REDMON, J. *YOLO: Real-Time Object Detection*. 2013–2018. <https://pjreddie.com/darknet/yolo/>. Citado na página 24.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 779–788. Citado 3 vezes nas páginas 15, 22 e 23.

REDMON, J.; FARHADI, A. Yolo9000: better, faster, stronger. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 7263–7271. Citado 3 vezes nas páginas 15, 23 e 50.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. Citado 3 vezes nas páginas 15, 23 e 50.

REN, S. et al. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015. Citado na página 21.

RISTANI, E. et al. Performance measures and a data set for multi-target, multi-camera tracking. In: SPRINGER. *European Conference on Computer Vision*. [S.l.], 2016. p. 17–35. Citado 3 vezes nas páginas 11, 33 e 34.

SAGARIKA, G.; PRASAD, S. K.; KUMAR, S. M. Paddy plant disease classification and prediction using convolutional neural network. In: IEEE. *2020 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*. [S.l.], 2020. p. 208–214. Citado na página 40.

SAHA, S. *A Comprehensive Guide to Convolutional Neural Networks*. 2018. Disponível em: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. Acesso em: 2020-03-09. Citado na página 19.

SALVI, A. de A.; BARROS, R. C. An experimental analysis of model compression techniques for object detection. In: SBC. *Anais do VIII Symposium on Knowledge Discovery, Mining and Learning*. [S.l.], 2020. p. 49–56. Citado na página 27.

SANDLER, M. et al. Mobilenetv2: Inverted residuals and linear bottlenecks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2018. p. 4510–4520. Citado na página 24.

SANTOS, R. M.; FLÔR, D. E. Raspcar–carro robótico guiado remotamente. *Revista Mundi Engenharia, Tecnologia e Gestão (ISSN: 2525-4782)*, v. 4, n. 4, 2019. Citado na página 39.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. Citado na página 20.

SINGH, A. et al. Animal detection in man-made environments. In: *The IEEE Winter Conference on Applications of Computer Vision*. [S.l.: s.n.], 2020. p. 1438–1449. Citado na página 14.

SOUSA, M. C. F. *Visualizing the Fundamentals of Convolutional Neural Networks*. 2019. Disponível em: <https://towardsdatascience.com/visualizing-the-fundamentals-of-convolutional-neural-networks-6021e5b07f69>. Acesso em: 2020-03-09. Citado na página 19.

SUN, K. et al. Deep high-resolution representation learning for human pose estimation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2019. p. 5693–5703. Citado 2 vezes nas páginas 48 e 49.

SUN, S. et al. Deep affinity network for multiple object tracking. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, 2019. Citado 5 vezes nas páginas 13, 14, 31, 46 e 47.

TAN, M.; LE, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. Citado na página 14.

VELASCO-MONTERO, D. et al. Performance analysis of real-time dnn inference on raspberry pi. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Real-Time Image and Video Processing 2018*. [S.l.], 2018. v. 10670, p. 106700F. Citado 2 vezes nas páginas 15 e 16.

WANG, G. et al. Exploit the connectivity: Multi-object tracking with trackletnet. In: *Proceedings of the 27th ACM International Conference on Multimedia*. [S.l.: s.n.], 2019. p. 482–490. Citado na página 14.

WOJKE, N.; BEWLEY, A.; PAULUS, D. Simple online and realtime tracking with a deep association metric. In: IEEE. *2017 IEEE international conference on image processing (ICIP)*. [S.l.], 2017. p. 3645–3649. Citado 3 vezes nas páginas 32, 46 e 47.

WONG, A. et al. Yolo nano: A highly compact you only look once convolutional neural network for object detection. *arXiv preprint arXiv:1910.01271*, 2019. Citado 6 vezes nas páginas 14, 42, 48, 49, 50 e 51.

WONG, A. et al. Ferminets: Learning generative machines to generate efficient neural networks via generative synthesis. *arXiv preprint arXiv:1809.05989*, 2018. Citado na página 49.

WU, B.; NEVATIA, R. Tracking of multiple, partially occluded humans based on static body part detection. In: IEEE. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. [S.l.], 2006. v. 1, p. 951–958. Citado 2 vezes nas páginas 11 e 33.

XU, J. et al. Spatial-temporal relation networks for multi-object tracking. In: *Proceedings of the IEEE International Conference on Computer Vision*. [S.l.: s.n.], 2019. p. 3988–3998. Citado 3 vezes nas páginas 14, 46 e 47.

XU, Y. et al. Deep learning for multiple object tracking: a survey. *IET Computer Vision*, IET, v. 13, n. 4, p. 355–368, 2019. Citado 2 vezes nas páginas 31 e 46.

YIM, J. et al. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2017. p. 4133–4141. Citado na página 28.

YOON, K. et al. Data association for multi-object tracking via deep neural networks. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 19, n. 3, p. 559, 2019. Citado na página 13.

YOON, Y.-c. et al. Online multi-object tracking with historical appearance matching and scene adaptive detection filtering. In: IEEE. *2018 15th IEEE International conference on advanced video and signal based surveillance (AVSS)*. [S.l.], 2018. p. 1–6. Citado na página 14.

YU, T. et al. Distributed data association and filtering for multiple target tracking. In: IEEE. *2008 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.], 2008. p. 1–8. Citado na página 31.

ZHANG, S. et al. Tiny yolo optimization oriented bus passenger object detection. *Chinese Journal of Electronics*, IET, v. 29, n. 1, p. 132–138, 2020. Citado 2 vezes nas páginas 48 e 51.

ZHAO, H. et al. Mixed yolov3-lite: A lightweight real-time object detection method. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 20, n. 7, p. 1861, 2020. Citado 4 vezes nas páginas 41, 42, 48 e 50.

ZHAO, Z. et al. Ecrt: an edge computing system for real-time image-based object tracking. In: *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. [S.l.: s.n.], 2018. p. 394–395. Citado na página 39.

ZHU, J. et al. Online multi-object tracking with dual matching attention networks. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. [S.l.: s.n.], 2018. p. 366–382. Citado 3 vezes nas páginas 32, 46 e 47.