DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Análise de viabilidade no uso de Deep Learning para contagem de pessoas com câmeras de segurança

Relatório submetido à Universidade Federal de Santa Catarina
como requisito para a aprovação da disciplina:
DAS 5511: Projeto de Fim de Curso

Angelo Baruffi Nogueira

Florianópolis, Novembro de 2018

Análise de viabilidade no uso de Deep Learning para contagem de pessoas com câmeras de segurança

Angelo Baruffi Nogueira

Esta monografia foi julgada no contexto da disciplina

DAS 5511: Projeto de Fim de Curso

e aprovada na sua forma final pelo

Curso de Engenharia de Controle e Automação

Prof. Aldo von Wangenheim

Banca Examinadora:

Igor Marques de Souza Gois
Orientador na Empresa

Prof. Aldo von Wangenheim
Orientador no Curso

Prof. Hector Bessa Silveira
Responsável pela disciplina

Prof. Flávio Gabriel Oliveira Barbosa, Avaliador

Andrei Donati, Debatedor

Abner do Canto Pereira, Debatedor

Resumo

O objetivo desse trabalho é analisar a viabilidade técnica de implementação de *deep learning* para detecção e contagem de pessoas em câmeras de segurança. Toda a implementação e viabilidade estudada levaram em consideração o a velocidade de implementação do sistema e processamento do algoritmo, com o objetivo de conseguir escalar o processo comercialmente, podendo implementar o sistema proposto em lojas do varejo físico. Foram analisados diversos modelos convolucionais, atualmente considerados estado da arte para detenção de objetos, implementando-os para detecção de pessoas. As diversas arquiteturas foram comparadas em velocidade e qualidade de predição, utilizando a medida de similaridade *mean average precision* e a raiz do erro quadrático médio.

Diversos modelos tiveram resultados satisfatórios em velocidade e acuracidade, chegando a atingir 13,26 FPS em uma CPU, 56% de mAP e erro médio percentual de $8,2\%$ por *frame* processado efetuando um pós processamento. Tais resultados obtidos sem a necessidade de retreinamento dos modelos, otimizando assim o tempo e custo de implementação.

Palavras-chave: Deep Learning, contagem de pessoas e processamento de vídeo.

Abstract

The goal of this work is to analyze the technical feasibility of implementing deep learning for detection and people counting in security cameras. All the implementation and feasibility studied took into consideration the speed of system implementation and algorithm processing, with the aim of being able to scale the process commercially, being able to implement the proposed system in physical retail stores. We analyzed several convolutional models currently considered state-of-the-art for object detection, implementing them for people couting. The different architectures were compared in speed and prediction quality, using the average precision measure and the root mean square error.

Several models had satisfactory results in speed and accuracy, reaching 13,26 FPS in a CPU, 56 % of mAP and error of $8,2\%$ in average per frame processed by performing a post processing. These results were obtained without the need to retrain the models, thus optimizing the implementation time and cost.

Keywords:Deep Learning, people counting and video processing

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

BI - Business Intelligence

CNN - Convolutional Neural Network

HOG - Histogram of oriented gradients

SVM - Support vector machine

GPU - Graphics Processing Unit

ANN - Artificial Neural Network

MLP - Multilayer Perceptron

RPN - Region Proposal Network

YOLO -You only look once

SSD - Single Shot MultiBox Detector

FIFO - First In, First Out

FFT - Fast Fourier Transform

Sumário

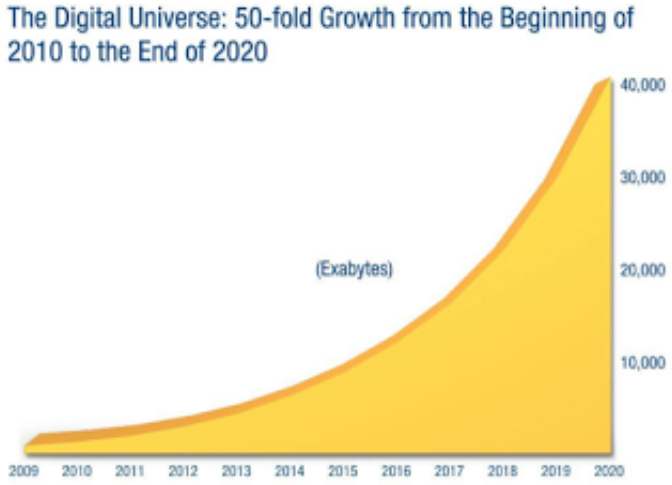
1 Introdução

O mundo dos negócios sempre é competitivo e de crescimento acelerado. Os gestores das grandes empresas estão sempre preocupados em entender melhor seus processos, os pontos que podem maximizar as receitas e minimizar os custos. Para ter essa visão clara dos processos é fundamental quantificar numericamente a realidade para que se consiga saber se as ações estão indo no caminho correto [1].

"Não se gerencia o que não se mede, não se mede o que não se define, não se define o que não se entende, e não há sucesso no que não se gerencia" [1]. Essa frase foi dita por William Edwards Deming, um dos grandes gurus da administração, e é por esse conceito que as grandes empresas buscam adquirir e guardar a maior quantidade de dados possível, para assim tentar quantificar os seus processos. Em adição a isso, a tecnologia viabilizou a aquisição de um grande volume de dados e de diferentes fontes, criando uma vasta gama de possibilidades de análise e inferência que podem auxiliar o gestor na tomada de decisão.

O estudo desenvolvido pela consultoria EMC [2] apontou que de 2006 a 2010, o volume de dados digitais gerados cresceu de 166 Exabytes para 988 Exabytes. Além disso, o estudo prevê que em 2020 o volume de dados alcance o patamar de 40.000 Exabytes, como pode ser visto na figura 1

Figura 1 – Crescimento de dados

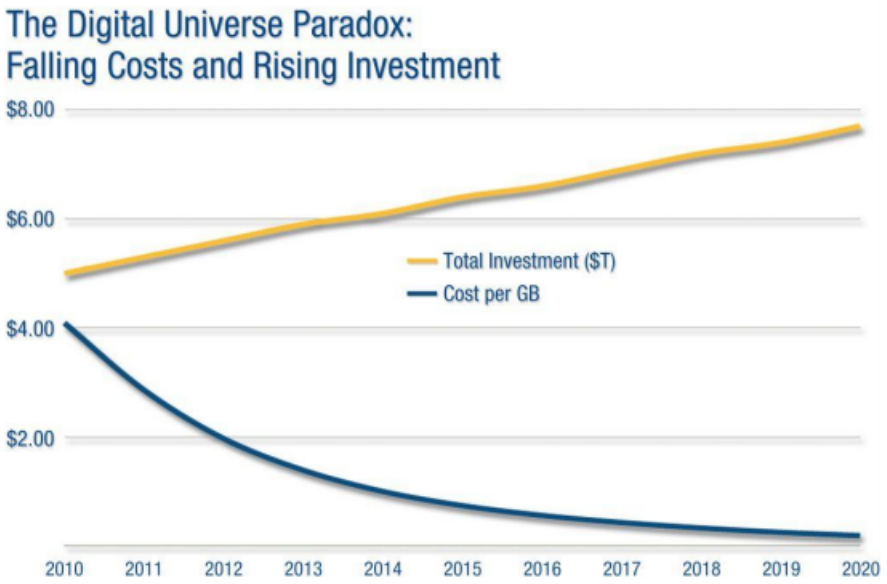


Fonte: EMC [2]

As somas desses fatores fez com que termos como *Data Science, Machine Learning, Big Data, IA e BI(Business Intelligence)* tivessem uma crescente busca e investimento nos últimos anos [2]. As empresas passaram a investir grandemente nesses setores a fim de entender seus dados e não apenas armazena-los. A figura 2 mostra o aumento do investimento em dados enquanto o custo de armazenamento ter caído drasticamente.

Ambas as curvas também são reflexo do desenvolvimento tecnológico e do alto investimento em pesquisas relacionadas a dados.

Figura 2 – Investimento em Dados



Fonte: EMC [2]

1.1 Visão Computacional

A visão computacional pode ser descrita como um processo de modelagem e identificação que consegue obter informações a partir de imagens e vídeos [3]. Muito similar ao olho humano que consegue identificar padrões, objetos, pessoas, cores e entre outras coisas, a visão computacional tenta trazer essa expertise para um algoritmo de máquina, criando um processo automatizado, muitas vezes melhor e mais rápido que o ser humano. Esse tipo de tecnologia tem sido usado grandemente para a aquisição de dados, seja para gerar tags em imagens de redes sociais ou até mesmo em processos industriais.

A importância do crescimento da visão computacional está principalmente ligada ao fato de que grande parte do volume de dados que estão sendo gerados são não estruturados, sendo muitas vezes no formato de vídeo e imagem. Esse grande volume de dados precisa ser armazenado, disponibilizado e analisado, que devido a quantidade é inviável que seja feito por pessoas.

Diversas são as aplicação de visão computacional. Na biomedicina, ela pode ser utilizada para detecção de câncer em imagens de Ráio-X ou identificação de problemas oculares [4]. Na industria, é comum ver aplicações de inspeção visual automática de produtos, separação de peças e identificação de problemas na linha produtiva [5]. Já em mídias sociais, que são responsáveis por produzir grande volume de imagens e vídeos,

aplicações como identificar pessoas, ocasiões, locais e tipo do conteúdo passaram a ser comuns de serem encontradas nessas grandes mídias. Aplicações estas que permitem até mesmo gerar acessibilidade para pessoas com deficiência visual, descrevendo imagens em forma de texto.

Uma área onde a visão computacional possui grande importância é a de sistema de segurança e vigilância. Devido a vasta quantidade de câmeras de segurança já implementadas no mercado são desenvolvidos cada vez mais métodos de inferência a partir dessa fonte de dados, seja com identificação facial, de pessoas e objetos ou até mesmo de situações de risco ou roubo. A quantidade desses equipamentos e a necessidade repetitiva de acompanhamento por pessoas criam um cenário perfeito para a automatização dos processos usando processamento de vídeo.

1.2 Varejo no Brasil

Segundo a SBVC [6], "o varejo é toda atividade econômica da venda de um bem ou um serviço para o consumidor final, ou seja, uma transação entre um CNPJ e um CPF". Duas categorias podem ser definidas, o varejo restrito e o varejo ampliado, sendo que o segundo inclui o primeiro com a adição das concessionárias de veículos e lojas de material de construção. Segundo a SBVC o varejo restrito teve 20,25% de impacto no PIB com um volume de R$ 1,34 trilhão e crescimento real de 2%. Portanto, é um mercado em alta acensão e com grande capacidade de investimento em novas tecnologias [6].

O E-commerce, que segundo a Neoatlas [7], teve um crescimento em 2017 de 12%, possui um elevado índice de investimento em tecnologia, principalmente em dados devido a facilidade e volume de aquisição de novas informações. Segundo uma pesquisa desenvolvida pela Forrester [8], o investimento em tecnologia pelas lojas virtuais corresponde, em média, a 9% da receita. Isso cria uma grande vantagem competitiva em relação ao varejo físico já que é possível entender melhor o cliente final e como agir para maximizar as vendas.

O varejo físico, portanto, passa por uma fase de grande investimento e inovação em novas tecnologias a fim de conseguir entender melhor o comportamento do cliente e seus padrões de consumo. Assim, tentando acompanhar a crescente inovação do e-commerce de utilizar uma vasta quantidade de dados para trazer ideias e informações sobre o negócio.

1.3 Problemática

Um dos grandes problemas do varejo físico é conseguir entender e quantificar o comportamento do consumidor final [9]. Identificar quantas pessoas vão à loja, quais os picos e vales de demanda, horários com maior conversão e fluxo de movimentação são

exemplos de dúvidas dos grandes varejistas e que são complexas de se quantificar, porém poderosas informações se conhecidas.

Em uma grande loja de varejo, por exemplo, entender em que horários há um maior fluxo de pessoas na loja é fundamental para poder gerir a quantidade de funcionários alocados, garantindo a qualidade sem aumentar os custos desnecessariamente. Tal processo é comumente chamado de *Workforce Management* (WFM), ele consiste em alocar de forma otimizada os trabalhadores da empresa, utilizando assim previsão de demanda e performance dos funcionários. Esses sistemas são muito comuns e utilizados em centrais de atendimento, onde se prevê a quantidade de ligações e o tempo médio de atendimento, como consta no artigo apresentado pela Callcentre Helper [10].

Para centrais de atendimento esse processo é facilitado já que a aquisição de dados é mais simples, como: tempo de cada ligação e quantidade de ligações por minuto. No varejo físico esse processo é mais complexo, não se sabe exatamente quantas pessoas estavam na loja ou quantas pessoas estavam na fila do caixa para alocar atendentes. Se utiliza para a previsão geralmente a quantidade de vendas no tempo, porém esse indicador pode mascarar a demanda já que o fator de conversão de venda pode estar alterado pela falta de atendentes no local.

1.4 Objetivo Geral

Análise de viabilidade da utilização de *deep learning* para fazer contagem de pessoas em tempo real no varejo físico, tendo um baixo custo de implementação.

1.5 Objetivos Específicos

- Analisar o estado da arte em visão computacional, focando em detecção de objetos.
- Sistema de processamento poder ser conectado em uma câmeras de segurança.
- Definir um ambiente de comparação entre os modelos escolhidos bem como uma métrica de comparação.
- Analisar e comparar os modelos com base nas métricas levantadas.
- Construir uma arquitetura para a inserção de um sistema de *tracking* de pessoas em próximos trabalhos.

1.6 Justificativa

Implementando a contagem de pessoas é possível identificar picos de demanda de vendas em tempo real, pode-se prever ações imediatas, por exemplo, na alocação de caixas em supermercados ou de atendentes. Além disso, uma arquitetura preparada para a adição de sistemas de rastreamento de pessoas, permite fazer análise de fluxo e quantidade de pessoas que vieram a loja, calculando assim a taxa de conversão que a loja possui e ainda podendo quantificar aumentos de fluxo na loja devido a contagem única de pessoas que compareceram ao estabelecimento.

O baixo custo de implementação do sistema aumenta o mercado potencial de compra do serviço, diminuí o tempo de retorno sobre o investimento e assim reduz a barreira de entrada em grandes companhias.

2 A Empresa

A empresa Bix Tecnologia é uma consultoria de *business intelligence*(BI) e *analytics* e visa criar todo a estrutura do BI, desde a extração dos dados de negócio do cliente, até sua transformação e visualização. A empresa tem como foco apresentar uma visão analítica dos problemas juntamente com uma consultaria de negócios. Grande parte dos clientes da Bix são considerados grandes varejistas do mercado nacional. Como por exemplo, a Studio Z, Animale, Farm, Portobello Shop e Officina. Além de outros clientes como a promotora de crédito Fontes e o porto Portonave.

O problema apresentado nesse trabalho tem origem em uma demanda apresentada por um cliente varejista, que apresentou a necessidade de saber quantas pessoas estão nas filas e na loja em tempo real. Por isso, o presente projeto foi desenvolvido para se tornar um produto comercial a ser vendido e implementado em empresas com necessidades similares.

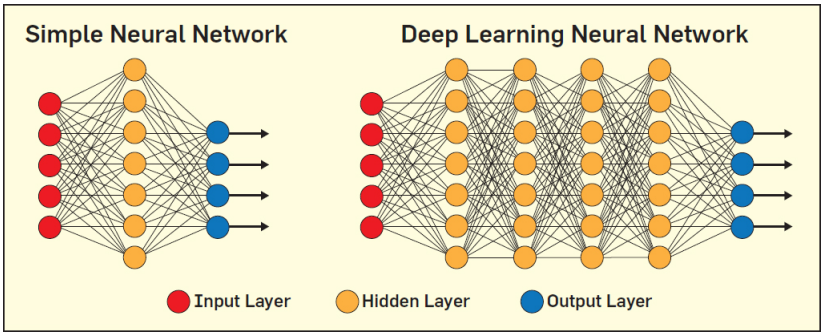
3 Fundamentação Teórica

Deep larning tem sido muito estudada e implementada nos últimos anos, representando uma evolução significativa nos métodos de aprendizado de máquina. Sua grande evolução tem por origem a implementação de algoritmos de treinamento de redes neurais, como o *back propagation* [11]. Além disso, o crescimento em processamento e memoria dos computados permitiu que as redes neurais pudessem se tornar mais complexas e com maior quantidade de neurônios podendo assim desempenhar processos mais complexos.

Uma rede de *Deep larning* se diferencia devido a quantidade de camadas internas para processamento das entradas inicias. Tal quantidade de camadas faz com que as regras de predição deixem de ser explicitas, entretanto adiciona um maior grau de liberdade para que a rede consiga aprender sistemas mais complexos e menos lineares [12]. Na figura 3 é possível observar a maior quantidade de *hidden layers*(camadas internas). Esse tipo de rede, por ter que ajustar diversos parâmetros são bem mais pesadas para treinamento dado a quantidade de dados necessária e geralmente mais lentos para predição, por isso, dependeram do avanço tecnológico para serem implementadas em larga escala.

O avanço do *Deep larning* permitiu grandes melhorias em sistemas de reconhecimento de fala, detecção de objetos com o processamento de imagens, processamento de linguagem natural e entre outras aplicações [11, 13].

Figura 3 – Treino e teste de uma rede neural artificial



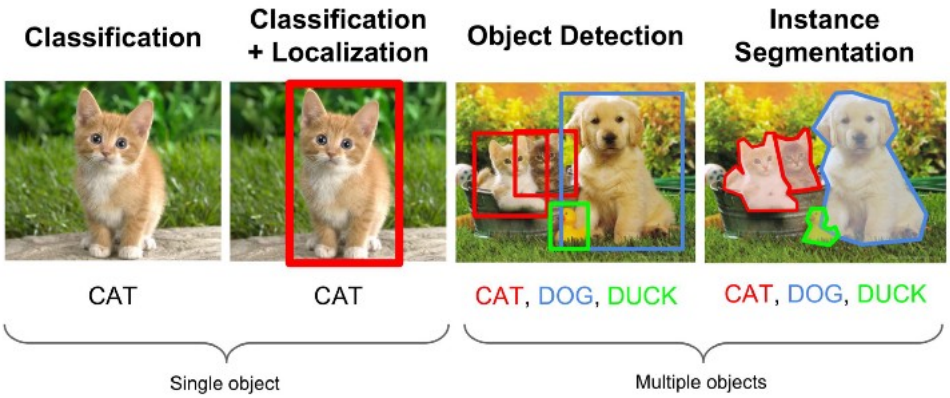
Fonte: Communications of the AC [14]

3.1 Categorias de algoritmos

Aplicações de *Deep larning* e visão computacional podem ser separadas em dois principais grupos: analise de um objeto ou analise de mais de um objeto na imagem. O processo mais simples de predição é a classificação de uma imagem quanto ao seu tipo. Basicamente é um processo de criação de rótulos para descrever o que há na imagem ou se

ela tem ou não uma característica. Melhorando esse processo pode-se também localizar aonde está essa característica/objeto na imagem, sendo conhecido como localização. Porém, quando se deseja processar e localizar mais de um objeto na imagem, setem um processo de predição mais complexo. Um dos exemplos é a necessidade de se obter a localização de múltiplos objetos na imagem, esse processo é chamado de detecção de objetos, ou então do inglês *object detection*. Há ainda a possibilidade de não apenas detectar a região de um determinado objeto, mas os exatos pixeis que o compõem(segmentação), como mostrado na figura 5.

Figura 4 – Comparação entre os tipos de objetivo na visão computacional



Fonte: Arthur Ouaknine [15]

O objetivo desse trabalho está voltado a fazer a contagem de pessoas em diferentes *frames* dos vídeos. Para esse tipo de abordagem não seria necessário a implementação de um detector de objetos já que saber a posição da pessoa não é um requisito funcional para a contagem. Uma tarefa como essa seria possível implementar uma rede convolucional para a extração das *features* e um algoritmo de regressão para predição do número de pessoas [16]. Conhecer a posição das pessoas na imagem, entretanto, permite com que se consiga desconsiderar pessoas em determinadas regiões e também dar as detecções como entrada para um algoritmo de *tracking*, podendo assim analisar a movimentação de uma pessoa no local analisado, os locais com maior fluxo de pessoas, entrada e saídas de ambientes e a identificação única de indivíduos.

A utilização de detecção de objetos passa a ser uma abordagem boa para a contagem de pessoas já que a elaboração de uma boa arquitetura de detecção permite utilizar as predições como entrada para outros futuros sistemas.

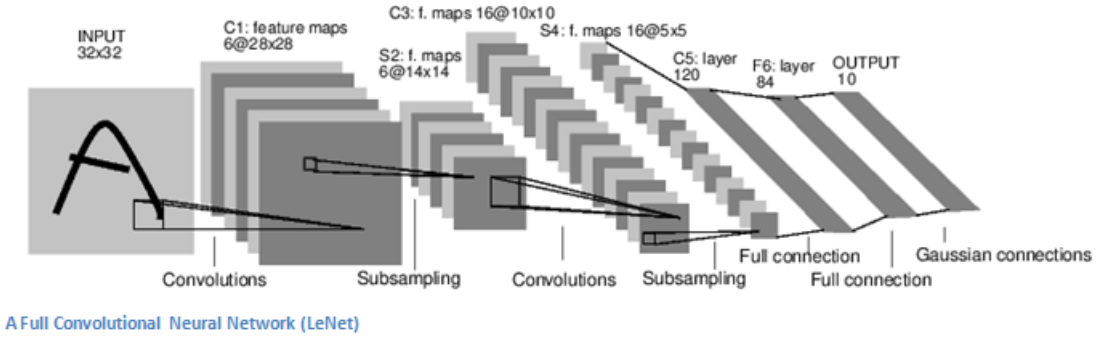
Quanto a utilização de segmentação de objetos com a finalidade de contagem de pessoas se mostrou desnecessária tendo em vista que não é necessário saber exatamente quais pixeis compõem uma determinada pessoa. Além disso, a arquitetura dessas redes tendem a ser mais complexas, como é o caso da Mask R-CNN, que adiciona uma camada a mais em uma arquitetura de *object detection* apenas para fazer a segmentação, portanto,

tendo um tempo de predição maior.

3.2 Redes Neurais Convolucionais

As redes convolucionais são a base de implementação da maioria das arquiteturas estado da arte em visão computacional e mais especificadamente detecção de objetos. Sua grande relevância está no fato de permitir a extração das *features* de uma imagem sem a definição formal do que deve ser analisado [17]. Antes de seu surgimento, as características a serem utilizadas para classificação de uma imagem necessitavam ser descritas manualmente, como cor, textura, bordas, gradientes, entre outras. Com o desenvolvimento das CNNs, o processo de geração de *features* se tornou também treinável de forma a identificar os pesos de diferentes filtros aplicados a imagem antes de ser colocado como entrada no classificador [17].

Figura 5 – Estrutura geral de uma rede convolucional



Fonte: Neural Networks and Deep Learning [17]

Na figura 5 há uma CNN simples para a classificação de imagens, por exemplo. A entrada da rede é um vetor de pixeis. No caso de um arquivo jpg há a entrada de um array de 32 x 32 x 3, onde o 3 representa os valores RGB. Os valores desse array são números inteiros que variam entre 0 e 255.

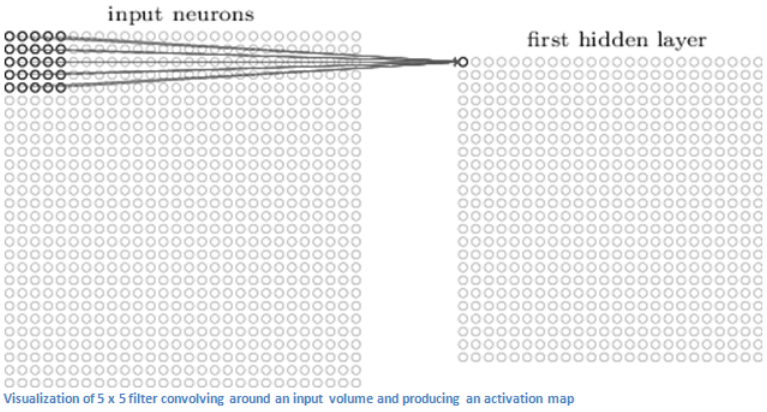
A arquitetura de uma CNN é composta por várias camadas que tem como objetivo, extrair as informações da entrada, reduzir a quantidade de processamento, generalizar a predição do modelo e também colocar componentes não lineares no processo.

3.2.1 Camada de convolução de uma CNN

Uma das camadas mais importantes em uma CNN é a de convolução [17]. Na figura 6 há um exemplo desse tipo de camada. O processo de convolução é a aplicação de um filtro deslizante que pondera com base em seus pesos as entradas produzindo um único valor de saída. No exemplo da figura 6 é usado um filtro 5 x 5 x 3, isso produz

como resultado uma imagem de 28 x 28 x 1, já que há 784 locais diferentes que o filtro pode ser aplicado(passo(*stride*) de 1). O filtro aplicado na convolução possui seus próprios pesos que são ajustados no processo de treinamento da rede. Esse filtro pode ser chamado também de *kernel*. O resultado desse processo gera o que é chamado de *feature map* ou *activation map*.

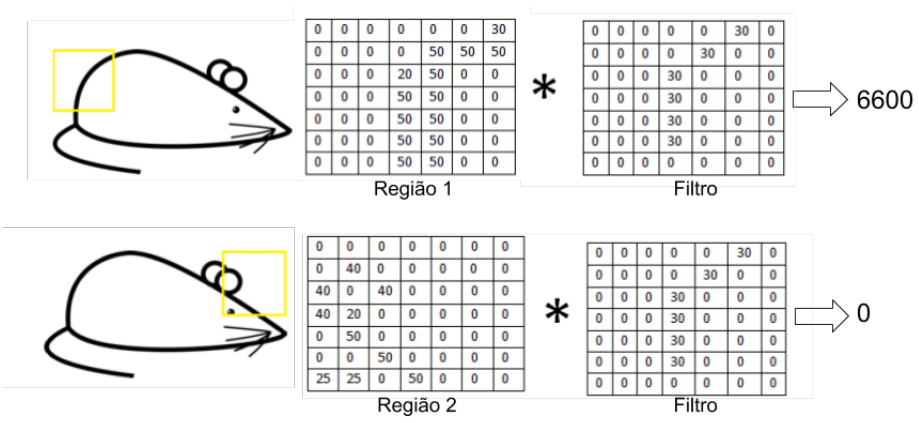
Figura 6 – Visualização de um filtro 5 x 5 sendo aplicado e produzindo um mapeamento de *features*



Fonte: Neural Networks and Deep Learning [17]

Na prática o que o processo de convolução está fazendo é a identificação de *features* com base nos pesos do filtro. Na figura 7 há um filtro que é aplicado em duas partes distintas da imagem. Ao fazer o produto do filtro pelos pesos dos pixeis da imagem é realçado um padrão da imagem, que no primeiro teste tem um valor bem alto, ou seja, identificou o padrão buscado, mas na segunda imagem resultou em 0 já que o padrão não se apresenta.

Figura 7 – Visualização de um filtro sendo aplicado a dois locais diferentes, ressaltando uma *feature* específica



Fonte: Adaptado de Neural Networks and Deep Learning [17]

O processo de visualização das features maps é um importante fator para conseguir entender o que a rede está utilizando como entrada para fazer as predições. No entanto, saindo das primeiras camadas, o processo de visualização do que está sendo produzido pelos filtros e entender o seu significado é complexo e muitas vezes não interpretável [18].

Em uma arquitetura de rede convolucional, não é aplicado apenas um filtro em cada camada, mas um conjunto k de *kernels*, que combinados vão resultar em, para a situação apresentada, um *feature map* de 28 x 28 x k.

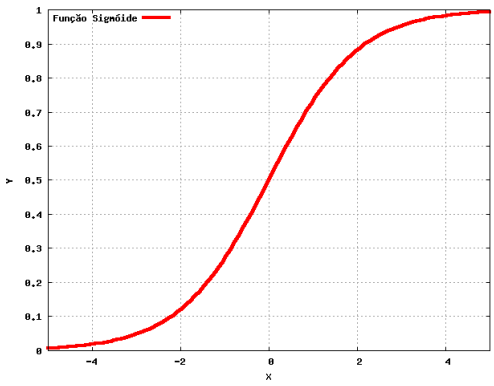
3.2.2 Camadas de Ativação

As camadas de ativação, também conhecidas como *activation layers*, são etapas fundamentais que adicionam robustez para basicamente qualquer tipo de rede neural e estão presentes também nas convolucionais [17]. Apenas com os cálculos de convolução da rede aplicando os *kernels* para filtragem, a saída dessas redes seriam composições lineares das imagens de entrada. Isso significa que só seria possível predizer saídas que possuem composição linear com as *features* de entrada. Para resolver isso e assim aumentar a robustez das redes neurais, são adicionados função para gerar não linearidades.

Uma função muito comum para adicionar a não linearidade é a sigmóide. Ela adiciona a não linearidade transformando os sinais de entrada em um intervalo de $[0, 1]$. Devido a esse comportamento, ela é muito utilizada em modelos de predição onde a saída é uma porcentagem de chance de o valor estar correto, como por exemplo em um modelo de regressão logística. A função sigmóide é descrita na equação 3.1 e há um exemplo de seu comportamento para $\lambda = 1$ na figura 8:

$$f(x) = \frac{1}{1 + e^{-\lambda x}} \tag{3.1}$$

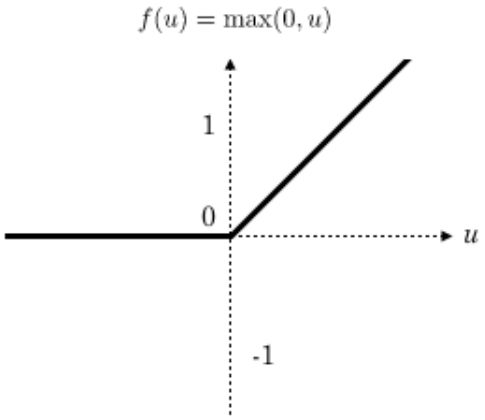
Figura 8 – Gráfico de uma função sigmóide para $\lambda = 1$



Atualmente, a função de ativação mais utilizada é a função de Unidade Linear Retificada (*ReLU - Rectified Linear Unit*) já que é mais rápida para o processo de

treinamento da rede sem perder muita acuracidade. Sua equação é o $Max(0, x)$, sendo então bem simples computacionalmente.

Figura 9 – Gráfico de uma função ReLU

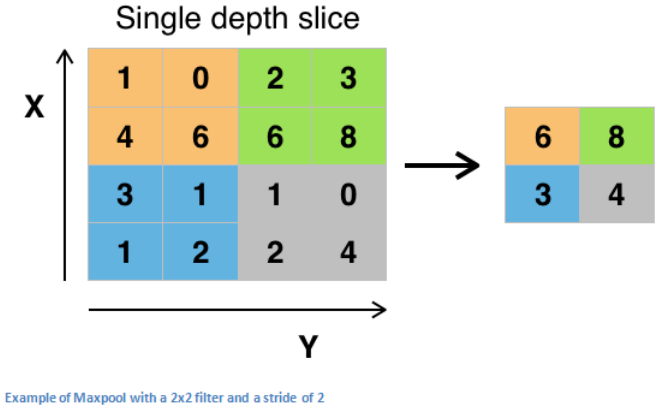


3.2.3 Camada de Agregação

No processo de convolução os filtros identificam características específicas de acordo com seus pesos, por exemplo identificando uma mão humana na imagem. Entretanto, essa característica específica pode estar apresentada na imagem não na exata forma com o que o filtro foi treinado a identificar, ou seja, a mão pode ser apresentada em diversos ângulos e iluminações diferentes, o que no final do processo de convolução, resultaria em pesos menores comparados com a imagem de treino. Para criar esse generalização é utilizado camadas de *pooling* ou também conhecidas como *downsampling* [17].

O processo de *pooling* mais utilizado é o *maxpooling* que tem como saída o máximo valor de uma determinada região de filtragem.

Figura 10 – Função *Maxpooling* sendo aplicada em uma camada



Fonte: Neural Networks and Deep Learning [17]

Além de ter o papel de reduzir dimensionalmente as camadas, otimizando assim o processo de treino, o *pooling* tem um importante papel para diminuir o *overfitting* do modelo treinado.

3.2.3.1 Sobre-ajuste ou Overfitting

Ao treinar uma rede com base em um conjunto de dados para treino, um problema muito comum é o alto ajuste do modelo com base nos dados de treino, impedindo assim que ele consiga generalizar para outras entradas diferentes das treinadas. Esse processo é chamado de *overfitting*. Um modelo com essa característica possui uma alta acuracidade para a base de treino, porém uma baixa acuracidade para o conjunto de teste.

3.2.4 Camada de Dropout

Nessa camada de uma rede convolucional, alguns pesos são descartadas aleatoriamente. Esse descarte é feito colocando zero para esses respectivos pesos o que torna o processamento das camadas subsequentes mais rápido, já que o produto por outros presos sempre vai ser zero. Essa etapa é fundamental para reduzir o *overfitting*, pois força a rede a se ter redundâncias e a se adequar a diferentes tipos de entrada, além de aumentar a velocidade de treinamento e predição.

3.2.5 Camada Totalmente Conectada

Essas são geralmente as últimas camadas de uma rede convolucional. Elas possuem diversos neurônios totalmente conectados entre si, onde na maior das vezes tem a função de criar uma regressão ou uma classificação em sua saída com base nas *feature* extraídas pelas camadas convolucionais. Esse tipo de rede foi representada na figura 3.

3.2.6 CNN para Detecção de Objetos

O problema de detecção de objetos é um pouco mais complexo do que um modelo de classificação. Isso se deve ao fato da necessidade, além de prever a classe do objeto, também prever a sua localização. Além disso, uma imagem pode ter diferentes números de objetos em sua imagem e portanto o tamanho de saída da rede precisa ser variável.

Para solucionar esse problema, uma proposta simples seria obter diferentes regiões em uma imagem e classifica-las utilizando a CNN. A dificuldade com essa técnica é que os objetos possuem diferentes tamanhos e proporções, além de que devido a quantidade de imagens a serem selecionadas, o processamento de detecção pode ser bem lento.

Diversas outras abordagens mais eficientes foram propostas para a detecção de objetos utilizando CNN. Nas próximas sessões desse capitulo serão apresentados a R-CNN e a YOLO, que propõem resolver esse problema com uma arquitetura diferenciada.

3.3 Principais Arquiteturas

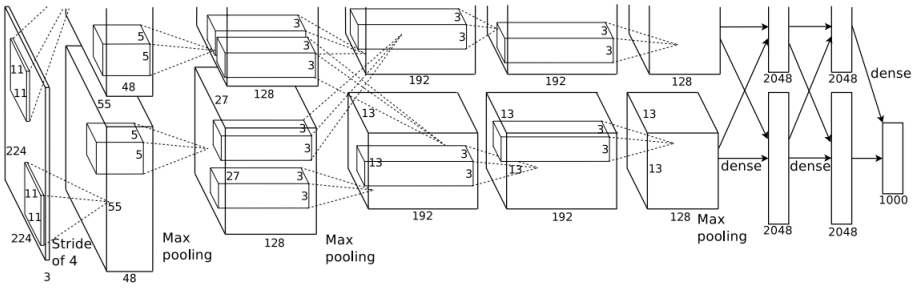
Tendo como base uma CNN, diversas arquiteturas de camadas e redes são implementadas para conseguir atingir um objetivo de modelo específico. A seguir serão aprensados algumas arquiteturas voltadas a detecção de objetos.

3.3.1 AlexNet

A AlexNet [19] foi proposta em 2012 por Alex Krizhevsky, Ilya Sutskever e Geoffrey Hinton. Essa arquitetura teve grande destaque e difundiu as CNNs como uma ótima alternativa para visão computacional. Sua estrutura é bem simplificada comparada com as mais atuais e pode ser descrita como:

- Duas primeiras camadas convolucionais, que foram seguidas de uma ativação ReLU, uma de normalização e uma de *maxpooling*;
- Terceira e quarta camada de convolução, seguida de uma ativação ReLU;
- Quinta camada de convolução, seguida por ReLU e *maxpooling*;
- *Dropout* antes da primeira camada completamente conectada;
- Duas camadas completamente conectadas com cinco neurônios de saída;

Figura 11 – Arquitetura de uma AlexNet



Fonte: Imagenet classification with deep convolutional neural networks [19]

3.3.2 VGG Net

A rede VGG [20] foi criada em 2014 e foi vencedora do desafio ILSVRC de 2014 com 7,3% de taxa de erro. Sua estrutura é composta por 16 camadas que usam filtros de 3 x 3 e camadas de *maxpooling* de 2 x 2. A combinação de dois filtros menores tem grande destaque nessa arquitetura se comparado com a primeira camada 11 x 11 da AlexNet. Segundo os autores, a composição de dois filtros 3 x 3 sequências tem uma área de análise

de 5 x 5, obtendo assim a vantagem de obter *features* em regiões maiores da imagem, porém sem perder a vantagem de treinar filtros menores, já que possuem menos pesos para serem ajustados. Da mesma forma, a composição de 3 filtros sequências tem como consequência um campo receptivo de 7 x 7. A arquitetura pode ser observada em detalhe na figura 12

Figura 12 – Tipos de VGG com destaque para a VGG-16

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB imag)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

The 6 different architecures of VGG Net. Configuration D produced the best results

Fonte: Very deep convolutional networks for large-scale image recognition [20]

Interessante observar que à medida que vai sendo aplicado agregação nas camadas, o tamanho espacial de volume vai diminuindo, enquanto que devido a quantidade de filtros aplicados, a profundidade dos volumes aumenta. Dessa forma, tendo como volume de entrada na rede completamente conectada um tamanho de 7 x 7 x 512. Esse conceito foi fundamental para apresentar que as CNNs, para terem bons resultados, precisam ter camadas com grande profundidade, para assim manter as características observadas nas primeiras camadas.

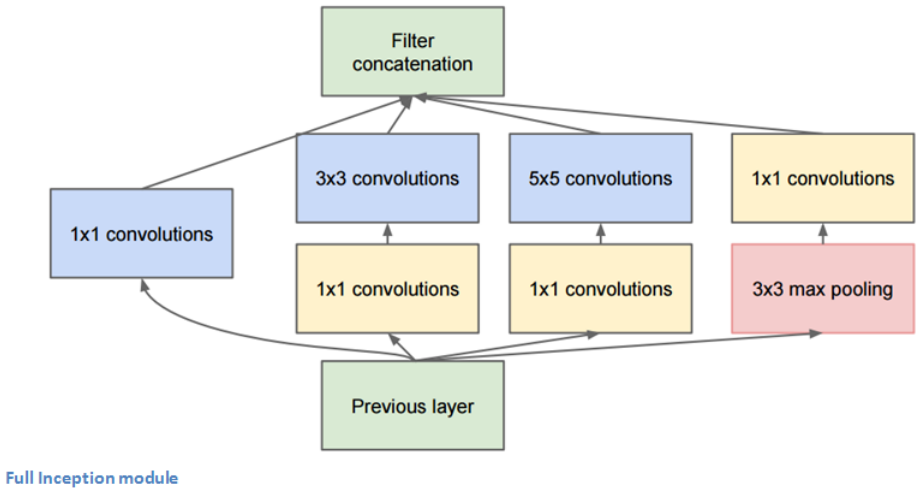
3.3.3 GoogLeNet

Essa arquitetura, desenvolvida pela Google [21] em 2015, tem grande destaque pois apresentou o conceito de não apenas encadear uma camada atrás da outra, dentro de uma

arquitetura convolucional, mas criar diferentes caminhos para informação. Sua arquitetura passa a ser bem mais complexa contando com 22 camadas.

A arquitetura é composta por vários módulos chamados de *inception* que é descrito na figura 13. Basicamente sua estrutura se baseia no conceito de possuir camadas de efeito distinto em paralelo, podendo ter assim um filtro de menor tamanho e de maior tamanho obtendo *features* da mesma imagem. A consequência negativa dessa arquitetura é o aumento no volume do dado, tornando assim a rede mais lenta. Para melhorar isso é necessário adicionar camadas de redução dimensional, como por exemplo o *maxpooling* para reduzir altura e largura e a camada de convolução 1 x 1, que é usada para reduzir a profundidade.

Figura 13 – Módulo *inception* com redução de dimensionalidade



Fonte: Going Deeper with Convolutions [20]

Essa arquitetura possuí como grande destaque a reformulação de como estruturar camadas de uma rede convolucional, além disso, apresentou um grande avanço em tempo de processamento já que sua estrutura conta com cerca de 12 vezes menos parâmetros que a AlexNet e por consequência melhor que a VGG, que por sua vez tem cerca de três vezes a quantidade de parâmetros que a AlexNet.

Após a apresentação da *Inception* v1 [21], várias outras modificações foram propostas para tentar otimizar o tempo e acurácia desse módulo de extração de features. As variações mais famosas são a *Inception* v2 [22], *Inception* v3 [22], *Inception* v4 [23] e *Inception-ResNet* [23]

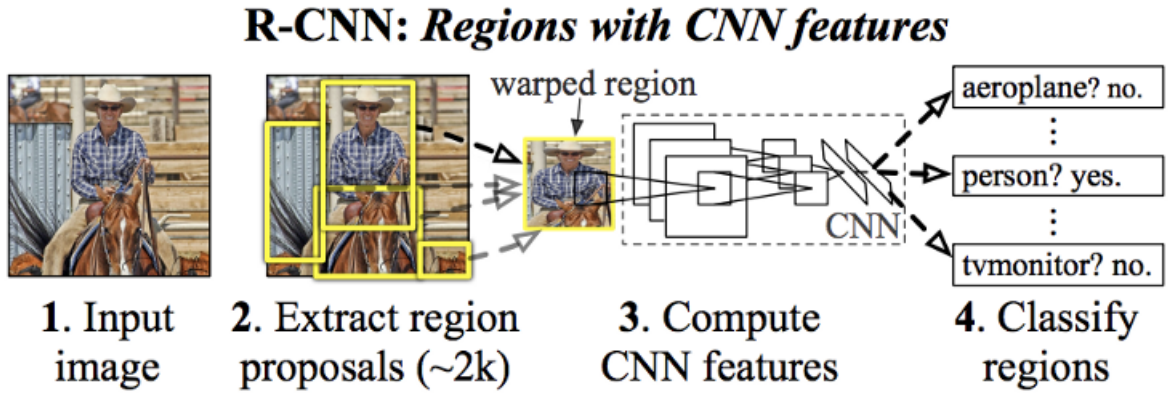
3.3.4 CNNs Baseadas em Região

Em 2013 foi proposto a arquitetura R-CNN [24] que tem como finalidade resolver o problema de *object detection*. Sua arquitetura pode ser descrita em duas etapas, a primeira

é responsável por prever possíveis localizações de caixas delimitadoras e a segunda tem como tarefa classificar essas localizações.

A arquitetura começa selecionando possíveis regiões de objetos na imagem utilizando a técnica de *selective search* [25]. Com as regiões propostas, é utilizado uma AlexNet para a extração das *features* de cada região, onde o resultado dessa extração é um vetor de *features*. Esse é então aplicado a um classificador binário do tipo SVM linear, tendo assim um para cada classe de objeto a ser detectado. Há ainda um regressor que tem como finalidade ajustar as caixas preditas. Para finalizar é feito uma análise de sobreposição de caixas, eliminando as redundâncias. A figura 14 representa o processo de predição de caixas da R-CNN.

Figura 14 – Modelo R-CNN



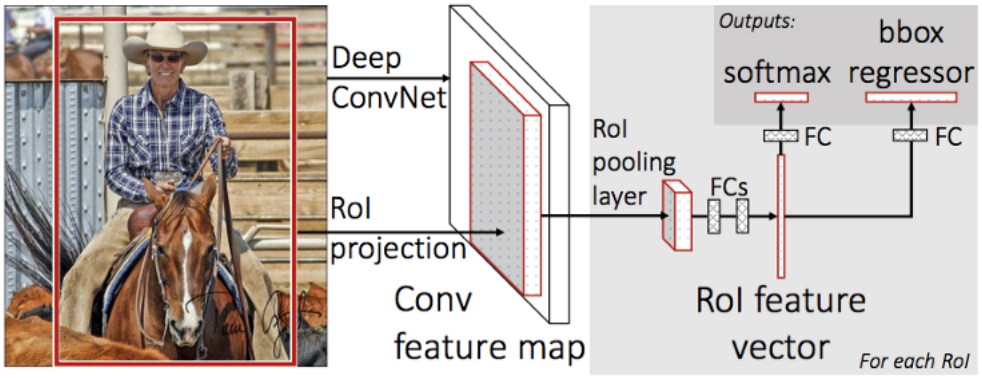
Fonte: Rich feature hierarchies for accurate object detection and semantic segmentation [24]

Uma das desvantagens dessa rede é o seu tempo de treinamento. Essa arquitetura possui diversos modelos internos que precisam ser treinado individualmente, como a AlexNet, a SVM e o regressor de caixas. Ela também possuí um tempo de predição bem lendo devido a quantidade de regiões propostas a serem analisadas pelo *selective search*, sendo cerca de 2 mil regiões.

Para otimizar a velocidade da rede foi proposto em 2015 a Fast R-CNN [26]. Essa arquitetura primeiramente aplica toda a imagem na rede de convolução extraindo assim as *features* de toda a imagem. Utilizando as *features* extraídas é obtido as propostas de região que são então aplicadas em camadas totalmente conetadas que propõem as caixas delimitadoras e as classes dessas caixas identificadas.

Tanto a R-CNN quanto a Fast R-CNN utilizam o *selective search* para fazer as previsões das regiões de interesse. No entanto, essa técnica é custosa computacionalmente afetando assim o desempenho dessas redes. Devido a isso, em 2015 foi proposto a Faster R-CNN [27] que diferente de suas antecessoras, implementa sua própria estrutura para

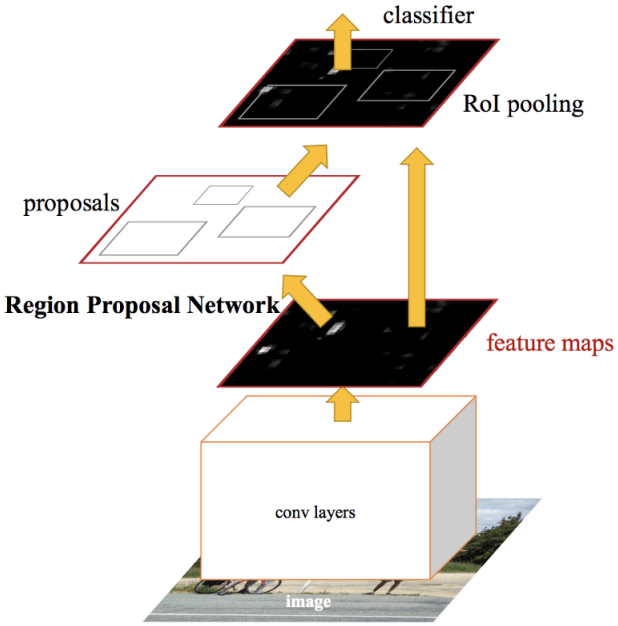
Figura 15 – Modelo Fast R-CNN



Fonte: Fast R-CNN [26]

prever possíveis regiões de objetos. Para isso é implementado após a extração das *features* uma rede RPN(*Region Proposal Network*) que tem por finalidade propor as regiões de possíveis objetos. Após a RPN o processo de classificação e regressão é similar a R-CNN.

Figura 16 – Modelo Faster R-CNN com sua nova camada RPN



Fonte: Faster R-CNN [27]

Com todas as melhorias propostas, a Faster R-CNN passou a ter condições de ser utilizada em processamentos de tempo real(em uma NVIDIA TITAN X) já que o tempo de predição da R-CNN é cerca de 245 vezes maior.

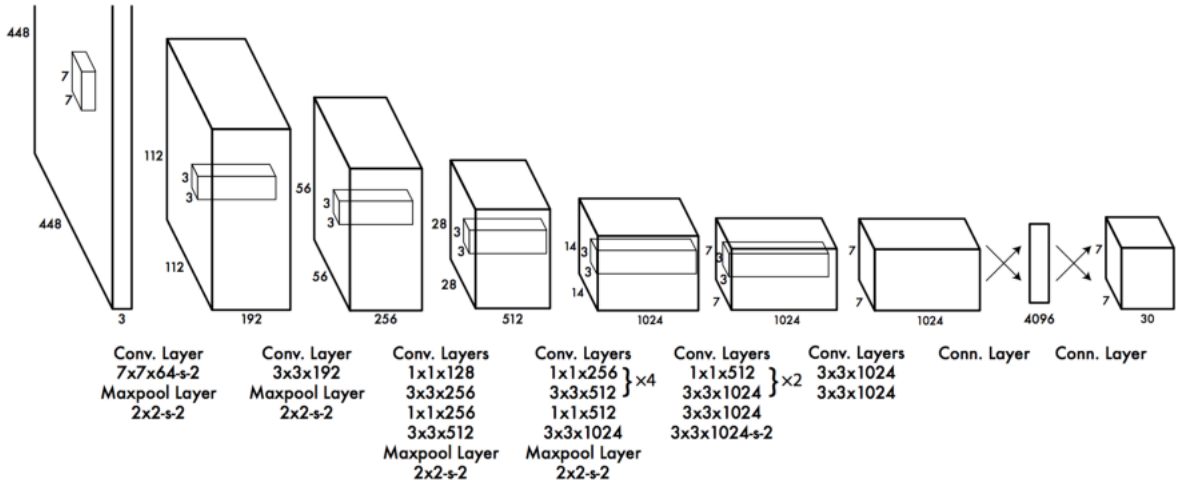
3.3.5 YOLO

Todas as arquiteturas apresentadas anteriormente tem como proposta utilizar regiões de interesse para aplicar suas classificações, nenhuma delas olha para a imagem como um todo. Em 2016, foi apresentado por a *You Only Look Once*, ou chamada de YOLO [28]. Essa arquitetura se apresentou como grande destaque e é um dos estados da arte em detecção de objetos em tempo real.

Primeiramente a YOLO divide a imagem em uma grade de SxS, onde para cada célula da grade é predito apenas um objeto. Para cada célula também, é predito uma quantidade limitada de *boundary box*(B), onde cada caixa tem uma nota de confiança para cada classe de objetos a serem detectados(C classes).

Um dos principais conceitos da YOLO é construir uma rede para reduzir o volume de entrada para S x S x (B*5 + C). Por exemplo, em um cenário onde tem-se 20 classes para predição, duas caixas por célula e uma grade de 7 x 7, tem-se ao final a rede apresentada na figura 17

Figura 17 – Arquitetura da CNN aplicada na YOLO reduzindo a dimensão de entrada para 7 x 7 x 30



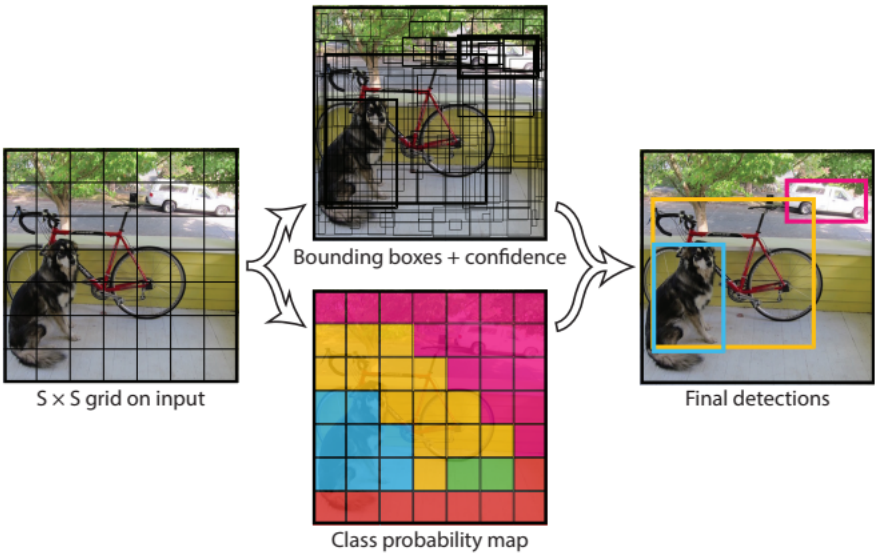
Fonte: YOLO [28]

A estratégia de definir diversas caixas por célula faz com que diversos objetos sejam detectados mais de uma vez por várias *boundary box* como pode ser observado na figura 18. É então necessário aplicar o processo de *non-maximal suppression*. Basicamente ele é constituído em duas etapas:

1. Ordena as predições com base em suas notas de confiança;
2. Começa com as primeiras detecções de maior confiança, ignorando as subsequentes que possuem uma sobreposição maior que 50% com a mesma classe;

A grande vantagem que a YOLO se destaca é sua alta velocidade de predição já que a imagem é passada pela rede convolucional apenas uma vez. Esse tipo de rede é chamada de *Single Shot detectors* e é de destaque para problemas de detecção de objetos em tempo real.

Figura 18 – Estrutura geral da YOLO



Fonte: YOLO [28]

Essa arquitetura também teve grandes melhorias em suas posteriores versões, como a YOLO v2 [29] e a YOLO v3 [30].

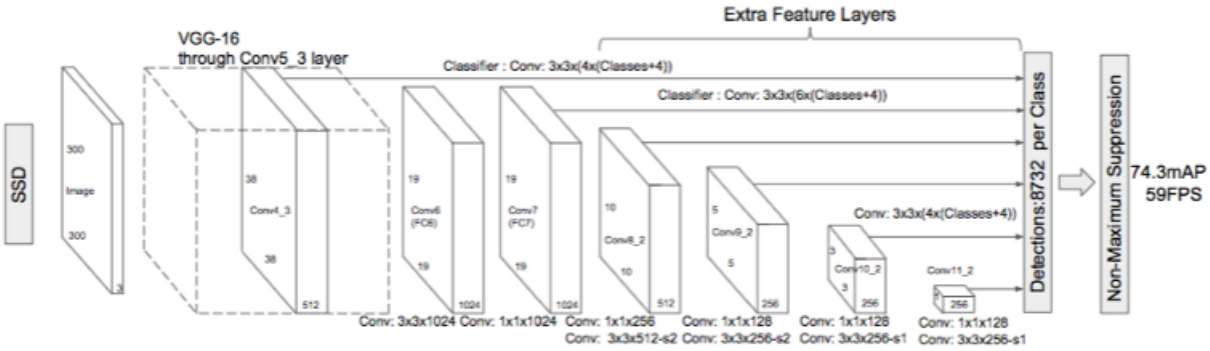
3.3.6 SSD

A *Single Shot MultiBox Detector* [31], também conhecida como SSD, tem como destaque também ser uma rede *single shot* e assim ter grande desempenho em tempo real, de forma parecida com a YOLO, eliminando também as regiões propostas para detecção. A SSD se mostrou bem mais rápida que a Faster R-CNN alcançando melhor acuracidade em imagens de mais baixa resolução.

A SSD é composto por duas partes. A primeira é uma extração de *features* utilizando a VGG-16 e após isso passa por filtros convolucionais extras para fazer a classificação e predição das *boundary box*. Um de seus grandes diferencias é que ela descarta o uso no final de sua arquitetura de uma rede completamente conectada.

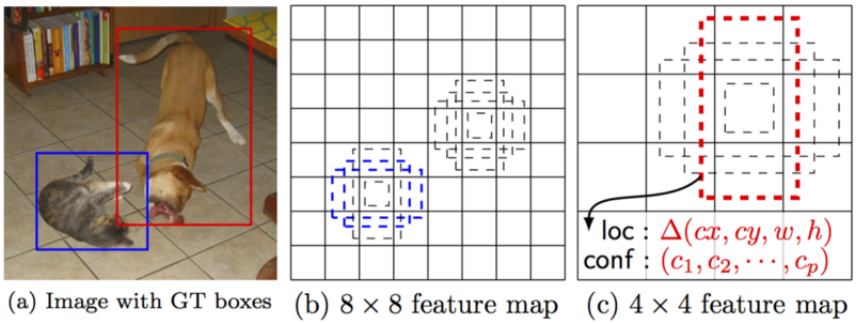
O conjunto de camadas extras adicionadas na SSD pode ser observado na figura 19 após a estrutura VGG-16. Uma de suas funções é fazer a regressão e predição das *boundary box* em uma arquitetura chamada de *MultiBox* [32]. Sua estrutura é em cima de uma *inseption* e que começa as predições utilizando caixas padrões que são então ajustadas para aquelas que tiveram confiança alta.

Figura 19 – Estrutura geral da SSD que utiliza inicialmente a VGG-16 e algumas camadas



Fonte: Single Shot MultiBox Detector [31]

Figura 20 – Predição de caixas padrões para cada célula do *feature map*



Fonte: Single Shot MultiBox Detector [31]

Na figura 20 é possível observar uma das características da rede SSD. Para cada célula do *feature map* é predito algumas caixas padrões que são futuramente ajustadas. Isso permite a rede se adaptar a diferentes tipos de entrada, sem precisar retreina-la para um tamanho de imagem diferente.

Dois pontos são de destaque nessa rede:

1. Cerca de 80% do tempo de processamento da rede é destinado a *feature map* do VGG-16. Tendo isso como base, há várias adaptações que utilizam extratores mais simples ou de maior acuracidade.

2. A SSD não se comporta bem com objetos pequenos nas imagens, já que seu mapeamento de *features* pode diluir a informação do objeto a ser detectado nas próximas camadas.

3.3.7 MobileNet

Essa estrutura de camadas foi proposta pela Google em 2017 [33] e tem como objetivo reduzir o tamanho e complexidade das camadas profundas, otimizando assim o

tempo de processamento para hardwares mais simples, por exemplo, celulares.

Em uma estrutura de camadas normais com entrada de altura H_i, largura W_i e profundidade C_i, se é desejado obter uma saída de profundidade C_o, então deve-se aplicar C_o *kernels* do tipo K x K x C_i, resultando assim em uma saída de H_o x W_o x C_o. Sendo assim, o custo de processamento de uma camada como essa é calculado como $H_i W_i C_i K^2 C_o$.

Na aplicação da MobileNet é utilizado a camada de filtragem *depthwise convolution*. Nessa camada com entrada H_i x W_i x C_i, onde se deseja uma saída de C_o, primeiramente é aplicado C_i vezes o filtro do tipo K x K x 1 tendo como saída H_o x W_o x C_i. Após isso é aplicado um filtro *pointwise convolution* que tem C_o *kernels* de K x K x C_i. A saída final desse processo continua sendo H_o x W_o x C_o, porém com redução nos parâmetros e cálculos da rede, diminuindo o tempo de predição, mas também a acuracidade do modelo. O custo computacional desse tipo de camada pode ser calculado como $H_i W_i C_i (K^2 + C_o)$ considerando que só há aliteração na profundidade da rede nessa camada.

3.4 Métrica de acuracidade para detecção de objetos

Tendo o valor verdadeiro da localização de objetos em uma imagem e tendo a predição feita por um modelo, é necessário quantificar o acerto da predição. Para isso tem de se utilizar uma medida de acuracidade. Uma das mais utilizadas para cálculo de acuracidade de detectores de objetos é o *mean Average Precision*(mAP). Para seu cálculo é necessário utilizar a precisão o *recall* e o IoU das predições.

3.4.1 Precisão

A precisão de um modelo mede quão corretos são as predições feitas, ou seja, de todas as predições feitas, quantas realmente são verdadeiras [11]. Seu cálculo é definido como a seguinte equação:

$$Precision = \frac{Verdadeiros\ Positivos}{Verdadeiros\ Positivos + Falsos\ Positivos} \tag{3.2}$$

3.4.2 Recall

O *Recall* de um modelo mostra o quão bem ele detecta os valores que são verdadeiros [11]. Seu cálculo é definido como a seguinte equação:

$$Recall = \frac{Verdadeiros\ Positivos}{Verdadeiros\ Positivos + Falsos\ Negativos} \tag{3.3}$$

3.4.3 Intersection over Union

A medida *Intersection over Union*(IoU) é utilizada para comparar as posições detectadas pelo modelo com as posições reais dos objetos. Seu cálculo considera as áreas

preditas e reais de forma a considerar a sobreposição dessas regiões [34].

$$IoU = \frac{Area\ de\ Intersecao}{Area\ de\ Uniao} \tag{3.4}$$

A medida IoU mede quão bom foi a predição da localização do objeto. Seu uso é feito considerando um parâmetro de escolha, onde para IoU maior que esse parâmetro a detecção é considerada como verdadeira. Na maioria das vezes esse parâmetro é 0.5 ou 50%.

3.4.4 Cálculo do Average Precision

Tabela 1 – Exemplo de predições feitas ordenadas pela confiabilidade

Classificação	Acerto	Precisão	Recall
1	Sim	1.0	0.2
2	Sim	1.0	0.4
3	Não	0.67	0.4
4	Não	0.5	0.4
5	Não	0.4	0.4
6	Sim	0.5	0.6
7	Sim	0.57	0.8
8	Não	0.5	0.8
9	Não	0.44	0.8
10	Sim	0.5	1.0

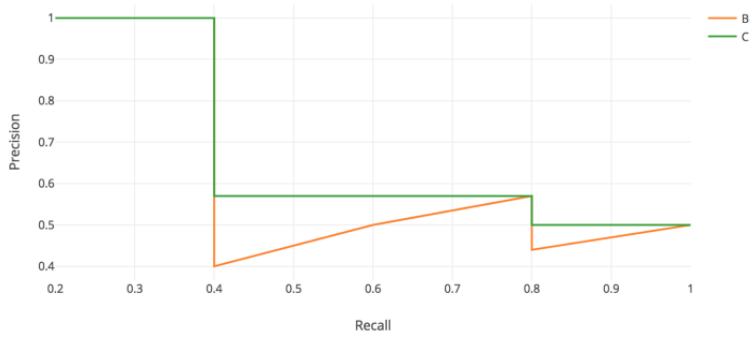
Fonte: Cálculo do mAP [35]

A tabela 1 apresenta um exemplo de possíveis resultados preditos. Para ser calculado o mAP é necessário primeiro ordenar em ordem decrescente pela confiabilidade das predições. Após, deve-se identificar utilizando a medida IoU se o objeto predito foi correto e com isso cálcular a precisão e *recall* para o registro considerado [35]. Interessante observar que o *recall* cresce a medida que mais predições são consideradas, já a precisão pode variar.

O cálculo de precisão média pode ser visto como a área de um gráfico de precisão por *recall*. Na prática, as variações acendentes e descendentes do valor de precisão são aproximados de acordo com a equação 3.5, onde $P'(r)$ é o novo valor de precisão para r e $P(r')$ é o valor de precisão não suavizado para r'. O gráfico da suavização pode ser visto na figura 21

$$P'(r) = \max_{r' \geq r}(P(r')) \tag{3.5}$$

Figura 21 – Valor de precisão em relação ao aumento do *recall*(B) e sua suavização(A)



Fonte: Cálculo mAP [35]

Com o valor de precisão suavizado pode-se então calcular a média de precisão com base na equação 3.6, onde N é o número de predições.

$$AP = \frac{1}{N} \sum_{r \in (0.0,...,1.0)} P'(r) \tag{3.6}$$

O cálculo mAP é então a média do AP de todas as clases preditas, ou seja, para o problema apresentado nesse trabalho, onde existe apenas a classe pessoas, o mAP é igual ao AP.

4 Metodologia

Diversas escolhas foram tomadas durante o desenvolvimento do trabalho para atender os problemas levantados pelos clientes e também para direcionar o escopo de implementação do projeto, dado a vasta área que é a visão computacional.

4.1 A escolha do Deep Learning

Antes da expansão da utilização de redes neurais profundas para soluções de visão computacional, processamentos mais simples de imagem eram utilizadas para detecção de pessoas em uma imagem ou a previsão de quantas pessoas haviam em um determinado frame. Processos como a utilização de *Histogram of oriented gradients*(HOG) para extração de *features* da imagem e após isso a aplicação de um classificador *Support vector machine*(SVM), foram processos comumente utilizados, como o apresentado pela primeira vez oficialmente em 2005 por Dalal and Triggs [36].

Foi apenas em 2012 que as *Convolutional neural networks*(CNNs) se tornaram grandes opções para visão computacional. Quando Alex Krizhevsky, Geoff Hinton, e Ilya Sutskever venceram a famosa competição ImageNet [19]. A partir desse momento surgiram diversas melhorias onde a arquitetura base da CNN é modificada colocando novos *layers*, modificando a forma de conexão dos neurônios e entre outros processos.

Tendo em vista que os processos de visão computacional que não utilizam *Deep Learning* já estarem mais maduros em pesquisa e desenvolvimento, o objetivo desse trabalho é testar modelos mais novos, que são classificados como estado da arte para inferências em imagens e portanto ainda estão sendo validados para diferentes aplicações.

4.2 Uso de câmeras de segurança

É comum encontrar produtos de contagem de pessoas que são vendidos para serem implementados nas portas das lojas em um formato *plug and play*. Esses produtos são pequenas câmeras com um micro processador que faz a aquisição das imagens, processa localmente e envia os dados processados através de uma conexão remota, como por exemplo um Wi-Fi. Na figura 22 há um exemplo de hardware desse tipo que pode ser encontrado a venda em sites como AliExpress por mais de R$1.500 [37].

Apesar da facilidade de implementação de um produto desses em uma loja ser alta, quando se trata de uma rede varejista esse processo precisa ser escalado e acaba sendo altamente custoso. Em um cenário em que há 100 lojas em uma rede, o investimento

Figura 22 – Exemplo de produto de contagem de pessoas

Fonte: Highlight HPC008 [37]

apenas de aquisição dessas câmeras sai por entorno de R$150.000. Porém, além do preço de aquisição, é necessário considerar o custo da logística(pessoas, deslocamento, instalação) dessas câmeras em toda a rede de lojas.

Outro ponto crítico em relação a utilização de um hardware específico para isso é o custo de manutenção desses equipamentos, tanto para possíveis defeitos, mas também para eventuais atualizações de hardware ou até mesmo software.

Tendo em vista os problemas de escalabilidade desse tipo de solução, se propôs nesse trabalho maximizar a arquitetura já implementada na vasta maioria dos grandes logistas, que são as câmeras de segurança, a fim de minimizar os investimentos inicias.

Outra vantagem de utilização das câmeras de segurança é que a maioria delas já possui uma interface de disponibilização de suas imagens via *stream*. Isso permite centralizar o processamento em um servidor que pode processar diversas câmeras ao mesmo tempo e facilitando assim a manutenção dos sistemas. Com um hardware potente centralizado, é possível utilizar algoritmos mais robustos que são inviáveis em sistemas embarcados.

A utilização de câmeras de segurança com imagens via *stream* permite ainda levar o processamento a qualquer local, sendo o único empecilho o atraso de envio dos *frames* pela rede. Pode-se então processar as imagens em sistemas altamente performáticos da *cloud* utilizando *Graphics Processing Units*(GPUs) e com tempo de *setup* baixo. Essa fácil escalabilidade do hardware permite escalar a arquitetura de processamento podendo adicionar novas funcionalidades assim que essas forem desenvolvidas, sem a necessidade da troca física por um hardware mais potente.

4.3 Baixa necessidade de processamento

A utilização das câmeras de segurança já dispostas nas lojas é um grande fator de economia da solução proposta, porém não suficiente. Uma loja pode possuir inúmeras câmeras de segurança e assim o servidor de processamento tem de processar em tempo real não apenas uma imagem, mas múltiplas delas. Tanto a utilização de um servidor físico comprado pelo cliente(*on primese*), tanto a implementação na *Cloud*, podem se tornar bem caro caso um único servidor não consiga escalar a quantidade de câmeras a serem processadas em tempo real.

Por esse fator, a premissa considerada nesse projeto é que o algoritmo de visão computacional deve ser computacionalmente leve o suficiente para rodar em tempo real em uma CPU, ou seja, para os requisitos desse projeto, mais de 3 *frames* por segundo. Isso da um total de 0.33 milissegundos entre cada *frames*. A velocidade média de uma pessoa em um shopping é de $1.4m/s$ [38], portanto, no intervalo de um *frames* e outro o deslocamento médio de uma pessoa será entorno de $46cm$ tornando possível a utilização de um algoritmo de *tracking*, por exemplo.

O valor de deslocamento de uma pessoa pode ser ainda menor que o calculado tendo em vista que a locomoção dentro de uma loja é possivelmente mais lenta e o deslocamento real da pessoa não é o deslocamento observado em pixeis na câmera. Para gravações a maior distância, o deslocamento relativo da pessoa em pixeis será menor que o deslocamento real.

A CPU utilizada como base para os parâmetros aqui apresentados é um Intel® Core™ i7 7500U (2.7 GHz até 3.5 GHz, 4 MB L3 Cache) e com 8GB de RAM. Sendo a memória do computador utilizado de 8GB.

4.4 Rápida implementação

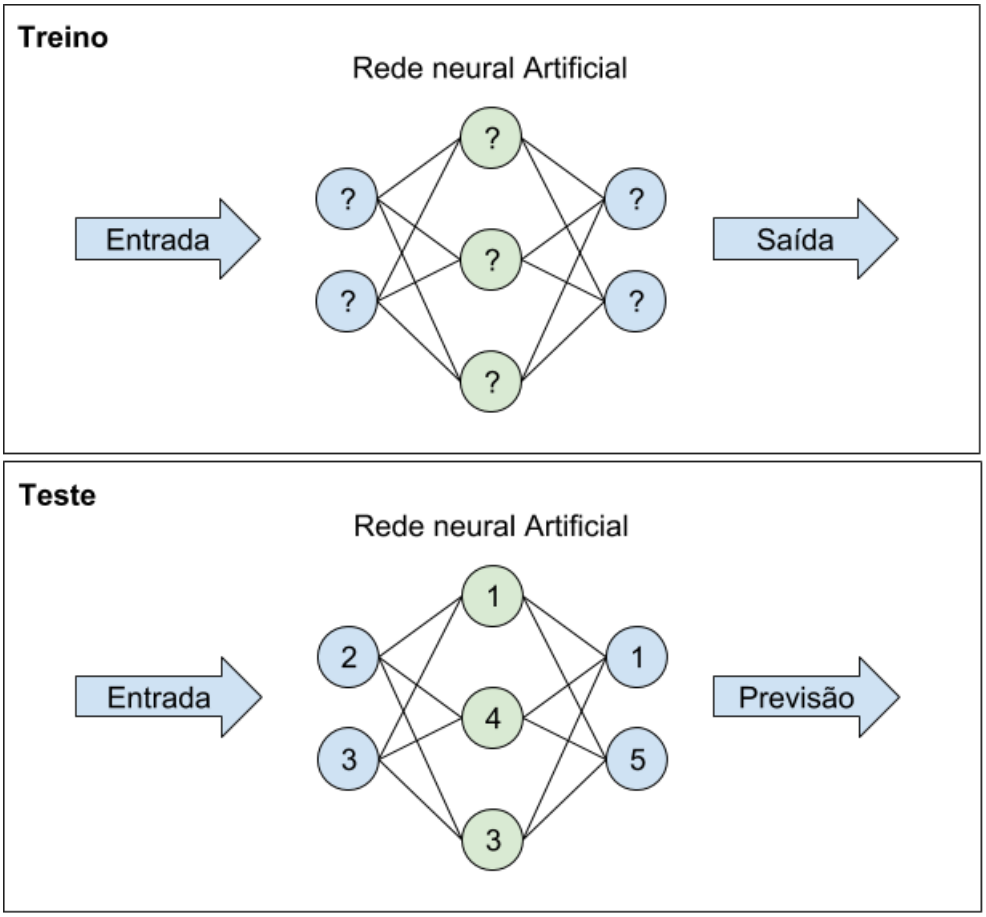
As redes profundas implementadas para a detecção de pessoas e objetos são algorítimos de aprendizado supervisionado. Nesse tipo aprendizado, os pesos da rede são ajustados para atender um comportamento de predição específico. Esse ajuste dos pesos é chamado de treino da rede, onde é passado os valores de entrada e os de saída, e através de uma iteração de propagações de erros, os pesos da rede vão sendo ajustados para se comportar como o desejado. A figura 23 descreve o processo de treinamento [1].

4.4.1 Dificuldades de treino de uma rede neural

Na grande parte das vezes as redes utilizadas para visão computacional são redes neurais convolucionais. Esse tipo de arquitetura apresenta diversas camadas com diversos pesos que devem ser setados ao longo do treinamento da rede (algumas arquiteturas

[1] Toda figura apresentada nesse trabalho sem sua respectiva fonte são de autoria do autor desse projeto

Figura 23 – Treino e teste de uma rede neural artificial



serão descritas no capítulo 3). Todo esse processo de treinamento possui um alto custo computacional devido as diversas operações numéricas e matriciais. Treinar uma rede para identificação de pessoas em uma imagem pode, por exemplo, levar alguns dias de alto processamento, mesmo utilizando GPUs que aceleram o processo.

Outra situação custosa no treinamento de uma rede é o levantamento do conjunto de entrada e saída para ser passado ao algoritmo. Para criar uma rede robusta é necessário passar centenas de imagens para a rede, além de que para se ter uma rede robusta ao final do treinamento, é fundamental que essas imagens sejam bem diversificadas, com fatores de iluminação, contraste e tipos de pessoas diferentes. Essa variedade é fundamental para a aplicação apresentada nesse projeto já que as há uma grande variação de câmeras e ambientes de lojas.

Definir o *Ground Truth*(valores de saída verdadeiros) do conjunto de treino também não é um processo simples. Em um sistema de identificação de objetos(melhor descrito no capítulo 3) é necessário passar para rede no processo de treino a imagem como entrada, e como saída todos os objetos que se deseja identificar, bem como as respectivas posições em pixeis na imagem. Fazer o levantamento disso manualmente em centenas ou até milhares de imagens pode ser bem custoso e ainda por cima, se feito por uma única pessoa, ser

adicionado erros aleatórios no processo de treino devido a tendencias no processo de rotulagem.

Para contornar essa situação de implementação e uso de redes mais complexas. A comunidade de *machine learning* disponibiliza na internet os pesos de redes pre treinadas em bancos de imagens disponibilizados também pela comunidade. Esses *data sets* de imagens já são rotuladas e possuem diversos objetos já levantados que podem ser utilizados para treino, como pessoas, cachorros, gatos, carros, entre outros. Com essas redes pré treinadas, é necessário apenas importar os pesos no algoritmo desenvolvido para poder passar a fazer predições em novas imagens. Importante ressaltar que esse processo é viável devido a padronização de algumas arquiteturas, que portanto, já possuem pesos treinados disponíveis na comunidade *web*.

Esse projeto usufruiu de arquiteturas famosas já treinadas, não necessitando passar por esse processo de treino. Importante ressaltar que treinar uma rede para o ambiente específico levantado iria melhorar os resultados obtidos, porém se definiu deixar fora do escopo o treinamento da rede para se ter um sistema mais rápido para implementação e poder assim focar em outras etapas do projeto.

4.5 Linguagem

Para o desenvolvimento desse projeto optou-se por utilizar a linguagem de programação Python. Isso se deve ao fato de que grande parte das bibliotecas e *frameworks* para *Data Science* e machine learning possuem uma interface com essa linguagem.

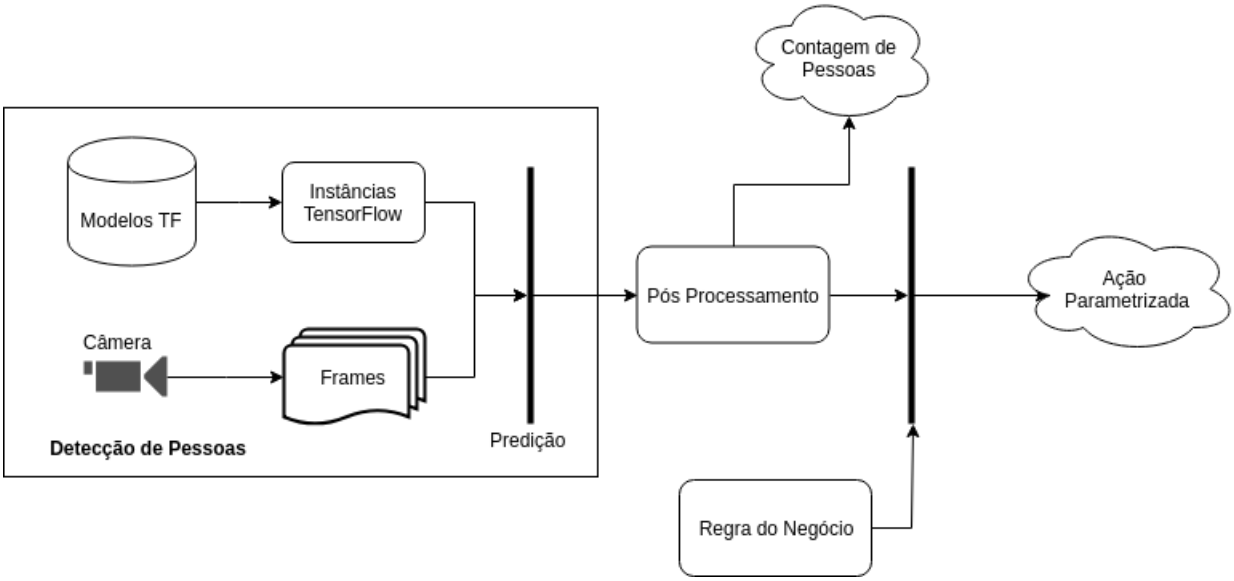
4.6 Arquitetura

A figura 24 apresenta o conceito da arquitetura de forma simplificada. A primeira estrutura identificada como o sistema de detecção de pessoas é o responsável por capturar os *frames* dos vídeos e processá-los em um sistema de detecção de objetos. Individualmente cada frame é encaminhado para uma instância *TensorFlow* [39] que detecta as pessoas na imagem com base no modelo que gerou a instância. A saída desse processo é um conjunto de posições na imagem contendo a informação se é ou não uma pessoa.

O módulo subsequente ao de detecção é o de pós processamento das localidades identificadas como pessoas. Esse módulo tem como finalidade contar a quantidade de pessoas detectadas obtendo assim um sinal de medição do sistema analisado. Esse sinal pode ser também processado para melhorar a acuracidade da contagem de pessoas.

A partir dos valores de contagem de pessoa, isso pode ser combinado com parâmetros definidos pelo negócio de aplicação. Por exemplo, em uma loja, pode-se definir uma quantidade n de pessoas que podem estar ao mesmo tempo na fila, caso esse valor seja

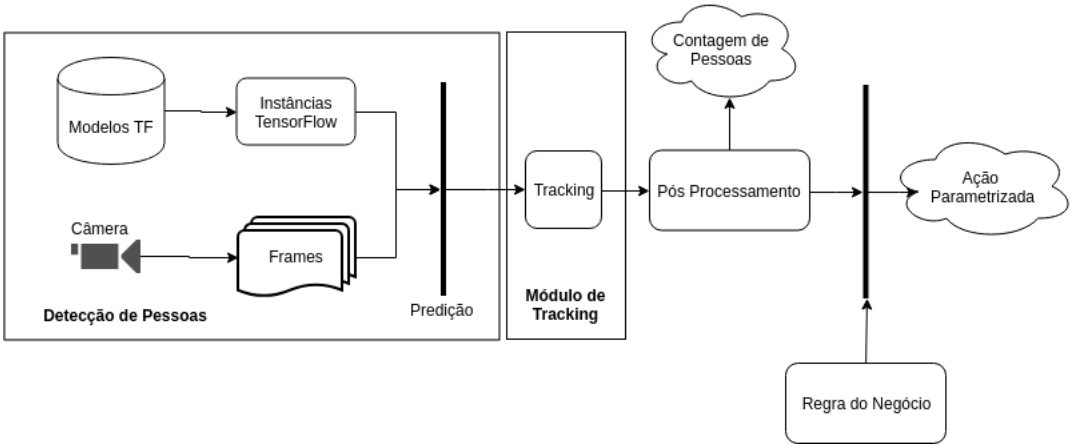
Figura 24 – Arquitetura de processamento do sistema



maior que n, o sistema encaminha um aviso para a loja indicando a necessidade de novos operadores de caixa.

Para a implementação de um sistema de *tracking* de pessoas com base nesse arquitetura, é necessário ter como saída do sistema de aquisição a posição exata das pessoas identificadas nos *frames*. É por esse fator que se optou por utilizar modelos de detecção de objetos, que identificam as pessoas antes de propriamente fazer a contagem. A desvantagem desse sistema é que não existe uma identificação única. Os *frames* são individualmente processados e por isso, o sistema não reconhece que é uma mesma pessoa em dois *frames* seguidos. Isso não permite fazer, por exemplo, a contagem de quantas pessoas estiveram na loja, já que para esse tipo de processo é necessário uma identificação única.

Figura 25 – Arquitetura de processamento com sistema de *tracking* de pessoas



Para uma identificação única pode-se utilizar um sistema de *tracking*, já que para

efetuar esse processamento o modelo precisa reconhecer pessoas iguais em uma sequência de detecções. Com a arquitetura desenvolvida, pode-se facilmente implementar um módulo de *tracking* de pessoas. Sua entrada é as caixas delimitadoras e os *frames* do vídeo capturado. Com essas informações, o modulo de *tracking* irá comparar com base nas características dos *frames*, as pessoas que já foram detectadas em *frames* anteriores adicionando a elas um identificador único. A arquitetura atualizada com um módulo de *tracking* é representado na figura 25.

4.7 Sistema de Detecção de Pessoas

O sistema de aquisição recebe como entrada um vídeo, seja ele em tempo real(*streaming*) ou gravado, e tem como saída as caixas delimitadoras de cada pessoa encontrada em cada *frame* do vídeo. Essa saída foi registrada em um arquivo para futura análise, como será descrito a seguir.

Há várias implementações *open source* de redes convolucionais disponíveis pela comunidade e que facilitam o tempo de desenvolvimento das arquiteturas mais complexas. Durante o desenvolvimento do projeto foram testados várias implementações diferentes que utilizam a linguagem Python, ao final foi elencado a melhor implementação, considerando velocidade de processamento do algoritmo e flexibilidade para possíveis alterações.

A implementação utilizada nesse trabalho é a criado por Léo Beaucourt [40] em 2018 e que tem seu código disponível no GitHub. A arquitetura desenvolvida contempla o uso de ferramentas como Docker, TensorFlow e OpenCV, sendo desenvolvida em Python 3. Mais detalhes sobre a importância de cada módulo será descrito a seguir.

4.7.1 Docker

Docker é uma plataforma *Open Source* desenvolvida pela Google na linguagem Go. Sua função é facilitar a criação e administração de ambientes isolados [41]. O Docker possibilita isolar uma aplicação ou um ambiente dentro de um *container*, este pode ter seu próprio sistema operacional, seus compiladores e *frameworks* instalados, sem afetar o ambiente externo. Na prática, não é necessário instalar, por exemplo, a linguagem python no sistema operacional. É possível, em vez disso, utilizar inúmeros *containers* que possui dentro dele a linguagem instalada de formas diferentes, se necessário.

A grande vantagem de utilização do Docker é a possibilidade de criar uma imagem de um *container*. Essa imagem contém toda a característica do ambiente gravado e dessa forma pode ser replicado em qualquer outro sistema apenas executando a imagem gravada.

A escolha de utilização do Docker nesse projeto é em decorrência do desejo de rapidamente poder implementar a arquitetura desenvolvida em qualquer máquina. Além

disso, como o processamento das imagens será feito de forma descentralizada, a utilização de uma imagem Docker permite criar de forma rápida e escalável contêineres já prontamente configurados em serviços *cloud* como o Google Cloud ou a Amazon Web Service.

Além da escalabilidade, o Docker foi de suma importância para a implementação dos testes desenvolvidos. Diversas foram as bibliotecas, linguagens e *frameworks* utilizados para rodar redes convolucionais, onde muitos desses serviços tinham dependências de versões diferentes. Dessa forma, a separação de ambientes utilizando *containers* foi fundamental para não haver conflitos de versão entre as diferentes ferramentas testadas.

4.7.2 Open CV

Open CV é uma biblioteca grátis de visão computacional, desenvolvida em C++ e que possui interface de utilização em linguagens como Python e JAVA. Sua utilização está relacionada a captura do vídeo em diferentes *frames* facilitando nesse processo a obtenção de vídeos de diferentes fontes e facilitando também o pré processamento dos *frames*, que podem ser redimensionados ou sofrerem tratamento de cor. Os *frames* já processados são então aplicados a rede neural. Com a predição concluída, tem-se as posições dos objetos nas imagens, que pode-se desenhar no frame a caixa delimitadora encontrada e então visualizar essa frame em tempo de execução, acompanhando as previsões do algoritmo.

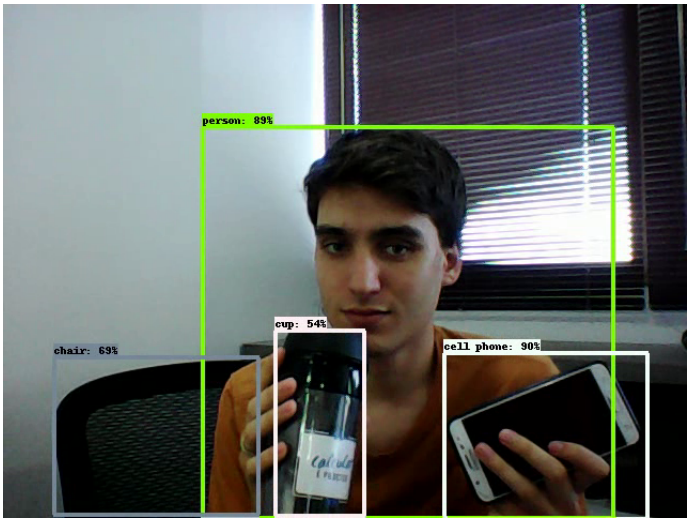
O sistema de aquisição de *frames* com base em vídeos precisou ser desenvolvido para suportar vídeos já gravados em diferentes formatos, vídeos sendo gravado em tempo real por uma câmera conectada no computador, como uma *Webcam* e também suportar a aquisição de vídeos de stream disponibilizado por câmeras de segurança. Com o Open CV essa situação foi feita utilizando a função *VideoCapture* da biblioteca.

A rede neural implementada tem como saída de forma simplificada três vetores, um contém as caixas delimitadoras encontradas, outro contém as classes dessas caixas e o terceiro vetor a confiabilidade de cada predição feita. Com essas informações é possível mostrar na tela os *frames* a medida que esses são processados. A figura 26 contém um exemplo da saída mostrada ao usuário. Além disso, o vídeo processado é gravado com as caixas delimitadoras para futura análise.

4.7.3 TensorFlow

Para auxiliar na implementação de redes neurais e principalmente redes profundas, foram desenvolvidas inúmeras bibliotecas ou *frameworks* que auxiliam nesse processo, abstraindo toda a implementação dos cálculos matriciais e otimizando o processo de rodar em diversas *threads* e em GPUs. Uma dessas bibliotecas é o TensorFlow, que foi desenvolvido pela Google e é de código aberto para uso. Ele facilita na implantação de arquiteturas já consolidadas, mas também permite criar suas próprias arquiteturas

Figura 26 – Visualização da saída da rede neural para um frame



fornecendo funcionalidades para treino e uso delas para predição.

Utilizando a biblioteca TensorFlow é possível carregar uma arquitetura e os pesos pré treinados. Com a instância TensorFlow inicializada e com os pesos do modelo em memória, é possível então fazer as predições para cada *frame* capturado da imagem.

4.7.3.1 Model Zoo

Uma das grandes vantagens da utilização do TensorFlow para a implementação desse projeto foi o repositório disponibilizado pela Google *TensorFlow detection model zoo* [42]. Esse repositório no GitHub da Google contém diversos modelos pré treinados em bases famosas de imagens que por coincidência possuem como classe(objeto) de treino as pessoas. Há diversos modelos com diferentes acuracidade e velocidade de predição que podem ser utilizados da mesma forma que disponibilizados ou podem ser usados para iniciar os pesos de uma rede que será treinada.

O repositório disponibiliza os modelos apresentados na tabela 2 no formato de um modelo *frozen TensorFlow*. Com esses pesos do modelo baixados, basta alterar a fonte do peso do modelo para o novo baixado no código descrito em A.3, que desse forma já se terá uma nova arquitetura implementada para fazer predições sobre os *frames*. Portanto, o aquivo baixado contempla não apenas o peso da rede, mas toda a configuração de sua arquitetura, que ao ser carregado pelo TensorFlow, já adapta a instância para a nova estrutura, facilitando assim o teste de diferentes modelos de arquitetura convolucionais.

Importante ressaltar que os tempos acuracidades apresentadas na tabela foram parâmetros levantados em *data sets* famosos de treino e em uma Nvidia GeForce GTX TITAN X card, bem diferente da implementação proposta nesse trabalho. Por esse fato, fez sentido para esse projeto reavaliar os diferentes modelos para a predição exclusivamente de pessoas e rodando na CPU já apresentada.

Tabela 2 – Modelos disponibilizados no Model Zoo

Modelo	Velocidade	mAP_{COCO}	Tipo
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco	26	18	Boxes
ssd_mobilenet_v1_quantized_coco	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco	29	16	Boxes
ssd_mobilenet_v1_ppn_coco	26	20	Boxes
ssd_mobilenet_v1_fpn_coco	56	32	Boxes
ssd_resnet_50_fpn_coco	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_low...	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes

Fonte: *TensorFlow detection model zoo* [42]

4.7.4 Implementação com multi processos

A implementação proposta por Léo Beaucourt [40] é rápida comparada com outras implementações comumente feitas. Na implementação original que utilizava uma rede SSD com extração das *features* pela MobileNet v2 o FPS na CPU apresentada foi de 10.65, atingindo o requisito do projeto e mostrando grande destaque frente a uma implementação comum.

O alto desempenho dessa implementação é em decorrência de ter sido implementado com um biblioteca de *multithreading* no Python. A abordagem feita é criar um *pool* de *workers* para processamento dos *frames*, ou seja, instânciando vários modelos que podem predizer ao mesmo tempo sobre *frames* distintos, paralelizando assim o processo de inferência. Esses *workers* obtém os *frames* de uma pilha FIFO lidos continuamente, que após serem processados, são colocados em uma segunda pilha de *frames* já preditos.

Listing 4.1 – Criação de um *pool* de *workers*

```
import argparse
from multiprocessing import Queue, Pool

#Inicializa as pilhas de frames com queue_size de tamanho
```

```
#Inicializa a pool com num_workers trabalhadores
input_q = Queue(maxsize=args["queue_size"])
output_q = Queue(maxsize=args["queue_size"])
pool = Pool(args["num_workers"], worker, (input_q,output_q))
```

O código 4.1 apresenta a criação de duas pilhas, uma de entrada e uma de saída. Apresenta também a criação do *pool* que recebe como parâmetro a função *worker*, função essa definida para instanciar o modelo e prever em *loop* os *frames* adicionados na pilha de entrada.

4.7.5 Saída

Uma das adaptações necessárias no código original foi a geração de um arquivo de saída com as posições das caixas delimitadoras de todas as pessoas detectadas frame a frame. Esse arquivo de exportação gerado foi utilizado para comparar com as detecções verdadeiras conhecidas e assim calcular a acuracidade do modelo.

As informações necessárias de saída do algoritmo para poder calcular a acuracidade do modelo são as posições de cada caixa delimitadora e a confiabilidade da predição daquela caixa. Os modelos utilizados nessa implementação foram treinados no famoso *data set* COCO. Essa base de dados possui inúmeros objetos e, portanto, a rede é capaz de detectar não apenas pessoas, mas cachorros, gatos, celulares, cadeiras, entre outros. Porém, na saída do algoritmo, é filtrado apenas pessoas já que os outros objetos não tem aplicação nesse projeto.

O formato do arquivo de exportação foi padronizado para ser utilizado por um outro código que será apresentado na próxima sessão. A exportação não é um arquivo único, mas um conjunto de n arquivos, onde n é a quantidade de *frames* gravados. Cada arquivo deve conter as pessoas detectadas no respectivo *frame* com uma linha para cada detecção no seguinte formato:

$$< Nome\_Classe > < confiabilidade > < esquerda > < cima > < direita > < baixo >$$

A figura 27 apresenta o que significa cada posição de saída das caixas delimitadoras. Os valores são representados em pixeis e são referenciados na borda superior e esquerda da imagem. No exemplo da figura 27, com uma imagem de tamanho 768 x 576, os valores de esquerda, cima, direita e baixo foram respectivamente 256, 220, 286 e 308.

4.7.6 Implementação da YOLO

Um dos modelos estado da arte para detecção de objetos em tempo real é a arquitetura YOLO e era uma das grandes apostas como melhor modelo a ser testado. O

Figura 27 – Formato de localização de uma caixa delimitadora

framework TensorFlow utilizado, no entanto, foi desenvolvido para rodar as arquiteturas apresentadas no formato do Model Zoo sendo necessário adaptações para a implementação de uma arquitetura como a YOLO. Devido a isso, foi utilizado uma implementação secundária com a finalidade de testar a arquitetura YOLO v3 e sua versão mais leve, a YOLO v3 Tiny.

Essa nova implementação foi feita utilizando como base o código apresentado por Sunita Nayak [43] e disponível no GitHub. A implementação proposta utiliza originalmente a YOLO v3 para detecção de objetos, porém foi adicionado a possibilidade de utilizar a YOLO v3 Tiny baixando os pesos e configurações da arquitetura no site oficial da YOLO.

Nessa segunda implementação não foi utilizado o *framework TensorFlow*. Nessa situação foi utilizado o módulo de *deep learning* do OpenCV, o *dnn*. Utilizando esse módulo é possível carregar arquiteturas desenvolvidas em diferentes *frameworks*, como TensorFlow, caffe, pythorch e Darknet, como é o caso da YOLO, que foi originalmente desenvolvida em Darknet.

4.8 Métricas de avaliação dos modelos

Para ser possível comparar diferentes modelos convolucionais é necessário ter um processo de teste e uma métrica de qualidade dos modelos, dessa forma conseguindo quantificar as qualidades das arquiteturas. Em uma algoritmo de *machine learning* o processo de análise da qualidade das predições é feita utilizando o conceito de treino e teste apresentado na sessão 3.4.1. Para efetivar a etapa de teste é necessário que se tenha os valores verdadeiros que devem ser preditos pelo algoritmo, dessa forma pode-se comprar

as previsões com os valores reais e assim calcular a acuracidade do modelo.

Para uma rede convolucional o processo de teste precisa também conhecer os valores reais a serem preditos, ou seja, para a aplicação de identificação de pessoas em uma imagem, é necessário conhecer essas identificações previamente para poder comparar com as predições. Por esse motivo, o teste dos modelos foram feitos em cima de um vídeo já previamente conhecido e não sobre um vídeo em *streaming* de uma câmera de segurança. Conhecendo previamente a quantidade e localização das pessoas no vídeo foi possível calcular a qualidade das predições feitas. Além disso, não há perda de representatividade nas análises utilizando um vídeo pré gravado já o modelo de predição é o mesmo, recebendo um frame de entrada e prevendo os objetos.

Como fonte de vídeo para a contagem de pessoas uma possibilidade era gerar esse vídeo utilizando uma câmera comum e então regirar as posições e quantidade de pessoas manualmente. O problema dessa abordagem é os erros de classificação manual que podem ser gerados se apenas uma pessoa fizer a identificação. É necessário para diminuir essa tendência de erro efetuar as localizações por diferentes pessoas, obtendo os pontos em comum, porém, se tornando um processo custoso para ser corretamente implementado. Além disso, a qualidade do arquivo teste precisa contemplar todos os testes situacionais possíveis de ocorrerem, portanto, utilizar poucas pessoas para gerar o vídeo, com roupas sempre iguais, se movendo de forma parecida e em distâncias fixas da câmera, podem gerar tendências no processo de teste e não contemplar a qualidade final dos modelos.

Como fonte de teste dos modelos foi utilizado uma base reconhecida de teste de algoritmos de *tracking* de pessoas. O MOTChallenge é um *framework* desenvolvido para testar e comparar modelos em bases de *multiple object tracking*. Nesta base há um conjunto vasto de vídeos voltados a identificação de pessoas em ambientes diversos, onde o grande destaque é ter oficialmente os valores verdadeiros de predição de pessoas. Na vasta quantidade de vídeos disponíveis foi escolhido o PETS09-S2L1 [44] devido a sua similaridade a câmeras de segurança localizadas a uma maior distância das pessoas. O vídeo também contempla uma variada quantidade de pessoas em diferentes sobreposições e movimentações. Além disso, diferentes de muitos vídeos do MOTChallenge, o vídeo escolhido tem uma angulação de gravação similar a câmeras de segurança.

4.8.1 Medida de Acuracidade

Com as informações verdadeiras e as preditas ainda é necessário se ter uma medida de comparação para verificar a taxa de acerto do modelo testado. Para quantificar a qualidade geral do modelos com as detecções de pessoas foi utilizado a medida de acuracidade mAP(*mean Average Precision*) devido a ser uma das medidas de acuracidade mais comuns para detecção de objetos. Basicamente quanto maior o mAP do modelo testado, maior é sua capacidade de detectar bem objetos da classe analisada.

A implementação do cálculo dessa medida para os modelos testados foi feita utilizando a linguagem python e usou como base o código desenvolvido por João Cartucho [45] e disponibilizado no GitHub. O algoritmo recebe como entrada a localização real de todos os objetos de cada frame, com a descrições de suas classes. Da mesma forma, recebe como entrada os objetos preditos, suas classes e os valores de confiança, seguindo o formato descrito anteriormente como saída do sistema de aquisição.

A saída desse algoritmo implementado é o cálculo do *average precision* para cada classe detectada, calculando ao fim a média desses valor, obtendo o mAP. Nos testes relativos ao trabalho, apenas a classe pessoa foi utilizada, tenso assim o valor de AP igual ao mAP.

Um dos parâmetros de cálculo utilizados para obter o mAP dos modelos testados é mínimo valor de IoU para ser considerado como uma detecção verdadeira. Com base no que geralmente é utilizado para esse parâmetro, o valor definido de sobreposição foi de 50% entre as caixas reais e as preditas. Isso impacta grandemente no valor final de mAP do modelo testado, já que se o modelo tiver uma tendencia de predizer caixas menores para as pessoas detectadas, as predições são consideradas como falsas, mesmo que a pessoa tenha sido detectada(um exemplo será mostrado no capítulo de resultados). Portanto, o calculo de mAP demostra a habilidade e a qualidade de detecção do modelo testado, basicamente sua robustez, não sendo apenas uma medida relativa a capacidade de detecção.

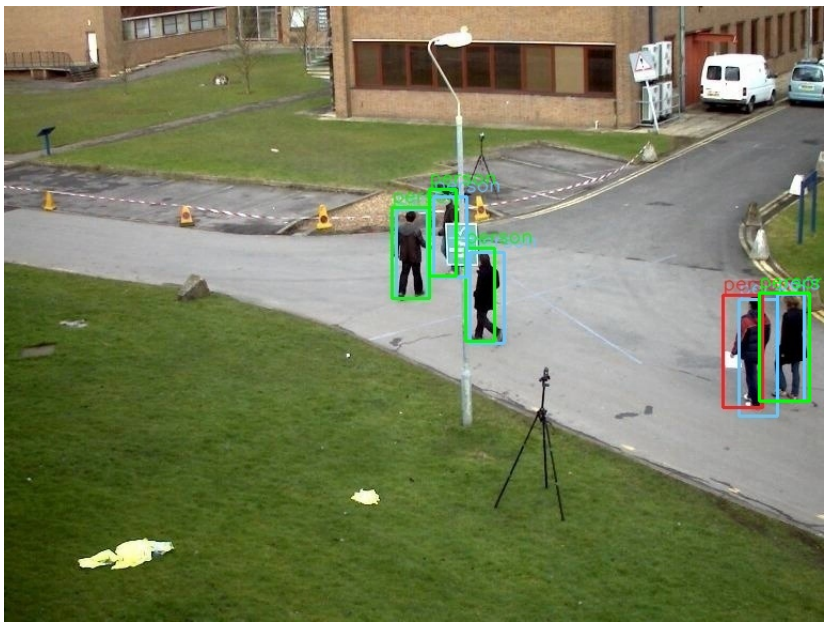
Outras saídas do algoritmo utilizado é o cálculo do verdadeiro positivo e falso positivo de cada modelo, podendo assim calcular o valor de precisão e *recall* dos modelos testados. Essas medidas não foram utilizadas como fundamentais na análise de resultado, mas podem ser utilizadas como critério de decisão dependendo dos requisitos de negócio da implementação estudada.

Por fim, o algoritmo disponibiliza como saída uma análise visual *frame* a *frame* dos valores preditos e dos valores reais, mostrando a qualidade de detecção do *frame*(figura 28). Essa análise é fundamental para entender as dificuldades do modelo testado e será abordado em mais detalhes no capítulo de resultados.

4.8.2 Erro na contagem de pessoas

A medida mAP não é suficiente para analisar a qualidade dos modelos testados. Como já apresentado, essa medida quantifica a qualidade da detecção. Entretanto, o objetivo final não é necessariamente obter a maior qualidade possível de localização das caixas delimitadoras, o objetivo final é obter o valor correto de pessoas em cada *frame*. Para ter uma medida mais correlacionada com o objetivo final, foi proposto a utilização do erro de predição em cada frame. A equação 4.1 apresenta o cálculo do erro de contagem de cada modelo, onde K é o número de *frames* do vídeo analisado. Sua interpretação é

Figura 28 – Análise do Frame 22 usando YOLO v3. Em azul o valor real, em verde a detecção correta e em vermelho a considerada errada



mais direta retratando o erro médio de pessoas a cada *frame*.

$$Erro = \frac{1}{K}\sum_{i=1}^{K}|Pred - Real| \tag{4.1}$$

Para se analisar esse indicador de acuracidade é importante ter como base a aplicação que está sendo analisada. Um modelo de $Erro = 1$ é ruim para uma câmera em um corredor de loja, porém pode ser um excelente valor para uma câmera na entrada de um show. Quanto maior a quantidade de pessoas em uma imagem, maior vai ser o número de pessoas possivelmente não detectadas, sendo isso proporcional a qualidade do modelo. Essa medida então pode ser utilizada nesse trabalho pois a situação aplicada é a mesma para todos os modelos, entretanto não é uma boa medida para comparar com outras cenas e câmeras.

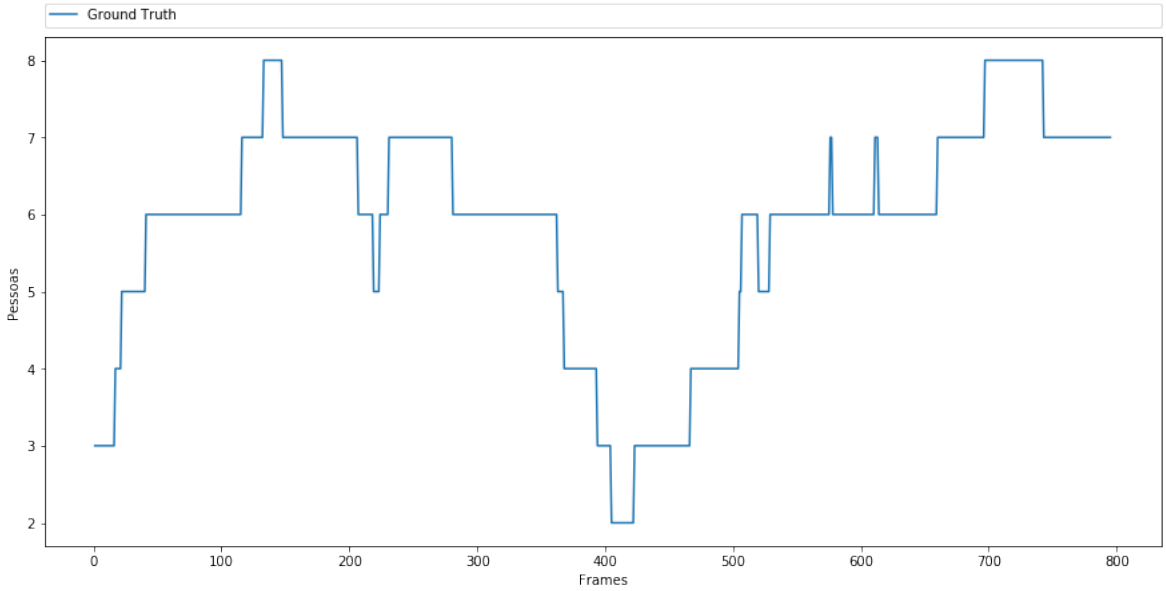
Para normalizar essa medida a proposta é analisar o erro percentual em relação a quantidade de pessoas em cada frame, quantificando assim o quanto percentualmente o modelo erra em média na contagem de cada frame. A equação 4.2 descreve o cálculo do erro percentual onde q_i é a quantidade real de pessoas no frame i.

$$Erro\% = \frac{1}{K}\sum_{i=1}^{K}\frac{|Pred - Real|}{q_i} \tag{4.2}$$

4.9 Sistema de pós processamento

Tendo como entrada a localização de cada pessoa detectada na imagem e o fator de confiabilidade da respectiva detecção, esse módulo tem como finalidade fazer a contagem total de pessoas em cada frame. A simples contagem das pessoas é um processo simples já que basta contar quantas detecções foram feitas para o frame. Entretanto, como nem sempre as detecções são feitas com alta qualidade, podendo de deixar de detectar uma pessoa ou detectar duas vezes a mesma pessoa, é possível fazer um sistema para tentar minimizar os erros de contagens e localizações que não fazem sentido prático para a aplicação analisada.

Figura 29 – Quantidade real de pessoas por frame

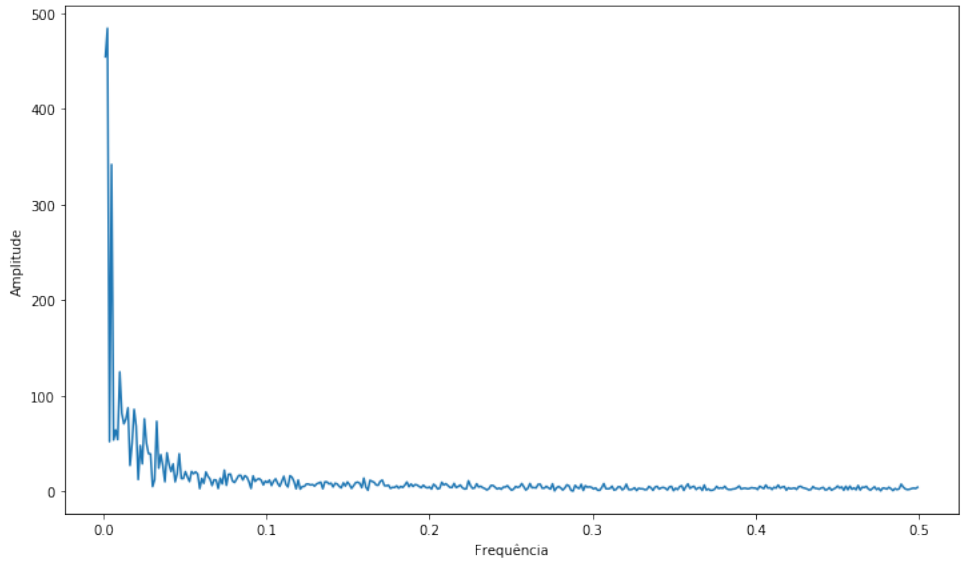


A figura 29 apresenta a quantidade de pessoas reais no tempo, ou seja, o sinal real a ser medido pelo sistema de aquisição. Tendo esse sinal, é possível observar as frequências do sistema e comparar com as frequências apresentadas pelo sinal predito. Essa análise de frequência possibilita auxiliar no desenvolvimento de um filtro para melhorar o sinal predito.

O foco de desenvolvimento desse trabalho não é o pós processamento de um sinal, entretanto, uma análise simples foi efetuada para auxiliar a criação de um filtro que já garantiu bons resultados.

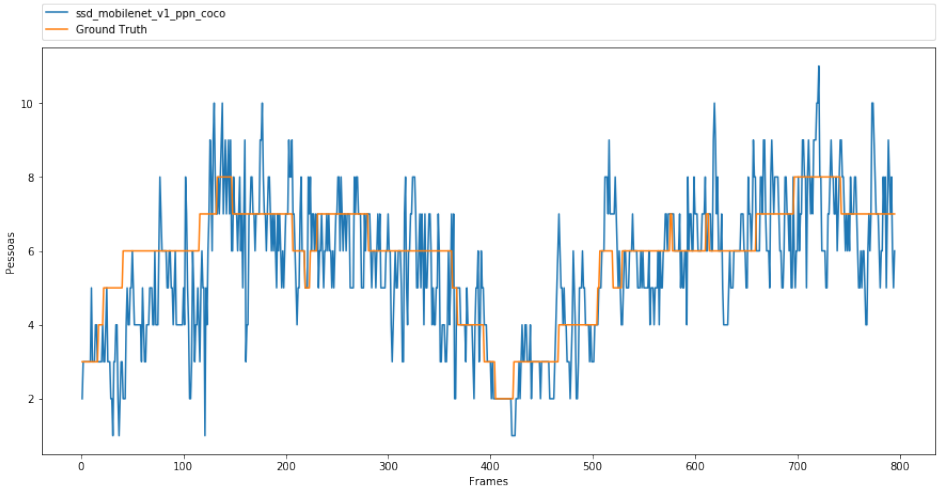
A figura 30 apresenta a analise em frequência do sinal de contagem verdadeiro. Aplicou-se uma *Fast Fourier Transform*(FFT) [46] e observou-se apenas os valores positivos de frequência devido ao espelhamento no eixo y. Observa-se na análise maiores componentes em baixa frequência, estando de acordo com o esperado já que não há variações tão bruscas de contagem de pessoas em uma câmera de segurança. Algumas frequências maiores

Figura 30 – Análise de frequência da contagem real de pessoas nos frames



acabam aparecendo no espectro devido a momentos de variação repentina na quantidade contada.

Figura 31 – Contagem de pessoas por frame utilizando o modelo PPN



A figura 31 apresenta a predição de um dos modelos utilizados para avaliação, nela é possível observar o alto ruído. Já a figura 32 apresenta a análise frequencial do valor predito, que comparando com o FFT do valor real, observa-se componentes em alta frequência, podendo classificá-los como os ruídos do sistema de medição. Com base nisso, optou-se por aplicar um filtro passa baixa, a fim de reduzir as componentes de maior frequência e assim obter um sinal mais parecido com o valor real.

O filtro aplicado foi um *butterworth* [47] passa baixa com ordem 2 e frequência de corte em 0.05. Na figura 33 observa-se o resultado frequência da aplicação do filtro sobre o sinal com a redução das componentes de alta frequência.

Figura 32 – Análise de frequência do modelo PPN

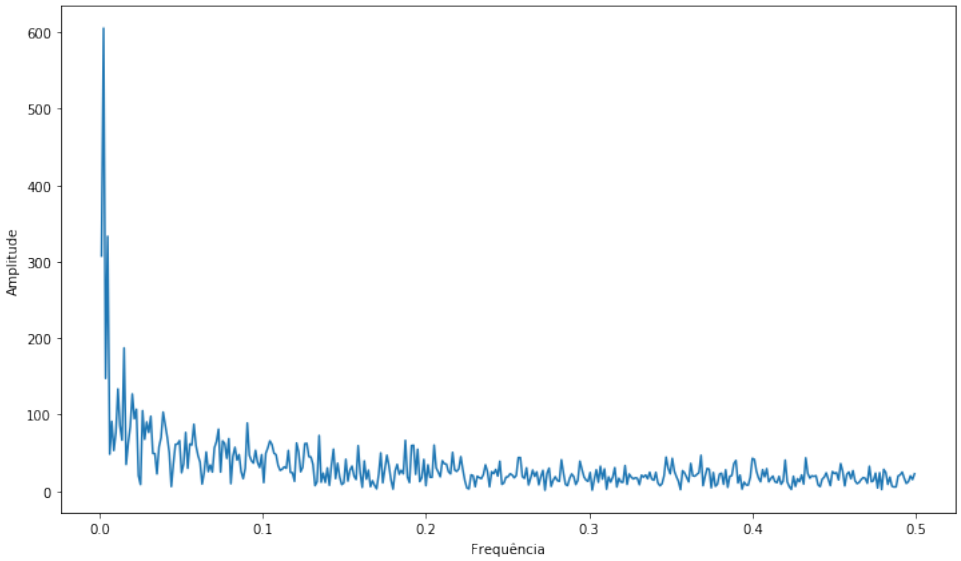
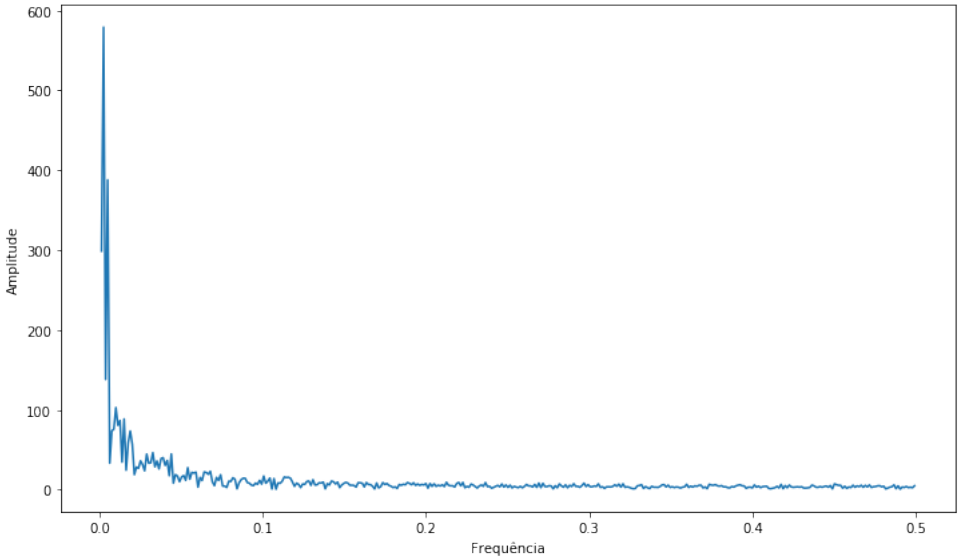


Figura 33 – Análise de frequência do modelo PPN filtrado



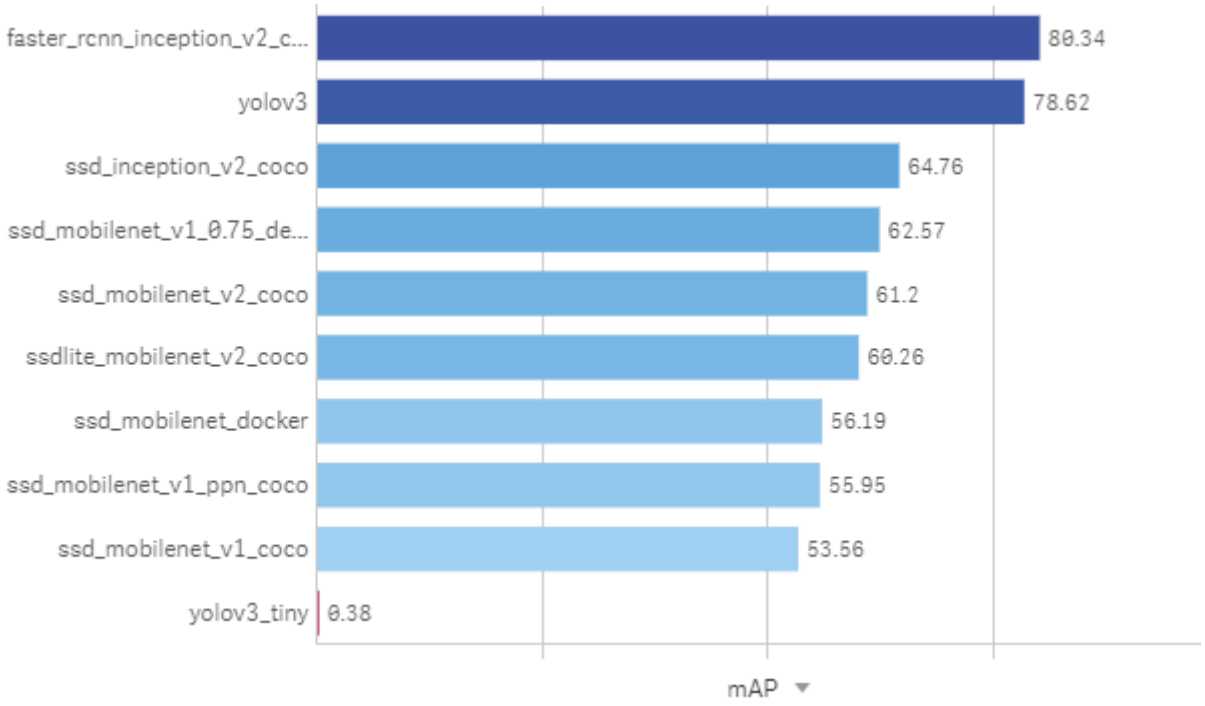
5 Resultados

O sistema de aquisição e o algoritmo de mAP geraram diferentes informações sobre os modelos testados que podem então ser usadas para comparar e escolher o melhor modelo para a aplicação apresentada.

5.1 Modelos e erros

A imagem abaixo apresenta o valor de mAP para todos os modelos testados no vídeo do *data set* MOT. Na figura 34 é possível observar que os modelos mais robustos, ou seja, com maior mAP foram os considerados mais pesados, que podem ser usados para tempo real, mas geralmente não suportam esse tipo de aplicação para processamentos em CPU. Tanto Faster-RCNN quanto a YOLO v3 são considerados estado da arte em qualidade de detecção e a análise comprova isso.

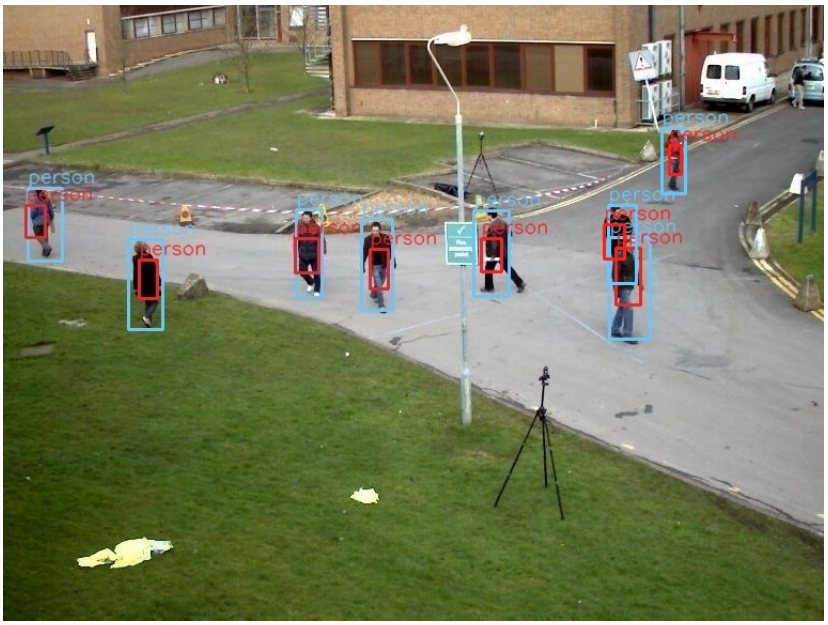
Figura 34 – mAP dos modelos testados



Na figura 34 a YOLO v3 Tiny aparece também com um valor muito baixo de mAP o que pode levar a mostrar que essa arquitetura tem baixa qualidade de predição. Entretanto, como já mencionado anteriormente, o mAP mede não só se as pessoas são detectadas, mas mede também a qualidade dessa detecção. Na figura 35 há um exemplo de um frame onde essa arquitetura detectou corretamente a quantidade de pessoas, entretanto

ela teve a característica de colocar pequenas caixas delimitadoras, não atingindo o IoU de 50%, caindo assim bastante o valor do mAP.

Figura 35 – YOLO v3 Tiny detectando todas as pessoas, mas com baixo AP. As caixas azuis representam o *Ground Truth* e as vermelhas os erros de detecção



Já na figura 36 pode-se observar a velocidade dos modelos testados. Alguns modelos atingiram alta velocidade de predição, passando o mínimo estabelecido, processando em uma CPU e com uma imagem de 768 x 576. Infelizmente o R-CNN e o YOLO v3 não atingiram o mínimo de velocidade e irão servir apenas como comparação de um modelo de alta acuracidade.

Tendo a velocidade dos modelos e sabendo que são factíveis de serem implementados em uma CPU, é necessário analisar a acuracidade desses modelos e quais possuem o maior custo benefício. Entretanto, como já mencionado anteriormente, a medida mAP não é satisfatória para ser usada como comparação da qualidade de contagem das pessoas. Na figura 37 observa-se a pouca relação entre os dois indicadores, destacando essa discrepância para o caso da YOLO v3 Tiny. Portanto, a medida de acuracidade a ser utilizada como prioritária é o erro de predição médio percentual.

Quando se trata do erro médio percentual dos modelos, a melhor resposta obtida é da YOLO v3, com uma média percentual de $6,7\%$ por frame predito, representando $0,36$ de erro por frame, porém ao mesmo tempo teve o pior tempo de predição. O pior modelo em erro foi a arquitetura SDD Mobilenet v1 com Depth com $33,7\%(1,81)$. Essa arquitetura apresenta um processo de otimização de velocidade em sua arquitetura, mostrando assim o melhor tempo de predição dos modelos, mas uma baixa acuracidade.

Observa-se com os resultados iniciais apresentados que existe uma ponderação na escolha do modelo, já que mais velocidade representa menor acuracidade e vice versa.

Figura 36 – FPS dos modelos testados

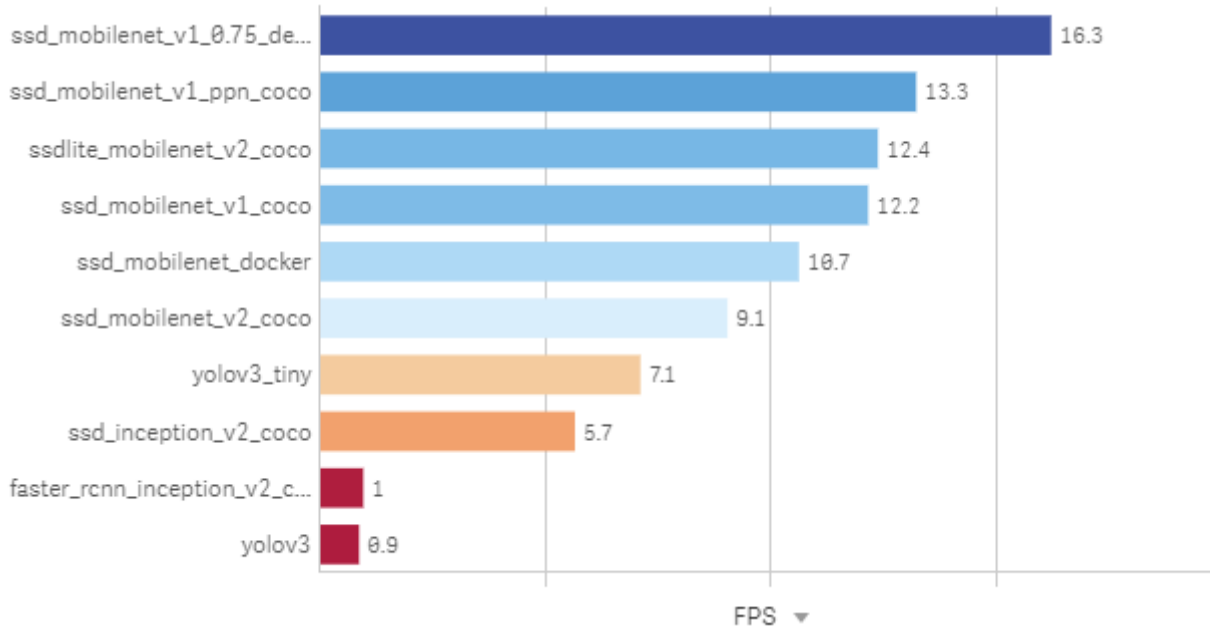
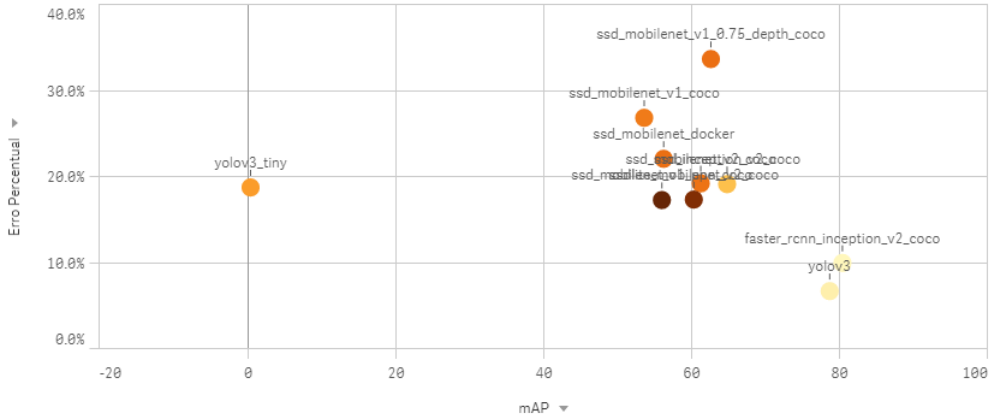


Figura 37 – Relação entre o mAP e o erro percentual dos modelos

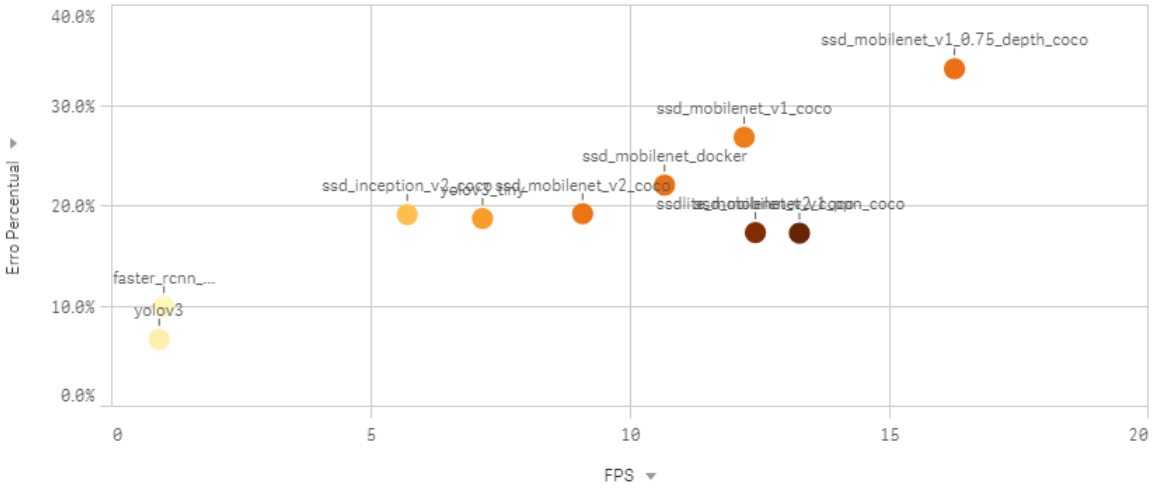


Para então obter a melhor opção entre os modelos pode-se analisar a dispersão dessas arquiteturas quanto a sua velocidade e seu erro de predição. A figura 38 apresenta um gráfico de dispersão onde é possível ver a tendência de modelos mais rápidos terem pior acuracidade.

A intensidade das cores na figura 38 representa, quanto mais escura, uma maior relação entre $FPS/Erro$, podendo-se chamar de ganho do modelo. Quanto maior o ganho do modelo, maior é o custo benefício de sua utilização, obtendo a maior acuracidade possível com o menor custo de processamento.

Dois modelos destacaram-se dos demais devido ao seu alto ganho de custo benefício. Sendo na ordem de maior erro para menor erro o SSD Lite Mobilenet v2 e o SSD Mobilenet v1 PPN. Ambos obtiveram uma velocidade e um erro muito similar, porém se destacaram

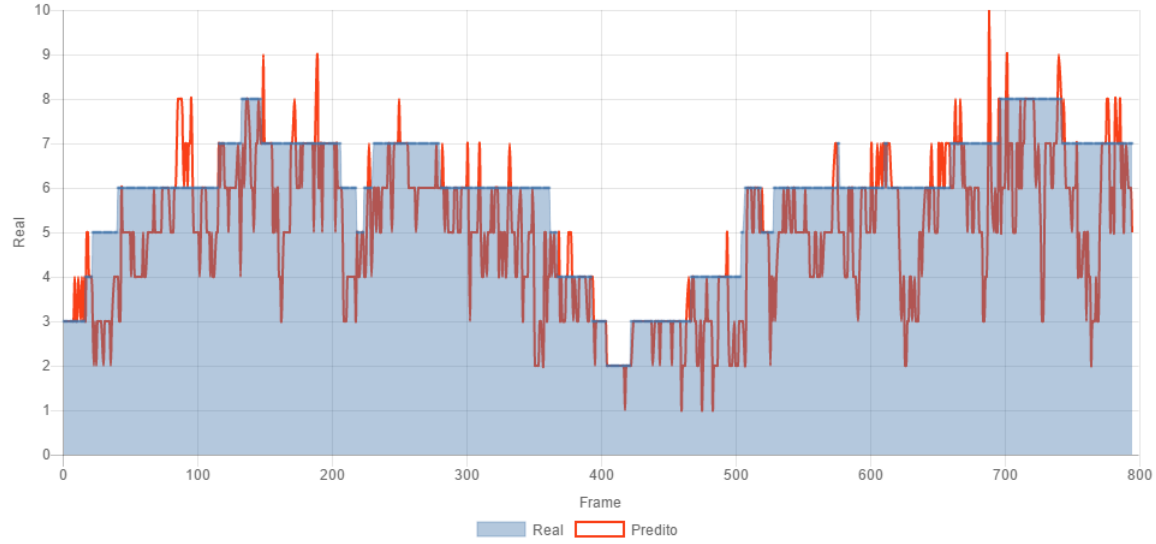
Figura 38 – FPS dos modelos versus o erro médio. Quanto mais escuro a cor, maior é a relação $FPS/Erro$



em relação aos outros modelos já que possuem o segundo menor erro entre os modelos possíveis de serem usados e por serem o segundo mais rápido.

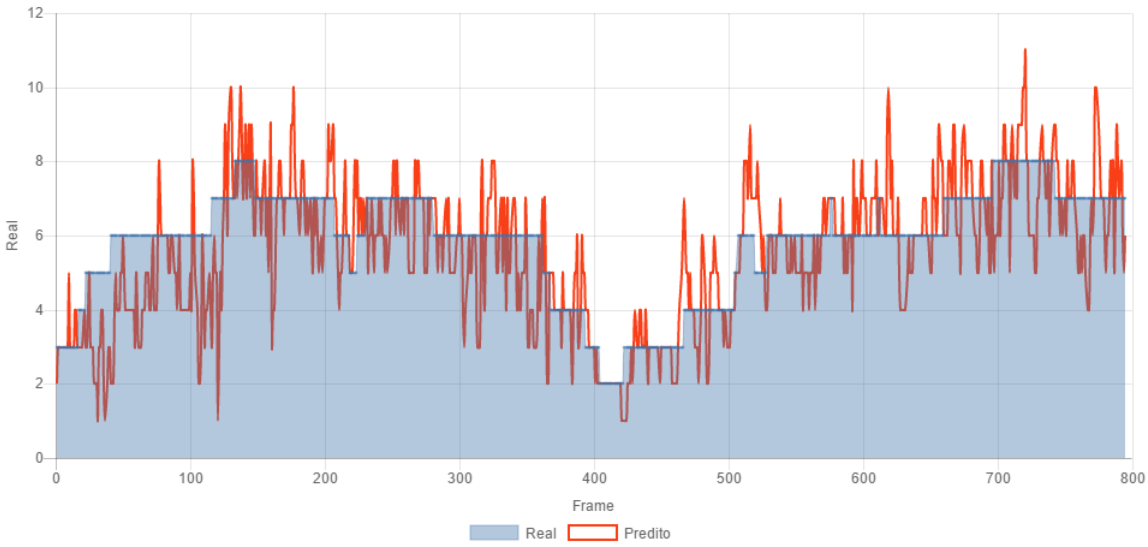
A decisão do melhor modelo entre os dois que se descaram não é fundamental com base no FPS ou no erro médio. Ambas os modelos também possuem uma alta similaridade entre mAP e erro percentual($17,3\%$ e $17,4\%$). Portanto, a escolha do melhor modelos foi com base no comportamento preditivo dessas arquiteturas. Na figura 39 observa-se a predição do modelo com arquitetura de otimização Lite. Em grande parte das vezes o modelo tem a característica de detectar uma menor quantidade de pessoas do valor verdadeiro dificultado assim melhorar o sinal. Isso caracteriza o modelo como sendo mais conservador para as detecções e isso pode ser interessante para uma aplicação específica.

Figura 39 – Em azul tem-se a quantidade de pessoas reais por frame. Em vermelho a predição do SSD Lite Mobilenet v2



Já na figura 40, observa-se o comportamento de predição do modelo com otimização PPN. Observa-se que a predição do modelo, apesar de possuir um erro devido a um tipo de ruído na medição, tem o seu valor em torno do real. Dessa forma, o modelo tem conseguido detectar grande parte das pessoas, entretanto, possui faltas durante alguns *frames*, deixando de detectar, detectando outras coisas como pessoas ou até mesmo detectando uma pessoa duas vezes.

Figura 40 – Em azul tem-se a quantidade de pessoas reais por frame. Em vermelho a predição do SSD Mobilenet v1 PPN



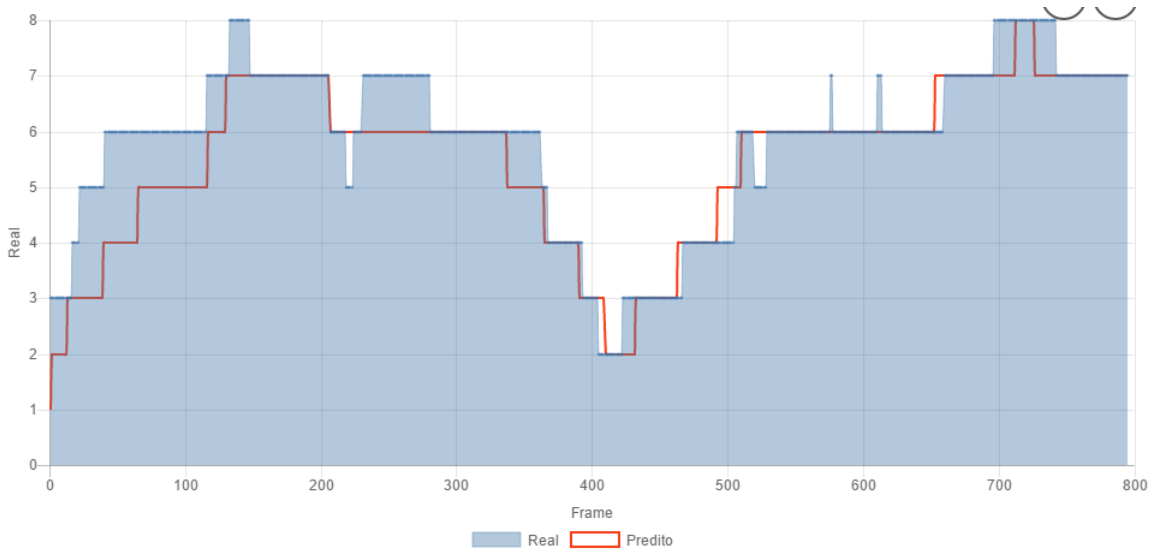
O comportamento do SSD Mobilenet v1 PPN em torno da quantidade real de predição torna esse modelo mais eficiente ao uso, já que uma utilização simples de um filtro pode melhor a qualidade de predição.

Aplicando o filtro butterworth passa baixa obtém-se uma resposta bem melhor, como descrito na figura 41. O erro do modelo cai de $17,3\%$ de erro percentual por frame, para $8,2\%$, ou seja, mais que dobrando a qualidade da predição efetuada. Além disso, a filtragem desse modelo, considerado um dos mais rápidos, gerou um erro percentual melhor que o Faster R-CNN, que teve 10% de erro.

Observando o comportamento do modelo ao longo dos *frames* é fácil observar que há um pior comportamento no início, onde nem mesmo o filtro conseguiu corrigir. Esse erro inicial, por ser grande, afeta a medida global de erro. Por isso, analisando a partir do frame 150, tem-se que o erro do modelo com PPN filtrado é de $4,9\%$, enquanto o do melhor modelo sem filtro, o YOLO v3, é de $5,5\%$.

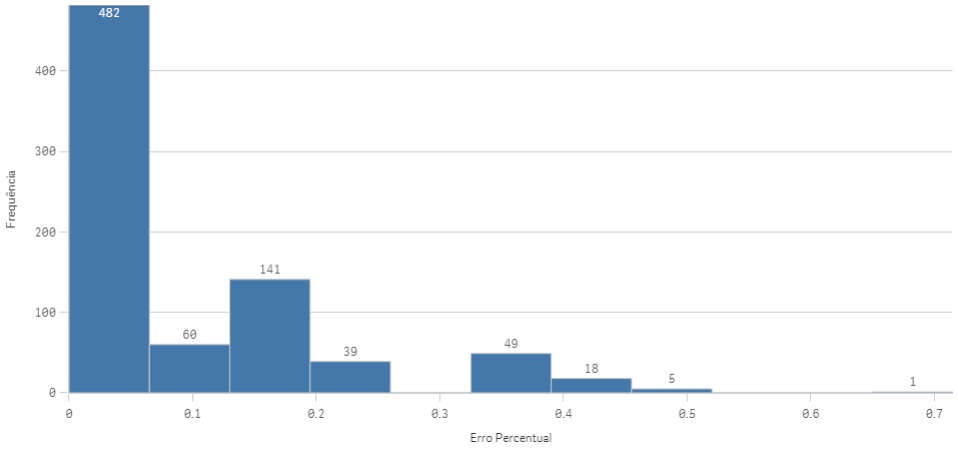
Outro ponto interessante de observar é o da figura 42. Nela tem-se o histograma da franquênia dos erros percentuais. Observa-se que cerca de 97% das vezes, o erro percentual não ultrapassa 39%. Esse valor é ainda menor quando analisado a partir do *frame* 150, tendo em 97% o valor de erro menor que 28%. Esse dado pode ser útil para ser usado como

Figura 41 – Em azul tem-se a quantidade de pessoas reais por frame. Em vermelho a filtragem do SSD Mobilenet v1 PPN



margem de segurança, por exemplo, em um sistema de contagem de pessoas em uma fila.

Figura 42 – Histograma de frequência dos erros percentuais



O ótimo comportamento do filtro se deve ao fato de que a frequência observada nos erros de medição são altas em relação a frequência de alteração real da quantidade de pessoas. Em um cenário como uma loja, a frequência de saída e entrada de pessoas tende a ser lenda comparando com a variação da predição dos modelos.

A decisão de o SSD Mobilenet v1 PPN ser considerado o modelo de destaque nesse trabalho tem como referência o menor erro de predição possível na quantidade de pessoas. Em uma situação onde a precisão ou o *recall* são mais importantes, essa decisão poderia ser diferente.

Na tabela 3 pode-se ver o resultado final de todos os modelos testados.

Tabela 3 – Tabela de resultados dos modelos testados

Modelo	mAP	Acertos	Erros	Total	FPS	Erro	Erro %
faster_rcnn_inception...	80,34%	4015	855	4870	0,99	0,59	10,02%
yolov3	78,62%	4017	777	4794	0,9	0,36	6,75%
ssd_mobilenet_v1_coco	53,56%	2847	622	3469	12,19	1,6	26,89%
ssd_mobilenet_v2_coco	61,20%	3240	574	3814	9,07	1,16	19,27%
ssd_inception_v2_coco	64,76%	3318	567	3885	5,69	1,14	19,18%
ssdlite_mobilenet_v2...	60,26%	3236	725	3961	12,41	1,04	17,39%
ssd_mobilenet_v1_0,75...	62,57%	3494	2418	5912	16,25	1,81	33,72%
ssd_mobilenet_docker	56,19%	2996	656	3652	10,66	1,34	22,12%
ssd_mobilenet_v1_ppn...	55,95%	3128	1263	4391	13,26	1	17,33%
yolov3_tiny	0,38%	297	4866	5163	7,14	0,98	18,80%

5.2 Dificuldades comuns dos modelos

Quantificar o resultado dos modelos em seus respectivos erros não é suficiente, já que esses podem ser implementados em diferentes ambientes e câmeras. Os erros dos modelos de detecção de objetos geralmente estão atrelados a comportamentos similares na imagem e que em sua grande maioria podem ser estudados e prevenidos. Em um cenário em que fosse interessante treinar esses modelos, tendo as principais dificuldades de detecção, pode-se treinar os modelos focando na resolução desses problemas, dando para o modelo um grande conjunto de treino nessas situações.

Figura 43 – Exemplos da diferentes dificuldades dos modelos em detectar pessoas

A figura 43 contém alguns exemplos práticos onde a arquitetura utilizada(SSD Mobilenet v1 PPN) teve dificuldade em prever uma determinada situação. Essas dificuldades geralmente estão presentes nos modelos não treinados para lidar com tais situações. O exemplo *A* demostra a dificuldade de detecção quando duas pessoas estão muito próximas uma da outra. Há uma tendência desse comportamento se acentuar dependendo da distância que as pessoas estão da câmera. No exemplo *C* duas pessoas estão muito próximas uma da outra no canto inferior esquerdo da imagem, entretanto elas são individualmente detectadas muito possivelmente devido a distância que estão em relação a câmera. Isso permite ao modelo ter um nível maior de detalhas da pessoa para efetuar a predição.

O exemplo *B* da figura 43 apresenta um exemplo diferente do apresentado na situação *A*. A falha de detecção de uma das pessoas no canto inferior esquerdo da imagem está relacionada a sua sobreposição em relação a outra pessoa detectada. Nessa situação o modelo não tem a característica de corpo inteiro da pessoa para considerar, tendo assim que conseguir classifica-lo apenas utilizando o dorso superior e a cabeça. Se o modelo não foi grandemente treinado a isso, ele terá dificuldade. Além disso, o elemento de redução das caixas delimitadoras preditas, o *Non-max Suppression*, pode ter eliminado a caixa que delimitava a pessoa de trás. É possível assim ajustar esse parâmetro se necessário.

Outra situação comum de acontecer é a apresentada no exemplo *C*. No centro da imagem, uma das pessoas não foi detectada muito possivelmente devido a estar atrás de um outro objeto. Semelhante a situação apresentada no exemplo *B*, o modelo tem dificuldade em obter as características da imagem, não sendo suficiente para classificar como pessoa.

No exemplo *D* tem-se uma situação onde uma pessoa é detectada duas vezes pelo modelo. Geralmente tal situação está relacionada com o algoritmo de *Non-max Suppression* não ter desconsiderado completamente todas as caixas delimitadoras redundantes. Essa situação é o oposto apresentado no caso do exemplo *B*. É por isso que o parâmetro de exclusão de caixas redundantes deve ser cuidadosamente estipulado.

Os exemplos apresentados na figura 43 mostram como o vídeo utilizado como base de análise dos modelos foi propositalmente desenvolvido para testar diversas situações onde geralmente os modelos possuem dificuldades de predição.

6 Conclusões

O objetivo geral do trabalho foi implementar um sistema de contagem de pessoas utilizando *deep learning* e câmeras de segurança. A ideia principal é que detectando pessoas utilizando as câmeras da loja, pode-se estimar fluxo de pessoas e calcular conversão da loja, tendo uma implementação fácil, barata e escalável.

O projeto contou com a implementação de uma arquitetura escalável e flexível para a detecção de pessoas. Dois pontos sendo como prioridade para a viabilização da implementação: o tempo de processamento e as qualidades de detecção. Diversos modelos de redes convolucionais de *deep learning* foram estudadas e implementadas, podendo comparar os modelos considerados atuais estado da arte. Ficou claro que a escolha de qual arquitetura utilizar deve ponderar entre velocidade e qualidade de predição, já que os modelos com maior qualidade tentem a ser mais pesados computacionalmente. Portanto, a viabilização do projeto dependeu da escolha de um modelo que consegui-se equilibrar os dois pontos.

O método de acurácia para comparação entre os modelos se mostrou efetivo e levantou dois modelos com alto custo benefício. Ambos com boa qualidade comparada com sua alta velocidade de processamento em CPUs. Com o pós processamento em um dos modelos selecionados, foi possível obter uma excelente qualidade de contagem de pessoas utilizando uma arquitetura que rodou a 13.3 FPS e com um erro médio percentual de $8,2\%$ pessoas por frame. Sendo que o melhor modelo em qualidade possui $6,7\%$ de erro médio percentual, mas com menos de um frame por segundo de processamento.

As análises desenvolvidas permitiram encontrar uma arquitetura rápida(SSD Mobilenet v1 PPN) que atendeu os requisitos de velocidade e foi boa o suficiente para atender grande parte dos requisitos de qualidade de detecção para uma implementação em uma loja. Devido a velocidade do modelo o custo para rodar os modelos desejados foi reduzido pois um bom servidor com GPU pode rodar paralelamente diversas câmeras.

A arquitetura desenvolvida foi pensada para suportar uma futura implementação de um módulo de tracking. Além disso, é estruturada utilizando docker, permitindo assim o rápido *deploy* para qualquer tipo de ambiente.

Os modelos sem treino mostraram dificuldade em algumas situações, sendo a de sobreposição de pessoas a possivelmente mais agravante. A contagem de fila pode ser bem prejudicada devido a dificuldade dos modelos em se comportarem bem com esse tipo de situação. Entretanto, em uma situação de implementação como essa, é possível treinar o modelo para se adequar com o requisito da sobreposição de pessoas, melhorando bastante os resultados finais.

A implementação de um modelo de *deep learning* em câmeras de segurança para a contagem de pessoas se mostrou viável em grande parte das situações. Concluí-se que uma das melhores alternativas é optar por um modelo mais rápido, como o SSD Mobilenet v1 PPN, e garantir o bom desenvolvimento de um filtro para pós processamento, que pode dobrar a qualidade do modelo tornando rápido e com alta acuracidade.

6.1 Próximos trabalhos

Varias premissas foram levantadas como metodologia base para o teste e escopo desse projeto. Para futuros trabalhos, diversas dessas premissas podem ser alteradas testando assim as implementações em diferentes cenários. Para futuros trabalhos é possível:

- Utilização de um conjunto vasto de vídeos e imagens de câmeras de segurança de uma loja. A geração do *ground truth* para essas imagens pode permitir o teste dos modelos em um cenário de implantação real, levantando assim as limitações e dificuldades dos modelos frente a um ambiente específico de implementação.
- Treinar os melhores modelos descritos nesse trabalho. Com a geração de um processo de treinamento é possível preparar as arquiteturas para ter um bom comportamento preditivo nas situações relatadas na sessão 5.2. Pode-se então reavaliar a performance dos modelos treinados aplicando em câmeras reais de lojas.
- Aplicar um modelo de *tracking* após a arquitetura de detecção de pessoas. Pode-se então avaliar e levantar a melhor combinação de modelo de *tracking* e detecção que contempla o melhor resultado preditivo. Com uma implementação desse tipo, diversas funcionalidades práticas passam a ser viáveis para implementação em uma loja, como identificação única de pessoas e análise do fluxo pela loja.
- Utilizando um modelo de alta acuracidade, pode-se criar uma arquitetura onde esse modelo gera o *ground truth* para o treinamento de uma arquitetura mais simples. A geração do *ground truth* não precisa ser em tempo real e pode acontecer em períodos de baixa demanda de processamento do modelo simples.

Referências

1 WALTON, M. *O método Deming de Administração*. 5nd ed. ed. [S.l.]: Marques Saraiva, 1989. Citado na página 15.

2 GANTZ, J.; REINSEL, D. *THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadow s, and Biggest Grow th in the Far East*. 2012. Acessado em 13/10/2018. Disponível em: <www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf>. Citado 2 vezes nas páginas 15 e 16.

3 RIOS, L. R. S. *Visão Computacional*. Acessado em 21/11/2018. Disponível em: <http://bit.do/visaocomputacional>. Citado na página 16.

4 TAVARES, J. M. R. S. *Aplicações da Visão Computacional em Biomedicina*. 2011. Acessado em 13/10/2018. Disponível em: <https://web.fe.up.pt/~tavares/downloads/publications/comunicacoes/Apresentacao_JT-IIIJorBioengUBI.pdf>. Citado na página 16.

5 LOTUFO, R. de A. *Visão Computacional com aplicações industrias*. 1996. Acessado em 13/10/2018. Disponível em: <http://www.dca.fee.unicamp.br/~lotufo/visao-ipt/index.htm>. Citado na página 16.

6 CONSUMO, S. B. de Varejo e. *O Papel do Varejo na Economia Brasileira*. 2018. Acessado em 13/10/2018. Disponível em: <http://sbvc.com.br/estudo-o-papel-do-varejo-na-economia-brasileira-atualizacao-2018/>. Citado na página 17.

7 NEOATLAS. *E-COMMERCE RADAR - GERAL CONSOLIDADO 2017*. 2017. Acessado em 13/10/2018. Disponível em: <http://materiais.neoatlas.com.br/ecommerce-radar-consolidado2017>. Citado na página 17.

8 VARON, F. S. L.; CAIN, M. *Retail eCommerce In Brazil: Marketing And Site Features*. 2017. Acessado em 13/10/2018. Disponível em: <www.forrester.com/report/Retail+eCommerce+In+Brazil+Marketing+And+Site+Features/-/E-RES135564#figure2>. Citado na página 17.

9 LUI, D.; ZAMBERLAN, L. *O VAREJO DE MODA EM ANÁLISE: Um Estudo do Comportamentos dos Consumidores com Relação às lojas de Vestuário de Santa Rosa*. 2015. Acessado em 21/11/2018. Disponível em: <http://bit.do/perfilconsumidor>. Citado na página 17.

10 DURR, S. M. B.; SYSTEMS, B. *What Is Workforce Management?* 2017. Acessado em 13/10/2018. Disponível em: <https://www.callcentrehelper.com/what-is-workforce-management-57249.htm>. Citado na página 18.

11 LECUN, Y. B. Y.; HINTON, G. *Deep learning*. 1. ed. [S.l.]: Nature, 2015. Citado 2 vezes nas páginas 23 e 38.

12 FACURE, M. *Introdução às Redes Neurais Artificiais*. 2017. Acessado em 21/11/2018. Disponível em: <https://matheusfacure.github.io/2017/03/05/ann-intro/>. Citado na página 23.

13 UDACITY. *Conheça 8 aplicações de deep learning no mercado*. 2018. Acessado em 21/11/2018. Disponível em: <https://br.udacity.com/blog/post/aplicacoes-deep-learning-mercado>. Citado na página 23.

14 EDWARDS, C. *Deep Learning Hunts for Signals Among the Noise*. 2018. Acessado em 14/10/2018. Disponível em: <https://cacm.acm.org/magazines/2018/6/228030-deep-learning-hunts-for-signals-among-the-noise/fulltext>. Citado na página 23.

15 OUAKNINE, A. *Review of Deep Learning Algorithms for Object Detection*. 2018. Acessado em 14/10/2018. Disponível em: <https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852>. Citado na página 24.

16 GHOSH PETER AMON, A. H. e. A. K. S. Reliable pedestrian detection using a deep neural network trained on pedestrian counts. *IEEE*, 2017. Citado na página 24.

17 NIELSEN, M. A. *Neural Networks and Deep Learning*. [S.l.]: Determination Pess, 2015. Citado 4 vezes nas páginas 25, 26, 27 e 28.

18 ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. *Cornell University Libary*, 2013. Citado na página 27.

19 KRIZHEVSKY, G. H. e. I. S. A. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012. Citado 2 vezes nas páginas 30 e 41.

20 SIMONYAN, S.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015. Citado 3 vezes nas páginas 30, 31 e 32.

21 SZEGEDY WEI LIU, Y. J. P. S. S. R. D. A. D. E. V. V. C.; RABINOVICH, A. Going deeper with convolutions. *Google*, 2015. Citado 2 vezes nas páginas 31 e 32.

22 SZEGEDY VINCENT VANHOUCKE, S. I. J. S. C.; WOJNA, Z. Rethinking the inception architecture for computer vision. *Cornell University Library*, 2015. Citado na página 32.

23 SZEGEDY VINCENT VANHOUCKE, S. I. A. A. C. Inception-v4, inception-resnet and the impact of residual connections on learning. *Cornell University Library*, 2016. Citado na página 32.

24 GIRSHICK JEFF DONAHUE, T. D. R.; MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation. Citado 2 vezes nas páginas 32 e 33.

25 UIJLINGS K.E.A. VAN DE SANDE, T. G. J.; SMEULDERS, A. Selective search for object recognition. Citado na página 33.

26 GIRSHICK, R. Fast r-cnn. Citado 2 vezes nas páginas 33 e 34.

27 REN KAIMING HE, R. G. S.; SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. Citado 2 vezes nas páginas 33 e 34.

28 REDMON SANTOSH DIVVALA, R. G. J.; FARHADI, A. You only look once: Unified, real-time object detection. Citado 2 vezes nas páginas 35 e 36.

29 Citado na página 36.

30 REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. Citado na página 36.

31 LIU DRAGOMIR ANGUELO, D. E. C. S. S. R. C.-Y. F. A. C. B. W. Ssd: Single shot multibox detector. Citado 2 vezes nas páginas 36 e 37.

32 SZEGEDY SCOTT REED, D. E. D. A. C.; IOFFE, S. Scalable high quality object detection. Citado na página 36.

33 HOWARD MENGLONG ZHU, B. C. D. K. W. W.-T. W. M. A. A. G.; ADAM, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. Citado na página 37.

34 ROSEBROCK, A. *Intersection over Union (IoU) for object detection*. 2016. Acessado em 22/11/2018. Disponível em: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>. Citado na página 39.

35 HUI, J. *mAP (mean Average Precision) for Object Detection*. 2015. Acessado em 27/10/2018. Disponível em: <https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173>. Citado 2 vezes nas páginas 39 e 40.

36 DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. *INRIA Rhone-Alps*, 2005. Citado na página 41.

37 STORE, H. P. C. *EAS HPC008*. 2018. Acessado em 22/11/2018. Disponível em: <https://pt.aliexpress.com/item/Highlight-EAS-HPC008-Camera-People-counter-Video-people-counter-Overhead-people-counter-High32457682599.html>. Citado 2 vezes nas páginas 41 e 42.

38 PACHI AIKATERINI;JI, T. Frequency and velocity of people walking. *The Structural Engineer*, 2005. Citado na página 43.

39 GOOGLE. *TensorFlow*. 2018. Acessado em 20/12/2018. Disponível em: <https://www.tensorflow.org/?hl=pt-br>. Citado na página 45.

40 BEAUCOURT, L. *Object-detection*. 2018. Acessado em 27/10/2018. Disponível em: <https://github.com/lbeaucourt/Object-detection>. Citado 2 vezes nas páginas 47 e 50.

41 DOCKER. *Get Started, Part 1: Orientation and setup*. 2018. Acessado em 22/11/2018. Disponível em: <https://docs.docker.com/get-started/#conclusion-of-part-one>. Citado na página 47.

42 GOOGLE. *Tensorflow detection model zoo*. 2018. Acessado em 27/10/2018. Disponível em: <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md#open-images-models>. Citado 2 vezes nas páginas 49 e 50.

43 NAYAK, S. *Deep Learning based Object Detection using YOLOv3 with OpenCV (Python / C++)*. 2018. Acessado em 02/11/2018. Disponível em: <https://github.com/spmallick/learnopencv/tree/master/ObjectDetection-YOLO>. Citado na página 52.

44 MOTCHALLENGE. *PETS09-S2L1*. 2015. Acessado em 27/10/2018. Disponível em: <https://motchallenge.net/vis/PETS09-S2L1>. Citado na página 53.

45 CARTUCHO, J. *mAP (mean Average Precision)*. 2018. Acessado em 02/11/2018. Disponível em: <https://github.com/Cartucho/mAP>. Citado na página 54.

46 WORNER, S. *Fast Fourier Transform*. 2012. Acessado em 22/11/2018. Disponível em: <http://pages.di.unipi.it/gemignani/woerner.pdf>. Citado na página 56.

47 WANG, R. *Butterworth filters*. 2018. Acessado em 22/11/2018. Disponível em: <http://fourier.eng.hmc.edu/e84/lectures/ActiveFilters/node6.html>. Citado na página 57.

Apêndices

APÊNDICE A – Exemplos de Códigos utilizados no desenvolvimento do projeto

No código A.1 há um exemplo da implementação da captura de frames de um vídeo capturado em tempo real por uma *Webcam*. Já no código A.2 é possivel ver a variação de parametros da função *VideoCapture* para se adaptar a fontes de vídeo diferentes, sendo elas straming, arquivo e um *device* de vídeo conectado no computador.

Listing A.1 – Exemplo da função VideoCapture para captura de vídeo de uma *Webcam*

```
import cv2

cap = cv2.VideoCapture(0) #Inicializacao da Captura da fonte de video
while(True):
    # Captura dos Frames
    ret, frame = cap.read()
    # Pre processamento transformando o frame em preto e branco
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Mostra o resultado do frame
    cv2.imshow('frame',gray)
    #Apertar em "q" para finalizar o programa
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Finaliza a captura dos frames e libera a fonte de video
cap.release()
cv2.destroyAllWindows()
```

Listing A.2 – Exemplo de diferentes fontes de captura de vídeo

```
import cv2

#Captura de video de uma webcam
cap1 = cv2.VideoCapture(0)

#Captura de video de um arquivo chamado video.mp4
cap2 = cv2.VideoCapture("video.mp4")

#Captura de video em streaming de uma camera de seguranca
cap3 = cv2.VideoCapture("http://ip-camera:porta")

...
```

O código A.3 representa a inicialização de uma instância de Tensorflow, carregando os pesos de uma rede pré treinada e disponibilizando essa instância para a predição de um

frame que receber como entrada.

Listing A.3 – Inicialização de uma rede neural utilizando Tensorflow

```
#Path do arquivo com os pesos do modelo.
PATH_TO_CKPT = 'model/frozen_inference_graph.pb'

# Carrega o modelo para memoria
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')
    #Instancia Tensorflow para prever objetos em uma imagem
    sess = tf.Session(graph=detection_graph)
```

O código A.4 mostra a implementação dessa inferência, onde o frame é representado pelo objeto *image_np*, que com a instância ativa *sess*, aplica a função *run* para obter as caixas preditas, as classes e as confiabilidades da predição.

Listing A.4 – Predição utilizando um modelo Tensorflow

```
# Inicializa alguns tensores
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
scores = detection_graph.get_tensor_by_name('detection_scores:0')
classes = detection_graph.get_tensor_by_name('detection_classes:0')
num_detections = detection_graph.get_tensor_by_name('num_detections:0')

# Apos inicializar as variaveis aplica a predicao na imagem
(boxes, scores, classes, num_detections) = sess.run(
    [boxes, scores, classes, num_detections],
    feed_dict={image_tensor: image_np})
```