UNIVERSIDADE FEDERAL DO ABC

UFABC

BACHARELADO EM ENGENHARIA DE INFORMAÇÃO

RELATÓRIO DO TRABALHO DE GRADUAÇÃO III

SISTEMA EMBARCADO PARA MONITORAMENTO INTELIGENTE DE PESSOAS

Alexandre de Sousa Santos

Rodrigo Hiroaki Ideyama

ORIENTADOR: Prof. Dr. Ricardo Suyama

COORIENTADOR: Prof. Dr. Murilo Bellezoni Loiola

São Paulo – SP

2021

Sumário

2. Lista de Figuras

2. Lista de Tabelas

3. Introdução

Segundo os indicadores da 2021 Global Peace Index, o Brasil está na 128º posição entre os 163 países mais seguros do mundo [29], ou seja, um ambiente de insegurança que necessita de atenção dos governantes. Essa insegurança é percebida por meio de diversos produtos e serviços que tentam intimidar a criminalidade e promover uma maior sensação de segurança.

Diante disso, o setor de segurança privada brasileira tornou-se um mercado bilionário, que somente em 2020, Somente em 2020, segundo a ABESE (Associação Brasileira das Empresas de Sistemas Eletrônicos de Segurança), foram R$ 8,10 bilhões de faturamento, 13% maior que em 2019 e as projeções para os próximos anos são de crescimento [30].

Diversos fatores contribuíram para esse mercado prosperar, desde a piora nos indicadores da Global Peace Index [29] até o surgimento de novas tecnologias que estão revolucionando o setor, como por exemplo: o reconhecimento de placas de veículos de forma automática, identificação por câmeras da utilização de máscaras pelas pessoas, bem como o monitoramento da temperatura corporal por meio de câmeras equipadas com sensores [40]. Além disso, com o avanço das tecnologias de rede 5G e *cloud*, as soluções tornaram-se cada vez mais integradas e conectadas entre si, fornecendo cada vez mais dados sobre os ambientes.

Nesse contexto, modelos de Inteligência Artificial (IA) têm se mostrado promissores, tendo em vista a sua capacidade de resolução de problemas complexos por meio da análise de dados estruturados e desestruturados, como imagens. Desta forma, cada vez menos pessoas serão necessárias para realizar tarefas repetitivas e monótonas, como o monitoramento integral de câmeras de segurança. Portanto, a tendência desse setor [40], e de outros, é utilizar das

tecnologias disponíveis, seja rede 5G, cloud e a IA para tornar as soluções de segurança mais assertivas e competitivas frente ao mercado dinâmico.

3.1. Objetivos

O objetivo geral do trabalho é elaborar uma solução utilizando modelo de inteligência artificial para detecção automática de pessoas por meio de imagens de câmeras de segurança e analisar aspectos relacionados à performance dessa solução a partir de imagens coletadas dos *datasets COCO* e *WiderPerson*.

Como objetivos específicos, podemos mencionar:

- Implementação do algoritmo *Single Shot Detection* (SSD) para a classificação e localização de pessoas, a fim de determinar a quantidade de pessoas no ambiente;
- Criação de alguns *datasets* para treinamento do algoritmo;
- Implementação do *software* em dispositivo embarcado (smartphone) e avaliação de seu desempenho

4. Evolução dos detectores de objetos

A detecção de objetos é uma das principais tarefas da visão computacional, na qual durante as últimas décadas esse campo de pesquisa avançou absurdamente e diferentes técnicas já foram desenvolvidas para tal finalidade. Mais recentemente, com o avanço das pesquisas de métodos de inteligência artificial, novos algoritmos baseados em redes neurais artificiais profundas têm se demonstrado bastante robustos e precisos para detecção de objetos em cena.

Quando tratamos de detecção de objetos, estamos interessados em encontrar objetos de uma certa classe (como pessoas, carros e animais) em uma imagem digital. Para isso, as técnicas extraem características (features) relevantes da imagem que permitam identificar e localizar o objeto corretamente na imagem. Na sequência apresentamos alguns dos principais métodos para detecção de objetos em imagens.

4.1. Detector Viola-Jones

A duas décadas atrás, P. Viola and M. Jones [49] apresentou um detector de faces humanas com precisão e necessitando de poucos recursos computacionais, que mais tarde ficou conhecido como VJ Detector em homenagem aos autores. Ele baseia-se em *sliding windows*, ou seja, percorre a imagem em busca de características que indiquem uma face humana. Nele temos três técnicas importantes que aumentaram muito sua performance: integral da imagem, seleção de características e classificadores em cascata.

1) Com a utilização da imagem integral, a complexidade computacional do algoritmo *VJ Detector* fica desassociado da dimensão da imagem,

desta forma, temos um incremento de desempenho, o filtro *Haar wavelet* é utilizado para representar as características da imagem.

2) Durante a seleção das características, o algoritmo *AdaBoost* seleciona uma pequena quantidade de características que serão úteis na detecção das faces, como pode ser verificada na figura 1 temos na figura superior os features selecionados pelo AdaBoost enquanto nas imagens inferiores temos os features sobrepostas a imagem da pessoa, mostrando um típico treinamento de face.
3) Os classificadores em cascata são utilizados visando a performance da solução. Nele todas as potenciais sub-janelas são analisadas e caso falhe durante uma etapa, essa sub-janela é descartada, logo apenas as sub-janelas que passarem por todos os classificadores são consideradas.

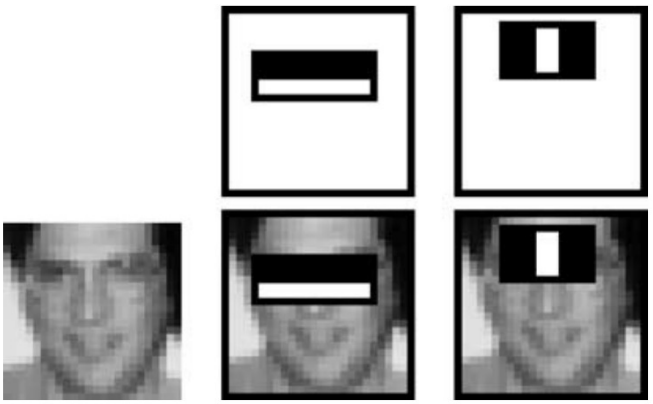

Figura 1: Representação do funcionamento do algoritmo adaboost [49]

4.2. Histograma de gradientes orientados (HOG)

Outro tipo de representação da imagem para detecção de objetos se baseia na informação do gradiente, que dá origem ao método conhecido como Histograma de gradientes orientados (HOG). Proposto por Dalal e Triggs [50], inicialmente o

método calcula os gradientes horizontais e verticais, que podem ser simplesmente encontrados filtrando a imagem original pelos kernels da figura 2:

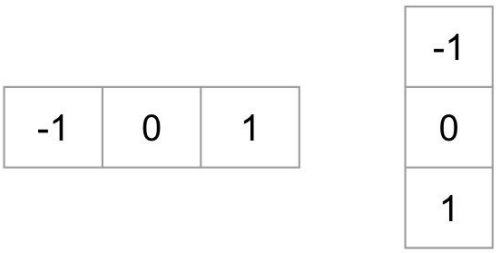

Figura 2: Kernel horizontal e vertical, *respectivamente,* do HOG *feature descriptors*

Ao calcular o histograma temos uma representação da imagem ainda compacta e menos sensível ao ruído. Então, a magnitude e a direção dos gradientes são simplesmente encontrados pela aplicação da Equação 1 e Equação 2, que são apenas conversões de coordenadas cartesianas para polares:

$$g = \sqrt{g_x^2 + g_y^2} \quad (1)$$

$$\theta = \arctan \frac{g_y}{g_x} \quad (2)$$

A ideia por trás do gradiente é indicar onde ocorrem mudanças bruscas de intensidade, na próxima etapa do algoritmo, precisamos encontrar o histograma com base nos valores dos gradientes. É necessário dividir a imagem original de dimensão *largura* x *altura* para células de tamanho 8 x 8, por exemplo, para que possamos calcular o histograma. Na esquerda da figura 3, temos uma imagem de um atleta que foi dividida em células de regiões 8 x 8, na imagem do meio temos a representação do gradiente de uma das células, indicado pelas setas, sendo a inclinação o ângulo e o comprimento sua magnitude. Por fim, na imagem à direita, temos as tabelas dos gradientes de magnitude e direção do trecho analisado.

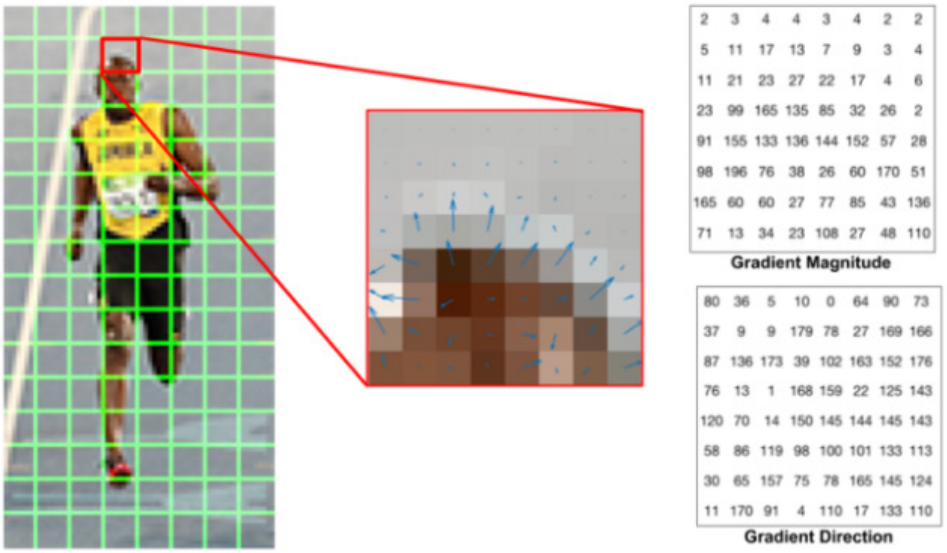

Figura 3: Representação do gradiente, ângulo e magnitude.

Ao invés de trabalhar diretamente com os valores do gradiente, o método utiliza a informação do histograma dos valores (magnitude e ângulo), produzindo assim um vetor com um número reduzido de componentes, indicando a frequência dos valores obtidos para o gradiente do patch analisado.

4.3. Deformable Part-based Model (DPM)

O próximo algoritmo analisado será *Deformable Part-based Model (DPM)*, ganhador da competição de detecção de VOC-07, -08 e -09, inicialmente proposto em 2008 P. Felzenszwalb, pensando em ser uma extensão do detector Datal-Triggs que por trás utiliza o HOG para obter uma representação de características da categoria do objeto.

Esse algoritmo baseia-se na abordagem *sliding window*, ou seja, o filtro é aplicado para todas as posições e escalas da imagem. Para realizar a detecção o DPM usa a ideia que todos os objetos são compostos por partes que possuem

locais relativos específicos, desta forma a inferência de uma determinada classe de objeto é realizada pela composição das partes do objeto, como ilustrado na figura 4. Além do filtro raiz (HOG) temos vários filtros parciais e em vez de especificar manualmente cada filtro, utilizamos um método de aprendizado fraco, desta forma podemos obter a configuração do filtro de forma automática. O autor R. Girshick, melhorou precisão de detecção com a introdução de algumas técnicas importantes:

- Hard negative mining, a meta dessa técnica é resolver o problema de dados desbalanceados durante o treinamento.
- Bounding box regression, nesta técnica buscamos obter melhores resultados para a localização do *bounding box* considerando como um problema de regressão linear.

Figura 4: Ilustração da detecção usando algoritmo DPM

Esse mesmo autor, desenvolveu uma técnica para incrementar a velocidade do algoritmo, chamada de "*compiling*" do modelo de detecção baseando-se em uma arquitetura em cascata. Desta forma, conseguiu aumentar a performance do modelo sem prejudicar sua precisão.

4.4. Redes neurais convolucionais (CNN)

No ano 2012, às redes neurais convolucionais ressurgiram em um momento onde as pesquisas na área não estavam tendo avanços significativos. E como as redes neurais convolucionais são extremamente robustas e de alto nível para extração de características de uma imagem, é plausível se questionar se seria uma boa opção para ser utilizada em detecção de objetos.

Quando tratamos de redes neurais convolucionais, elas se dividem em dois grupos: as de duas etapas e a de etapa única. Na CNN de duas etapas, a 1º etapa é composta por uma rede que propõe regiões de interesse, enquanto na 2º etapa são utilizadas essas regiões de interesse para realizar a detecção do objeto e a regressão do *bounding box*. Enquanto nas CNN's de uma única etapa, a detecção de objetos é tratado como um simples problema de regressão. Portanto, dada uma imagem de entrada, uma única rede fornecerá as probabilidades das classes assim com as coordenadas dos *bounding box*, em contrapartida as CNN's de duas etapas, essa abordagem apresenta ótimos resultados de performance. No Capítulo 3, as redes neurais serão abordadas em mais detalhes em relação ao seu funcionamento e estrutura.

4.4.1. R-CNN (Region-based CNN)

Em 2014, R. Girshick propôs a RCNN (Region-based CNN) para detecção de objetos, desde então a detecção de objetos começou a evoluir sem precedentes.

Foi a primeira CNN de duas etapas que apresentou destaque na detecção de objetos foi a R-CNN, o seu funcionamento é relativamente simples. Ela começa gerando as regiões de interesse (ROI) por meio da pesquisa seletiva na imagem de entrada, com isso ela indica ROI's onde existe a probabilidade de se encontrar um

objeto. Em seguida, cada região de interesse é redimensionada para uma dimensão padrão, de modo a alimentar o modelo CNN treinado no ImageNet para extração das características desses ROI's. Por fim, os classificadores SVMs são usados para predizer se existe um objeto na região de interesse e assim determinar qual classe ele pertence.

Apesar dos grandes avanços RCNN trouxe para detecção de objetos na época, ainda existia um ponto de atenção relacionada a sua performance, pois para cada ROI, em média 2.000 por imagem, era necessário passar pelas camadas de convolução o que tornava a detecção de objetos lenta.

4.4.2. Fast-RCNN (Fast Region-based CNN) [31]

O autor R. Girshick, no ano de 2015, propôs algumas melhorias na R-CNN, de forma a permitir que simultaneamente pudesse treinar o detector e o regressor de bounding box usando a mesma configuração de rede. Além disso, conseguiu um incremento de mAP de 58.5% para 70.0% além de aumentar a velocidade de detecção 200 vezes a mais que uma R-CNN.

Apesar das melhorias significativas de velocidade de detecção e acurácia, essa abordagem estava sendo limitada pelas regiões interesse, pelo menos em termos de performance de detecção.

4.4.3. Faster-RCNN (Faster Region-based CNN) [32]

Ainda em 2015, S. Ren propôs o detector Faster-RCNN brevemente após o lançamento do detector Fast RCNN. Nessa proposta, ainda de duas etapas, temos duas grandes evoluções, a primeira rede CNN end-to-end e primeira CNN a

fornecer uma detecção próxima ao real time. Isso só foi possível com a introdução da *Region Proposal Network* (RPN) que fornece regiões de interesse com um custo irrisório. Desta forma, temos uma única rede que contém todas as etapas de um RCNN, isto é, detecção de regiões de interesse, extração de características e regressão para obter os *bounding boxes*.

4.4.4. You Only Look Once - YOLO [34]

Em 2015, R. Joseph propôs o Yolo para detecção de objetos. Essa proposta foi a primeira CNN de uma única etapa, tendo como principal característica sua performance, sua versão mais rápida (YOLOv3-tyne) roda a 220fps com o COCO trainval mAP=33.1%, enquanto sua versão de melhor acurácia (YOLOv3-spp) roda a 20fps e mAP=60.6%.

O YOLO é uma abreviação para You Only Look Once, nota-se uma mudança de paradigma em relação às abordagens anteriores, pois nesta temos uma única rede neural para uma imagem completa, desta forma a imagem de entrada é dividida em regiões e avaliada por uma única rede que prever os bounding boxes e as probabilidades da classe para cada região de forma simultânea, desta forma consegue ser mais rápida ao realizar a detecção. Na figura 5 temos uma representação desse processo.

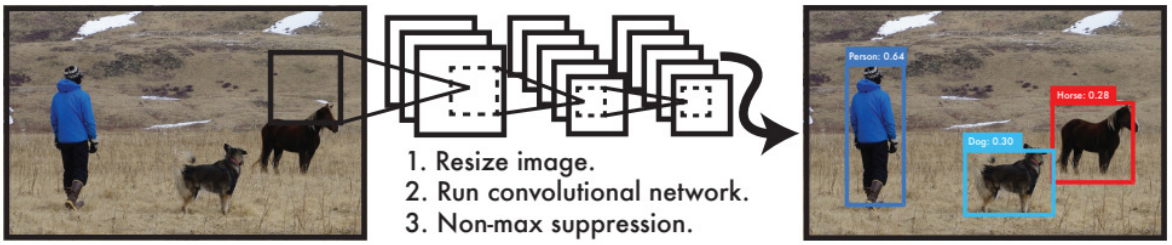

Figura 5: Sistema de detecção do YOLO [34]

Nessa abordagem, a imagem é dividida em S x S células e para cada célula temos a predição de B bounding boxes, além da confiança para esse boxes e as probabilidades das C classes, resultando em tensor de S x S x (B x 5 + C). Na figura 6 temos a representação desse processo, para determinar qual dos bounding boxes será utilizado como resposta final, ele combina os índices de confiança e a previsão da classe, que são calculados em conjunto, para gerar uma pontuação final que vai trazer a probabilidade da caixa delimitadora ter um tipo específico de objeto, então após todos os cálculos, a rede mantém as caixas que obtiveram a maior pontuação, como pode ser verificado à direita da figura 6.

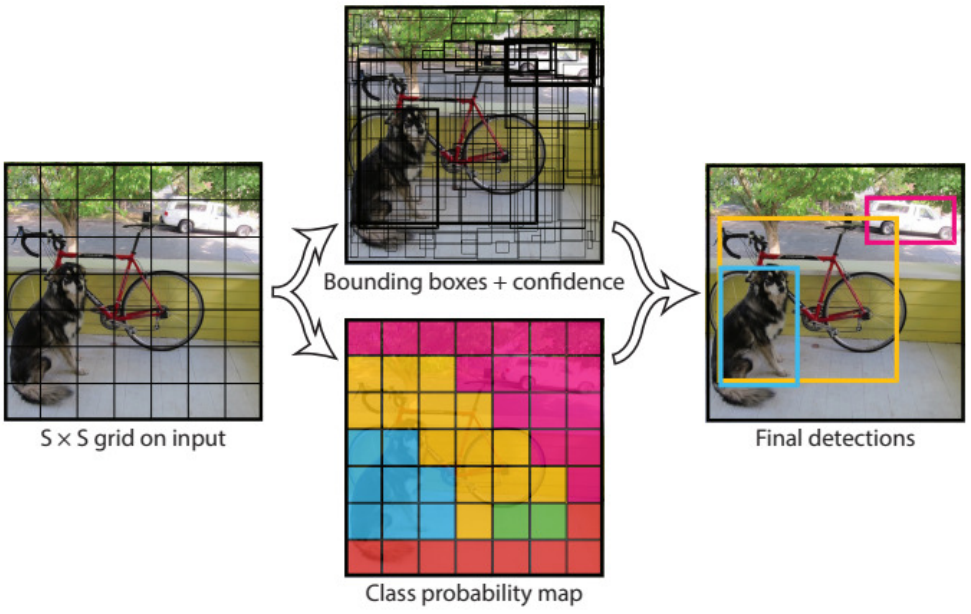

Figura 6 - O modelo de predição do YOLO

Vale ressaltar que após o lançamento, R. Joseph fez diversas melhorias e propôs as versões YOLO v2 e v3 que melhoram a acurácia da detecção e manteve alta a velocidade de detecção. Mesmo performático na detecção, ele ainda perdia em nível de acurácia, principalmente para objetos pequenos quando comparado aos métodos de duas etapas, em especial na sua primeira versão, algo que ajustado e melhorado nas versões subsequentes.

4.4.5. Single Shot MultiBox Detector (SSD)

Ainda em 2015, W. Liu propôs o Single Shot MultiBox Detector (SSD), sendo a segunda CNN de etapa única na era dos detectores usando *deep learning*. Essa abordagem é atualmente um dos modelos mais avançados para reconhecimento de objetos.

A mudança fundamental desse modelo está na melhora de velocidade que foi possível com a eliminação da proposta de *bounding boxes* e a introdução das técnicas *multi-reference* e *multi-resolution* que ajudaram a melhorar a acurácia nas detecções de objetos, principalmente de objetos menores (59 FPS e mAP 74.3% on VOC2007 test).

À medida que a imagem de entrada vai passando de camada para camada sua escala é reduzida, o que ajuda as *multi boxes* localizar objetos maiores que não eram identificados em imagens grandes quando utilizado redes sem essa característica. [22]

As primeiras camadas do SSD contém o *Feature Extractor*, geralmente, baseia-se no modelo pré-treinado do VGG no ResNet, rede neural só convolutiva, que como diferença, tem algumas camadas de convolução extras que realmente auxiliarão a lidar com objetos de diferentes escalas, em geral objetos maiores. Na figura 7 está representada a topologia da rede.

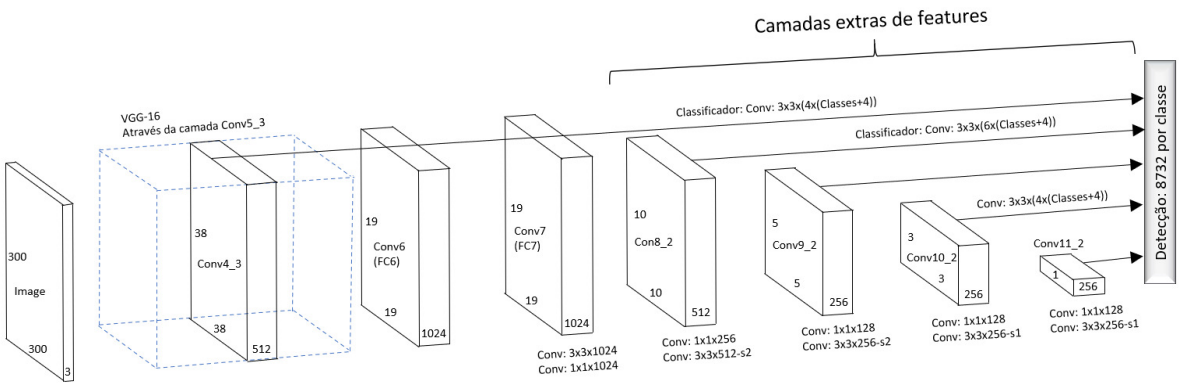

Figura 7: Representação da rede SSD

Além da arquitetura descrita no tópico acima, em 2017 pesquisadores do google publicaram uma nova arquitetura [25], conhecida como *MobileNet*, que posteriormente foi utilizada nas primeiras camadas (*Feature Extractor*) da arquitetura SSD. Entre os principais avanços desse modelo, destaca-se: desempenho e sua baixa latência, desta forma, possibilitando a aplicação de soluções de visão computacional em dispositivos embarcados, como por exemplo: telefones celulares, raspberry e outros.

Nesse novo modelo, foram introduzidos os hiperparâmetros α (width multiplier) e ρ (resolution multiplier) ambos definidos em (0, 1] permitem diminuir a complexidade da arquitetura MobileNet, descrita na Tabela 1. A ideia por trás do α é uniformizar cada camada da rede, dado uma camada da rede e um α, o número de “canais” de entrada M torna-se αM e o número de “canais” de saída N torna-se αN. Enquanto, ρ busca reduzir a resolução da imagem de entrada e consequentemente toda representação interna em cada camada é reduzida por ρ. Logo, esses hiperparâmetros reduzem o poder computacional e o tamanho da rede, tornando-a mais rápida, nota-se que ρ e α devem ser escolhidos de modo a não prejudicar significativamente a acurácia do modelo.

Tabela 1: corpo da arquitetura da MobileNet

Camada	Shape do filtro	Shape Entrada
Conv / s2	3 x 3 x 3 x 32	224 x 224 x 3
Conv dw / s1	3 x 3 x 32 dw	112 x 112 x 32
Conv / s1	1 x 1 x 32 x 64	112 x 112 x 32
Conv dw / s2	3 x 3 x 64 dw	112 x 112 x 64

Conv / s1	1 x 1 x 64 x 128	65 x 56 x 64
Conv dw / s1	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 128	56 x 56 x 128
Conv dw / s2	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 256	28 x 28 x 128
Conv dw / s1	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 256	28 x 28 x 256
Conv dw / s2	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 512	14 x 14 x 256
5 x Conv dw / s1 Conv / s1	3 x 3 x 512 dw 1 x 1 x 512 x 512	14 x 14 x 512 14 x 14 x 512
Conv dw / s2	3 x 3 x 512 dw	14 x 14 x 512
Conv / s1	1 x 1 x 512 x 1024	7 x 7 x 512
Conv dw / s2	3 x 3 x 1024 dw	7 x 7 x 1024
Conv / s1	1 x 1 x 1024 x 1024	7 x 7 x 1024
Avg Pool / s1	Pool 7 x 7	7 x 7 x 1024
FC / s1	1024 x 1000	1 x 1 x 1024
Softmax / s1	Classifier	1 x 1 x 1000

Essa abordagem foi selecionada para esse trabalho, pois permite uma maior precisão em objetos com diferentes escalas, devido a utilização de várias camadas de convolução de diferentes formatos, além da acurácia e performance em relação a outras abordagens.

5. Redes Neurais

Neste capítulo apresentamos a fundamentação teórica das redes neurais artificiais, incluindo a descrição das redes convolucionais, que formam a base do algoritmo de deteção estudado no trabalho.

5.1. Introdução e estrutura básica das redes neurais

Uma Rede Neural Artificial (RNA) é um modelo computacional que se inspira na estrutura neural e na forma como os organismos inteligentes processam informações, na literatura é comum relacionar o neurônio artificial ao neurônio biológico, como pode ser observado na figura 8.

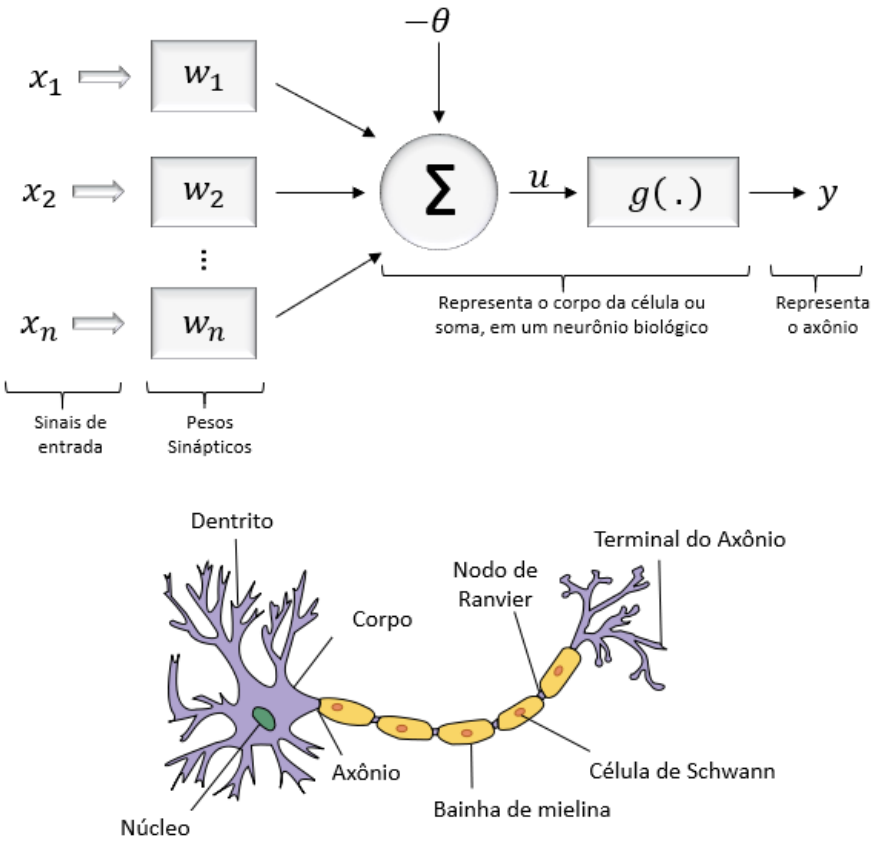

Figura 8*: Neurônio artificial acima e neurônio biológico abaixo*

Na figura 9, está representado em detalhes um neurônio artificial, que é composto por três elementos básicos:

- Conjunto de pesos associados às entradas (x_1,x_2, …, x_n);
- Um somador para acumulação dos n sinais de entrada;
- Uma função de ativação que pode limitar o valor de saída.

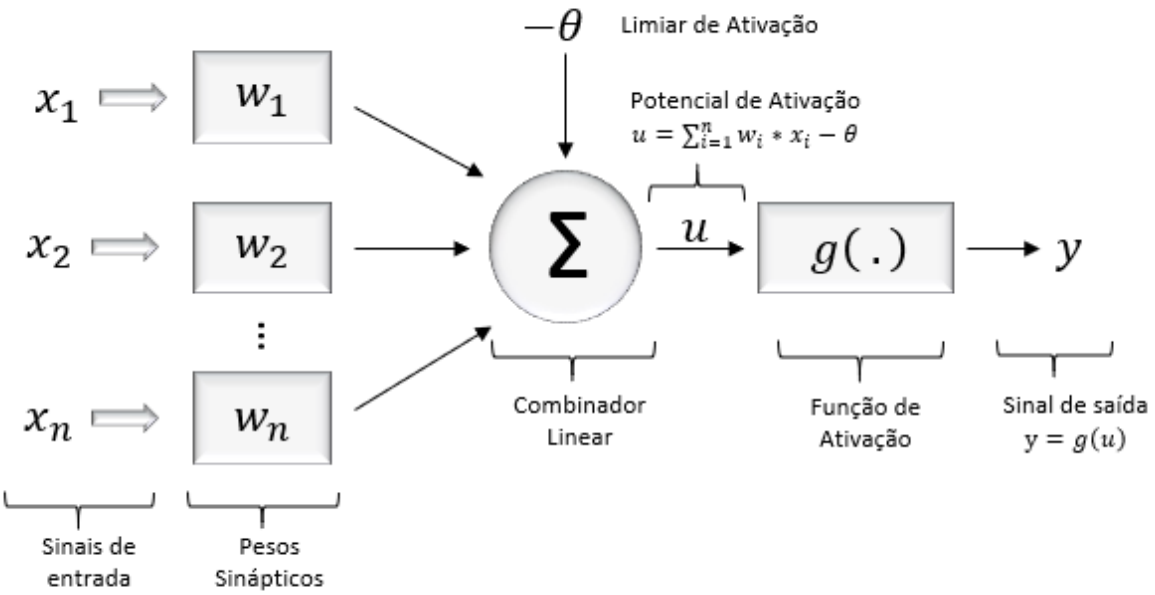

Figura 9 - *Representação de um neurônio artificial*

Matematicamente, a saída do neurônio é dada por :

$$y = g\left(\sum_{i=1}^{n} w_i * x_i - \theta\right) \quad (3)$$

O modelo apresentado corresponde ao Perceptron, que individualmente consegue realizar a classificação de padrões simples. Entretanto, em tarefas mais complexas, é necessário considerar uma estrutura mais elaborada, para que seja possível prover um mapeamento entrada-saída mais flexível. Para isso, pode-se considerar uma rede, composta de vários neurônios, conforme ilustra a Figura 10.

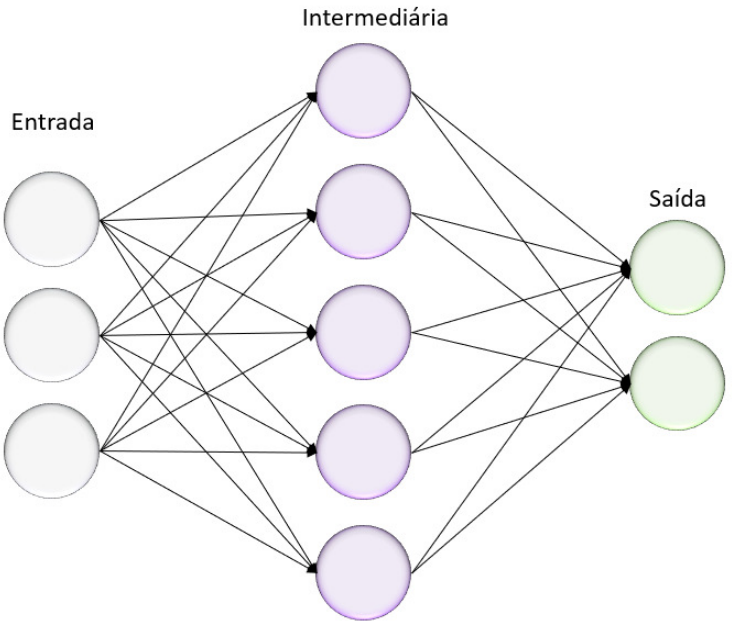

Figura 10 - *Representação da estrutura básica de uma rede neural*

> *As redes neurais artificiais se diferenciam pela sua arquitetura e pela forma como os pesos associados às conexões são ajustados durante o processo de aprendizado. A arquitetura de uma rede neural restringe o tipo de problema no qual a rede poderá ser utilizada, e é definida pelo número de camadas (camada única ou múltiplas camadas), pelo número de nós em cada camada, pelo tipo de conexão entre os nós (feedforward ou feedback) e pela sua topologia (HAYKIN, 2001, p. 46-49).*

5.2. Rede convolucional

Uma das primeiras publicações relacionada a redes neurais convolucionais surgiu em 2012, quando o Alex Krichevsky utilizou tais estruturas para ganhar a competição anual do ImageNet, conhecida como *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) que tem como objetivo avaliar algoritmos de

detecção de objeto e classificação de imagens, nessa competição ele conseguiu uma boa redução dos erros de classificação de 26% para 15%.

A partir disso, várias empresas começaram a oferecer serviços que utilizam esse tipo de rede neural - considerado um exemplo de rede neural profunda - como por exemplo, o Google para sua pesquisa de fotos, o Facebook usa para os algoritmos de marcação automática e a Amazon para sua recomendação de produtos. [22]

As redes neurais convolucionais são apropriadas para trabalharem com datasets que possuem características espaciais, como imagens. Elas baseiam-se em operações similares à convolução. No caso de sinais bidimensionais, f[x,y] e g[x,y], a convolução é definida como

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2] \quad (4)$$

Em essência, a operação de convolução é utilizada no processamento de imagens para aplicar filtros à imagem, podendo assim extrair características relevantes. Um exemplo desse tipo de operação é representado na figura 11, onde temos a matriz do filtro Sobel e a imagem a ser convoluída, utilizado quando deseja-se obter contornos de objetos.

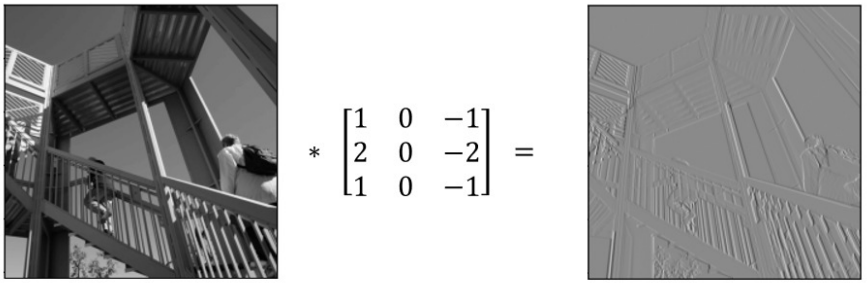

Figura 11 - *Exemplo de operação de convolução em uma imagem*

5.3. Estrutura da CNN

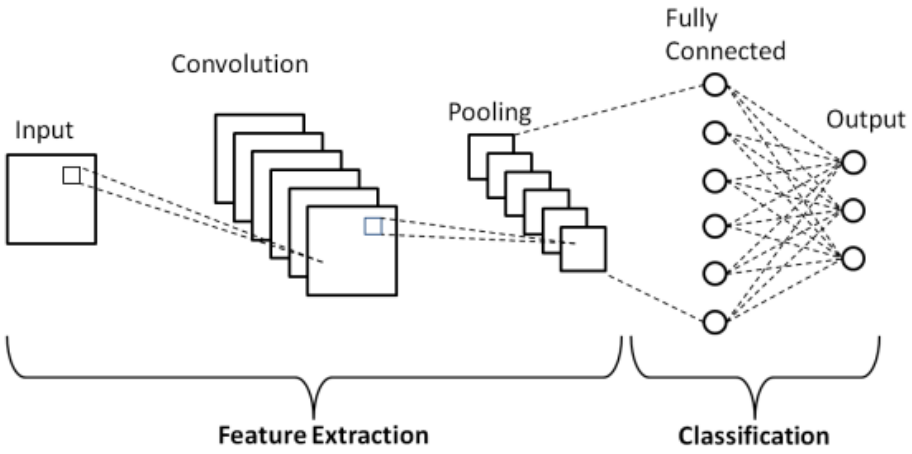

Figura 12 - *Estrutura base de de uma CNN*

A CNN apresenta uma estrutura base conforme ilustrado na figura 12. Nela é possível identificar os seguintes blocos funcionais:

- Camada de convolução: onde ocorre os movimentos dos filtros (stride) sobre toda a imagem que foi passada como entrada. Cabe destacar, entretanto, que embora o nome da estrutura receba o nome “convolucional”, a operação efetivamente realizada no caso da CNN é uma operação de correlação entre o *kernel* e a imagem. Matematicamente, a operação é definida por

$$f[x,y] * g[x,y] = \sum_{n_1=-a}^{a} \sum_{n_2=-b}^{b} f[n_1,n_2] \cdot g[x+n_1, y+n_2] \quad (5)$$

onde f[x,y] é o kernel do filtro, e “*a”* e “*b”* determinam o tamanho do kernel. A figura 12 abaixo ilustra um exemplo de operação realizada na camada de convolução, no qual o kernel do filtro é representado pela matriz [1 1; 0 0].

Figura 13 - *Exemplo de convolução*

Como pode observar na figura 13, na convolução faz-se os seguintes cálculos para cada uma das etapas:

1. 0*1 + 0,5*1 + 0*0 + 0*0 = 0,5
2. 0,5*1 + 0,5*1 + 0*0 + 0,5*0 = 1,0
3. 0*1 + 0*1 + 0*0 + 0*0 = 0
4. 0*1 + 0,5*1 + 0*0 + 0,5*0 = 0,5

- Função de ativação: uma escolha bastante comum para função de ativação no caso de CNNs é a função ReLu (unidade linear rectificada), que é uma função linear para valores de x>0 e zero para x<0, como mostra a figura 14:

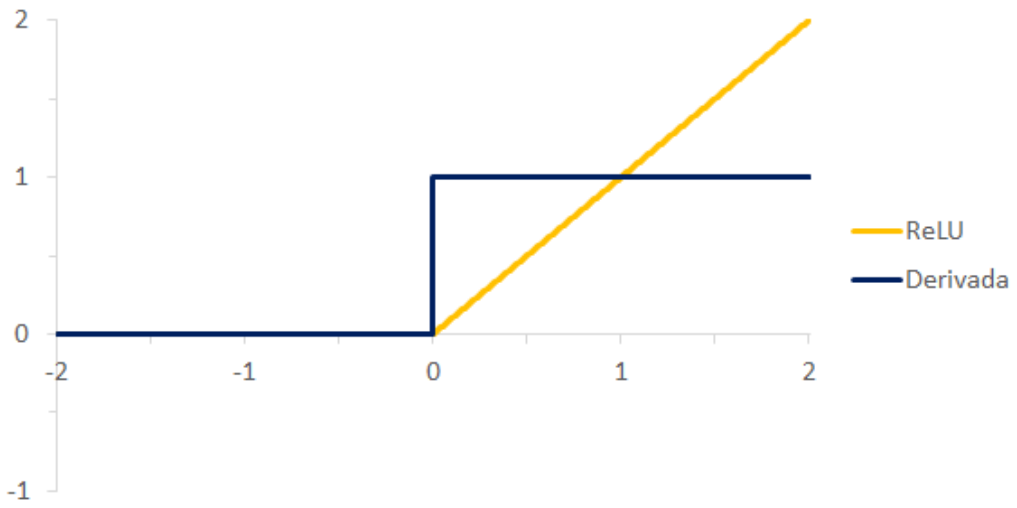

Figura 14 - *ReLu e sua derivada*

O comportamento dela nos garante que alguns neurônios não serão ativados quando a entrada for negativa e ativados quando a entrada for positiva. Ela possui uma derivada estável e grande para metade do domínio, o que ajuda na aprendizagem de uma rede neural com as atualizações dos pesos.

- Camada de Max-pooling: camada que é usada após a convolução para reduzir (downsampling) o mapa de ativação. Resumidamente, vai analisar em uma região quais são os valores maiores e descartando os menores, a fim de ficar com as partes mais relevantes. A Figura 15 apresenta um exemplo de operação considerando uma região de análise de tamanho 2x2.

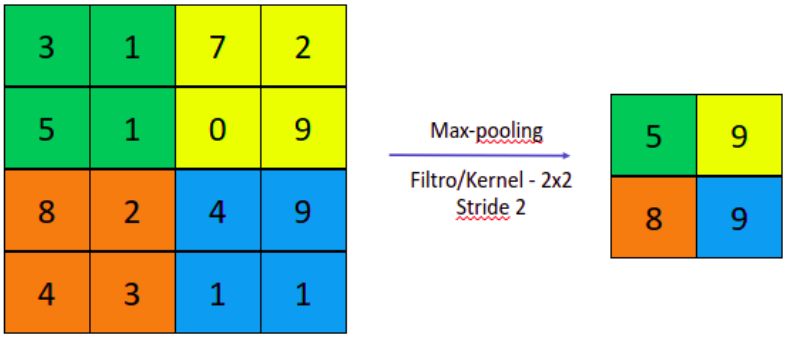

Figura 15 - *Max-pooling*

- Camada densa: também chamada de camada totalmente conectada, pois todos os neurônios estão conectados entre si. A camada analisa a relação entre os vetores de características retiradas das camadas anteriores com as classes dos objetos para conseguir realizar a classificação. Por exemplo, se nas camadas anteriores forem produzidos mapas de atributos correspondentes a nariz, perna e braço, é muito provável que a imagem corresponda a uma pessoa.

5.4. Treinamento da Rede

Para realizar uma tarefa específica é preciso ajustar adequadamente os parâmetros (pesos) da rede. Para isso, são necessários dois ingredientes básicos: um critério para o ajuste dos parâmetros, que é expresso pela minimização de uma função custo (ou loss function); e um algoritmo de otimização, que nos fornecerá a melhor configuração de pesos e bias que minimize a loss function, para que o modelo forneça um bom valor de acurácia.

- **Loss function**

Quando estamos lidando com modelos estatísticos, uma forma de examinar se estamos no caminho correto é por meio do seu desempenho, ou seja, quão assertivo está nosso modelo. Desta forma, faz-se necessário uma maneira de medir seu desempenho em valores reais, é aqui que as funções de perda entram em ação. Em resumo, as funções medem a distância entre o valor previsto e o valor real. Temos diversas funções de perda, no caso do modelo ssd, *classification loss* (*weighted sigmoid* e *weighted sigmoid focal*), *localization loss (weighted smooth L1*) e regularização, no tópico 6.2.3 - Losses functions, esses conceitos serão descritos em mais detalhes.

- **Algoritmo de Otimização**

De modo geral, é o método que realiza os ajustes dos pesos de uma rede neural com base em conjunto de dados de treinamento. A figura 16 ilustrada a ideia explorada em grande parte dos algoritmos para ajuste dos parâmetros da rede, que se baseiam no algoritmo backpropagation.

Essencialmente, a ideia pode ser descrita da seguinte forma: para um dado vetor de entrada, cada neurônio calcula a sua saída com base no valor de entrada, peso (W) e bias (B). Esse processo se repete para todo neurônio até que cheguemos na camada de saída, onde calculamos o valor da função de perda (comparando a saída atual e a saída desejada para a rede, definida no conjunto de treinamento). A partir desse ponto, deve-se obter a informação para o ajuste dos pesos de todos os neurônios, que é baseado na derivada da função custo em relação a cada parâmetro da rede.

O procedimento para obtenção das derivadas da função custo pode ser bastante otimizado considerando-se a estrutura em camadas da rede: a informação obtida com o erro na saída da rede é retropropagada pelas camadas da rede, obtendo de maneira recursiva o valor do gradiente da função custo e assim realizando o ajuste dos pesos. Repetimos esse processo até que os valores estimados estejam com um erro mínimo aceitável.

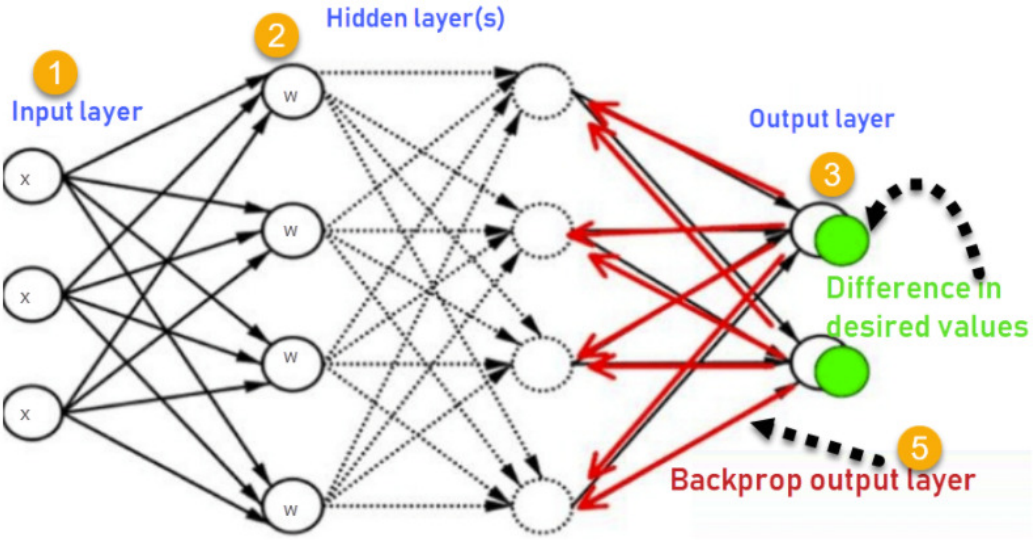

Figura 16 - Funcionamento do algoritmo *back propagation*

Há diferentes algoritmos que se baseiam no mesmo procedimento, uma opção de algoritmo usualmente adotada é o algoritmo SGD (Stochastic Gradient Descent). Nesse caso, para um conjunto de B dados de treinamento (*mini-batches*), o algoritmo realiza a adaptação dos parâmetros da rede (representados pela matriz W) de acordo com a seguinte equação:

$$W_{t+1} = W_t + \eta \sum_{j=1}^{B} \nabla L\left(W; x_j^B\right) \qquad (6)$$

onde η é o *learning rate*, ou seja, parâmetro que define o tamanho do passo que será dado na direção apontada pelo gradiente, e $\nabla L(W; x_j^B)$ representa a estimativa do gradiente para cada amostra considerada.

Uma variação do algoritmo, denominado de *Adaptive Moment Estimation* (ADAM) [47] considera o ajuste dos parâmetros com learning rates adaptativos para cada parâmetro. Essa abordagem leva, em geral, a uma convergência mais rápida do que o SGD.

5.5. Framework

O TensorFlow, ferramenta *open-source* desenvolvida pela Google, é considerado o que apresenta a melhor estrutura quando se trata de Deep Learning, devido a sua alta flexibilidade. Ele é o recomendado entre os três frameworks, em razão de diversas citações da comunidade Deep Learning, várias documentações disponíveis para o auxílio e é baseado em Python. A aplicação mais famosa que foi utilizado o TensorFlow é o Google Tradutor com o uso de técnicas como processamento de linguagem natural, reconhecimento de fala e outras. Além disso, está disponível para desktop e dispositivos móveis e também suporta linguagens como Python, C++ e R para a construção de modelos de Deep Learning. [9]

Para esse projeto foi utilizado a versão do Tensorflow 1.14 junto com as seguintes ferramentas para seu funcionamento na GPU:

- Python 3.7: linguagem de programação muito utilizada nos dias de hoje principalmente para área de ciência de dados
- CUDA (Compute Unified Device Architecture) Toolkit 10.0: extensão para a linguagem de programação C que permite o uso de computação paralela com o GPU [48]
- CuDNN 7.6.5: biblioteca da empresa NVidia que auxilia para processamento de redes neurais profundas em GPUs

Além disso, esse framework apresenta uma API que ajuda na realização de tarefa de reconhecimento de vários tipos de objetos chamada *TensorFlow Object Detection API* que disponibiliza módulos que ajudaram na realização do projeto como exportar o modelo para o grafo de inferência (utilizada para testar o modelo em vídeos) e para o formato *tflite* (usado para o modelo ser executado em dispositivos móveis). [46]

Por fim, ele ainda disponibiliza modelos pré-treinados no repositório TensorFlow Model Garden [1] para facilitar usuários a realizar treinamentos dos modelos conforme os problemas que desejam solucionar com eles, apenas configurando e adaptando conforme o necessário.

6. Detecção e Localização de Objetos em Imagens

Quando lidamos com modelos que realizam reconhecimento de objetos, é necessário definir 3 conceitos importantes: classificação, localização e detecção de objetos. No primeiro, a ideia é predizer qual tipo ou classe de objeto temos na imagem, a entrada do modelo será uma imagem e a saída um rótulo, ou mais, indicando as classes presentes na imagem. Enquanto na localização indicamos onde o objeto está localizado na imagem, isto é feito desenhando o *bounding box* em volta dele.

Quando falamos de detecção de objetos, estamos interessados em saber quais classes de objetos estão presentes na imagem, assim como sua localização. O problema, portanto, consiste em realizar a detecção de pessoas em um vídeo em *real time*, tanto quanto realizar esse processamento em um dispositivo de baixo poder computacional, neste caso um smartphone. Pretende-se também, analisar performance e acurácia variando alguns hiperparâmetros do modelo.

6.1. Pré processamento dos dados

A solução adotada neste trabalho baseia-na em uma rede neural convolucional para a detecção e classificação de objetos. Dessa forma, uma etapa importante quando lidamos com CNN é a elaboração de uma base de dados que posteriormente será utilizada para o treinamento da estrutura.

Para isso, utilizamos uma ferramenta gratuita, chamada LabelImg, para identificar os objetos e os seus respectivos *bounding boxes* na imagem, conforme apresentado a seguir.

6.1.1. LabelImg

A ferramenta, desenvolvida em Python, foi usada para a construção do primeiro dataset (200 imagens) do projeto. Através dele, pode-se indicar onde estão as pessoas e salvar as coordenadas de cada pessoa em um arquivo XML que, posteriormente, vai ser utilizado para produzir os *datasets*. Na figura 17 apresentamos uma das telas da ferramenta, onde é possível ver uma cena de jogadores em campo, e os respectivos *bounding boxes*. Cada *bounding box* deve ser criado pelo usuário por meio da ferramenta de desenho, e deve ser atribuído um rótulo a cada objeto destacado em cena. No caso, há 5 rótulos possíveis (dog, person, cat, tv, car), e é selecionado o rótulo *person* (pessoa).

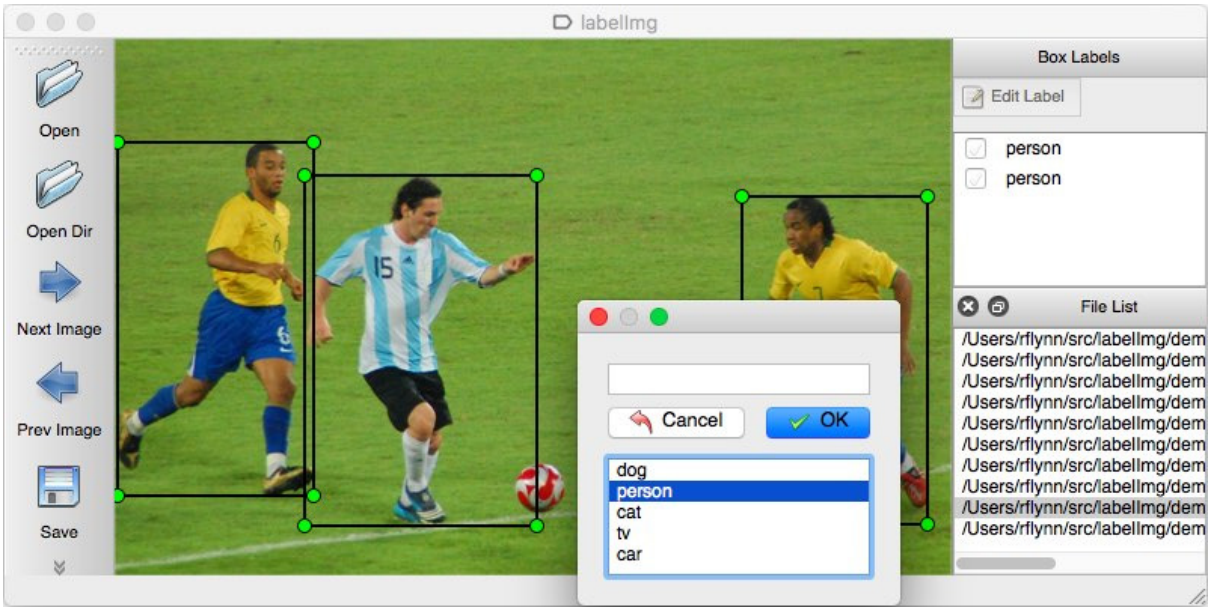

Figura 17: Ferramenta labelImg [26]

6.1.2. Pandas

Pandas é uma biblioteca *open source* desenvolvida em *Python* para análise e manipulação de dados. Entre seus grandes diferenciais, destacam-se: performance, flexibilidade e facilidade de uso.

No contexto do projeto, utilizamos o Pandas para organizar as informações que são necessárias para a geração dos datasets de treinamento da rede neural. A Figura 18 ilustra um *Dataframe*, que é a estrutura básica utilizada pelo Pandas para organizar as informações. No exemplo apresentado, cada linha contém a informação relativa a uma imagem: qual a classe do objeto a ser detectado na imagem (no caso uma pessoa), informações referentes à localização da pessoa na imagem (xmin, ymin, xmax e ymax), além da descrição do nome da imagem (filename) e suas dimensões (width e height).

	filename	width	height	class	xmin	ymin	xmax	ymax
0	000000000036.jpg	481	640	person	167	162	478	628
1	000000000086.jpg	512	640	person	152	183	281	559
2	000000000113.jpg	416	640	person	4	31	156	486
3	000000000113.jpg	416	640	person	341	100	416	428
4	000000000113.jpg	416	640	person	148	44	343	431

Figura 18: DataFrame contendo o layout do arquivo csv

Enquanto na figura 19 temos um arquivo json elaborado pelos criadores do dataset coco, por se tratar de um dataset multi classe, temos um nível (*categories*) para descrever todas as classes existentes e outro (*images*) para detalhar todas as imagens do dataset além das dimensões. Por fim, no último nível (*annotations*) está descrito todos os objetos e suas respectivas localizações na imagem.

```
{
    "info": {...},
    "licenses": [],
    "categories": [
        {
            "id": 0,
            "name": "airplane",
            "supercategory": null
        },
        ...
    ],
    "images": [
        {
            "id": 1,
            "file_name": "001631.jpg",
            "height": 612,
            "width": 612,
            "license": null,
            "coco_url": null
        },
        ...
    ],
    "annotations": [
        {
            "id": 1,
            "image_id": 1,
            "category_id": 9,
            "bbox": [
                92.14,
                220.04,
                519.86,
                61.89000000000001
            ],
            "area": 32174.135400000006,
            "iscrowd": 0
        },
        ...
    ]
}
```


Figura 19: Arquivo json contendo as anotações do dataset coco

A estrutura provida pelo Pandas é mais conveniente para a geração de arquivos que são utilizados no treinamento da rede - que foi implementada utilizando o framework Tensorflow (que foi descrito em mais detalhes no capítulo 5.5). Além disso, a utilização dos dataframes possibilita uma maior facilidade na manipulação dos dados, incluindo operações usualmente empregadas quando usamos bancos de dados.

6.2. Métricas de Avaliação

A avaliação dos métodos de deteção de objetos pode ser feita por meio de métricas adequadas. O método de detecção aplicado às imagens deve retornar se há ou não um objeto na imagem e a sua localização por meio de uma bounding box, conforme ilustrado na Figura 20. Dessa forma, diferentes métricas podem ser utilizadas para avaliar os resultados obtidos, dentre as quais utilizamos duas: a Intersection over Union (IoU) e a Mean Average Precision (mAP).

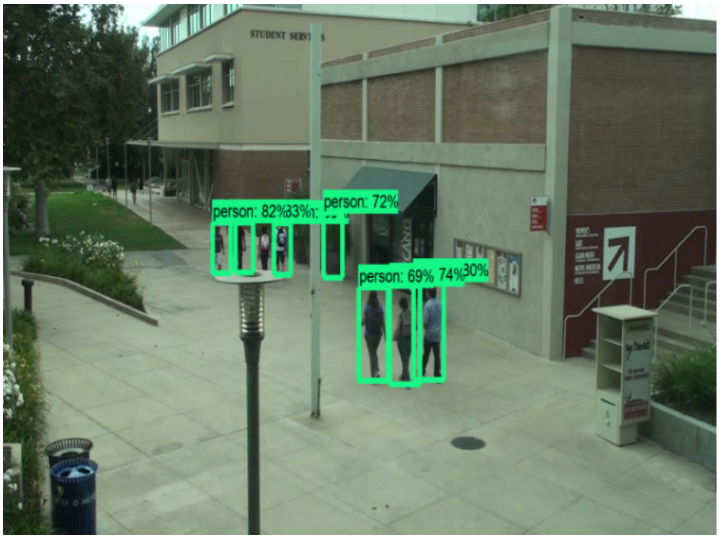

Figura 20: Ilustração da detecção e localização de uma classe

6.2.1. IoU

A métrica IoU (*Intersection of Union*) permite avaliar se a bounding box retornada pelo método de detecção está perfeitamente alinhada com a bounding box “ideal”, conforme definida pelos dados de treinamento. A ideia de métrica é ilustrada na figura 17, onde se observa que a *IoU* avalia a razão entre a área de sobreposição (numerador) e a área de união (denominador) com a *bounding box* predita e a *bounding box* rotulada. Um exemplo de cálculo é apresentado na Figura 18.

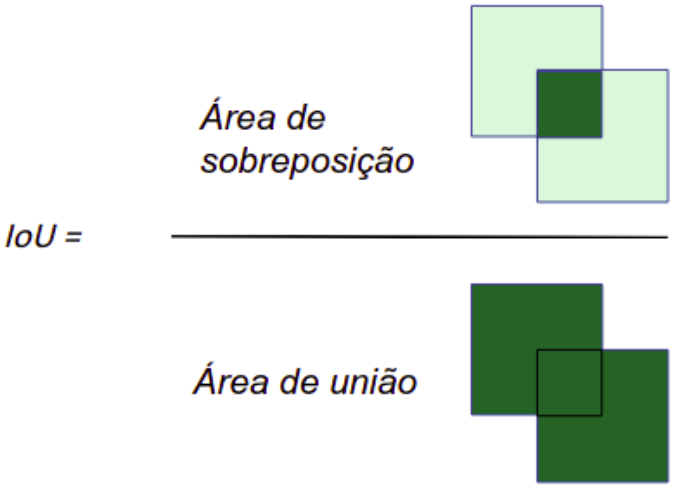

Figura 21: *Intersection of Union*

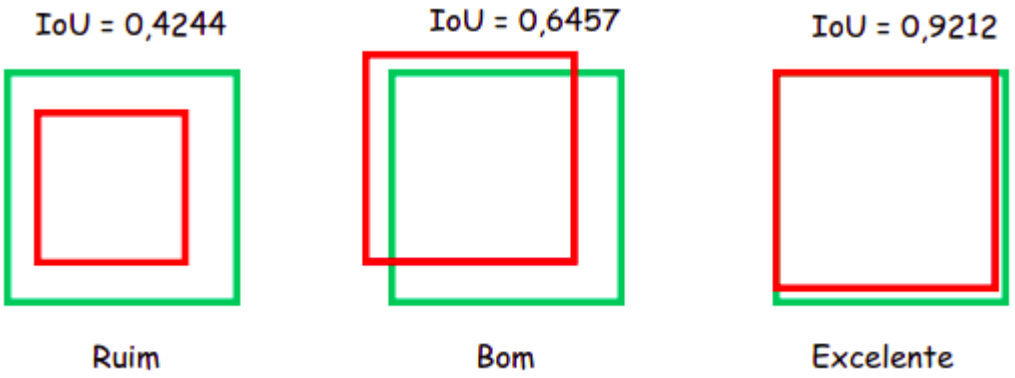

Figura 22: Exemplo de IoU

6.2.2. mAP

É interessante ressaltar que a IoU permite definir critérios objetivos para avaliar se ocorreu a detecção correta ou não do objeto. Por exemplo, se definirmos que uma *IoU* igual ou superior a 0,4 indica a detecção correta, então todas as três detecções estarão corretas, e correspondem ao que denominamos de resultados verdadeiros positivos (VP), i.e., o algoritmo detectou um objeto que realmente estava presente na imagem. Caso contrário, o resultado é um falso negativo (FN).

Se o algoritmo indica que há um objeto na imagem, mas na verdade não existe objeto algum, então o algoritmo apresentou um resultado falso positivo (FP).

De forma semelhante, se não há objeto na imagem e o algoritmo não aponta objeto algum, então temos um verdadeiro negativo (VN).

Os número de VP, VN, FP e FN permitem obter duas métricas importantes para avaliar classificadores, precision e recall, definidas como

$$\text{precision} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos} + \text{Falsos positivos}} \quad (7)$$

$$\text{recall} = \frac{\text{Verdadeiros positivos}}{Todososcasos(VP + FP + VN + FN)} \quad (8)$$

Observe que se variarmos o limiar do IoU para definir se a detecção foi correta ou não, teremos uma curva, denominada curva precision recall, conforme ilustrada na Figura XXX. Assim, a métrica mAP (*mean Average Precision*), muito utilizada para medir acurácia de detectores de objetos, pode ser calculada pela área abaixo da curva do gráfico precision-recall, i.e.:

$$AP = \int_0^1 p(r)dr \quad (9)$$

e quanto mais próximo de 1, melhor a classificação.

6.2.3. Losses Functions

Conforme mencionado anteriormente, as *loss functions* são importantes para o treinamento do sistema, uma vez que definem a função a ser minimizada com a otimização dos parâmetros da rede. Mesmo que não se trate de uma métrica de avaliação de desempenho propriamente dita, a avaliação dos valores das *loss functions* permite verificar se o processo de treinamento está sendo bem sucedido ou não.

A métrica classification loss, como o próprio nome indica, busca quantificar o desempenho da classificação [39], e é definida como:

$$
\begin{aligned}
&L_{conf}(x,c) = -\sum_{i\in+}^{N} x_{ij}^{p} \log_{10}\left(\hat{c}_{ij}^{p}\right) - \sum_{i\in-} \log_{10}\left(\hat{c}_{i}^{0}\right) \\
&\text{onde } \hat{c}_{i}^{p} = \frac{\exp(c_i^p)}{\Sigma_p}, x_{ij}^{p} = \{1,0\}, \\
&i = \text{ bounding box predita} \\
&j = \text{ bouding box rotulado} \\
&p = \text{ categoria e } \hat{c}_{i}^{p} \text{ a probabilidade de um objeto estar no bounding box i}
\end{aligned}
\tag{10}
$$

Os tipos de *classification loss* que foram usados nesse projeto foram:

- *Weighted Sigmoid:* também conhecido como *Sigmoid Cross Entropy Loss* ou *Binary Cross Entropy Loss*, pesa mais as predições erradas do que as predições corretas quando se trata de problemas de classificação binárias. Ela é a junção da função de ativação *Sigmoid* e *Cross Entropy Loss*, segue a figura com suas fórmulas [52]:

Figura 23: *Weighted Sigmoid*

$$f(s_i) = \frac{1}{1+e^{-s_i}} \tag{11}$$

$$\text{Cross Entropy Loss} = \sum_{i=1}^{C'=2} t_i \log(f(s_1)) \tag{12}$$

$$\text{Cross Entropy Loss} = -t_1 \log(f(s_1)) - (1-t_1)\log(1-f(s_1)) \tag{13}$$

onde s_1 é o *score* e o t_1 é a *groundtruth label* para a Classe C_1.

Agora s_2 é a classe *background* (aquela classe que representa a não classe que quer identificar)

- *Weighted Sigmoid Focal*: também é um tipo de *Cross Entropy Loss* que procura levar em conta a contribuição de cada amostra para a perda com base no erro de classificação. Sua fórmula é:

$$FL = -\sum_{i=1}^{C=2} (1 - s_i)^{\gamma} t_i \log(s_i) \quad (14)$$

onde $(1 - s_i)^{\gamma}$é o fator de modulação para reduzir a influência das amostras classificadas corretamente na perda quando o parâmetro de focagem $\gamma \geq 0$. Pode reparar que quando $\gamma = 0$, ele é igual a *Weighted Sigmoid*.

Já o localization loss, busca quantificar o erro na localização do objeto detectado, e baseia-se na métrica *Smooth L1*, definida da seguinte forma para dois valores x e y:

$$\begin{cases} 0.5(x-y)^2, \text{ se } |x-y| < 1 \\ |x-y| - 0.5, \text{ caso contrario} \end{cases} \quad (15)$$

Dessa forma, caso o erro absoluto entre x e y for menor que um, a smooth L1 usa o termo quadrático; caso contrário, usa o valor absoluto menos 0.5. Isto ajuda a evitar explosão de gradientes e se torna menos sensível a outliers. [38]

Dessa forma, a localization loss é formalmente definida como:

$$L_{loc(x,l,g)} = \sum_{i\varepsilon P_{os}}^{N} m\{cx,cy,w,h\} \sum_{ij}^{p} \text{smooth}_{L1}\left(l_i^m - \hat{g}_j^m\right)$$

Onde:

$$\hat{g}_j^{cx} = \frac{\left(g_j^{cx} - d_i^{cx}\right)}{d_i^w}$$

$$\hat{g}_j^{cy} = \frac{\left(g_j^{cy} - d_i^{cy}\right)}{d_i^h} \qquad (16)$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right)$$

$$\hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^n}\right)$$

onde *l* é o box predito e *g* é o box verdadeiro. Já o cx, cy, w e h representam as seguintes características do bounding box *d*: coordenadas x e y do centro, largura e altura.

7. Metodologia

As etapas utilizadas na elaboração dessa solução estão esquematizadas na figura 24. Nota-se que os resultados da etapa 4 serviram de subsídio para as etapas 2 e 3 na tentativa de buscar uma maior acurácia.

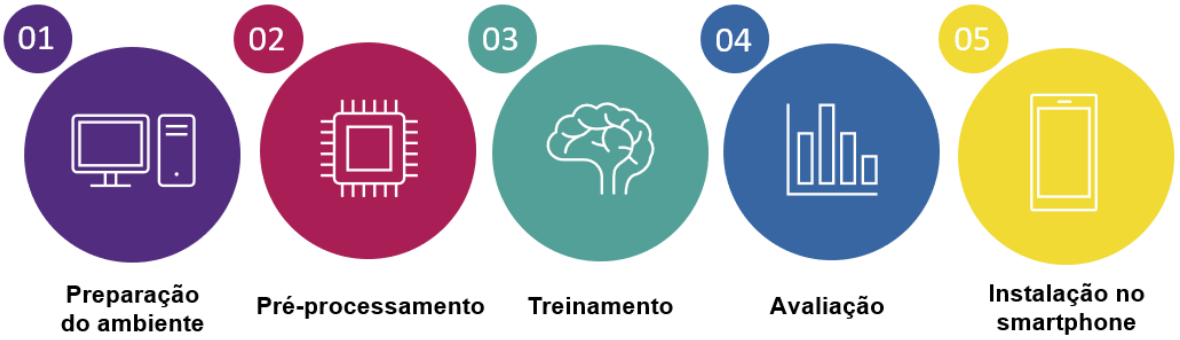

Figura 24: Etapas do processo

Na etapa 1 realizamos a configuração e instalação de todos os pacotes necessários para o projeto, enquanto no pré-processamento realizamos a

elaboração dos *datasets*. Na etapa 3, realizamos diversos treinamentos variando alguns hiperparâmetros do modelo. Na avaliação, coletamos dados de performance e desempenho do modelo e por fim, na última etapa, realizamos a instalação desse modelo no smartphone.

7.1. Preparação do ambiente

Na preparação do ambiente precisamos instalar e configurar os pacotes necessários para o desenvolvimento do solução:

- Anaconda 3 2019.03;
- virtualenv;
- Tensorflow 1.14;
- Tensorflow Object Detection API;
- Tensorflow Model Garden;
- Protobuf 3.14;
- COCO API;
- Object Detection API

Após ambiente configurado, criamos o sistema de pastas representado na figura 25, de maneira a facilitar a organização do projeto.

```
people_detection/
├─ annotations/
├─ exported-models/
├─ images/
│  ├─ test/
│  └─ train/
├─ models/
└─ pre-trained-models/
```

Figura 25: Sistema de pasta do projeto

7.2. Produção dos datasets

Com objetivo de avaliar a importância e o impacto dos datasets em modelos de machine learning utilizamos 4 tipos diferentes, variando quantidade de imagens e pessoas, diversos cenários e o tamanho das pessoas. Abaixo encontram-se em detalhes cada um dos *datasets* elaborados.

- 200 Imagens

  Coletamos 200 imagens do google imagens contendo pelo menos 70% de um pessoa em cenários diversos. Em seguida, para cada imagem rotulamos classe e posição de todas pessoas presentes na imagem, com o auxílio da ferramenta *LabelImg*.

Figura 26: Amostras do dataset de 200 imagens

- Dataset COCO V0

  O *dataset* COCO possui mais de 330 mil imagens e 1.5 milhão de objetos que atendem diversos propósitos, como: detecção de objetos, segmentação e criação de legendas. Como nosso objetivo é apenas identificação de pessoas, selecionamos entre todas as imagens apenas aquelas que temos objetos pessoas. Após essa etapa, o dataset resultante foi reduzido para 59.146 mil imagens.

Figura 27: Amostras do dataset COCO v0

- Dataset COCO V1

  Entre todas as imagens do dataset anterior, desprezamos objetos pessoas menores que 1% da resolução da imagem. Isso foi possível pois para cada objeto pessoa na imagem, as informações de localização estavam presentes no rótulo da imagem.

Figura 28: Amostras do dataset COCO v1

- Dataset COCO V2

  Para cada imagem do dataset anterior, realizamos uma inspeção manual para retirada de objetos contendo “pedaços de pessoas”, o que resultou em um banco com 34.940 imagens

Figura 29: Amostras do dataset COCO v2

- WiderPerson Dataset

  O *dataset* WiderPerson é uma referência para detecção de pedestres, suas imagens foram selecionadas de uma variedade de cenários, não se limitando a cenários de tráfego. Ele é composto por 13.382 imagens e possui 400 mil rótulos de objetos.

Figura 30: Amostras do dataset widerPerson

No dataset de 200 imagens usamos a proporção de 10% para validação e 90% para treinamento. No restante dos datasets, realizamos o particionamento na proporção de 4% para validação e o restante para treinamento, por conta do volume de imagens presentes neles.

Após essa etapa, criamos por meio do programa generate_tfrecord.py os arquivos *tfrecords*[1] de treinamento e validação com base nas informações de

[1] O *tfrecord* é um formato específico para armazenamento de uma sequência de registros binários, utilizado pelo TensorFlow.

posição e rótulo de cada objeto na imagem.

7.3. Treinamento

Quando falamos de modelos de inteligência artificial supervisionado, como é o caso da rede neural artificial desse projeto, uma etapa anterior importante a avaliação é a realização do seu treinamento, sendo por meio dele que o modelo “aprenderá” a generalizar para imagens com pessoas nunca vistas anteriormente, assim quando novas imagens com pessoas fossem utilizados no modelo ele seja capaz de detectar/localizar de forma correta. Logo realizar a configuração do modelo SSD *mobilenet* pré-treinado e criar estratégias para monitorar o seu desempenho e acurácia é tão importante. A seguir, detalhamos como foi realizado o ajuste no modelo padrão e como realizamos o monitoramento durante o treinamento.

7.3.1. Configuração

Inicialmente, realizamos o download do modelo pré-treinado SSD mobilenet [27]. Em seguida, realizamos a extração dos arquivos, sendo necessário ter realizado as etapas de preparação do ambiente e elaboração dos datasets descritas nos tópicos 7.1 e 7.2, e realizar a alteração das seguintes variáveis de configuração do modelo no arquivo `pipeline.config`

- num_classes: atribui o número de classes do modelo, no nosso caso, 1
- batch_size: quantidade de amostras que será alimentada na rede durante o treinamento.
- fine_tune_checkpoint: caminho do arquivo checkpoint do seu modelo

- fine_tune_checkpoint_type: colocar “detection” dado que nossa solução terá essa finalidade.
- use_bfloat16: usar false, pois não será treinado usando TPU
- label_map_path: caminho para o label_map.pbtxt
- input_path: caminho onde estão localizados os arquivos TFRecord

Após realizado os ajustes mencionados acima, executamos o programa python model_main.py para inicializar o treinamento. O hardware utilizado no treinamento foi um notebook com a seguinte configuração:

- I7 8ª geração, 12 CPUS
- 16gb de memória RAM
- Placa de vídeo GTX 1060, com 6GB de memória RAM e 1280 de NVIDIA cuda cores

É importante destacar que não é possível realizar o treinamento em uma máquina sem placa de vídeo e com menos de 8gb de memória RAM.

7.3.2. Monitoramento

Como mencionado acima, durante o treinamento realizamos o monitoramento de algumas métricas, em particular o mAP e as *losses functions*, por meio dos *dashboards* disponíveis no *tensorboard*. Nas figuras Figura 31 e Figura 31 temos os gráficos das métricas disponíveis para avaliação dentro da aplicação enquanto o modelo está sendo treinado.

Na figura 31 estão detalhadas as mAP. O gráfico *DetectionBoxes_Precision/mAP* representa o desempenho médio do sistema, enquanto os gráficos *DetectionBoxes_Precision/mAP (small, medium, large)* apresenta o desempenho considerando apenas tamanhos específicos de *bounding boxes*. Logo, para *small* consideramos *bounding boxes* de área menores que 32*32

pixels, enquanto para *medium* devemos ter área do *bounding boxes* entre 32*32 e 96*96 pixels. Por fim, l*arge* está associado a áreas maiores que 96*96 pixels.

O gráfico *DetectionBoxes_Precision/mAP@.50IOU* descreve o mAP considerando apenas o limiar para a IoU de 0.5, enquanto no gráfico *DetectionBoxes_Precision/mAP@.75IOU* usamos o limiar de detecção em IoU=0.75.

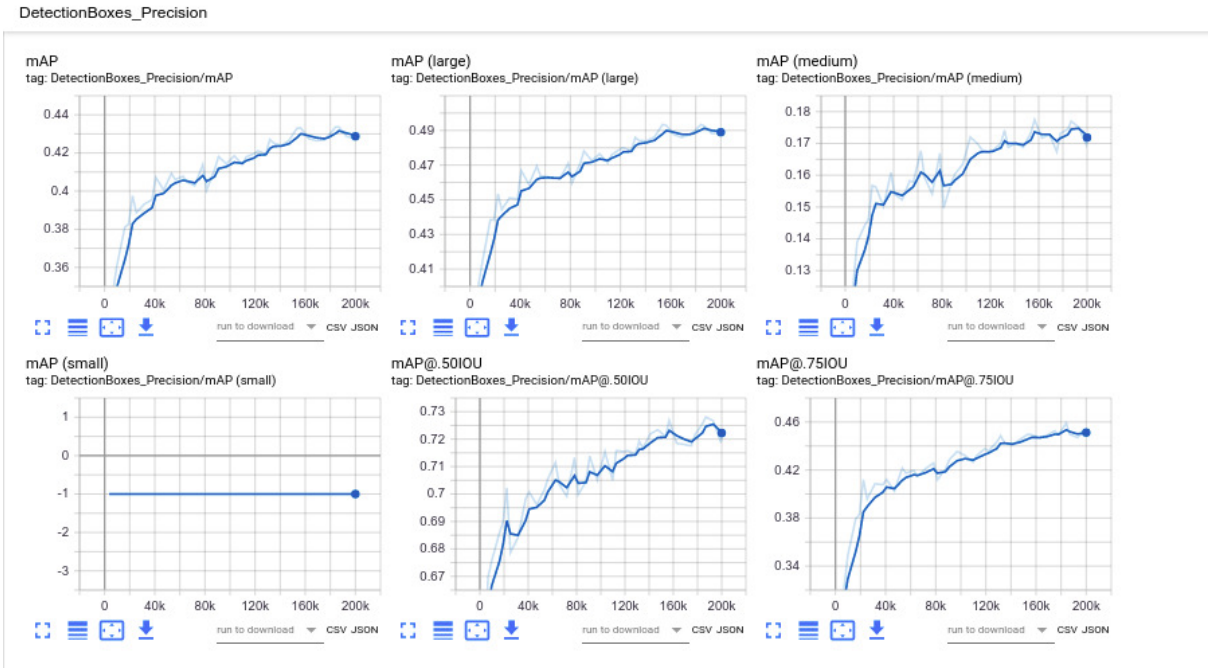

Figura 31: Métricas da precisão das caixas de detecção, medidos em mAP

Nos gráficos da figura 32 está representado todas as métricas relacionadas às perdas do nosso modelo (conforme definido no Capítulo 6.2). Como se tratam de métricas associadas à otimização do modelo, idealmente espera-se observar curvas decrescentes, que tendem a um valor baixo conforme o número de iterações do algoritmo de treinamento aumenta, conforme exemplificado na Figura 30.

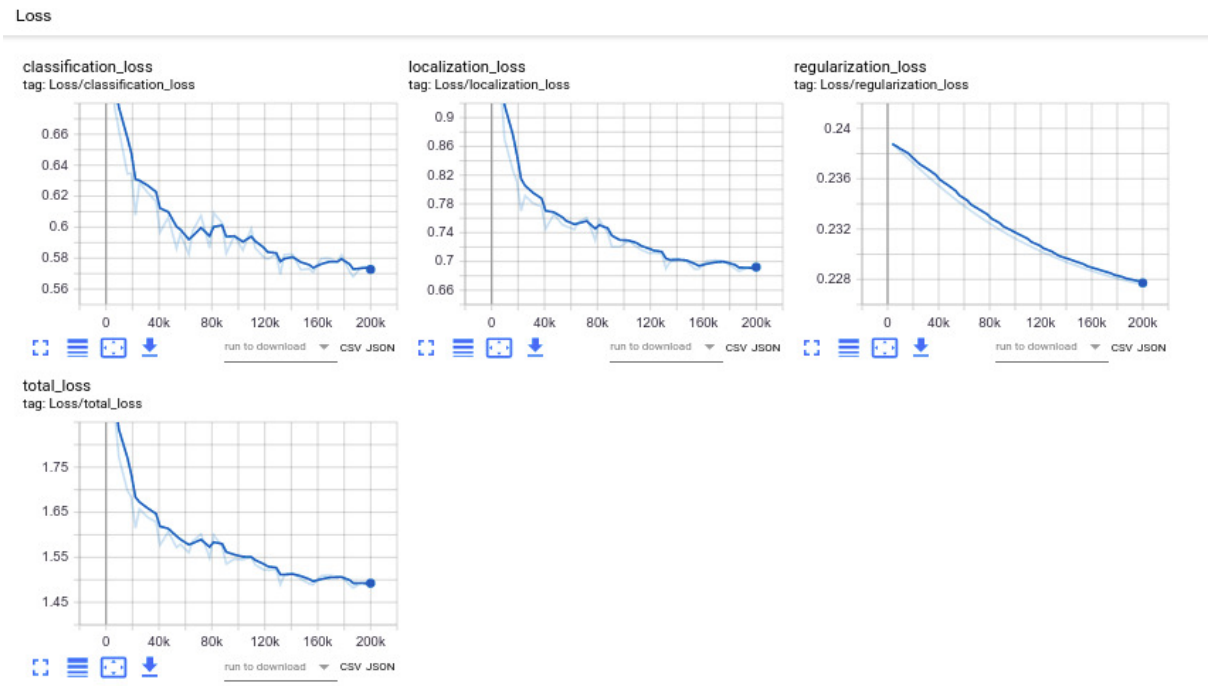

Figura 32: Métricas de avaliação, *loss*

Além de permitir a visualização das métricas facilmente, o *Tensorboard* permite fazer o download em formato csv de todos os dados que são utilizados no gráfico, o que ajuda a elaborar comparações entre configurações diferentes do modelo, alterando dataset e hiperparâmetros. Além disso, nele já é possível observar como será a detecção do modelo para uma imagem selecionada, como mostrado na figura 33.

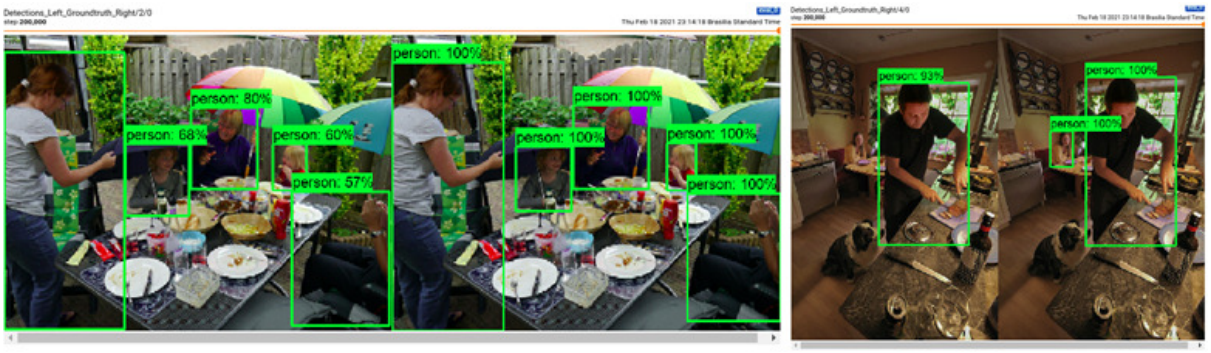

Figura 33: Exemplos de pessoas detectadas e a % de acurácia no tensorboard

Como pode ser observado na figura 33 da esquerda e direita, do lado esquerdo tem-se o que foi detectado pelo modelo e do lado direito é o que foi

rotulado no momento que foi construído o banco de imagens, isso para ambas as imagens.

Na figura da esquerda nota-se que o modelo conseguiu detectar todas as pessoas da imagem com uma boa acurácia, porém já na imagem da direita, a pessoa do fundo não foi corretamente detectada.

7.4. Avaliação do modelo

O treinamento da rede neural convolucional tratada nesse projeto necessita de extenso uso de recursos computacionais para realizar o treinamento. Por esse motivo, foi necessário adotar uma estratégia para encontrar a melhor configuração sem testar todas as combinações de configuração.

Inicialmente adotamos a configuração padrão para os seguintes parâmetros do modelo: batch_size=8, steps=200k, learning_rate=0.001, score_threshold=0.3 e iou_threshold=0.6 e variamos os seguintes parâmetros:

- freeze:
    - Quando foi colocado o valor de ".*FeatureExtractor.*", congelou todas as atualizações de pesos e bias na camada de *feature extractor* do modelo, caso contrário permitir a atualização)
    - Agora ao não colocar esse parâmetro, todas as camadas do modelo SSD podem ser atualizadas os seus pesos e bias.
- focal: explicado na seção 5.4
- dataset, podendo ser dataset COCO V0, dataset COCO V1 e 200 imagens

Todas as combinações possíveis entre esses 3 parâmetros foram realizadas, e após o término do treinamento de cada combinação de hiperparâmetros, armazenamos os valores finais de desempenho, ou seja, os valores de tempo de

treinamento, total_loss, mAP0.5 e mAP. Exportamos as informações de desempenho em formato csv do *tensorboard,* e selecionamos a configuração que apresentou melhor desempenho.

Em seguida, deixando fixo os parâmetros (batch_size, steps, learning_rate, freeze, focal e dataset), passamos para a avaliação de qual a melhor configuração dos hiperparâmetros score_threshold e iou_threshold. Após definir quais os melhores valores para ambos hiperparâmetros, começamos a avaliar os valores de batch_size e learning_rate. Em todas essas análises, anotamos os valores de desempenho (total_loss, mAP 0.5 e mAP) e tempo de treinamento.

7.5. Implementação no Smartphone

Uma vez escolhido qual dos treinamentos apresentou melhor desempenho com base na avaliação pelo *Tensorboard,* utilizou-se a ferramenta chamada Android Studio (Figura 32) e o template pronto do *Tensorflow Lite,* disponível na referência [23], para desenvolver a aplicação de detecção de pessoas no celular.

Nessa etapa fez-se uma pequena alteração na classe DetectorActivity e CameraActivity para a aplicação conseguir mostrar quantas pessoas está conseguindo detectar. Além disso, para cada modelo foi gerado um arquivo do modelo chamado *detect.tflite* e inserido na pasta *assets.*

Esse tipo de arquivo foi gerado, primeiramente, com o auxílio do script *export_tflite_ssd_graph.py* que realiza a conversão dos dados do modelo (*checkpoint* e *pipeline.config*) em *tflite.* Esse código tem como saída uma pasta chamada *tflite*, contendo *detect.tflite, tflite_graph.pb* e *tflite_graph.pbtxt*.

A figura 34 apresenta a tela do aplicativo implementado, onde é possível notar que o programa consegue identificar uma pessoa com um *bounding box*, e além disso traz as informações:

- *Frame*: Tamanho da imagem visualizada
- *Crop*: Imagem resultante do corte do frame para servir como entrada para o modelo
- *Inference Time*: Quanto tempo leva para detectar o número de pessoas
- *Number of people*: número de pessoas
- *Threads*: número de núcleos para realizar a inferência

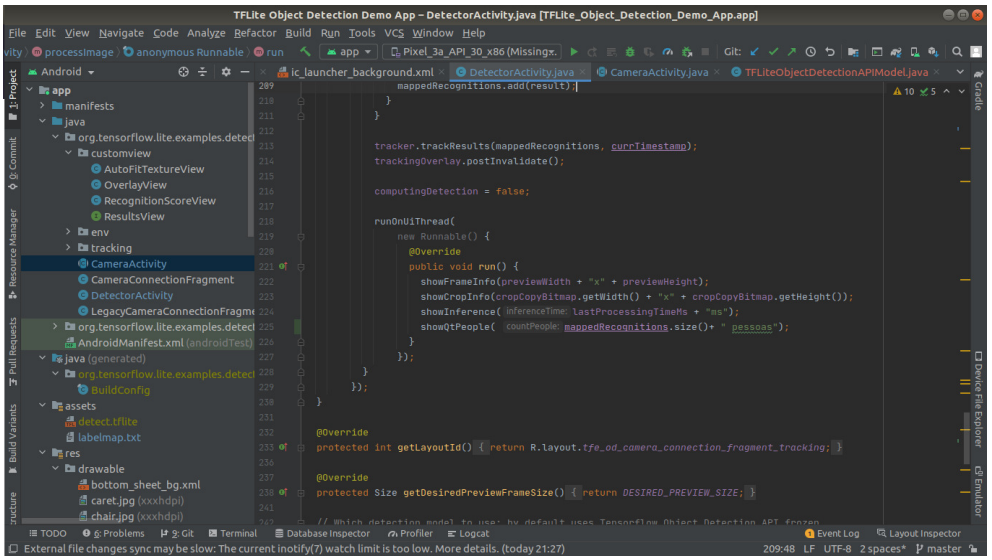

Figura 34: Tela do Android Studio

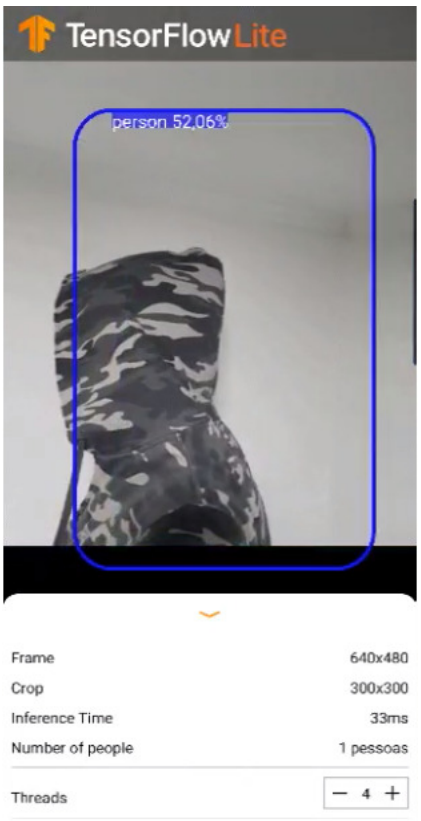

Figura 35 : App TensorFlow Lite com o modelo construído

8. Resultados e discussões

Conforme mencionado no Capítulo 7.4, realizamos diversos treinamentos variando alguns hiperparâmetros. Foram, aproximadamente, 290h de treinamento, com 24 diferentes configurações de hiperparâmetros. Uma alternativa, caso não tivéssemos a limitação de recursos computacionais, seria a aplicação da técnica *grid-search* para encontrar os melhores hiperparâmetros para o modelo. Desta forma, teríamos a melhor configuração que forneceria o melhor resultado de acurácia durante as detecções de pessoas na imagem.

8.1. Análise comparativa dos hiperparâmetros e datasets

Na tabela 2, temos descrito os hiperparâmetros fixos utilizados no treinamento e seus respectivos valores, os valores padrões do modelo foram utilizados, pelo menos inicialmente.

Tabela 2 - Parametrização fixa inicial

Batch size	Steps	learning_rate	score_threshold	iou_threshold
8	200k	0.001	0.3	0.6

Enquanto na tabela 3, temos os resultados do treinamento do modelo variando os hiperparâmetros freeze (se sim, congelar todas as atualizações de pesos e bias na camada de *feature extractor* do modelo, caso contrário permitir a atualização), e focal (se sim, utilizar a função de perda *weighted_sigmoid_focal*, caso contrário usar *weighted_sigmoid*) e variamos os *datasets* entre os dataset coco v0, dataset coco v1 e 200 imagens, e anotamos algumas métricas de performance do modelo além do tempo gasto no treinamento.

Tabela 3 - Tabela comparativa utilizando a parametrização da Tabela 2

Index	Freeze	Focal	Dataset	time	total_loss	mAP0.5	mAP
1	Sim	Sim	dataset coco v0	4h10min	2,799	0,3564	0,1706
2	Sim	Sim	dataset coco v1	3h22min	1,864	0,6435	0,3533
3	Sim	Sim	200 imagens	3h08min	2,891	0,3737	0,1588
4	Sim	Não	dataset coco v0	6h50min	6,164	0,2723	0,1443
5	Sim	Não	dataset coco v1	6h27min	5,075	0,5276	0,3028
6	Sim	Não	200 imagens	5h55min	7,465	0,3175	0,1378
7	Não	Sim	dataset coco v0	10h25min	2,417	0,4128	0,2071
8	Não	Sim	dataset coco v1	10h12min	1,687	0,6876	0.4038
9	Não	Sim	200 imagens	9h30min	2,843	0,4264	0,2137
10	Não	Não	dataset coco v0	14h15min	5,622	0,3291	0,184
11	Não	Não	dataset coco v1	13h10min	4,701	0,5919	0,3636
12	Não	Não	200 imagens	15h16min	5,624	0,3346	0,1834

Nota-se que o tempo de treinamento mais que dobra quando não utiliza-se o congelamento das camadas do *feature extractor*, o que é esperado dado a maior quantidade de pesos e bias a serem atualizados. Outro fator que contribui para o aumento do tempo do treinamento é a utilização de um dataset com mais imagens, porém notou-se que o dataset com 200 imagens e dataset coco v1 (39.386 imagens) apresentaram tempos semelhantes. No index 12, o tempo ficou maior, pois durante o treinamento ocorreu uma atualização do sistema operacional, desta forma os recursos de hardware competiram com o treinamento.

Nota-se pelas métricas de desempenho da tabela 3 que o dataset coco v1 apresentou os melhores resultados, principalmente quando utilizamos a função de perda *weighted_sigmoid_focal* e não congelamos as camadas do *feature extractor*. Contudo, nota-se que essa abordagem não estava completa, pois para aqueles datasets que utilizamos pessoas pequenas o gráfico de mAP *small* apresentou valores pequenos, diferente dos demais que apresentaram valores nulos (figura 39),

por conta disso os erros eram maiores naturalmente, devido a dificuldade de detectar pessoas pequenas, logo o mAP geral ficou penalizado, dando a falsa sensação que o melhor desempenho está naqueles que possui os melhores números na tabela 3.

Na tabela 4 utilizamos os melhores valores de desempenho apresentados na tabela 3 para compor os hiperparâmetros fixos, nessa etapa estávamos interessados em analisar o impacto no desempenho do modelo quando alterados os valores do *score_threshold* e *ou threshold.*

Tabela 4 - Parametrização com dataset coco v1

Batch size	Steps	learning_rate	Freeze	Focal	Dataset
8	200k	0.001	Não	Sim	dataset coco v1

Os valores usados para o *score_threshold* e *iou_threshold* foram 0.3 ou 0.6, desta forma, conseguimos 4 combinações de configuração, na tabela 5 seguimos as mesmas métricas de desempenho e avaliação do tempo de treinamento da tabela 3

Tabela 5 - Tabela comparativa utilizando a parametrização da Tabela 4

Index	score_threshold	iou_threshold	time	total_loss	mAP0.5	mAP
1	0.6	0.3	10h42min	1,669	0,5256	0,3278
2	0.6	0.6	9h29min	1,665	0,5228	0,3302
3	0.3	0.3	9h56min	1,657	0.663	0,3922
4	0.3	0.6	10h12min	1,687	0,6876	0.4038

Como resultados gerais, os tempos de treinamento ficaram semelhantes com variação de aproximadamente uma hora. Os valores de *total_loss* variaram pouco, dado que estamos utilizando o mesmo dataset, enquanto os valores de mAP sofreram uma variação de aproximadamente 24% quando abaixamos o valor do *score_threshold* de 0.6 para 0.3, pois estamos sendo menos criteriosos ao classificar um objeto pessoa na imagem, enquanto o valor de *iou_threshold* pouco interferiu no resultado do mAP.

A configuração 4 da tabela 5 apresentou o melhor resultado de mAP, por isso consideramos essa configuração de *score_threshold* e *iou_threshold* para analisar a variação dos próximos hiperparâmetros. Na tabela 6 temos os valores de configuração fixa para analisar as variações do *batch_size* e *learning_rate* ao treinar o modelo.

Tabela 6 - Parametrização fixa

Steps	Freeze	Focal	Dataset	score_threshold	iou_threshold
200k	Não	Sim	dataset coco v1	0.3	0.6

Tabela 7 - Tabela comparativa utilizando a parametrização da Tabela 6

Index	Batch_size	learning_rate	time	total_loss	mAP0.5	mAP
1	32	0,010	1d16h29min	2.134	0,6072	0,3531
2	32	0,001	1d18h30min	1,860	0,6393	0,3633
3	8	0,010	10h17min	1,658	0,6873	0,4118
4	8	0,001	10h12min	1,687	0,6876	0.4038

Observando a tabela 7, percebe-se que valores de batch_size 32 afetaram significativamente o tempo de treinamento do modelo, e em contrapartida não tivemos uma melhora nos indicadores de desempenho do modelo. Nesse ponto, criamos uma hipótese de que talvez pedaços de pessoas pudessem estar prejudicando o desempenho da solução. Por isso, elaboramos um dataset sem essas características, denominado dataset coco v2, e utilizamos os valores da etapa anterior, aumentado apenas o range de variação do *learning_rate*. Os parâmetros relativos a esse experimento são apresentados na Tabela 8.

Tabela 8 - Parametrização para o dataset coco v2

Steps	Freeze	Focal	Dataset	score_threshold	iou_threshold
200k	Não	Sim	dataset coco v2	0.3	0.6

Como esperado, tivemos uma melhora de quase 5% do valor de mAP comparado ao melhor resultado da tabela 7, conforme pode ser observado na Tabela 9. Porém, nos testes práticos percebemos que essa melhora acabou prejudicando o seu poder de generalização, principalmente quando as pessoas estão pequenas na imagem. Ainda assim, com relação ao tempo de treinamento, os valores permaneceram próximos.

Tabela 9 - Tabela comparativa utilizando a parametrização da Tabela 8

Index	Batch_size	learning_rate	time	total_loss	mAP0.5	mAP
1	8	0,1000	9h57m	2,312	0,4947	0,224
2	8	0,0100	10h24min	1,452	0,7216	0,4399
3	8	0,0010	10h32min	1,493	0,7215	0,4279

Por fim, decidimos testar o dataset wider person. Os hiperparâmetros utilizados durante o treinamento baseou-se na melhor configuração das etapas descritas anteriormente, ou seja, *batch_size*=8, *learning_rate*=0.01, *steps*=20.000, focal=sim, freeze=não. Os resultados obtidos foram os seguintes: mAP=0,301, mAP 0.5=0,606 e total loss=1,686 e tempo de treinamento 12h25min. Nota-se que os valores de mAP não foram superiores aos analisados anteriormente, em grande parte pelos valores baixos de mAP small que fizeram com que a média caísse, como pode ser verificado na figura 39.

A seguir, realizamos comparações dos indicadores de desempenho do modelo entre os principais datasets utilizados (dataset coco v2, dataset coco v1 e widerPerson dataset). Como mencionado no Capítulo 7.4, utilizamos os *csv* gerados

para realizar a comparação entre os modelos com os mesmos hiperparâmetros e diferentes datasets.

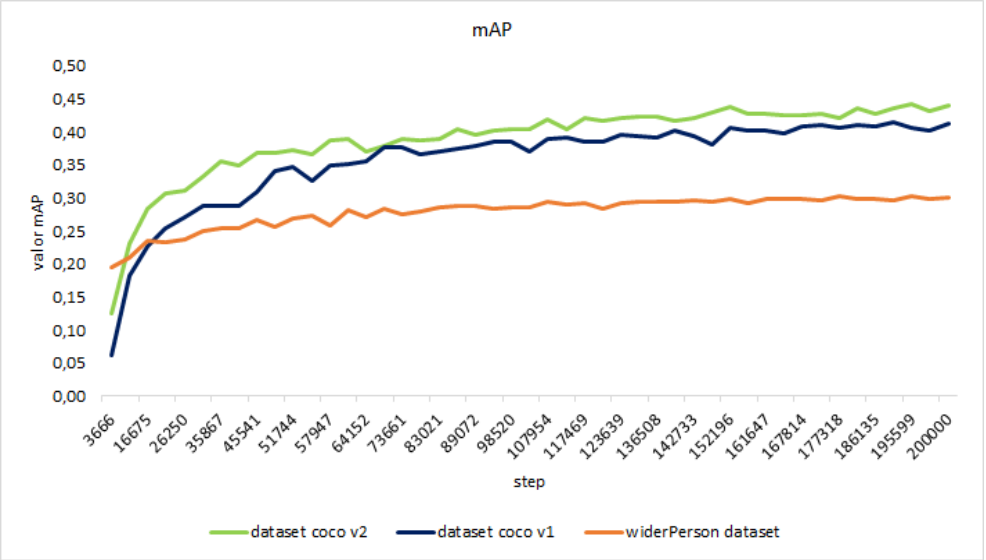

Figura 36 - Análise comparativa do mAP para 3 datasets

Na Figura 34 apresentamos a comparação do mAP para três datasets distintos. Independentemente da retirada de imagens que só representassem parte de uma pessoa (mão, pé), não houve uma diferença relevante entre os datasets *nice_people_coco* e *without_minion_coco.* Já para o *WiderPerson,* o único que contém um grande número de imagens pequenas e médias de pessoas, o valor médio do mAP acaba ficando abaixo dos demais datasets. Essa diferença de desempenho fica mais evidente ao considerarmos apenas o mAP para cada categoria de *bounding box* (*large*, *medium* e *small*), conforme apresentado nas figuras 35, 36 e 37. Observa-se que para o caso de *bounding boxes* grandes, o desempenho com os três datasets é muito semelhante, mas a diferença torna-se bastante acentuada ao avaliar apenas o desempenho para *bounding boxes* médios e pequenos - já que apenas o dataset *widerPerson* possui dados desse tipo para treinamento.

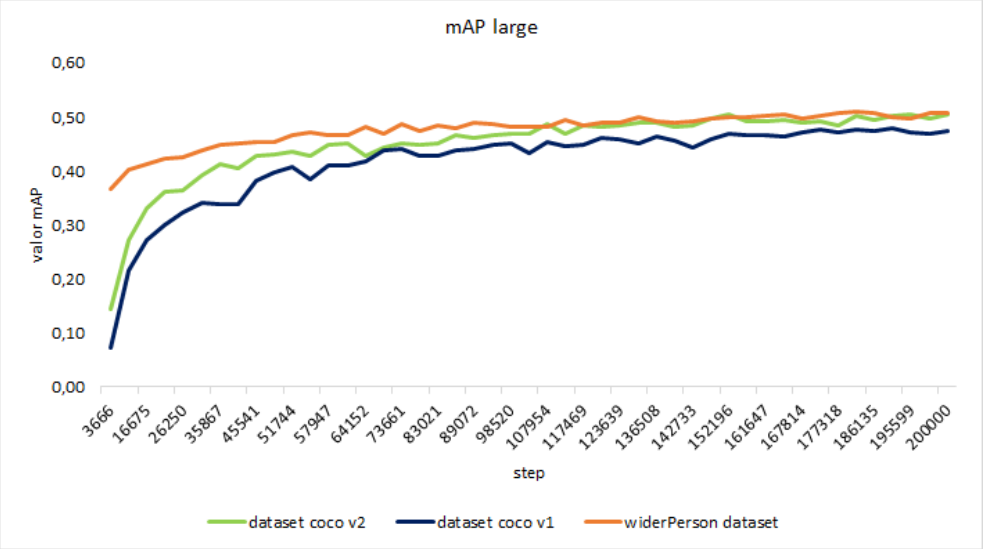

Figura 37 - Análise comparativa do mAP large para 3 datasets

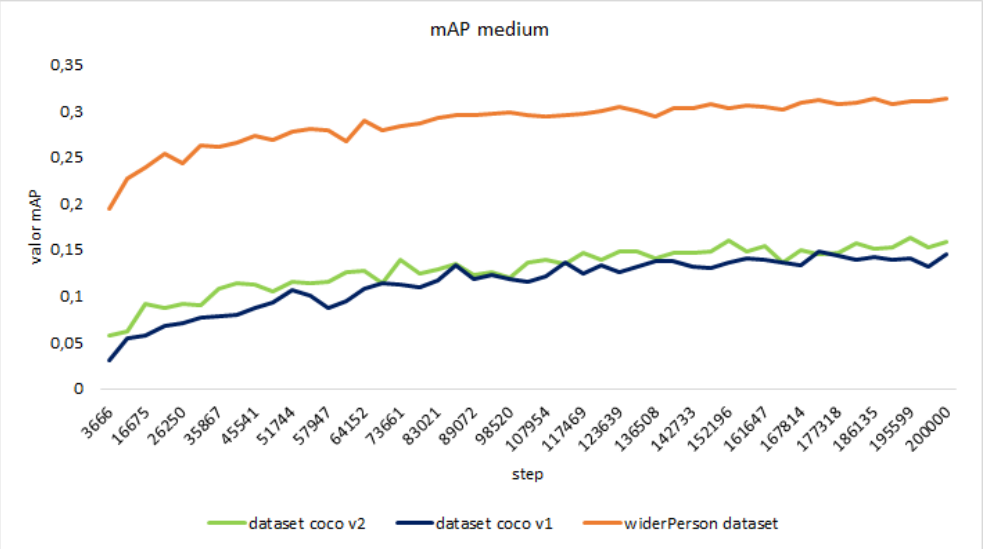

Figura 38 - Análise comparativa do mAP medium para 3 datasets

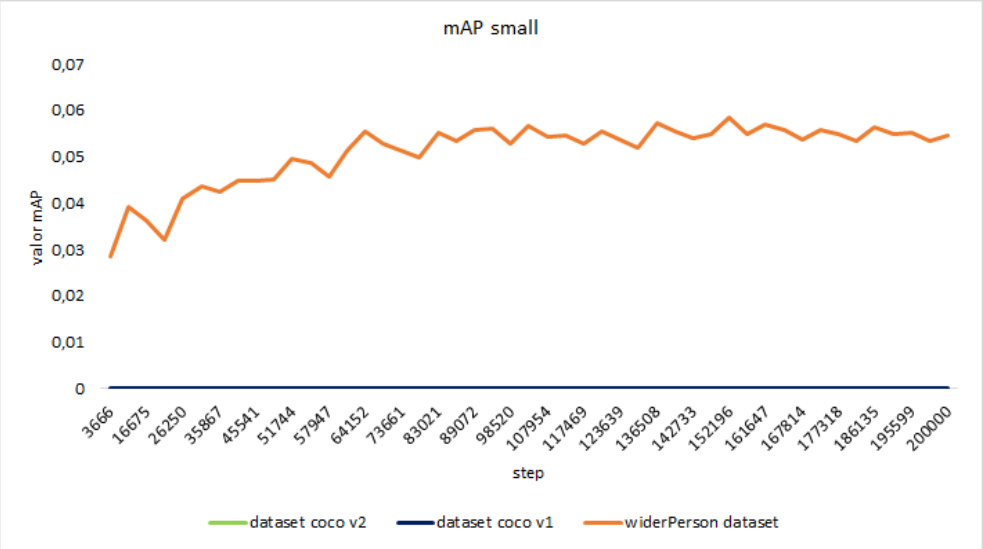

Figura 39 - Análise comparativa do mAP small para 3 datasets

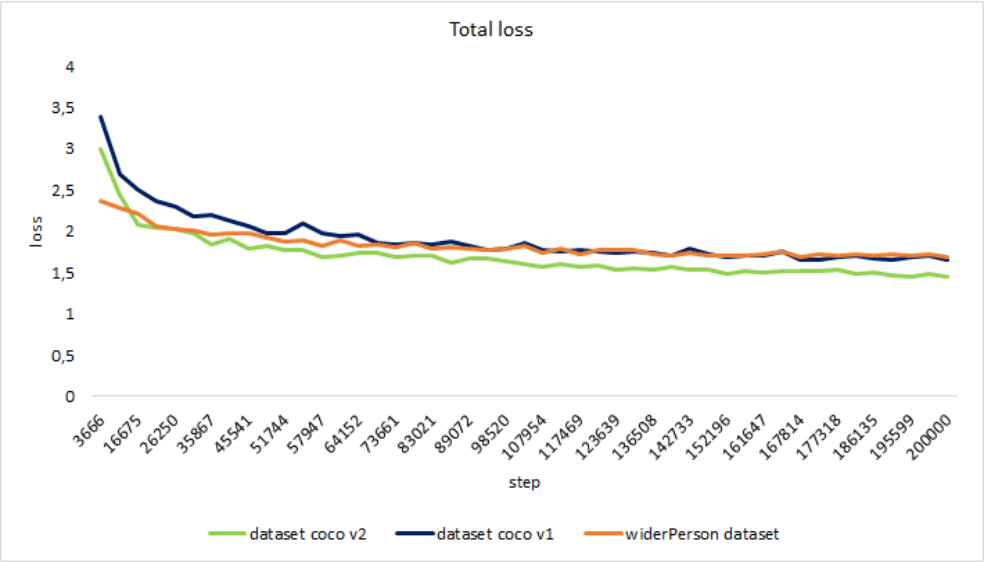

Figura 40 - Análise comparativa do total loss para 3 datasets

A evolução da total loss para os três cenários é apresentada na Figura 38. Observa-se que para os três datasets não houve muita diferença na evolução da total loss, indicando que o processo de ajuste dos parâmetros foi semelhante nos diferentes cenários.

Há, entretanto, uma pequena diferença entre os cenários quando avaliamos, separadamente, o classification loss e localization loss, conforme apresentado nas figuras 39 e 40. Observa-se que os resultados obtidos com o data set widerPerson atinge um valor da função de perda de classificação menor, mas atinge um patamar mais elevado da localization loss, possivelmente devido à maior dificuldade de encontrar coordenadas x e y de pessoas pequenas e médias.

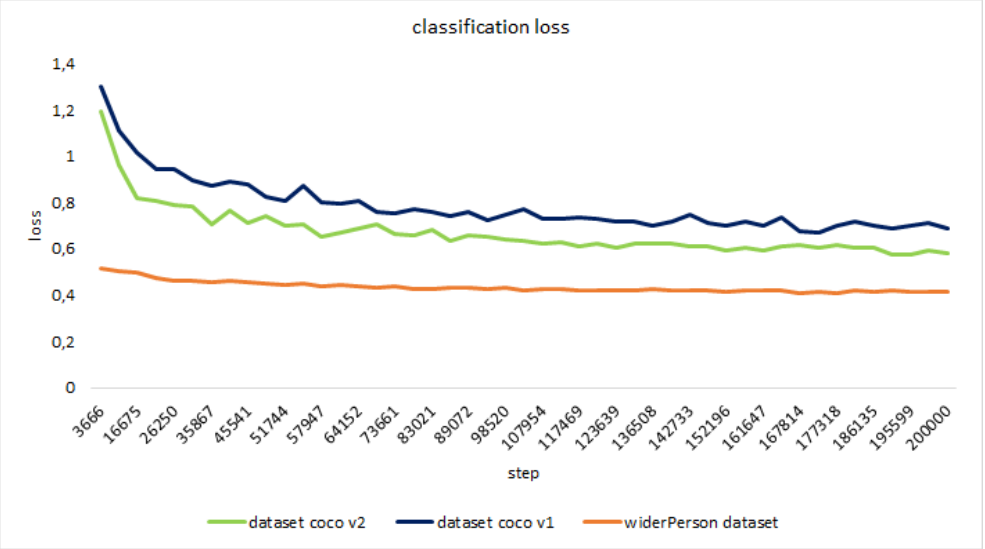

Figura 41 - Análise comparativa do classification loss para 3 datasets

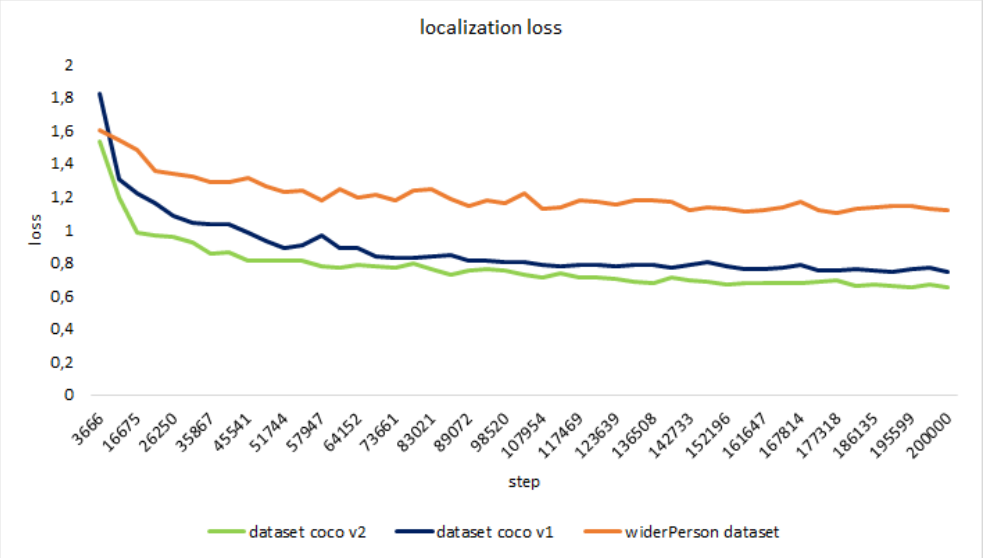

Figura 42 - Análise comparativa do localization loss para 3 datasets

8.2. Avaliação da solução nos dispositivos

A avaliação do aplicativo implementado foi feita de maneira subjetiva, utilizando os resultados de treinamento com cada um dos datasets do projeto.

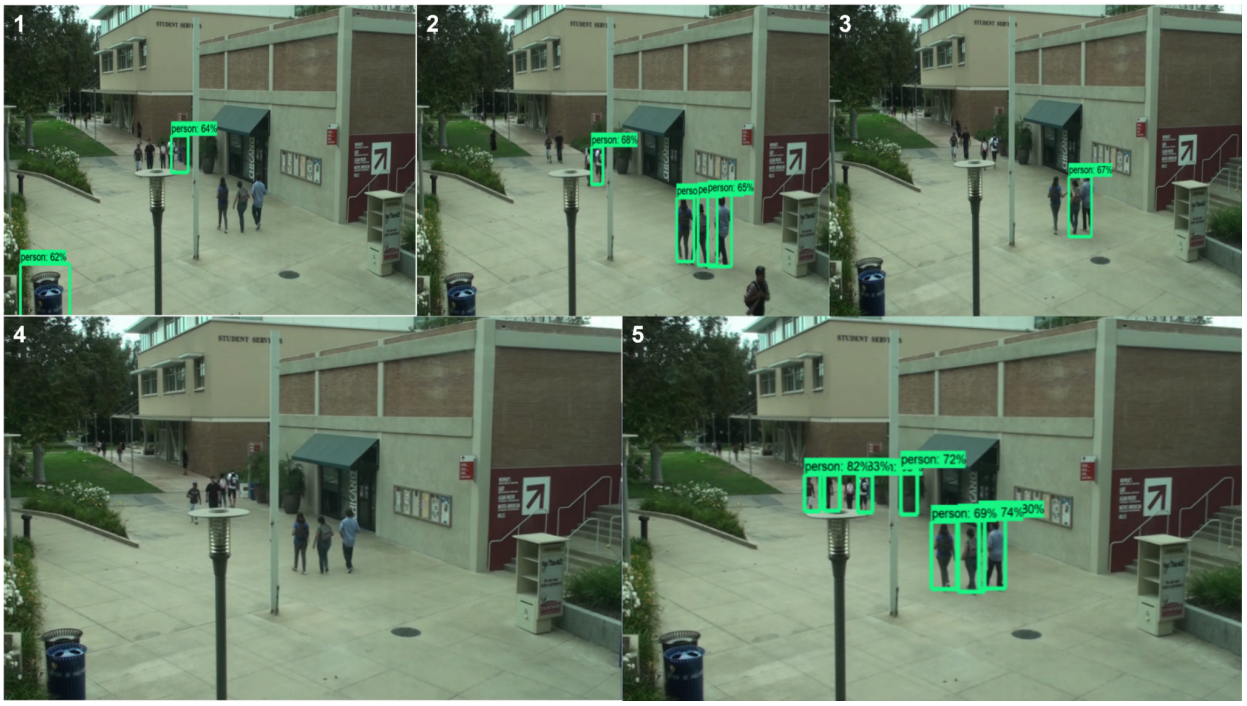

Figura 43: Resultados com os modelos em dispositivo móvel

Na figura 43, são apresentados resultados obtidos com os modelos treinados com cada um dos tipos de datasets do projeto:

1. Dataset de 200 imagens
2. Dataset V0
3. Dataset V1
4. Dataset V2
5. Dataset Wider

Nela percebe-se que o modelo 1 apresenta um falso positivo (diz que o lixo é uma pessoa) e a localização não está perfeita da pessoa. Tais erros ocorrem, em grande parte, pelo número reduzido de imagens no dataset, ao contrário do que se observa no exemplo obtido com o modelo 2, que foi treinado com muitas e variadas

imagens (59.146 imagens), que consegue detectar corretamente um maior número de pessoas e não apresenta falsos positivos. Mesmo assim, ele não consegue detectar algumas pessoas.

Já nos modelos 3 e 4, nota-se que mesmo com o tratamento de tirar imagens de pessoas pequenas e partes do corpo de uma pessoa (mão e ombro), o resultado não foi satisfatório. Finalmente, com o modelo 5, devido ao grande número de imagens, mais variadas (incluindo imagens de pessoas pequenas), o resultado foi melhor do que os modelos anteriores.

O comportamento descrito anteriormente reflete o desempenho geral observado em outros testes realizados. Entretanto, cabe ressaltar que, embora seja possível executar o modelo em um smartphone, o processamento das imagens exige bastante do dispositivo. Os testes foram realizados em um aparelho Octa-Core 2.7GHz, que contém 8GB de RAM, o que pode ser um fator importante a ser considerado em aplicações reais.

8.3. Possíveis aplicações

Após as análises de desempenho e testes realizados com vídeos do computador e usando a câmera do smartphone, conclui-se que essa solução seria adequada para monitoramento de pessoas em um determinado ambiente pelos circuito de câmeras, contudo a solução aqui apresentada, necessitaria de alguns ajustes quando trata-se de identificar pessoas que estejam pequenas na imagem.

Outra possível aplicação seria na contagem de pessoas que entram em um determinado ambiente, com algumas restrições, pois a solução não contaria corretamente caso uma multidão entrasse de uma única vez no ambiente, somente se as pessoas entrem uma após a outra. Esse tipo de solução proporcionará maior

segurança a todos que estão presentes no ambiente, principalmente, em um momento onde a aglomerações são um problema, pois facilitaria o contágio do vírus no ambiente, o que o projeto pode auxiliar na identificação da quebra dessa cautela que a população precisa ter e avisar as autoridades locais.

9. Conclusões e perspectivas

No presente trabalho foi explorado uma técnica de detecção de objetos utilizando o modelo Single-Shot Detector (SSD), uma das arquiteturas avançadas na área de detecção de objetos. Realizamos diversos experimentos variando datasets e parâmetros, sendo que melhor configuração foi utilizando o Dataset Wider e com os hiperparâmetros: batch_size=8, learning_rate=0.01, steps=200.000, focal=sim, freeze=não. Nota-se que os resultados obtidos mostram que a técnicas é capaz de detectar pessoas em imagens de vídeo, mas ainda pode ser ajustado para obter um melhor desempenho, principalmente quando utilizado para detecção de pessoas pequenas na imagem. A implementação dessa solução no smartphone foi bem sucedida, indicando que a aplicação pode ser implementada em dispositivos embarcados.

Por fim, nota-se que o objetivo da solução apresentada forneceu um resultado satisfatório, como perspectivas futuras podemos citar, um ajuste dos hiperparâmetros para encontrar melhor a solução, assim como utilizar novos *datasets* e testar outras arquiteturas, como por exemplo YOLO.

10. Bibliografia

[1] TENSORFLOW. **tensorflow/models: Models and examples built with TensorFlow**. Disponível em: <https://github.com/tensorflow/models>. Último acesso em 31/07/2021.

[2] ***Funções De Ativação,*** *Disponível em: <https://matheusfacure.github.io/2017/07/12/activ-func/>* . Último acesso em 31/07/2021

[3] ***Hui, Jonathan. "SSD Object Detection: Single Shot MultiBox Detector for Real-Time Processing."*** Medium, Medium, 28 Dec. 2018, https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06. Último acesso em 31/07/2021

[4] **"YOLO." YOLO - DI 0.0.1 Documentation**, https://deep-learning-study-note.readthedocs.io/en/latest/Part 2 (Modern Practical Deep Networks)/12 Applications/Computer Vision External/YOLO.html. Último acesso em 31/07/2021

[5] **Redmon, Joseph. YOLO: Real-Time Object Detection**, https://pjreddie.com/darknet/yolo/. Último acesso 31/07/2021.

[6] **"Common Objects in Context." COCO**, https://cocodataset.org/. Último acesso em 31/07/2021.

[7] L. Wei, A. Dragomir, E. Dumitru, S. Christian, R. Scott, F. Cheng-Yang, C. Alexander Berg. **SSD: Single Shot MultiBox Detector.**

[8] ANDRILUKA, M. et al. **2D Human Pose Estimation: New Benchmark and State of the Art Analysis.** 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014.

[9] Disponível em: https://people.eecs.berkeley.edu/~nzhang/piper.html. Último acesso em 07/04/2019.

[10] OH, S. J. et al. Person Recognition in Personal Photo Collections. **2015 IEEE International Conference on Computer Vision (ICCV)**, 2015.

[11] **Food image recognition using deep convolutional network with pre-training and fine-tuning** - Keiji Yanai, Yoshiyuki Kawano 2015 IEEE International Conference on Multimedia & Expo Workshops (ICMEW) - 2015

[12] **Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning** - Hoo-Chang Shin, Holger Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, Ronald Summers IEEE Transactions on Medical Imaging - 2016

[13] **balancap/SSD-Tensorflow: Single Shot MultiBox Detector in TensorFlow** GitHub.

[14] **You Only Look Once: Unified, Real-Time Object Detection** - Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) - 2016

[15] C. Nakajima, M. Pontil, B. Heisele, and T. Poggio. **People recognition in image sequences by supervised learning**. In MIT AI Memo, 2000.

[16] D. Gavrila. **Pedestrian detection from a moving vehicle**. In Proc. 6th European Conf. Computer Vision, Dublin, Ireland, volume 2, pages 37–49, 2000.

[17] LIU, X. et al. Detecting and counting people in surveillance applications. **Proceedings. IEEE Conference on Advanced Video and Signal Based Surveillance, 2005.**, [s.d.].

[18] S. Russell, A. Mykhaylo and N. Y. Andrew. **End-to-end people detection in crowded scenes. Stanford University**, USA, 2016.

[19] C. C. Caio. **Redes neurais em dispositivos raspberry PI para detecção de pessoas**. 2018.

[20] R. Joseph, Ali Farhadi. **YOLOv3: An Incremental Improvement**, University of Washington.

[21] Z. Iffat, T. Giounona, B. Richard, P. Nimesh, A. Leonardo. **Hands-On Convolutional Neural Networks with TensorFlow - Solve computer vision problems with modeling in TensorFlow and Python**. 2017.

[22] Mendes, D. Visão Computacional e Reconhecimento de Imagem. Disponível em:https://www.datascienceacademy.com.br/path-player?courseid=viso-computacional-e-re

conhecimento-de-imagem&unit=5a467f2a5e4cdeba928b456cUnit. Último acesso em 31/07/2021

[23] **Android quickstart : TensorFlow Lite**. Disponível em: <https://www.tensorflow.org/lite/guide/android> Último acesso em 31/07/2021

[24] CHAGAS, E. T. D. O. Deep Learning e suas aplicações na atualidade. **Revista Científica Multidisciplinar Núcleo do Conhecimento**, v. 04, n. 05, p. 05–26, 2019.

[25] Andrew G. Menglong Zhu , Bo Chen e Dmitry Kalenichenko. **MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications**

[26] TZUTALIN. **tzutalin/labelImg: LabelImg is a graphical image annotation tool and label object bounding boxes in images**. Disponível em: <https://github.com/tzutalin/labelImg>. Último acesso em 31/07/2021

[27] TENSORFLOW. **models/tf2_detection_zoo.md at master · tensorflow/models**. Disponível em: <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md> Último acesso em 31/07/2021

[28] GARCIA, M. **Spark - Saiba mais sobre essa poderosa ferramenta: Blog Cetax**. Disponível em: <https://www.cetax.com.br/blog/conheca-mais-sobre-o-framework-apache-spark/>. Último acesso em 31/07/2021

[29] **Global indexes**. Disponível em: <https://www.visionofhumanity.org/maps/#/> Último acesso em 31/07/2021

[30] SECURITY, E. P. O. R. R. D. et al. **Mercado de segurança eletrônica cresce 13% em 2020**. Disponível em: <https://revistadigitalsecurity.com.br/mercado-de-seguranca-eletronica-cresce-13-em-2020/> . Último acesso em 31/07/2021

[32] REN, S. et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 39, n. 6, p. 1137–1149, 2017.

[33] GIRSHICK, R. et al. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. **2014 IEEE Conference on Computer Vision and Pattern Recognition**, 2014.

[34] REDMON, J. et al. You Only Look Once: Unified, Real-Time Object Detection. **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, 2016.

[35]NAKAJIMA, C. et al. People Recognition in Image Sequences by Supervised Learning. 2000.

[36] LIU, X. et al. Detecting and counting people in surveillance applications. **Proceedings. IEEE Conference on Advanced Video and Signal Based Surveillance, 2005.**, [s.d.].

[37] JHA, P. **A Brief Overview of Loss Functions in Pytorch**. Disponível em: <https://medium.com/udacity-pytorch-challengers/a-brief-overview-of-loss-functions-in-pytorch-c0ddb78068f7#:~:text=Smooth L1 Loss&text=It uses a squared term,and an absolute term otherwise.&text=In mean square error loss,values result in exploding gradients> Último acesso em 31/07/2021

[38] LIU, W. et al. SSD: Single Shot MultiBox Detector. **Computer Vision – ECCV 2016 Lecture Notes in Computer Science**, p. 21–37, 2016.

[39] SECURITY, E. P. O. R. R. D.; SECURITY, R. D.; REDAÇÃO DIGITAL SECURITY REDAÇÃO DA REVISTA DIGITAL SECURITY. **10 principais tendências do setor de segurança em 2021**. Disponível em: <https://revistadigitalsecurity.com.br/10-principais-tendencias-do-setor-de-seguranca-em-2021/>. Último acesso em 31/07/2021.

[40] Zhengxia Zou, Zhenwei Shi, Member, IEEE, Yuhong Guo, and Jieping Ye. **Object Detection in 20 Years: A Survey**, 2019.

[4!] KUMAR, N. **Deep Learning Best Practices: Regularization Techniques for Better Performance of Neural Network**. Disponível em: <https://heartbeat.fritz.ai/deep-learning-best-practices-regularization-techniques-for-better-performance-of-neural-network-94f978a4e518>. Último acesso em 31/07/2021.

[42] MITTAL, K. **A Gentle Introduction Into The Histogram Of Oriented Gradients**. Disponível em: <https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa>. Último acesso em 31/07/2021.

[43] WAJIH et al. **Intersection over Union (IoU) for object detection**. Disponível em: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/> Último acesso em 31/07/2021.

[44] **Understanding Loss Functions in Machine Learning**. Disponível em: <https://www.section.io/engineering-education/understanding-loss-functions-in-machine-learning/>. Último acesso em 31/07/2021.

[45] ALMEIDA, R. P. D. Gerenciamento de dados de proveniência de workflow de bioinformática com banco de dados baseados em grafo. [s.d.]. Capítulo 3, Como funciona o Deep Learning.

[46] TENSORFLOW. **models/research/object_detection at master · tensorflow/models**. Disponível em: <https://github.com/tensorflow/models/tree/master/research/object_detection>. Último acesso em 31/07/2021.

[47] Diederik P. Kingma, Jimmy Lei Ba. **ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION, 2017.**

[48] **CUDA Toolkit**. Disponível em: <https://developer.nvidia.com/cuda-toolkit>. Último acesso em 31/07/2021.

[49] VIOLA, P.; JONES, M. Robust real-time face detection. **Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001**, [s.d.].

[50] DALAL, N.; TRIGGS, B. Histograms of Oriented Gradients for Human Detection. **2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)**, [s.d.].

[51] P. F. Felzenszwalb, R. B. Girshick, D. McAllester and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 9, pp. 1627-1645, Sept. 2010, doi: 10.1109/TPAMI.2009.167.

[52] Disponível em: <https://gombru.github.io/2018/05/23/cross_entropy_loss/#:~:text=is available here-,Binary Cross-Entropy Loss,affected by other component values.>. Último acesso em 31/07/2021.