CENTRO UNIVERSITÁRIO FEI

LUCAS RIBEIRO DE ABREU

REDES NEURAIS CONVOLUCIONAIS APLICADAS À DETECÇÃO DE OBJETOS NO DOMÍNIO DO FUTEBOL DE ROBÔS HUMANOIDES

São Bernardo do Campo

2021

LUCAS RIBEIRO DE ABREU

REDES NEURAIS CONVOLUCIONAIS APLICADAS À DETECÇÃO DE OBJETOS NO DOMÍNIO DO FUTEBOL DE ROBÔS HUMANOIDES

Dissertação de Mestrado, apresentada ao Centro Universitário FEI para obtenção do título de Mestre em Engenharia Elétrica. Orientado pelo Prof. Dr. Reinaldo A. C. Bianchi.

São Bernardo do Campo

2021

Ribeiro de Abreu, Lucas.
Redes Neurais Convolucionais aplicadas à Detecção de Objetos no Domínio do Futebol de Robôs Humanoides / Lucas Ribeiro de Abreu. São Bernardo do Campo, 2021.
100 f.

Dissertação - Centro Universitário FEI.
Orientador: Prof. Dr. Reinaldo Bianchi.

1. YoloV4. 2. Rede neural convolucional. 3. futebol de robôs. 4. mobilenet. I. Bianchi, Reinaldo, orient. II. Título.

Elaborada pelo sistema de geração automática de ficha catalográfica da FEI com os dados fornecidos pelo(a) autor(a).

centro universitário FEI

APRESENTAÇÃO DE DISSERTAÇÃO
ATA DA BANCA EXAMINADORA

Programa de Pós-Graduação Stricto Sensu em Engenharia Elétrica

Mestrado

PGE-10

Aluno: Lucas Ribeiro de Abreu **Matrícula:** 118114-8

Título do Trabalho: Redes neurais convolucionais aplicadas à detecção de objetos no domínio do futebol de robôs humanoides.

Área de Concentração: Inteligência Artificial Aplicada à Automação e Robótica

Orientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi

Data da realização da defesa: 21/12/2020

ORIGINAL ASSINADA

Avaliação da Banca Examinadora:

A banca foi realizada no dia 21 de dezembro às 14:00 horas, e se iniciou com a apresentação do aluno, e seguiu para a arguição, onde o aluno respondeu a todas as questões de forma adequada demonstrando conhecimento pleno do tema. Foram sugeridas melhorias em relação ao texto para a versão final. A aprovação foi por unanimidade.

São Bernardo do Campo, / / .

MEMBROS DA BANCA EXAMINADORA	
Prof. Dr. Reinaldo Augusto da Costa Bianchi	Ass.: _____
Prof. Dr. Flavio Tonidandel	Ass.: _____
Prof. Dr. Alexandre da Silva Simões	Ass.: _____

A Banca Julgadora acima-assinada atribuiu ao aluno o seguinte resultado:

APROVADO ☒ REPROVADO ☐

VERSÃO FINAL DA DISSERTAÇÃO

APROVO A VERSÃO FINAL DA DISSERTAÇÃO EM QUE FORAM INCLUÍDAS AS RECOMENDAÇÕES DA BANCA EXAMINADORA

_____

Aprovação do Coordenador do Programa de Pós-graduação

_____

Prof. Dr. Carlos Eduardo Thomaz

Dedico este trabalho à minha família.

AGRADECIMENTOS

Agradeço primeiramente à minha esposa, pai, mãe, irmão, amigos e colegas de trabalho. Agradeço também ao meu orientador Reinaldo Bianchi por ter acreditado em mim desde o início do mestrado. Finalmente, agradeço ao Centro Universitário FEI por ter me proporcionado uma bolsa de estudos por Mérito.

“Imagination is more important than knowledge. Knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution”

Albert Einstein, What Life Means to Einstein (1924)

RESUMO

A RoboCup é uma das maiores iniciativas no ramo de pesquisa em robótica. Essa iniciativa considera o futebol como um dos maiores desafios para robôs e tem o intuito de promover e ganhar um jogo de futebol entre humanos e robôs até o ano de 2050. O módulo de visão dos robôs é um sistema crítico, pois precisa localizar e classificar objetos de interesse ao robô em tempo real, com o objetivo de tomar a melhor ação dado o ambiente a sua volta. Este trabalho avalia redes neurais convolucionais profundas para detecção da bola de futebol e de robôs. Para tal tarefa, cinco arquiteturas da literatura foram escolhidas e treinadas utilizando conceitos de transferência de aprendizado e aumento de dados. Os modelos foram avaliados em um conjunto de dados de teste, gerando resultados promissores em termos de precisão e quadros por segundo. O melhor modelo atingiu um mAP de 0.98 com 50% de interseção a uma taxa de 14.7 quadros por segundo, sendo executado em uma CPU.

Keywords: Detecção de Objetos. Redes Neurais Convolucionais. MobileNet. YOLO.

ABSTRACT

The RoboCup Soccer is one of the largest initiatives in the robotics field of research. This initiative considers the soccer match as a challenge for the robots and aims to win a match between humans *versus* robots by the year of 2050. The vision module is a critical system for the robots because it needs to quickly locate and classify objects of interest for the robot in order to generate the next best action. This work evaluates deep neural networks for the detection of the ball and robots. For such task, five convolutional neural networks architectures were trained for the experiment using data augmentation and transfer learning techniques. The models were evaluated in a test set, yielding promising results in precision and frames per second. The best model achieved an mAP of 0.98 and 14.7 frames per second, running on CPU.

Keywords: Object Detection. Convolutional Neural Network. MobileNet.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AP	Precisão Média (*Average Precision*).
AWS	*Amazon Web Services*.
BN	Normalização de Lotes (*Batch Normalization*).
CNN	Redes Neurais Convolucionais - *Convolutional Neural Networks*.
COCO	*Common Objects in Context*.
CPU	Unidade de Central de Processamento (*Central Process Unit*).
EC2	*Amazon Elastic Compute Cloud*.
FC	Totalmente conectadas (*Fully connected*).
FEI	Centro Universitário FEI.
FPS	Frames por Segundo.
GPU	Unidade de Processamento Gráfico (*Graphics Processing Unit*).
ILSVRC	*ImageNet Large Scale Visual Recognition Competition*.
IoU	Interseção sobre União (*Intersection over Union*).
mAP	Média da Precisão Média (*Mean Average Precision*).
MLP	Perceptrons Multicamadas (*Multilayer Perceptron*).
NiN	*Network In Network*.
NLL	Probabilidade Negativa de Log *Negalive Log Likelihood*.
NMS	Supressão Não-Máxima *Non-maximum Suppression*.
ReLU	Unidade de Retificação Linear (*Rectified Linear Unit*).
RNA	Redes Neurais Artificiais.
SSD	*Single Shot MultiBox Detector*.
SSE	Soma do Erro Quadrátrico *Error Sum of Squares*.
VOC	*Visual Object Classes*.
WVC	Workshop de Visão Computacional.
YOLO	*You Only look once*.

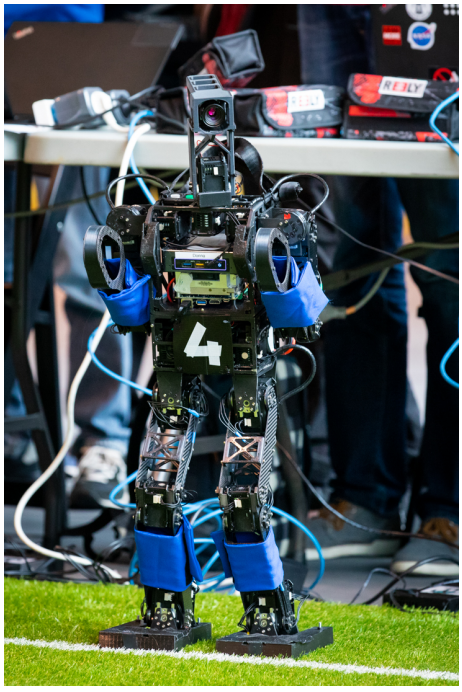
SUMÁRIO

1 INTRODUÇÃO

A principal competição global de futebol de robôs é organizada pela iniciativa *Robot World Cup Competition* (RoboCup) (ROBOCUP..., 2020). O objetivo da iniciativa é de incentivar o uso de uma gama de tecnologias relacionadas a robótica e inteligência artificial em problemas complexos e padronizados (KITANO et al., 1997). Para longo prazo, Kitano e Asada (1998) também propuseram que até o ano de 2050 uma equipe de futebol de robôs humanoides totalmente autônoma vença um jogo de futebol, cumprindo as regras oficiais da FIFA, contra o vencedor da Copa do Mundo mais recente. A primeira competição da RoboCup aconteceu em Nagoya em 1997 e acontece todos os anos contemplando diversas competições além do futebol de robôs.

Para que um robô autônomo seja funcional, alguns elementos como design de agentes autônomos, colaboração multi-agentes, raciocínio em tempo real, robótica, sensorização e inteligência artificial precisam ser combinados. Como as regras da RoboCup só permitem o uso de sensores com funções similares aos sentidos humanos, um importante elemento sensorial é a câmera, a qual pode ser observada na parte superior do robô na Figura 1.

Figura 1 – Foto de um robô humanoide. Pode-se observar a câmera na sua parte superior.
Fonte: (ROBOCUP, 2019)



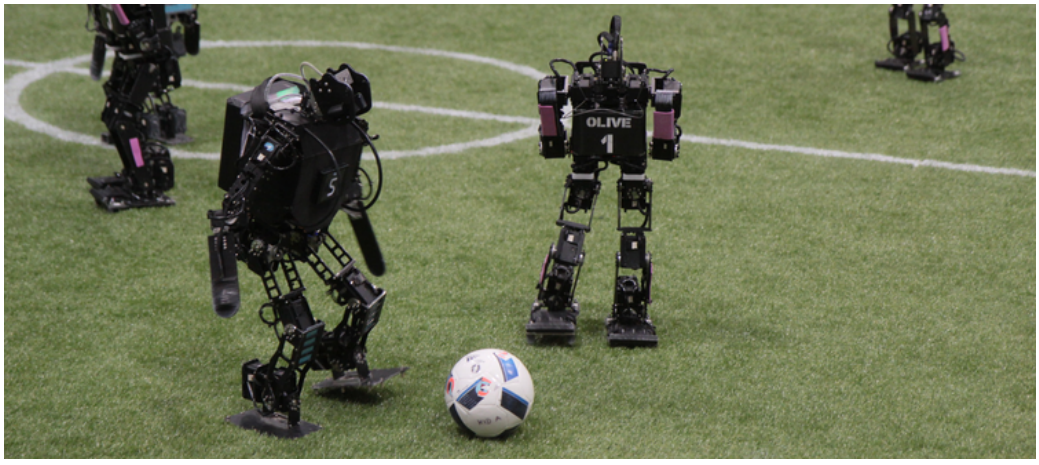
As câmeras são ricas fontes de informações e precisam ser exploradas no domínio do futebol de robôs para viabilizar a interpretação do ambiente. Os robôs não só precisam identificar

a posição da bola de futebol para conseguir se movimentar na direção correta e viabilizar um passe ou chute a gol mas também precisam evitar colisões com outros robôs a sua volta, tanto do próprio time quanto do time adversário. Logo, com a interpretação do ambiente, pode-se tomar as melhores decisões sobre a próxima ação a ser executada.

No entanto, existem alguns desafios na interpretação dos dados da câmera quanto a velocidade e precisão. Se a detecção de objetos acontece de forma lenta, o robô irá tomar decisões atrasadas. Por isso, a detecção precisa acontecer em uma alta taxa de quadros por segundo para que o robô consiga interpretar o ambiente e agir sem atrasos.

Além da velocidade, a precisão é um fator determinante para a interpretação do ambiente. Até o ano de 2016 a bola possuía coloração laranja, o que destacava a bola dentro do campo de futebol e facilitava a criação de algoritmos para detecção da bola com técnicas de segmentação de cores (WASIK; SAFFIOTTI, 2002). Todavia, a partir do ano de 2016, as bolas adotadas passaram a possuir padrões em branco e uma segunda cor predominante (MENASHE et al., 2017), como pode ser observado na Figura 2. Esta nova coloração que pode gerar detecções equivocadas com a marca do pênalti por exemplo. Além disso, as linhas do campo são brancas e alguns robôs podem possuir coloração similar. Isso gera um ambiente mais desafiador aos algoritmos de visão.

Figura 2 – Ilustração do ambiente do futebol de robôs em que a bola pode ser observada.
Fonte: (ROBOCUP, 2019)



Neste contexto, algoritmos mais complexos e com maior precisão ganharam popularidade nos últimos anos, como as redes neurais artificiais profundas. Embora essas redes tenham diversas aplicações, elas agora são sem dúvida a técnica mais popular no campo da percepção inteligente, especialmente a visão computacional, segundo Szemenyei e Estivill-Castro (2019a).

Pesquisas relacionadas a Redes Neurais Convolucionais (CNN - *Convolutional Neural Network*) tornaram-se populares desde que Krizhevsky, Sutskever e Hinton (2012) venceram a ILSVRC (*ImageNet Large Scale Visual Recognition Competition*) (DENG et al., 2009), em 2012, com uma arquitetura de CNN chamada AlexNet. Em uma análise da evolução dos algoritmos na ILSVRC, Russakovsky et al. (2015) consideram vitória da AlexNet um marco para tarefas de reconhecimentos de objetos em larga escala. As grandes contribuições da AlexNet foram a arquitetura inovadora atrelada a computação em unidades de processamento gráfico (GPU - *Graphics Processing Unit*) e taxa de acurácia significativamente superior aos vencedores anteriores.

A partir de 2012, o ajuste de arquiteturas neurais profundas para obter a máxima precisão equilibrada com um bom desempenho foi uma área de pesquisa bastante ativa no meio acadêmico. Tanto a pesquisa manual de arquitetura quanto as melhorias nos algoritmos de treinamento levaram melhorias significativas em relação aos projetos iniciais. A arquitetura convolucional profunda da AlexNet contém cinco camadas convolucionais e três camadas totalmente conectadas, totalizando oito camadas e 60 milhões de parâmetros. Em 2014, a GoogLeNet contendo 22 camadas (SZEGEDY et al., 2015) foi a vitoriosa no ILSVRC. No ano de 2015 a vencedora foi a ResNet (HE et al., 2016), que já incrementou a quantidade de camadas para 152.

As redes neurais artificiais evoluíram rapidamente e revolucionaram muitas áreas de inteligência de artificial, permitindo precisão sobre-humana para tarefas desafiadoras de reconhecimento de imagem. No entanto, o impulso para melhorar a precisão muitas vezes tem um custo: redes de última geração exigem recursos computacionais elevados, os quais estão além das capacidades de muitos dispositivos móveis e embarcados (SANDLER et al., 2018), que é o caso dos robôs humanoides.

1.1 OBJETIVO

Este trabalho tem o objetivo de aplicar técnicas de aumento de dados e transferência de aprendizado a arquiteturas de CNNs já existentes conhecidas como GoogLeNet, ResNet, YOLOv4, MobileNetV2 e MobileNetV3, realizando as adaptações de arquitetura necessárias para o contexto do do futebol de robôs humanoides.

As CNNs devem ser executadas em uma CPU (*Central Process Unit* ou Unidade Central de Processamento) para detectar (i) Robôs e (ii) Bolas de futebol. O processo de detecção

fornece coordenadas, além de largura e altura de um ou mais objetos dentro da imagem ou quadro de vídeo. Os dados de coordenadas, largura e altura formam o que chama-se de caixa delimitadora.

Adicionalmente, as CNNs devem ser robustas o bastante para serem capazes de detectar os objetos propostos em diferentes campos de futebol, com variações de movimento, iluminação e brilho.

1.2 MOTIVAÇÃO

A escolha dos dois objetos a serem detectados foi feita com a motivação de que a bola de futebol é um objeto fundamental a ser detectado no jogo. Toda a estratégia necessária para que robôs sejam capazes de jogar uma partida de futebol gira em torno da bola, que precisa ser detectada tanto para ataque ao gol adversário, quanto para defender contra ataques ao próprio gol. Diversos artigos relacionados e recentes (POPPINGA; LAUE, 2019) (TEIMOURI; DELAVARAN; REZAEI, 2019) (HOULISTON; CHALUP, 2019) (SZEMENYEI; ESTIVILL-CASTRO, 2019b) (SZEMENYEI; ESTIVILL-CASTRO, 2019a) (SPECK et al., 2017) (KUKLEVA et al., 2019) da área visam identificar apenas este objeto com uma CNN ou este objeto em conjunto a outros objetos como robôs e traves do gol.

Assim como existem desafios na detecção da bola, a detecção de robôs é dificultada pelo fato de que eles adotam posturas diferentes (por exemplo, em pé, deitado, levantando-se ou de perfil). Além disso, frequentemente só aparecerem no campo de visão algumas partes do robô quando estão próximos à câmera. Outro fator que torna mais complexa a identificação é pelo motivo deles não serem simétricos e, portanto, adotam aspectos diferentes dependendo do ângulo de visão (POPPINGA; LAUE, 2019).

O presente trabalho já obteve resultados preliminares que foram publicados em um artigo no Workshop de Visão Computacional (WVC) (ABREU; BIANCHI, 2019). O artigo avalia arquiteturas conhecidas de CNN para detecção de bolas de futebol em tempo real e obteve resultados promissores.

Este trabalho está organizado da seguinte maneira: o primeiro capítulo apresentou uma breve introdução sobre a RoboCup, um contexto sobre a evolução das redes convolucionais, seus problemas atuais e um caminho para resolvê-los. Além disso, foi exposta a motivação e o objetivo deste trabalho.

A seguir, este trabalho apresentará mais cinco capítulos, sendo o Capítulo 2 uma Fundamentação Teórica dos conceitos aplicados neste trabalho. O Capítulo 3 é composto por análise sobre Trabalhos Correlatos. O Capítulo 4 possui detalhes sobre a Metodologia do trabalho, seguido do Capítulo 5 com Resultados dos experimentos e Capítulo 6 contendo a Conclusão.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo será introduzido um breve contexto sobre a história das Redes Neurais Artificiais. Em sequência, alguns conceitos sobre Redes Neurais Convolucionais e seus principais componentes.

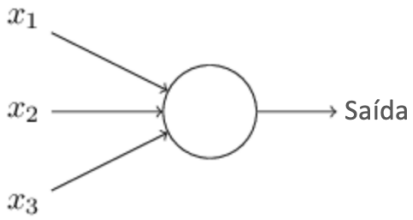
Após os principais conceitos teóricos serem apresentados, o capítulo explora algumas arquiteturas de Redes Neurais Convolucionais e seu funcionamento para detecção de objetos.

2.1 REDES NEURAIS ARTIFICIAIS

Algoritmos baseados no funcionamento biológico humano datam desde a década de 50, quando Rosenblatt (1957) propôs o Perceptron, um tipo de neurônio artificial que foi inspirado em trabalhos prévios de McCulloch e Pitts (1943) comparando a atividade neural à lógica proposicional.

Um perceptron recebe n variáveis de entradas binárias, $x_1, ..., x_n$ e produz uma única saída binária:

Figura 3 – Exemplo de um Perceptron que recebe três entradas e possui uma saída. Fonte: Autor.



No exemplo apresentado, o perceptron possui três entradas, x_1, x_2, x_3. Rosenblatt propôs uma regra simples para calcular a saída introduzindo pesos, $w_1, ..., w_n$, que são números reais expressando a importância das respectivas entradas em relação a saída. A saída binária do neurônio, 0 ou 1, é determinada se a soma ponderada $\sum_j x_j w_j$ é menor ou maior que um valor limiar. Assim como os pesos, o limiar é um número real, que é um parâmetro do neurônio. Para colocar em termos algébricos mais precisos:

$$f(x) = \begin{cases} 0, & \text{se } \sum_j x_j w_j \leq limiar \\ 1, & \text{se } \sum_j x_j w_j > limiar \end{cases} \tag{1}$$

Uma evolução dos neurônios Perceptrons são os Sigmóides. Apesar de semelhantes em termos de entradas e pesos, os sigmóides são modificados para que pequenas alterações em seus pesos causem apenas uma pequena alteração em sua saída. Esse é o fato crucial que permite que uma rede de neurônios sigmóides aprenda (NIELSEN, 2015).

Assim como um Perceptron, o neurônio Sigmóide possui entradas, $x_1, ..., x_n$ e possui pesos para cada entrada, $w_1, ..., w_n$, mas adiciona uma constante chamada de viés ou *bias*, denotada por b. Outra modificação é que as entradas $x_1, ..., x_n$ podem assumir valores reais quaisquer. Consequentemente, a saída do neurônio final assume valores da função $\sigma(w \cdot x + b)$, onde σ é definida por:

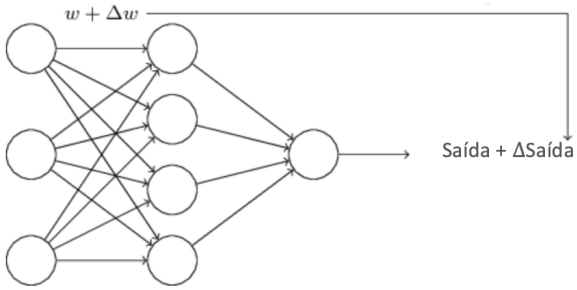
$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

Para observar como o aprendizado de redes compostas por neurônios interligados pode funcionar, suponha uma rede conforme mostra a Figura 4. A camada mais à esquerda nesta rede é chamada de camada de entrada e os neurônios dentro da camada são chamados de neurônios de entrada. A camada mais à direita ou de saída contém os neurônios de saída ou, como neste caso, um único neurônio de saída. A camada do meio é chamada de camada oculta, uma vez que os neurônios dessa camada não são entradas nem saídas. Nesse caso, a camada oculta também é conhecida como camada totalmente conectada por possuir conexões entre todos os neurônios da camada anterior.

Apesar da rede poder ser composta de sigmóides ou outros tipos de neurônios, esse tipo de rede também é conhecida como Perceptrons Multicamadas (MLP - *Multilayer Perceptron*) ou simplesmente Redes Neurais Artificiais (RNA) (NIELSEN, 2015).

Com o princípio de que cada pequena alteração no peso causa apenas uma pequena alteração no resultado, podemos usar esse fato iterativamente para modificar os pesos e os *bias* para que nossa RNA se comporte da maneira que desejamos (NIELSEN, 2015).

Figura 4 – Rede de neurônios artificiais. Fonte: (NIELSEN, 2015) adaptado.



Sendo assim, um necessita-se de um algoritmo que permita encontrar pesos e *bias* para que a saída da rede se aproxime de uma resposta desejada. Supondo que as entradas sejam um vetor x e as saídas desejadas um vetor $y(x)$, para quantificar quão bem a rede está estimando essas saídas desejadas, define-se uma função de custo quadrático ou erro médio quadrático:

$$C(w,b) = \frac{1}{2n}\sum_{x} ||y_i - a_i||^2 \tag{3}$$

Onde w denota a coleção de todos os pesos na rede, b todos os *bias*, n é o número total de entradas de treinamento, a é o vetor de saídas da rede (composta por w e b) quando x é entrada e y é a resposta esperada.

Com uma função de custo definida, o objetivo no aprendizado ou treinamento da rede neural passa a ser o de encontrar pesos e *bias* que minimizem a função de custo quadrático $C(w,b)$. Para tal tarefa de minimização, a técnica de gradiente descendente é utilizada associada a uma técnica que avalia as contribuições de cada peso w_i do erro por meio do cálculo de gradientes com a regra de cadeia. O processo é conhecido como retropropagação e foi desenvolvida por Rumelhart, Hinton e Williams (1986).

Em termos simples, após cada processamento direto (*feed foward*) da entrada x da rede através da RNA, a retropropagação executa um processamento reverso e ajusta os parâmetros do modelo (pesos w e vieses b) de acordo com uma taxa de aprendizado η . Este processo ocorre iterativamente até atingir um critério de parada ou até que a função custo obtenha seu valor mínimo sem grandes diferenças entre as iterações. Matematicamente, a definição da direção correta para otimização da função de custo é obtida através dos gradientes da função em relação a saída da rede neural, conforme equações abaixo, onde t denota cada iteração do algoritmo e k o índice do peso w no vetor de pesos.

$$w_k^{t+1} = w_k^t - \eta\frac{\partial C}{\partial w_k} \tag{4}$$

$$b_k^{t+1} = b_k^t - \eta\frac{\partial C}{\partial b_k} \tag{5}$$

Apesar dos avanços, as redes neurais profundas só ganharam mais popularidade em termos de aplicações práticas entre no final dos anos 2000 com a acessibilidade a GPUs e com o volume de dados disponíveis para treinamento das redes aumentando exponencialmente. Pesquisas com diversos tipos de RNAs, como as multicamadas, convolucionais e recorrentes evoluíram e receberam otimizações e inovações, além de receberem mais camadas. Segundo Mallat (2016), as RNAs que possuem mais de cinco camadas podem ser consideradas RNAs

profundas. As redes neurais convolucionais - CNN, foco deste trabalho, são um tipo de RNA e serão detalhadas na próxima seção.

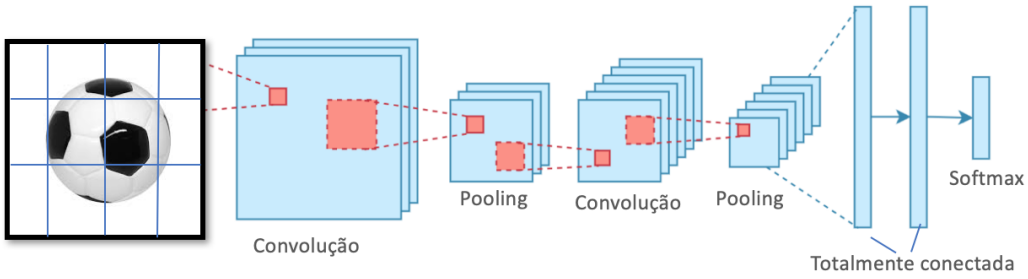
2.2 REDES NEURAIS CONVOLUCIONAIS

Segundo Goodfellow, Bengio e Courville (2016), Redes Neurais Convolucionais (CNN - *Convolutional Neural Network*) são simplesmente redes neurais que utilizam a convolução no lugar da multiplicação geral de matrizes em pelo menos uma de suas camadas.

Já para LeCun, Kavukcuoglu e Farabet (2010), CNN é uma arquitetura biologicamente inspirada que é treinável para aprender características invariantes baseados em dados de entrada e saída, caracterizando um problema de aprendizado supervisionado.

Modelos de CNN seguem arquiteturas semelhantes, conforme ilustrado na Figura 5. A Figura mostra uma imagem de entrada de uma bola de futebol seguida de uma série de operações de Convolução, *Pooling*, seguidas de camadas Totalmente Conectadas e uma última camada de *Softmax* para gerar probabilidades de classes para a imagem de entrada. Nas próximas seções os conceitos dessas camadas serão detalhados.

Figura 5 – Exemplo de rede neural convolucional com imagem de entrada e camadas de Convolução, *Pooling*, Totalmente Conectadas e *Softmax*. Fonte: Autor.



2.2.1 CONVOLUÇÃO

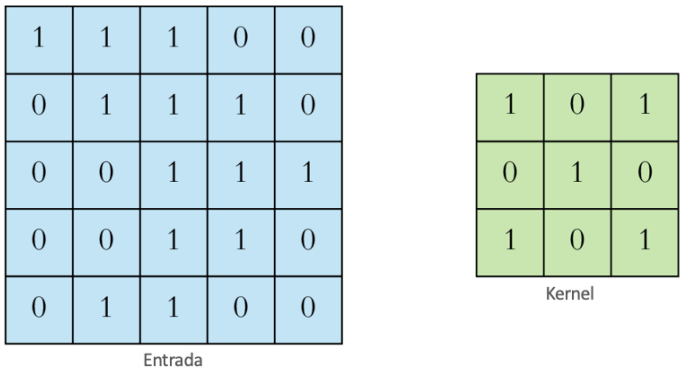
A convolução é uma forma matemática de combinar dois sinais (f e g) para formar um terceiro sinal ($f * g$). É a técnica mais importante em processamento digital de sinais segundo Smith (1997).

O principal componente da CNN é a camada convolucional, que em termos simples, utiliza-se da operação matemática de convolução entre uma matriz e um kernel (que também é

conhecido como filtro de convolução) para mesclar dois conjuntos de informações, onde o kernel é uma matriz de pesos w que será adaptada pelo algoritmo de retropropagação similarmente ao que foi introduzido no capítulo anterior para as RNAs.

Nas CNNs, a convolução é aplicada aos dados de entrada ou em camadas ocultas da rede. A Figura 6 exemplifica uma imagem de entrada à esquerda e o kernel à direita da Figura. O resultado da convolução entre a entrada e o filtro produz um mapa de características.

Figura 6 – Exemplo de entrada 5×5 e kernel 3×3. Este filtro de convolução é chamado de convolução 3×3 devido ao formato do filtro. Fonte: Autor.



Matematicamente, podemos definir a operação de convolução pela equação abaixo (GOODFELLOW; BENGIO; COURVILLE, 2016), onde K é um kernel bidimensional de tamanho $m \times n$, I é a matriz de entrada e (i, j) representa a posição da convolução na imagem de entrada.

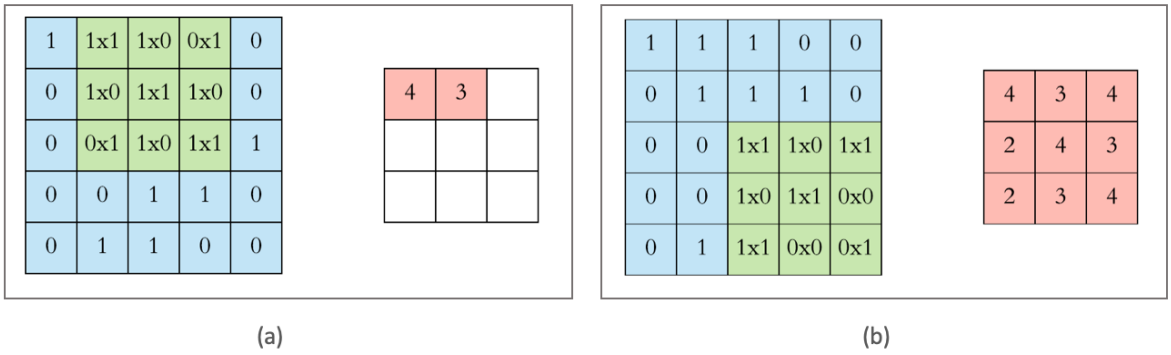
$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{6}$$

A operação de convolução desliza o kernel sobre a matriz de entrada. Em todos os locais, realiza-se a multiplicação de matrizes e soma-se o resultado. Essa soma compõe no mapa de características conforme apresentado na Figura 7 em coloração rosa.

A Figura 7 (a) ilustra a segunda janela deslizante da operação de convolução. Ao lado, a Figura 7 (b), mostra o resultado final após o deslize do kernel sobre todas as posições da imagem de entrada, finalizando com o mapa de características completo em coloração rosa.

Este foi um exemplo de operação de convolução em duas dimensões usando um filtro 3×3. Mas, na realidade, essas convoluções podem ser realizadas em mais dimensões. No caso de imagens, elas são representadas como uma matriz de três dimensões com dimensões de altura, largura e profundidade, onde a profundidade corresponde aos canais de cores RGB. Um filtro de convolução possui altura e largura específicas, como 3×3 ou 5×5 e deve possuir profundidade equivalente à da imagem de entrada.

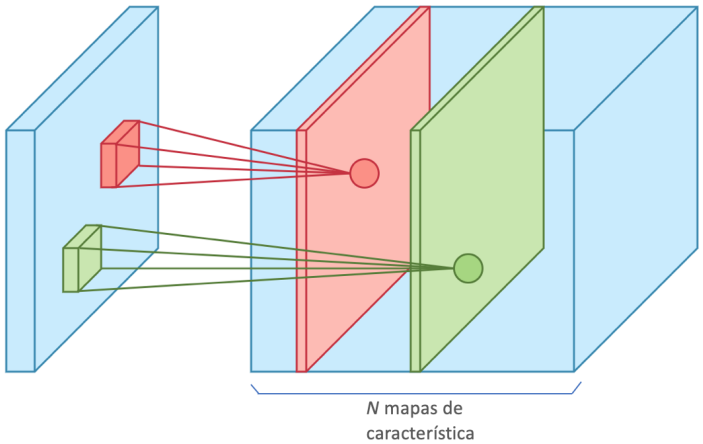
Figura 7 – Exemplo de convolução entre a entrada e o kernel. A Figura (a) ilustra a iteração S(0, 1) conforme a fórmula da convolução e a Figura (b) ilustra a convolução completa. Fonte: Autor.



Em aplicações de aprendizado de máquina, a entrada da CNN geralmente é uma matriz multidimensional de dados e o kernel é uma matriz multidimensional de parâmetros que são adaptados pelo algoritmo de aprendizado (GOODFELLOW; BENGIO; COURVILLE, 2016).

Na prática, cada camada convolucional realiza N convoluções em uma entrada em paralelo, cada uma utilizando um filtro diferente e resultando em um mapa de características distinto, conforme mostra a Figura 8, onde as cores representam os respectivos filtros e mapas de características. Em seguida, empilha-se todos esses mapas de características e isso se torna a saída final da camada de convolução. Por exemplo se utilizarmos N filtros diferentes, obtém-se N mapas de características que são empilhados na dimensão de profundidade.

Figura 8 – Exemplo de dois filtros sendo aplicados ao mesmo tempo e gerando dois mapas de características empilhados, onde as cores representam os respectivos filtros e mapas de características. Fonte: Autor.



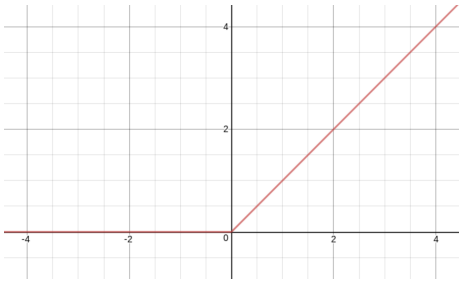
2.2.2 FUNÇÃO DE ATIVAÇÃO

O resultado da operação de convolução geralmente é passado por uma camada de não-linearidade, também conhecidas como funções de ativação. Portanto, os valores nos mapas de características finais não são apenas as convoluções entre a entrada e o kernel, mas a função de ativação aplicada a eles.

Um exemplo popular de função de ativação é a sigmóide, que foi apresentada na Equação 2. Outro exemplo é a Unidade de Retificação Linear (ReLU - *Rectified Linear Unit*), que foi introduzida inicialmente por Richard Hahnloser (2000). A função ReLU altera o sinal de acordo com o comportamento que pode ser observado na Equação 7 e Figura 9. Este estágio é chamado de estágio detector e e funciona transformando entradas menores do que o valor zero, no valor zero. Já para valores maiores do que zero, adota um comportamento linear.

$$f(x) = max(0, x) \tag{7}$$

Figura 9 – Comportamento da função de ativação linear retificada. Fonte: Autor.



Uma variação da ReLU é a ReLU6, que é definida por $f(x) = min(max(0, x), 6)$. Usa-se a ReLU6 por conta de sua robustez quando usado com computação de baixa precisão, segundo (SANDLER et al., 2018).

2.2.3 NORMALIZAÇÃO DE LOTES

Uma técnica opcional que é aplicada após funções de ativação e que pode auxiliar no treinamento das RNAs é a Normalização de Lotes (BN - *Batch Normalization*). Esta técnica permite a utilização de taxas de aprendizado mais altas e também atua como um regularizador, ajudando a prevenir um sobreajuste (*overfitting*) nos dados de treinamento da RNA (IOFFE; SZEGEDY, 2015).

Para um conjunto de ativações de entrada x e uma quantidade de imagens no lote de treinamento da CNN denominado m, calcula-se a Média e Variância do lote definidas nas Equações 8 e 9, respectivamente.

$$\mu \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \tag{8}$$

$$\sigma^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2 \tag{9}$$

Com as Equações 8 e 9 calculadas, podemos definir a Normalização do lote como na Equação 10, onde ϵ representa uma constante para evitar a divisão por zero:

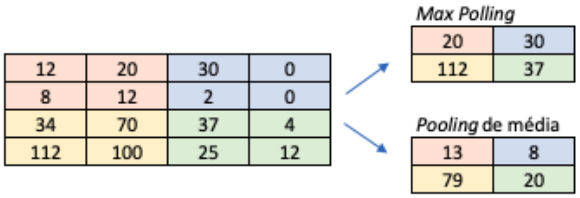
$$\hat{x}_i \leftarrow \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{10}$$

2.2.4 *POOLING*

Após a convolução, a função de ativação e opcionalmente a normalização de lotes, pode-se utilizar uma função de agregação (*pooling*) para executar mais uma modificação na saída da camada de convolução. De acordo com Goodfellow, Bengio e Courville (2016), um dos grandes benefícios do *pooling* é ajudar a tornar a representação aproximadamente invariável a pequenas translações da entrada. Invariância significa que, se transladarmos a entrada por uma pequena quantidade, os valores da maioria das saídas agrupadas não serão alterados.

Uma função de *pooling* basicamente substitui a saída da rede em um determinado local por uma estatística resumida das saídas próximas, o que reduz a dimensão da matriz que será processada pela próxima camada da rede. Por exemplo, o *max pooling* informa a saída máxima dentro de uma vizinhança retangular, conforme exemplificado na Figura 10.

Figura 10 – Exemplo de max pooling e pooling de média. Fonte: Autor.



Outras funções populares de agrupamento incluem a média de uma vizinhança retangular, a norma $L2$ de uma vizinhança retangular ou uma média ponderada baseada na distância do pixel central.

2.2.5 *SOFTMAX*

A camada de *Softmax* geralmente é utilizada como última camada da rede neural. Ela transforma todos os valores calculados pelas camadas anteriores em probabilidades de classes sendo inferidas. Por exemplo no escopo deste trabalho para detecção da bola de futebol e de robôs, temos duas classes e a *softmax* irá indicar a classe mais provável do objeto identificado. *Softmax* é definido formalmente pela Equação 11 abaixo. Com *i* sendo o índice do vetor de saída da CNN e *j* sendo o número total de saídas da rede neural.

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))} \tag{11}$$

Com todos os blocos da CNN introduzidos, agora podemos montar uma CNN que classifica uma ou mais classes com o auxílio da *softmax*.

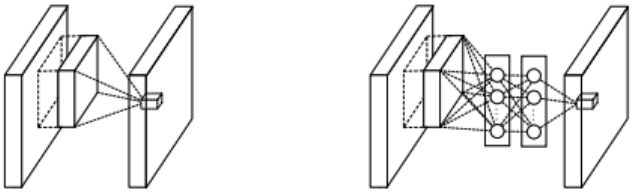
No entanto, quando queremos classificar mais de um objeto dentro de uma mesma imagem, surge o problema que a CNN só possui uma camada de *Softmax*. Sendo assim, devemos utilizar algoritmos que separam e cortam regiões que possuem maior chance de conter um objeto e só depois utilizar a CNN para gerar classificações para estes cortes selecionados. Nas próximas seções serão apresentadas algumas arquiteturas de CNNs e como elas resolvem este problema da detecção de objetos.

2.3 GOOGLENET

A GoogLeNet, desenvolvida por Szegedy et al. (2015), é uma importante arquitetura de CNN de 22 camadas e que foi premiada em primeiro lugar na competição ILSVRC 2014. Algumas das principais características dessa arquitetura são o processamento em várias escalas e a melhor utilização dos recursos de computação dentro da rede. Tais características foram alcançadas devido a um design cuidadosamente elaborado que permite aumentar a profundidade e a largura da rede, prezando pela eficiência computacional.

Um conceito que os autores da GoogLeNet utilizaram foi inspirado por Lin, Chen e Yan (2013), que agrega micro redes neurais *Network In Network* (NIN) com estruturas mais complexas para abstração de dados dentro do campo receptivo das convoluções. O grande benefício vem da redução de dimensionalidade dos dados, principalmente quando aplicado a CNNs. A operação pode ver vista na Figura 11.

Figura 11 – À esquerda a convolução tradicional e à direita a representação da NIN. Fonte: Lin, Chen e Yan (2013)



Szegedy et al. (2015) mostraram o artigo de Lin, Chen e Yan (2013) sob a perspectiva de que essas micro redes neurais na verdade poderiam ser vistas como convoluções 1×1 e a agregaram em sua rede juntamente à ideia de ter convoluções de diferentes escalas em paralelo, conforme ilustrado na Figura 12. À esquerda da Figura, um bloco chamado de *Inception* possui com convoluções 1×1, 3×3, 5×5 e 3×3 com *max-pooling*, seguidos de uma concatenação dos mapas de características desses blocos. À direita da figura, um bloco similar, porém com redução de dimensão a partir das convoluções 1×1.

Figura 12 – Blocos *Inception*. À esquerda um bloco com convoluções 1×1, 3×3, 5×5 e 3×3 com *max-pooling* com a concatenação da saída desses blocos. E à direita um bloco similar, porém com redução de dimensão a partir das convoluções 1×1. Fonte: (SZEGEDY et al., 2015).

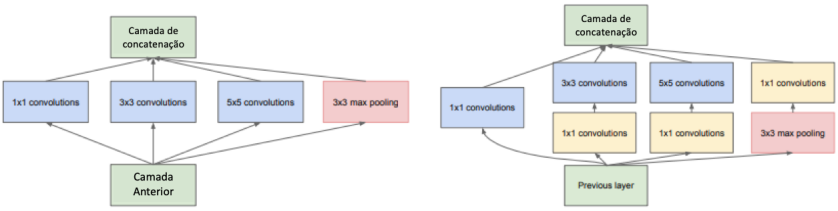


Figura 13 – Arquitetura da GoogLeNet. Fonte: (SZEGEDY et al., 2015).

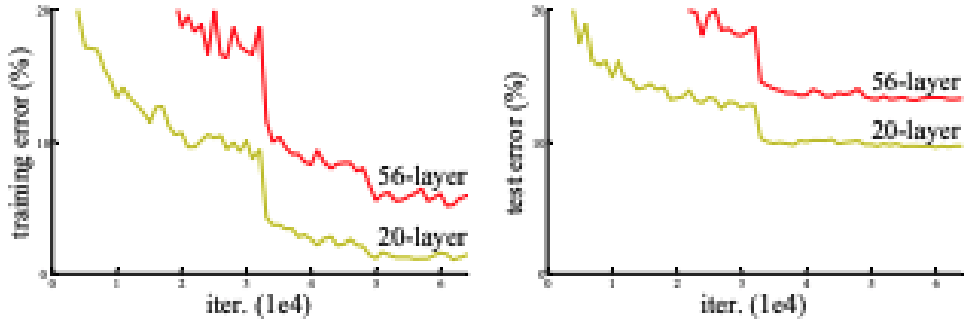
2.4 RESNET

A ResNet, desenvolvida por He et al. (2016), também é uma arquitetura de CNN que possui um histórico de vitórias em competições e uma grande importância para o cenário das redes neurais convolucionais profundas. A ResNet ganhou o primeiro lugar na competição ILSVRC 2015 e COCO 2015.

Motivada por avanços da época em que as melhores CNNs eram cada vez mais profundas com até trinta camadas, He et al. (2016) levantam a questão: "Aprender redes melhores é tão fácil quanto empilhar mais camadas?".

Entretanto, nem sempre é tão fácil e os autores evidenciam um obstáculo ao empilhamento de camada notado em alguns experimentos. Um desses experimentos é apresentado na Figura 14, onde o erro de treinamento (esquerda) e erro de teste (direita) no conjunto de dados CIFAR-10 com redes de 20 e 56 camadas. Ao contrário do que a hipótese de que quanto mais camadas, melhor o desempenho, a rede mais profunda tem maior erro de treinamento e de teste.

Figura 14 – Erro de treinamento (esquerda) e erro de teste (direita) no conjunto de dados CIFAR-10 com redes de 20 e 56 camadas. A rede mais profunda tem maior erro de treinamento e de teste. Fonte:He et al. (2016)



Os principais motivos da rede mais profunda não ter a perfomance esperada é atribuído a dois principais problemas: desaparecimento de gradiente (*vanishing gradient*) e a degradação da acurácia.

O problema do desaparecimento de gradiente é encontrado ao treinar RNAs com métodos de aprendizagem baseados em gradiente e retropropagação. Em tais métodos, cada peso da rede neural recebe uma atualização proporcional à derivada parcial da função de erro em relação ao peso atual em cada iteração de treinamento. O acontece quando, em alguns casos, o gradiente é um valor pequeno e evita que o peso mude de valor (GLOROT; BENGIO, 2010). Este problema, no entanto, foi amplamente estudado e superado por meio de inicialização normali-

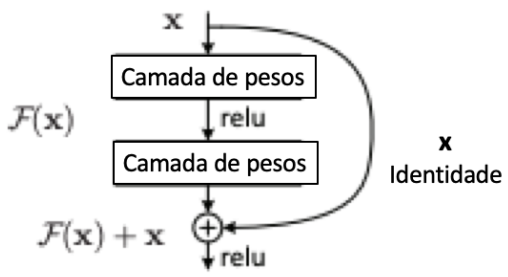
zada dos pesos das camadas, além de camadas de normalização intermediárias (Normalização de lotes, por exemplo).

Já em relação ao problema de degradação da acurácia, que se dá quando redes mais profundas começam a convergir e a precisão fica saturada e então se degrada rapidamente (HE et al., 2016). Esse é um segundo indicativo de que redes profundas não são tão fáceis de serem treinadas.

Considerando duas CNNs, uma com uma arquitetura rasa e a segunda seria uma CNN mais profunda. Existe uma solução por construção para que a acurácia do modelo mais profundo seja no mínimo igual ao modelo raso: as camadas adicionadas são mapeamentos de função identidade, ou seja, apenas replicam as entradas, e as outras camadas são copiadas da rede mais rasa. A existência desta solução construída indica que um modelo mais profundo não deve produzir nenhum erro de treinamento maior do que sua contraparte mais rasa (HE et al., 2016). Entretanto, as arquiteturas conhecidas não possuem esse tipo de função identidade.

Para resolver tal problema conexões residuais de salto ou conexões de atalho foram introduzidas. Conexões residuais de salto permitem que a informação do gradiente passe pelas camadas, criando curto-circuitos de informação, onde a saída de uma camada anterior é adicionada à saída de uma camada mais profunda. Isso permite que as informações das partes anteriores da rede sejam passadas para as partes mais profundas da rede, ajudando a manter a propagação do sinal mesmo em redes mais profundas. As conexões residuais são um componente crítico que permitem o treinamento bem-sucedido de redes neurais mais profundas. A Figura 15 ilustra um bloco com conexões residuais. As camadas de pesos podem ser tanto camadas totalmente conectadas quanto camadas convolucionais.

Figura 15 – Bloco de camadas com conexão residual de salto. Fonte: (HE et al., 2016) adaptado.



Formalmente se gostaríamos que o mapeamento de saída do bloco residual seja $\mathcal{H}(x)$, onde x representa a entrada do bloco, e se hipoteticamente múltiplas camadas não-lineares podem assintoticamente aproximar funções complicadas, então é equivalente a hipotetizar que

as camadas podem aproximar assintoticamente as funções residuais. Portanto, ao invés das camadas mapearem $\mathcal{H}(x)$, elas passam a mapear $\mathcal{F}(x) = \mathcal{H}(x) - x$ (HE et al., 2016).

Uma arquitetura final de exemplo da ResNet com 34 camadas é mostrada na Figura 16. A arquitetura vencedora das competições ILSVRC 2015 e COCO ILSVRC foi uma versão similar estendida da Figura 16 com 152 camadas.

2.5 MOBILENET

Nesta seção, apresentamos uma classe de modelos eficientes chamados MobileNets idealizados no ano de 2017 indicado para aplicativos móveis e de visão incorporados. Este tipo de CNN é baseado em uma arquitetura simplificada que usa convoluções separáveis em profundidade, uma evolução das convoluções que vimos na seção anterior. O objetivo é para construir redes neurais profundas leves, capazes de serem executadas em dispositivos com menor poder de processamento (HOWARD, A. G. et al., 2017).

Convoluções separáveis em profundidade

As MobileNets são baseadas na operação de convolução separável em profundidade, que é uma forma de convolução fatorada. A operação é dividida em dois fatores: (i) uma convolução em profundidade e (ii) uma convolução 1×1, chamada convolução no sentido do ponto.

Uma convolução padrão como vimos anteriormente multiplica a matriz de entrada e o kernel em uma única etapa, por meio de multiplicação de matrizes deslizantes e formando um volume de saída, como pode ser visualizado na Figura 17(a).

Já as Convolução separável em profundidade, como primeiro passo, aplica a convolução em profundidade, ou seja, um filtro a cada dimensão de profundidade da matriz entrada, conforme mostra a Figura 17 (b). Em seguida a convolução no sentido do ponto aplica uma convolução 1×1 para combinar as saídas da convolução em profundidade, como segundo passo. Essa fatoração tem o efeito de reduzir drasticamente o número de operações de multiplicação e adição, como veremos a seguir.

A equação do custo de operações de uma convolução padrão é definida pela Equação 12 (HOWARD, A. G. et al., 2017) abaixo, onde M: Número de canais de entrada da imagem, N:

Figura 16 – Arquitetura ResNet com 34 camadas. Fonte: He et al. (2016)

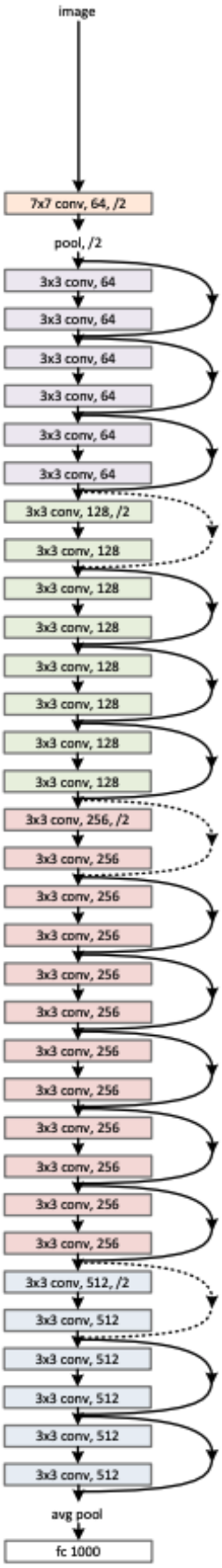
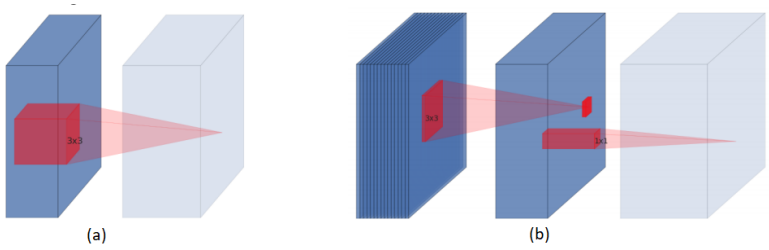


Figura 17 – Comportamento da (a) convolução padrão comparado à (b) convolução fatorada. Fonte: (SANDLER et al., 2018) adaptado.



número de canais de saída, D_k: tamanho do kernel e D_F: tamanho do filtro de características de entrada, e assumindo matrizes quadradas.

$$D_k^2 \cdot M \cdot N \cdot D_F^2 \tag{12}$$

Já a equação do custo de operações de uma convolução separável em profundidade, dividida nos dois fatores, é definida pela Equação 12:

$$D_k^2 \cdot M \cdot D_F^2 + M \cdot N \cdot D_F^2 \tag{13}$$

que é a soma das convoluções em profundidade e da convolução no sentido do ponto. Para mensurar a taxa de ganho da convolução separável em profundidade, realiza-se a divisão, na Equação 15, entre a Equação 13 e a Equação 12:

$$\frac{(D_k^2 \cdot M \cdot D_F^2 + M \cdot N \cdot D_F^2)}{(D_k^2 \cdot M \cdot N \cdot D_F^2)} \tag{14}$$

$$\frac{1}{N} + \frac{1}{D_k^2} \tag{15}$$

Como exemplo, a MobileNet com convoluções separáveis em profundidade 3×3 consome entre 8 a 9 vezes menos computação que as convoluções padrão (HOWARD, A. G. et al., 2017).

Para provar o quão mais rápidas as convoluções em profundidade são, a custo de uma baixa redução na precisão, Andrew G Howard et al. (2017) avaliou o desempenho da sua rede com Convoluções padrão e com Convoluções separáveis em profundidade no conjunto de dados ImageNet.

Com o intuito de facilitar a nomenclatura, neste trabalho denominou-se MobileNetConvencional a CNN implementando a convolução convencional e MobileNetSepProfund a CNN com convolução separável em profundidade. A MobileNetConvencional obteve apenas 1% a

de superioridade em termos de acurácia em relação à MobileNetSepProfund, conforme Tabela 1. A Tabela também mostra a quantidade de multiplicações na ordem de milhões de operações, além da quantidade de parâmetros também na ordem de milhões de parâmetros.

Tabela 1 – Métricas MobileNet Padrão vs Otimizada. Fonte: Autor.

Modelo	Acurácia ImageNet	Multiplicações	Parâmetros
MobileNetConvencional	71.7%	4866	29.3
MobileNetSepProfund	70.6%	569	4.2

A MobileNetSepProfund perde apenas 1% em termos de precisão, mas uma quantidade de multiplicações reduzidas a aproximadamente 10% se comparada a MobileNetConvencional. Além disso, a quantidade de parâmetros foram reduzidos a apenas 14 % da MobileNetConvencional.

No decorrer do trabalho, o termo MobileNet se refere às redes MobileNetSepProfund, que implementam as convoluções separáveis em profundidade.

Arquitetura

A estrutura MobileNet é construída em convoluções separáveis em profundidade, como mencionado na seção anterior, exceto a primeira camada que é uma convolução convencional. A arquitetura original (HOWARD, A. G. et al., 2017) da MobileNet criada para classificar imagens da base ImageNet com 1000 classes é definida na Tabela 2.

Todas as camadas são seguidas por Normalização de Lote e e ReLU, com exceção da camada final totalmente conectada que se conecta a uma camada *Softmax* para classificação de apenas um objeto. Contando com as convoluções em profundidade e em ponto, a MobileNet possui 28 camadas.

Em alguns casos de uso específicos a aplicação da MobileNet pode exigir que a rede seja menor e mais veloz, embora a arquitetura já possua uma estrutura pequena e de baixa latência. Para construir esses componentes menores e que requerem menor poder computacional, um parâmetro α chamado multiplicador de largura pode ser utilizado.

O papel do multiplicador de largura α é de afinar a rede de forma uniforme em todas as camadas. Para um dado multiplicador de largura α, o número de canais de entrada M se torna αM e o número de canais de saída N se torna αN.

Segundo Andrew G Howard et al. (2017), o multiplicador de largura tem o efeito de reduzir o custo computacional e o número de parâmetros quadraticamente por aproximadamente

Tabela 2 – Arquitetura MobileNet. Fonte: Autor.

Tipo	Tamanho Filtro	Tamanho Input
Conv	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32 dw$	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64 dw$	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128 dw$	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128 dw$	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 256$
Conv dw / s1	$3 \times 3 \times 256 dw$	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256 dw$	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5x Conv dw / s1	$3 \times 3 \times 512 dw$	$14 \times 14 \times 512$
5x Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512 dw$	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024 dw$	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

α^2, onde $\alpha \in (0, 1]$. As configurações típicas são 1, 0,75, 0,5 e 0,25. $\alpha = 1$ é o padrão da MobileNet e $\alpha < 1$ é utilizado em MobileNets reduzidas.

O segundo hiperparâmetro para reduzir o custo computacional o custo de uma rede neural é um multiplicador de resolução ρ, onde $\rho \in (0, 1]$.

Em outras palavras, ρ é a constante que altera as dimensões da matriz de entrada. Sendo assim, $\rho = 1$ é o padrão da MobileNet e $\rho < 1$ são MobileNets reduzidas. O multiplicador de resolução tem o efeito de reduzir o custo computacional em ρ^2 (HOWARD, A. G. et al., 2017).

MobileNetV2

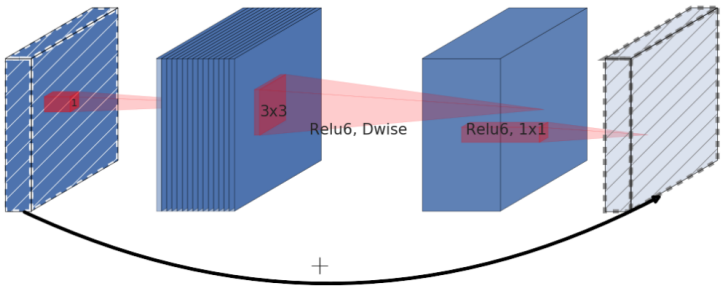
Uma evolução da Mobilenet é a MobilenetV2 criada por (SANDLER et al., 2018), que introduz uma inovação na implementação das camadas convolucionais, a Camada de gargalo linear e resíduos invertidos.

A principal vantagem do gargalo linear é codificar os mapas de características gerados pelas convoluções em subespaços de baixas dimensões. Já a vantagem de ter blocos residuais

é que eles conectam o início e o final de um bloco convolucional com uma conexão direta. Ao conectar esses o início e o fim do bloco, a rede tem a oportunidade de acessar ativações anteriores que não foram modificadas no bloco convolucional. Esses princípios orientam o design da nova camada convolucional Sandler et al. (2018).

A Camada de gargalo linear e resíduos invertidos recebe uma matriz com k canais e executa três etapas convoluções separadas, conforme Figura 18. Primeiro, uma convolução ponto a ponto 1×1 é usada para expandir o mapa de características de entrada para um espaço de dimensão superior. Um fator de expansão t eleva o número de canais a tk nesta primeira etapa. Então, a função de ativação ReLU6 é aplicada.

Figura 18 – Exemplo de Camada de gargalo linear e resíduos invertidos. Fonte: Sandler et al. (2018)



Em seguida, é realizada uma convolução em profundidade usando kernels 3×3. Por fim, o mapa de características é projetado de volta para um subespaço de baixa dimensão usando outra convolução no sentido do ponto e outra função ReLU6.

Como os mapas de características da primeira etapa e da etapa final têm as mesmas dimensões, a conexão residual é adicionada para ajudar no fluxo dos gradientes durante o processo de retropropagação (SANDLER et al., 2018).

Arquitetura MobileNetV2

A arquitetura da MobileNetV2 possui 21 camadas e é detalhada na Tabela 3. Cada linha da tabela descreve uma sequência de 1 ou mais camadas idênticas, repetidas n vezes, conforme a quinta coluna da tabela. Todas as camadas na mesma sequência possuem c de canais de saída, os quais servem como entrada para a próxima camada. O termo *bottleneck* se refere ao bloco de Camada de gargalo linear e resíduos invertidos introduzido na subseção anterior.

Tabela 3 – Arquitetura MobileNetV2. Fonte: Autor.

Operador	Input	t	c	n
Conv2d	$224^2 \times 3$	-	32	1
bottleneck	$112^2 \times 32$	1	16	1
bottleneck	$112^2 \times 16$	6	24	2
bottleneck	$56^2 \times 24$	6	32	3
bottleneck	$28^2 \times 32$	6	64	4
bottleneck	$14^2 \times 64$	6	96	3
bottleneck	$14^2 \times 96$	6	160	3
bottleneck	$224^2 \times 160$	6	320	1
Conv2d	$7^2 \times 320$	-	1280	1
avgPool	$7^2 \times 1280$	-	-	1
conv2d	$1^2 \times 1280$	-	-	-

MobileNetV3

As MobileNets foram desenvolvidas em blocos de construção cada vez mais eficientes. MobileNetV1 introduziu convoluções separáveis em profundidade como um substituto para camadas de convolução tradicionais. Convoluções separáveis em profundidade fatoram a convolução tradicional, separando a filtragem espacial do mecanismo de geração de características.

MobileNetV2 introduziu a camada de gargalo linear e a estrutura residual invertida para construir estruturas de camadas ainda mais eficientes que aproveitam resíduos para recuperar informações e características.

O MnasNet foi construído por Tan et al. (2019), inspirada na estrutura MobileNetV2 introduzindo módulos de atenção baseados em compressão e excitação na estrutura de gargalo.

Para MobileNetV3 (HOWARD, A. et al., 2019), usa-se uma combinação dessas camadas como blocos de construção a fim de construir uma arquitetura mais eficaz. Conforme Figura 19. A estrutura é semelhante à da MobileNetV2, porém inclui mais uma camada intermediária convolucional, operações de *pooling* e e camadas Totalmente Conectadas (FC - *Fully Connected*).

A MobilenetV3 se compara em termos de desempenho no conjunto de dados COCO (*Common Objects in Context*)(LIN et al., 2014) com a MobileNetV2 e a MobileNet original conforme Tabela 4.

A métrica de desempenho é semelhante entre as CNNs, porém com uma clara evolução na latência e na quantidade de parâmetros.

Figura 19 – Bloco de convolução *depthwise* da MobileNetV3 Fonte: Andrew Howard et al. (2019)

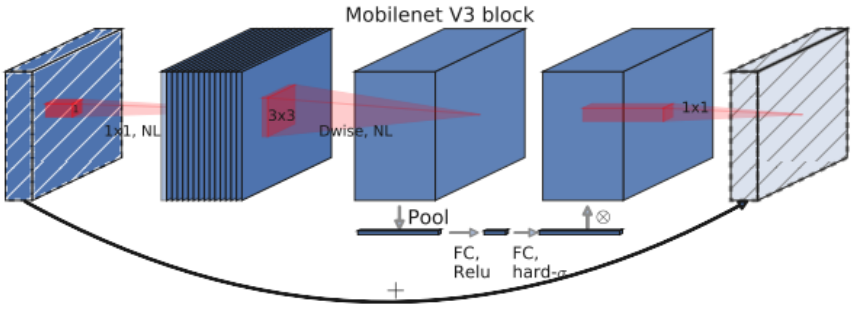


Tabela 4 – Comparativo entre MobileNets. Fonte: Autor.

Arquitetura	mAP	Latência (ms)	Parâmetros (M)
V1	22.2	228	5.1
V2	22.1	162	4.3
V2 0.5	16.6	79	1.54
V3	22.0	119	1.77

Arquitetura MobileNetV3

A arquitetura da MobileNetV3 possui 20 camadas e é detalhada na Tabela 5. Cada linha da tabela descreve uma camada com o tipo de operador com seu respectivo tamanho de kernel. O termo *bneck* se refere ao bloco de convolução da MobileNetV3 introduzido na subseção anterior.

2.5.1 DETECÇÃO DE OBJETOS

Até agora, foram dados exemplos de CNNs aplicadas ao problema de classificação de imagens com apenas um objeto a ser identificado. No contexto de futebol de robôs, objetos diferentes podem estar presentes em qualquer posição da imagem com diferentes proporções ou tamanhos. Por exemplo dois ou mais robôs e uma bola de futebol aparecem frequentemente juntos e em diferentes distâncias. Este tipo de problema não pode ser resolvido apenas com uma camada de *softmax* por conta da camada apenas ter as classes como saída. Neste caso, além das classes necessitamos de coordenadas na imagem dos objetos detectados.

A definição do problema de detecção de objetos é determinar onde os objetos estão localizados em uma determinada imagem e a classificação de qual categoria cada objeto pertence. Portanto, o passo a passo dos modelos tradicionais de detecção de objetos pode ser dividido

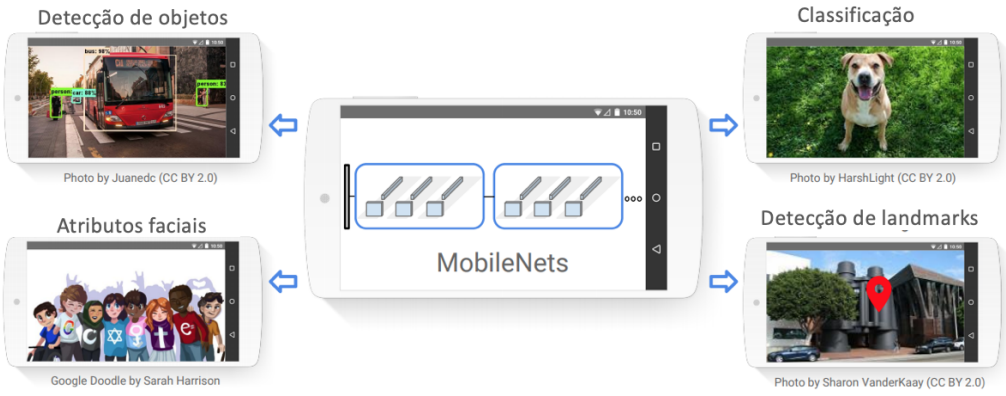
Tabela 5 – Arquitetura MobileNetV3. Fonte: Autor.

Operador	Input
Conv2d	$224^2 \times 3$
bneck, 3×3	$112^2 \times 16$
bneck, 3×3	$112^2 \times 16$
bneck, 3×3	$56^2 \times 24$
bneck, 5×5	$56^2 \times 24$
bneck, 5×5	$28^2 \times 40$
bneck, 5×5	$28^2 \times 40$
bneck, 3×3	$28^2 \times 40$
bneck, 3×3	$14^2 \times 80$
bneck, 3×3	$14^2 \times 80$
bneck, 3×3	$14^2 \times 80$
bneck, 3×3	$14^2 \times 80$
bneck, 3×3	$14^2 \times 112$
bneck, 3×3	$14^2 \times 112$
bneck, 5×5	$7^2 \times 160$
bneck, 5×5	$7^2 \times 160$
conv2d	$7^2 \times 160$
pool	$7^2 \times 960$
conv2d	$1^2 \times 960$
conv2d	$1^2 \times 1280$

principalmente em três etapas: seleção de regiões informativas, extração de características e classificação do objeto (ZHAO et al., 2019).

As MobileNets, ResNets e GoogLeNets podem ser utilizadas para diversas tarefas de visão computacional, como a Classificação de objetos como vimos até a seção anterior. Adicionalmente, pode ser aplicada a Detecção de objetos, a Detecção de *Landmarks* e a Detecção de atributos faciais (HOWARD, A. G. et al., 2017), conforme Figura 20.

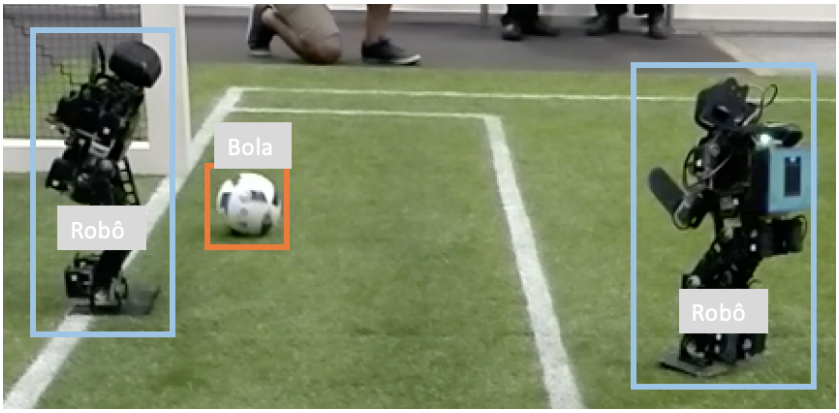
Figura 20 – Diferentes possíveis aplicações da MobileNet. Fonte: (HOWARD, A. G. et al., 2017).



Especificamente para detectar múltiplos objetos em imagens, as MobileNets geralmente utilizam uma abordagem denominada SSD (*Single Shot MultiBox Detector*). Segundo Liu et al. (2016), SSD é tão preciso quanto outras técnicas mais lentas de proposição de regiões de interesse e mais rápido que o algoritmo estado da arte anterior para detectores de disparo único.

Com o intuito de viabilizar o treinamento da detecção de objetos para que a CNN consiga prever coordenadas dos objetos detectados, além de suas classes, é necessário prover imagens anotadas com caixas delimitadoras reais, conforme Figura 21. O formato mais comum de coordenadas foi criado para o conjunto de dados PASCAL VOC (*Visual Object Classes*) e se tornou um padrão (EVERINGHAM et al., 2014).

Figura 21 – Imagem com objetos anotados com suas devidas caixas delimitadoras contendo classe e coordenadas. Fonte: (ROBOCUP, 2019) adaptado.



Para as três etapas de detecção de objetos a técnica SSD utiliza, de forma compartilhada, os mapas de características internos da CNN base. Essa CNN base pode ser uma das MobileNets apresentadas ou outras CNNs profundas. Com esses mapas de características compartilhados, adiciona-se seis outras camadas convolucionais para prever coordenadas e classes em cada segmento da imagem, como será explicado a seguir e pode ser visualizado na Figura 22.

Para a primeira etapa de seleção de regiões informativas, imagens anotadas com caixas delimitadoras para cada objeto a ser detectado são processados pela CNN, conforme Figura 23 (a). De maneira convolucional, avalia-se um conjunto pequeno de por exemplo quatro caixas delimitadoras em diferentes mapas de características da MobileNet. As caixas delimitadoras avaliadas também podem ser parametrizadas para assumirem diferentes proporções, como pode ser visualizado na Figura 23 (b) com escalas 8×8 e Figura 23 (c) com 4×4.

A Figura 23 mostra um exemplo com poucas escalas e caixas delimitadoras. Na realidade, algoritmo padrão da SSD prevê até 8732 possíveis coordenadas de caixas delimitadoras

Figura 22 – SSD acoplada a uma CNN base chamada VGG. Esta rede pode ser substituída pela MobileNet ou outra CNN. Fonte: (LIU et al., 2016)

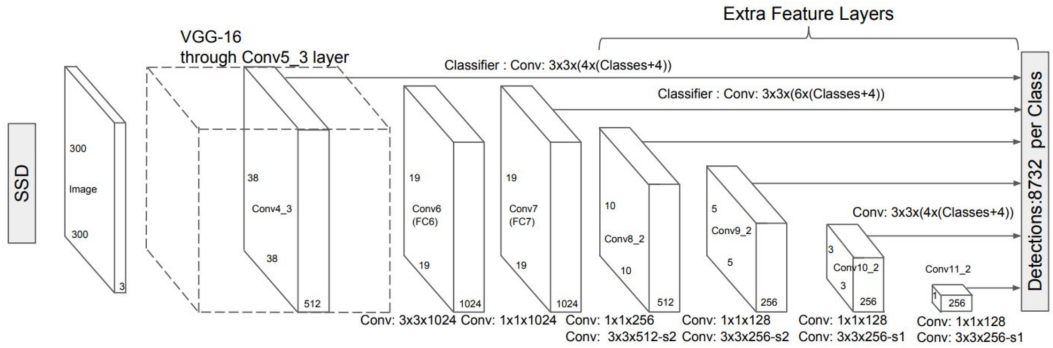
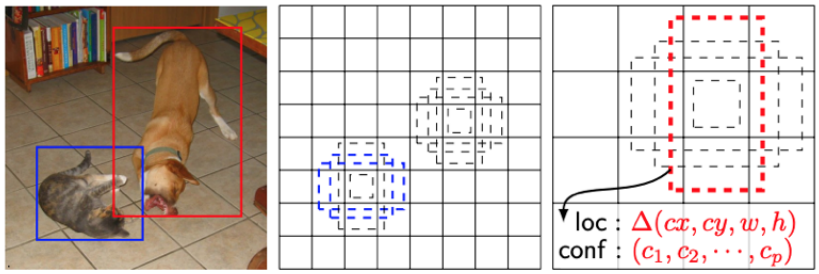


Figura 23 – (a) Imagem de treinamento com caixas delimitadoras em um exemplo de detecção de gatos e cachorros. (b) e (c) mostram caixas delimitadoras avaliadas pela rede em diferentes padrões e escalas. Fonte: (LIU et al., 2016) adaptado.



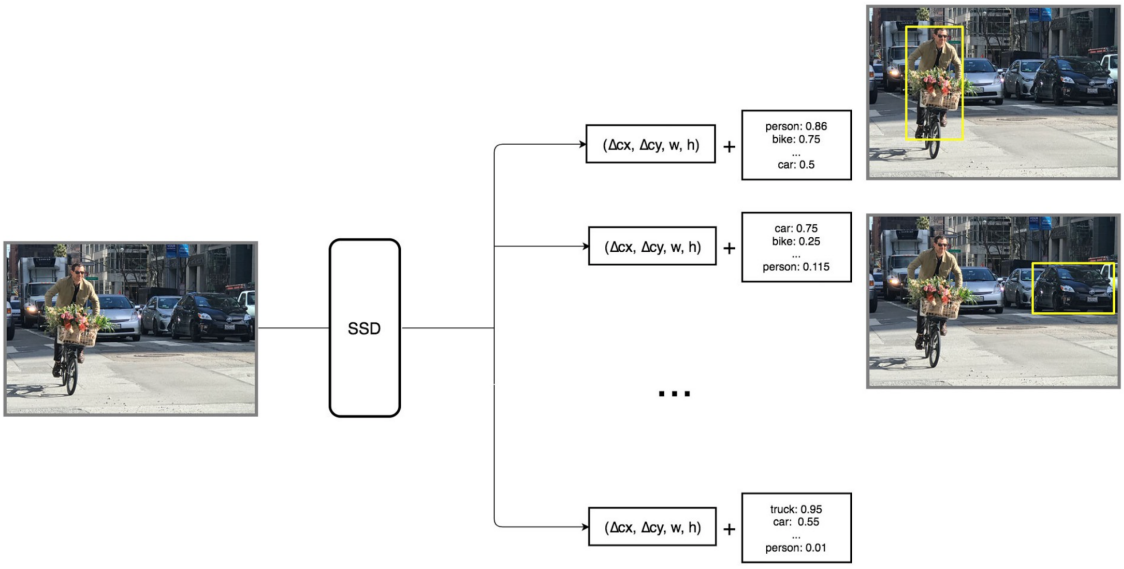
e também os níveis de confiança para todas as categorias de objetos. Avaliando para contexto desta dissertação, teremos uma classe para a bola de futebol, outra para robô e a SSD inclui outra classe para quando nenhum objeto é encontrado. Para sumarizar o processo, a Figura 24 ilustra o processo inteiro de entrada da imagem, separação das regiões de interesse e previsão das localizações e classes.

Para que as previsões sejam as melhores possíveis tanto em termos de localização quanto classificação, a função de custo a ser otimizada pela retropropagação é uma soma ponderada entre os erros de localização e os erros de classificação.

Para compor a função de custo, três caixas delimitadoras precisam ser definidas. Primeiramente existem N caixas delimitadora real anotada, cada uma denotada por: g. O segundo tipo são as caixas delimitadoras (d) propostas pela SSD com largura w e altura h. O terceiro tipo de caixa delimitadora l é estimada pela CNN com desvio c_x e c_y de d para ajuste fino entre o proposto e o estimado.

A parcela de Custo por conta da localização também leva em consideração um componente binário x_{ij}^{i} que indica se houve uma interseção maior que 50% entre d e g.

Figura 24 – Processo da SSD desde a entrada da imagem, separação das regiões de interesse e previsão das localizações e classes. Fonte: (LIU et al., 2016).



$$L_{loc} = \sum_{i \in Pos}^{N} \sum_{m \in c_x, c_y, w, h} x_{ij}^{k} smooth_{L1}(l_i^m - g_j^m) \tag{16}$$

onde $smooth_{L1}$ é definido por:

$$smoothL1 = \begin{cases} 0{,}5x^2 \; se \; |x| < 1 \\ |x| - 0{,}5 \; caso \; contrário \end{cases} \tag{17}$$

Já em relação à parcela de custo por conta da classificação, define-se c como o grau de confiança que a camada de *Softmax* estima para cada classe. Além disso, a classe c^0 refere-se à classe que identifica se nenhum objeto foi encontrado é utilizada para penalizar a função de perda, e c^p se refere à confiança quando existe um objeto, conforme Equação abaixo:

$$L_{conf} = -\sum_{i \in Pos}^{N} x_{ij}^{p} log(c_i^p) - \sum_{i \in Neg}^{N} log(c_i^0) \tag{18}$$

A equação final de custo é ponderada pelas componentes de localização L_{loc} e classificação L_{conf} é dada por:

$$L(x, c, l, g) = \frac{1}{N} \left(L_{conf}(x, c) + L_{loc}(x, l, g) \right) \tag{19}$$

Finalmente após o treinamento da SSD atrelada a uma CNN, as previsões de coordenadas e classes passam por um pós processamento, onde um conceito chamado de Supressão Não-Máxima (*Non-maximum Suppression*, NMS) (BODLA et al., 2017) para remover previ-

sões duplicadas apontando para o mesmo objeto é utilizado, conforme exemplificado na Figura 25.

Figura 25 – Ilustração da entrada e saída do algoritmo NMS. Fonte: (BODLA et al., 2017).

O algoritmo NMS ordena as previsões pelas pontuações de confiança em ordem decrescente. A partir da previsão mais confiante, avalia-se se alguma outra caixa delimitadora possui uma interseção significativa (geralmente maior que 50%) com a previsão atual para a mesma classe. Se encontrada, a previsão com menor confiança será ignorada.

2.6 MÉTODO DE AVALIAÇÃO DE DETECÇÃO DE OBJETOS

Na detecção de objetos a avaliação da saída da CNN não é trivial porque há duas tarefas distintas para medir: (i) se o objeto existe na imagem (classificação) e (ii) a localização do objeto, que é uma tarefa de regressão na imagem que define as coordenadas do objeto, além da largura e altura dentro da imagem.

Para a classificação, podemos medir por meio de duas métricas conhecidas, definidas como Precisão e Revocação, definidas nas Equações 20 e 21 abaixo, onde TP denota as detecções verdadeiras positivas, FN as falsas negativas, FP as falsas positivas e TN as verdadeiras positivas.

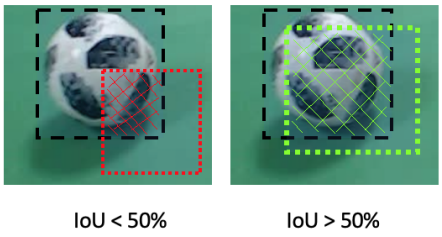
$$Precisão = \frac{TP}{TP + FP} \tag{20}$$

$$Revocação = \frac{TP}{TP + FN} \tag{21}$$

Já para garantir a qualidade na localização do objeto, a saída da CNN é comparada com a caixa delimitadora verdadeira considerando a Interseção sobre União (IoU - *Intersection over Union*), que é ilustrada na Fig. 26 e representada pela Equação 22. Com este conceito, pode-se medir a precisão e revocação com IoU maior ou igual a 50%, por exemplo.

$$IoU = \frac{\text{Área } Interseção}{\text{Área } União} \tag{22}$$

Figura 26 – Exemplo IoU de caixa delimitadora verdadeira e detectada com menos de 50% IoU à esquerda e mais de 50% IoU à direita. Fonte: Autor.



A principal métrica usada em competições de detecção de objetos como COCO (LIN et al., 2014) e PASCAL VOC que agrega os conceitos de precisão, a revocação e IoU é conhecido como Precisão Média (AP).

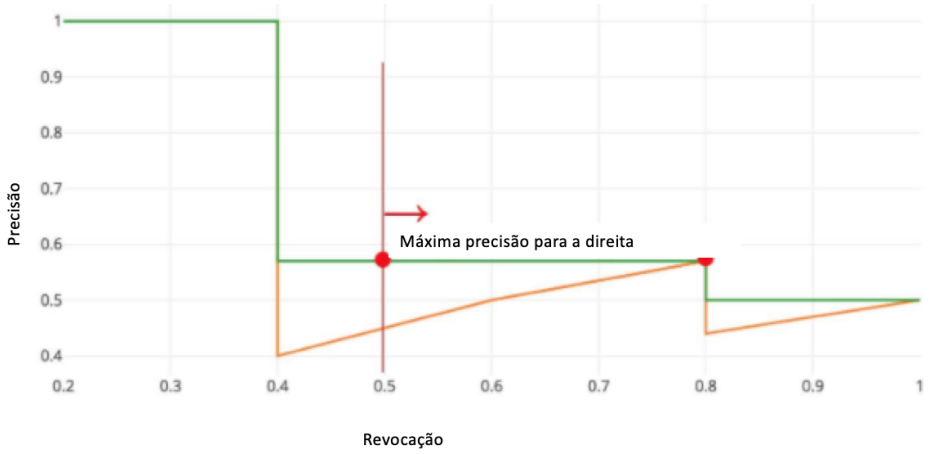
Para uma determinada tarefa e classe, a curva relacionando precisão e revocação (PR) é calculada a partir da saída da CNN considerando diferentes níveis IoU, variando de 50% até 95% com intervalos de 5%, totalizando dez curvas. O exemplo de uma dessas dez curvas PR está ilustrada na Figura 27 em cor laranja. Com essa curva, interpola-se a precisão máxima à direita para todos os pontos de revocação, obtendo assim uma nova curva em verde em coloração verde.

Com a curva PR interpolada, calcula-se a área abaixo da curva usando integração e obtém-se a AP. Ao obter a AP de cada curva de IoU, a AP final pode ser calculada pela Equação 23.

$$AP = \frac{AP_{0.5} + AP_{0.55} + ... + AP_{0.95}}{10} \tag{23}$$

No caso de existirem diversas classes a serem detectadas, a AP final é a média simples entre a AP de cada classe e também pode ser denotada por mAP (*Mean AP*).

Figura 27 – Curva de precisão $\times$ revocação para um limiar de IoU (por exemplo 50%) em laranja e interpolação para cálculo da AP em verde. Fonte: (PADILLA; NETTO; DA SILVA, 2020) adaptado.



2.7 SISTEMA YOLO PARA DETECÇÃO DE OBJETOS

Nesta seção, será explorado o sistema para de detecção de objetos chamada *You Only Look Once* ou YOLO. Segundo os autores Redmon et al. (2016), YOLO é um sistema para processamento e detecção de objetos em tempo real. Para a YOLO, basta processar a imagem apenas uma vez para detectar múltiplos os objetos, conceito semelhante ao da técnica SSD. Este fato de processar apenas uma vez a imagem serviu como fonte de inspiração para o nome da arquitetura, que em uma tradução literal significa "você olha apenas uma vez", e também é um dos motivos pelo qual a YOLO é um modelo que possui baixos tempos de execução.

A YOLO é um sistema que contempla uma CNN para realizar as previsões conhecida como Darknet. A arquitetura dessa CNN é inspirada na GoogLeNet para classificação de imagens e possui 24 camadas convolucionais seguidas por 2 camadas totalmente conectadas. Diferente dos módulos *Inception* usados pelo GoogLeNet, a YOLO simplesmente utiliza camadas de redução 1×1 seguidas por camadas convolucionais 3×3 (REDMON et al., 2016). A arquitetura final pode ser visualizada na Figura 28.

Para a detecção dos objetos, a YOLO divide a imagem de entrada na grade $S \times S$. Por exemplo, a Figura 29 abaixo é dividida em uma grade 5×5 (YOLO em sua primeira versão utiliza $S = 7$). Se o centro de um objeto está em uma das células da grade, essa célula da grade é responsável por detectar esse objeto.

Na sequência do processamento da Figura 29, a YOLO possui como saída da CNN as coordenadas e um grau de confiança para cada uma das $5 \times 5 = 25$ células da grade, que são

Figura 28 – Arquitetura Yolo com suas 24 camadas convolucionais. Fonte: Redmon et al. (2016)

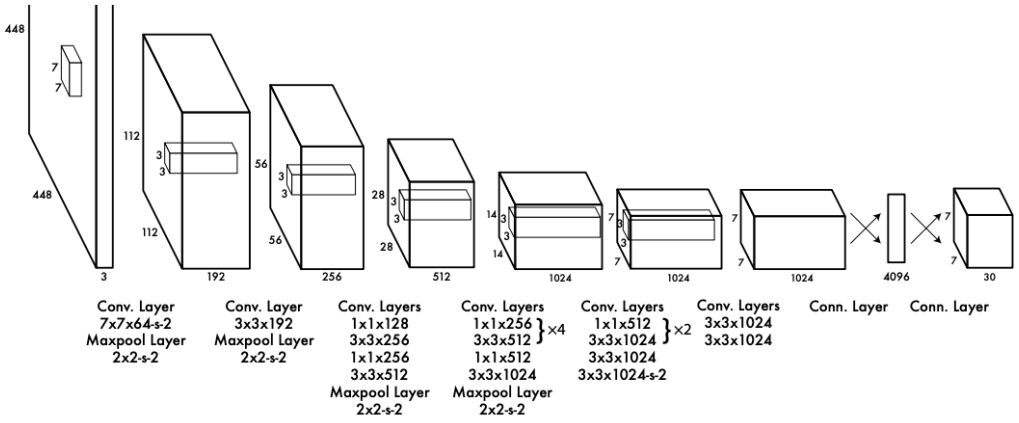
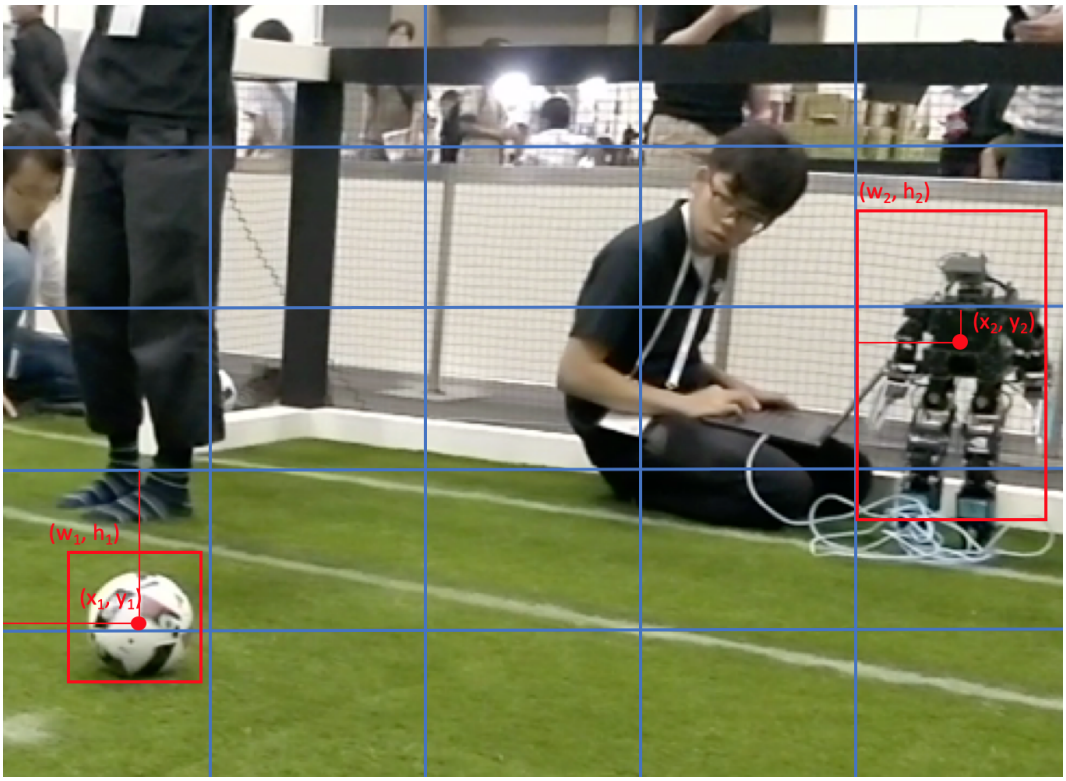


Figura 29 – Imagem de entrada dividida em uma grade 5×5. Fonte: ROBOCUP... (2020) adaptado



equivalentes a tarefas de classificação e localização. Desta forma, pode-se identificar diversos objetos em uma mesma imagem.

Para esse processo acontecer, a CNN estima B caixas delimitadoras para cada uma das células da grade 5×5, e, para cada caixa delimitadora, a CNN emite uma pontuação probabilística de confiança c. Essas pontuações de confiança refletem quão confiante A CNN

está de que a célula contém um objeto. Se nenhum objeto existir na célula, pode-se evitar que o modelo detecte planos de fundo utilizando essa pontuação de confiança.

Além da pontuação de confiança c, o modelo produz mais quatro números $((x, y), w, h)$ por célula. Os primeiros dois, (x, y), são as coordenadas da localização do centro do objeto dentro na célula da grade e são medidas em relativas às dimensões da própria célula, adotando valores entre zero e um. E as dimensões de largura w e altura h da caixa delimitadora prevista são geradas com valores entre zero e altura máxima e zero e a largura máxima da imagem, conforme pode-se observar na Figura 29.

O formato de saída final da rede define-se na Equação 24, onde o valor constante 5 representa as dimensões $(c, (x, y), w, h)$:

$$T_{out} = S \times S \times (B * 5 + Classes) \tag{24}$$

Sendo assim, para $Classes = 20$, $B = 2$ e $S = 7$ por exemplo, a saída da rede tem o formato $7 \times 7 \times 30$.

Com a estrutura da CNN definida, o próximo passo é definir a função de custo para otimizar os pesos da rede. A YOLO utiliza a Soma do Erro Quadrático (SSE - *Error Sum of Squares*) para a função de custo pela facilidade de otimização por gradiente descendente (REDMON et al., 2016). A SSE para a YOLO foi dividida em cinco termos, sendo o primeiro deles apresentado na Equação 25. A equação representa a SSE entre as caixas delimitadoras preditas (x, y) e as caixas delimitadoras verdadeiras $(\hat{x}, \hat{y})$, para cada célula na grade $S \times S$, para cada caixa delimitadora prevista B:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i) + (y_i - \hat{y}_i)] \tag{25}$$

O termo 1_{ij} denota que caixa delimitadora prevista j da célula i é responsável por aquele objeto, caso contrário, 0. Define-se que uma caixa delimitadora é responsável pelo objeto quando ela possui a maior interseção com a caixa delimitadora real dentre todas as possíveis caixas delimitadoras previstas. Já o termo λ_{coord} é uma constante para controlar o peso desta componente da função de custo em relação às outras componentes que veremos a seguir.

A segunda componente da função de custo é similar, porém calculada com relação as dimensões (w, h) das caixas delimitadoras:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \tag{26}$$

O principal motivo dos termos da Equação 26 estarem com o operador de raiz quadrada se dá por conta da medição do erro quando a dimensão da caixa delimitadora é grande ou pequena. Como exemplo, na Tabela 6, compara-se as larguras $w1$ e $w2$ de dois objetos e seus respectivos erros de $0{,}05$, onde mostra-se que o $Erro^2$ final é igual para ambos objetos

Tabela 6 – Tabela com exemplo de larguras de caixas delimitadoras e seus respectivos erros. Fonte: Autor.

	w1	**w2**
$\hat{w}_i$	0,64	0,09
w_i	0,59	0,04
Erro	0,05	0,05
Erro^2	0,25	0,25

Entretanto, quando toma-se as raízes quadradas de $w1$ e $w2$ antes da avaliação do erro, conforme Tabela 7, obtém-se que o $Erro^2_{w1} = 0{,}001024$, enquanto $Erro^2_{w2} = 0{,}01$. Sendo assim, erros de mesmo tamanho $0{,}05$ são relativizados ao tamanho da caixa delimitadora.

Tabela 7 – Tabela com exemplo de larguras de caixas delimitadoras e seus respectivos erros, sendo maior o erro da tabela de menor largura. Fonte: Autor.

	w1	**w2**
$\sqrt{\hat{w}_i}$	0,8	0,3
$\sqrt{w_i}$	0,768	0,2
Erro	0.032	0.1
Erro^2	0.001024	0.01

Os primeiros dois termos anteriores das Equações 25 e 26 representam o custo de localização dos objetos. Os próximos dois termos (Equações 27 e 28) representam o custo de confiança que existe um objeto em determinada célula:

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B} 1_{ij}^{obj}(C_i - \hat{C}_i)^2 \tag{27}$$

$$\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B} 1_{ij}^{noobj}(C_i - \hat{C}_i)^2 \tag{28}$$

onde C representa a confiança de que existe um objeto na célula i e 1_{ij}^{noobj} se não há ou não um objeto dentro da célula i, e que a caixa j para esta célula não é responsável por aquele objeto. Já a constante λ_{coord} representa o peso para este termo da função de custo por conta de que diversas células não contém objetos, então este peso auxilia na convergência do treinamento da CNN.

Por fim, o quinto e último termo é apresentado na Equação 29, a qual representa o custo de classificação dos objetos, e que pode ser traduzido como sendo a soma dos erros das probabilidades estimadas $p_i(c)$ contra a anotação verdadeira $\hat{p}_i(c)$:

$$\lambda_{noobj} \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{29}$$

A função final de custo da YOLO é a soma das Equações 25, 26, 27, 28 e 29:

$$\begin{aligned} F_{custo} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i) + (y_i - \hat{y}_i)] + \\ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] + \\ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \\ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2 + \\ \lambda_{noobj} \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \end{aligned} \tag{30}$$

Na época em que foi desenvolvida em 2016, a primeira versão da YOLO foi considerada extremamente rápida por processar aproximadamente 45 quadros por segundo em GPU, além de atingir o estado da arte em termos de acurária em alguns conjuntos de dados (REDMON et al., 2016). Entretanto, por conta do processamento da imagem utilizando uma grade S, a YOLO enfrenta alguns problemas:

a) Como utiliza-se grade de por exemplo 5×5, e qualquer grade pode detectar apenas um objeto, o número máximo de objetos que o modelo pode detectar é 25.
b) Se uma célula da grade contém mais de um objeto; o modelo não será capaz de detectar todos eles, o que causa um problema de detecção de objetos próximos uns aos outros.
c) O objeto pode se localizar em mais de uma grade (como o robô da imagem Figura 29), de modo que o modelo pode detectar o robô mais de uma vez (em mais de uma grade).
d) A versão comete um número significativo de erros de localização.

Visando corrigir alguns desses problemas, no ano seguinte, em 2017, Redmon e Farhadi (2016) lançaram a segunda versão do YOLO com melhorias de desempenho. Os autores se

concentraram principalmente em melhorar da localização de objetos, mantendo a precisão da classificação. Para alcançar um melhor desempenho, utilizaram as seguintes ideias:

a) Passaram a utilizar um modelo totalmente convolucional e eliminando as camadas totalmente conectadas da saída da CNN.
b) BatchNormalization: A adicão desta camada de normalização de lote em todas as camadas convolucionais no YOLO, gera mais de 2% de melhoria na métrica mAP;
c) Aumento da resolução das imagens de entrada, passando de 224×224 para 448×448;
d) Convolução com caixas de ancoragem substituem as camadas totalmente conectadas. Diferente da primeira versão onde a CNN realizava a previsão do centro (x, y) das caixas delimitadoras baseadas na célula da grade $S \times S$ e (w, y) em relação à altura e largura da imagem inteira, a YoloV2 gera as previsões de largura e altura baseadas em caixas de ancoragem.
e) Variação de dimensões de entrada: a cada 10 lotes, o algoritmo escolhe aleatoriamente um novo tamanho de imagem entre 320×320 e 608×608 pixels. Essa abordagem traz um ganho de mAP em 1,4%.

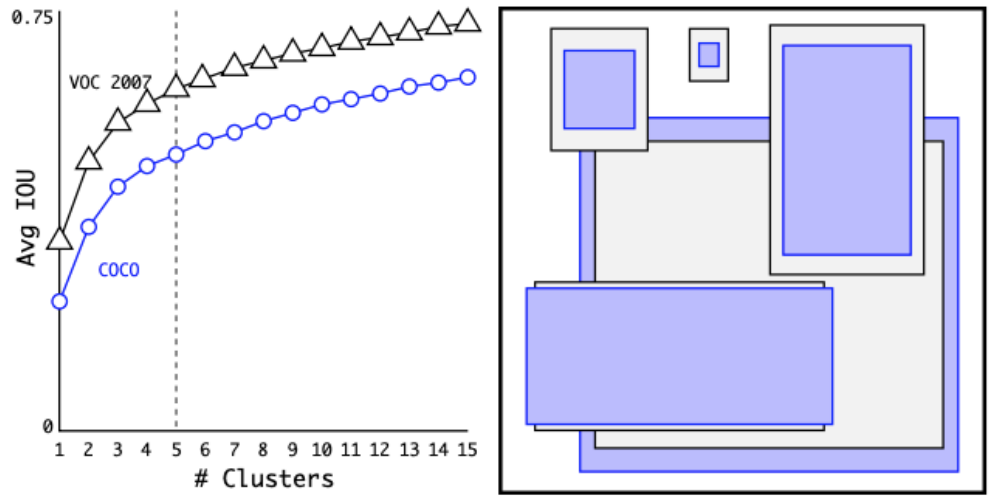
As caixas de ancoragem possuem o conceito similar a caixas delimitadoras avaliadas na técnica SSD. Entretanto, na YOLO, elas são caixas delimitadoras com localização e tamanho definidas pelo algoritmo K-médias (ou *Kmeans*), onde as localizações dos objetos durante o treinamento da rede servem de base para criação dessas caixas âncora, um exemplo de definição de caixas de ancoragem é ilustrado na Figura 30, onde o K-médias é executado para diferentes valores de k e os autores definiram $k = 5$ como uma quantidade de caixas de ancoragem que gera um bom balanço entre complexidade e revocação na identificação de objetos.

Em resumo, na primeira versão da YOLO cada objeto era designado a uma das células da grade $S \times S$, a qual contém o centro do objeto em questão. Com as caixas de ancoragem, cada objeto continua sendo designado para a célula que contem o centro do objeto, mas também é designado a uma caixa de ancoragem pré-definida que possua a maior interseção com a célula designada.

O novo formato de saída final da rede define-se na Equação 31, onde A representa a quantidade de caixas âncora:

$$T_{out} = S \times S \times A \times (5 + Classes) \tag{31}$$

Figura 30 – À esquerda, um gráfico demonstrando a média da interseção total entre as caixas âncora versus k caixas âncora. À direita, as caixas âncora com $k = 5$ para os conjuntos de dados VOC e COCO. Fonte:(REDMON; FARHADI, 2016)



Sendo assim, para $S = 3$, $A = 2$ e $Classes = 2$ e por exemplo, a saída da rede tem o formato $3 \times 3 \times \times 2 \times 7$ ou $3 \times 3 \times 14$. Esse tamanho de saída é capaz de comportar as informações de $(c, (x, y), w, h)$ para cada célula da grade, para cada caixa de ancoragem.

Comparando imagens com tamanhos 416×416, o YOLOv2 atingiu 67 FPS sendo executada em GPU, 22 FPS a mais em relação a primeira versão do algoritmo. E também superou em 13,4% a versão anterior termos de desempenho (mAP).

Porém, já em 2018, a YOLO foi atualizada para a sua terceira versão. Segundo Redmon e Farhadi (2018), além de diversas pequenas melhorias de desempenho, a versão utiliza um novo método para extração de mapas de características. A proposta apresenta a nova arquitetura de CNN, a qual utiliza camadas de filtros 3×3 e 1×1, além de conexões de atalho encontradas nas ResNets (HE et al., 2016), para a extração de características.

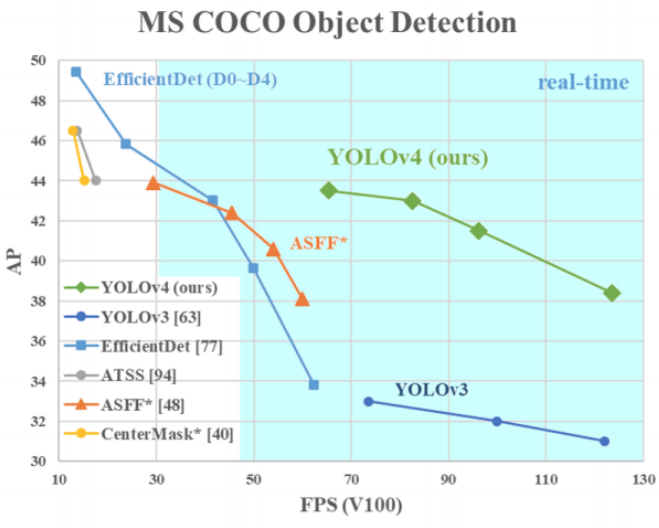
Redmon e Farhadi (2018) mostram nesta versão um aumento na capacidade de detectar objetos pequenos e destacam a velocidade de detecção do algoritmo que chega a ser cerca de três vezes mais rápida em relação ao algoritmo *Single Shot Multibox Detector* - SSD.

A quarta versão da YOLO, desenvolvida por Bochkovskiy, Wang e Liao (2020), foi lançada em abril de 2020. As principais características que podem ser destacadas nessa versão são a melhoria na acurácia e velocidade de inferência. Outra característica importante é que foi otimizada para utilizar menos memória, se tornando mais eficiente para rodar em GPUs.

O YOLOv4 demonstrou ser o melhor detector de objetos para testes em tempo real de acordo com as métricas do COCO (BOCHKOVSKIY; WANG; LIAO, 2020), um conjunto de

dados padrão para comparação de sistemas de detecção de objetos. Um *benchmark* entre as principais redes é apresentado na Figura 31.

Figura 31 – Comparativo entre a YoloV4 e outras arquiteturas de CNN.
Fonte:(BOCHKOVSKIY; WANG; LIAO, 2020)



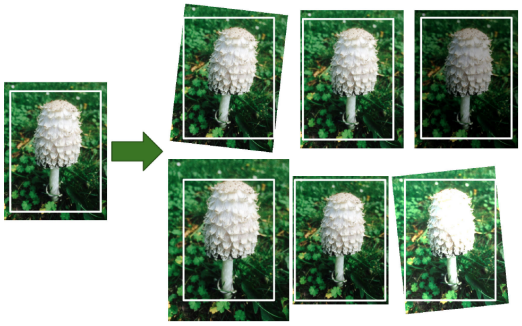
2.8 TRANSFERÊNCIA DE APRENDIZADO

Treinar uma rede neural profunda com pesos inicializados de forma aleatória pode exigir muito tempo de processamento. Uma alternativa é encontrar uma rede neural existente que realize uma tarefa semelhante àquela que se está tentando resolver (D. COOK K. FEUZ, 2013). A partir desta rede já treinada, é possível restaurar os parâmetros e pesos em uma nova rede. Essa é uma forma de reutilizar os pesos das camadas e é chamado de transferência de aprendizado. Isso não só acelera consideravelmente o treinamento, mas também permite o uso de bases de dados reduzidas. A transferência de aprendizado é utilizada principalmente para imagens, e possui algumas boas práticas para gerar bons resultados (GÉRON, 2019):

a) Descartar a camada de saída do modelo original, pois provavelmente não é útil para a nova tarefa, e pode não ter o número certo de saídas para a nova tarefa;
b) Criar uma nova camada de saída com a quantidade certa de classes a serem treinadas;
c) Congelar as camadas da rede, exceto a camada de saída. Ou seja, tornar os pesos não treináveis, para que a descida de gradiente modifique apenas a forma de interpretar os sinais recebidos das camadas de convolução para mapear as novas classes - e treinar o modelo com retro-propagação;
d) Após algumas rodadas de treino, reduzir a taxa de aprendizado e descongelar uma ou duas das camadas ocultas superiores para permitir que a retro propagação faça ajustes finos;

2.9 AUMENTO DE DADOS

Figura 32 – Exemplo de aumento de dados aplicado a imagens utilizando de operações de corte aleatório, rotação, translação, alteração de brilho. As técnicas também podem ser combinadas. (GÉRON, 2019).



Além da transferência de aprendizado, outra técnica bastante difundida no contexto de detecção de objetos é o aumento de dados (*data augmentation*). O principal objetivo é de gerar artificialmente mais imagens para o conjunto de dados de treinamento a partir da criação de variantes realistas de cada imagem de treinamento. Isso reduz o *overfitting*, tornando o aumento de dados uma técnica de regularização (GÉRON, 2019). As instâncias geradas devem ser as mais realistas possíveis, a ponto de um ser humano não ser capaz de dizer se foi aumentado ou não.

Exemplos de aumento de dados são operações de deslocar, girar, espelhar horizontalmente (exceto para texto e outros objetos não simétricos) e redimensionar levemente todas as imagens do conjunto de treinamento e adicionar as imagens resultantes ao conjunto de treinamento, conforme apresentado na Figura 32. Estas operações forçam o modelo a ser mais tolerante a variações na posição, orientação e tamanho dos objetos nas imagens (GÉRON, 2019). Para, por exemplo, aumentar a tolerância do modelo a diferentes condições de iluminação, pode-se gerar imagens com diferentes contrastes. Ao combinar essas transformações, aumenta-se muito a variedade e volume dos de dados de treinamento.

Umas das novidades mais importantes para a YOLOv4 foi a introdução de um novo método de data augmentation chamado de *Mosaic*. Basicamente o método mistura quatro imagens do conjunto de treinamento, como pode ser observado na Figura 33.

Figura 33 – Método de aumento de dados chamado *Mosaic*. Fonte: (BOCHKOVSKIY; WANG; LIAO, 2020).

Neste capítulo foi apresentada uma breve introdução à história das Redes Neurais Artificiais, bem como dos conceitos fundamentais por trás das Redes Neurais Convolucionais e alguns exemplos de arquiteturas, como a GoogLeNet, ResNet, MobileNet, e YOLO. Além disso, foram introduzidos os conceitos da métrica de avaliação de detecção de objetos, transferência de aprendizado e aumento de dados. Todos estes conceitos serão utilizados no desenvolvimento deste trabalho.

3 TRABALHOS CORRELATOS

Nos últimos anos, a detecção de objetos no contexto do futebol de robôs recebeu diversas contribuições de estudos que serão abordados abaixo, visando complementar técnicas e ideias para este trabalho.

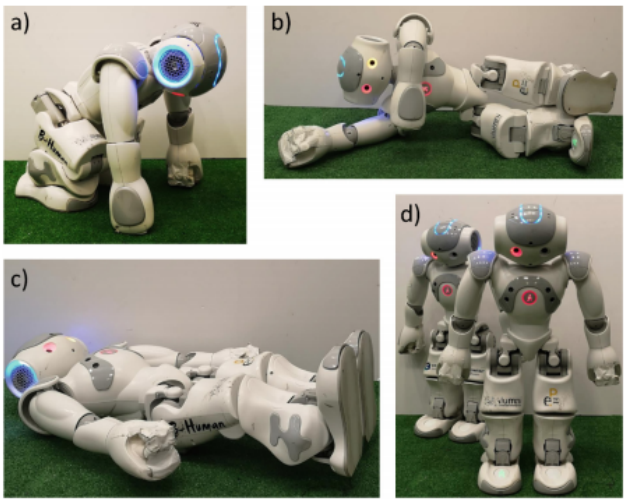
Em recentes estudos, Meneghetti et al. (2020) avaliaram seis diferentes arquiteturas de CNN no contexto de futebol de robôs, apenas para detecção da bola de futebol. A base de dados utilizada possui 4364 imagens com resolução 1920×1080 extraídas de vídeos com modelo de lente de olho de peixe, a qual, segundo os autores, maximiza o campo de visão dos robôs. Como base de teste, apenas 5% da base total de imagens foi utilizada e os resultados estão resumidos na Figura 34. Em termos de mAP, a melhor CNN foi a MobileNetV3 com um multiplicador de largura de $0{,}75$, seguido da MobileNetV2 com resultados similares. Considerando velocidade de inferência e precisão, os autores concluem que a MobileNetV3 possui melhor balanço de resultados.

Figura 34 – Resultados atingidos com o treinamento de seis tipos de CNN com variações de hiperparâmetros. Fonte: (MENEGHETTI et al., 2020)

Network	Width mult.	Input res.	mAP	Inference time (ms) Core i5-4210U	V100	Xeon
MobileNetV2	0.35	96	0.4065	99.964	65.37	64.959
		128	0.7095	101.166	51.062	49.642
		160	0.6304	88.651	52.642	50.97
		192	0.6756	87.006	53.613	52.232
		224	0.4984	**78.553**	42.852	**41.703**
	0.5	96	0.4065	91.403	58.919	57.626
		128	0.6986	81.08	44.529	43.388
		160	0.3361	91.775	58.288	58.616
		192	0.0944	116.076	65.629	64.828
		224	0.3253	**78.624**	42.759	43.18
	0.75	96	0.7284	86.569	51.065	51.528
		128	0.6954	84.159	**42.097**	**41.866**
		160	0.6679	81.351	**41.883**	**42.309**
		192	0.6952	**78.699**	**42.347**	**41.776**
		224	0.7874	85.186	48.343	47.854
	1	96	**0.8133**	122.853	56.992	57.83
		128	0.7672	82.277	46.921	47.799
		160	**0.8597**	88.886	52.278	52.569
		192	0.3632	110.75	61.263	60.052
		224	**0.8177**	79.547	42.438	**42.183**
MobileNetV3 (large min.)	1	224	0.6007	85.808	58.706	59.581
MobileNetV3 (large)	0.75	224	**0.8847**	89.362	63.515	63.703
MobileNetV3 (large)	1	224	0.6875	120.045	88.017	91.369
MobileNetV3 (small min.)	1	224	0.6024	**79.142**	48.68	49.236
MobileNetV3 (small)	0.75	224	0.7067	**60.654**	49.328	47.741
MobileNetV3 (small)	1	224	**0.8651**	96.975	70.689	70.017
TinyYOLOv3			0.3381	588.235	**33.557**	85.47
TinyYOLOv4			0.3504	714.286	**29.851**	119.048
YOLOv3			0.1355	5000	44.248	588.235
YOLOv4			0.1419	5000	50	833.333

Poppinga e Laue (2019) propõem uma nova arquitetura de CNN chamada JET-Net, a qual implementa convoluções inspiradas nas da arquitetura MobileNet. A JET-Net foi construída com o objetivo de detectar outros robôs em tempo real e foi embarcada no hardware de um robô *NAO V5*. Os experimentos foram realizados com a rede sendo treinada com dados dinâmicos e dados estáticos. Nos dados estáticos, o robô foi posicionado em quatro posições distintas conforme apresentado na Figura 35, e estas posições em cinco distâncias distintas (1,5m, 3,0m, 4,5m, 6,0 e 9 metros).

Figura 35 – Ilustração das quatro posições de treinamento para a JET-Net. Fonte: (POPPINGA; LAUE, 2019)

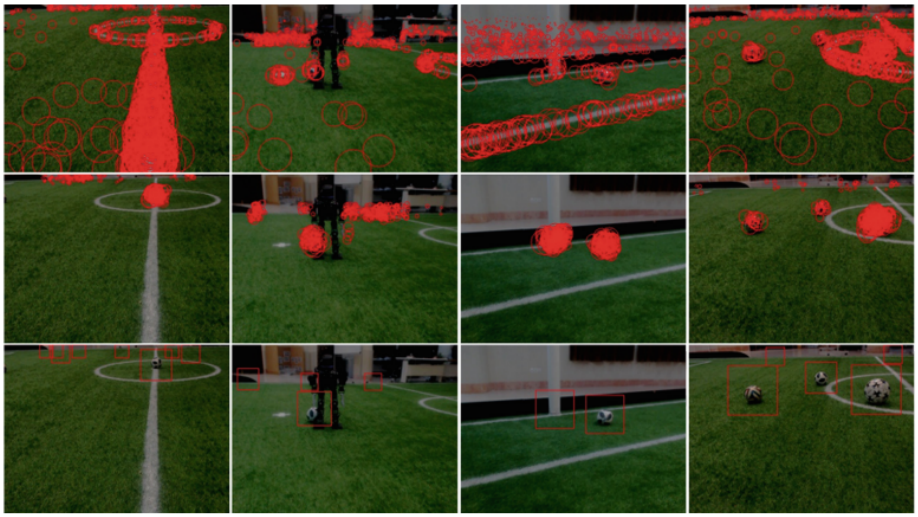


Os resultados mostram que a JET-Net é capaz de executar detecções em apenas $2{,}5ms$ no robô *NAO V6*, permitindo que o robô processe as imagens capturadas a 30 Frames por Segundo (FPS) e atingindo uma acurácia de aproximadamente 96%. Já com dados dinâmicos, que são imagens livres de robôs no ambiente do futebol de robôs adquiridos da plataforma ImageTagger (FIEDLER; BESTMANN; HENDRICH, 2019), o mAP atingido foi de 0,591.

O artigo de Teimouri, Delavaran e Rezaei (2019), propõe um novo método de detecção de bola para robôs humanoides com baixo custo computacional. O método proposto é dividido em duas etapas. Primeiro, algumas regiões de interesse que podem conter a bola de futebol são extraídos usando um método iterativo, conforme ilustrado na Figura 36.

Após a extração de regiões de interesse pelo método iterativo, as regiões extraídas são enviadas a uma pequena rede neural também inspirada nas MobileNets, as quais classificam a região selecionada para propor a caixa delimitadora da bola. Nos testes, o algoritmo detectou as bolas de futebol com uma taxa de acurácia de 91,43%, uma precisão de 89.72% e *5,13*

Figura 36 – Ilustração de propostas de regiões da bola de futebol. Fonte: (TEIMOURI; DELAVARAN; REZAEI, 2019)



ms de tempo de inferência por imagem. Infelizmente o resultado em termos de *mAP* não foi informado.

Analisa-se agora o artigo de Houliston e Chalup (2019), no qual foi desenvolvido um pré-processamento que transforma cada quadro de imagem capturada pelo robô em uma malha visual. A malha visa normalizar o tamanho dos objetos para facilitar o processo de detecção. Isso permite uma estrutura mais simples onde os dados existem como um grafo, contrário a uma grade de *pixels* normalmente utilizada. Os *pixels* geralmente têm nove vizinhos, mas na malha apenas seis elementos são conectados a um elemento central, conforme pode ser observado na Figura 37.

A malha visual é criada a partir da estimativa de um parâmetro $\Delta\phi_n$, representando a inclinação da câmera em relação ao plano do campo de futebol. O outro parâmetro estimado é o $\Delta\theta_n$ que representa o ângulo de anéis concêntricos a partir do ponto ocupado pela da câmera. Devido à perspectiva, o tamanho da bola diminui com a distância conforme pode ser observado nas Figuras 38(a) e 38(c). No entanto, nas Figuras 38(b) e 38(d), após criação da malha, a bola permanece com um tamanho semelhante se compararmos as respectivas figuras.

Por ter uma estrutura de dados mais simples, a malha visual é passada a uma CNN com estrutura modificada. Por exemplo, uma convolução 3×3 em uma CNN típica acessa oito *pixels* em torno de um *pixel* central. A operação equivalente no gráfico acessa pontos com a distância de uma aresta no grafo, totalizando seis vizinhos. Isso tem um impacto positivo no desempenho, pois dois valores a menos precisam ser considerados em cada operação.

Figura 37 – Imagem pré-processada transformada em uma malha visual. Fonte: (HOULISTON; CHALUP, 2019) adaptado.

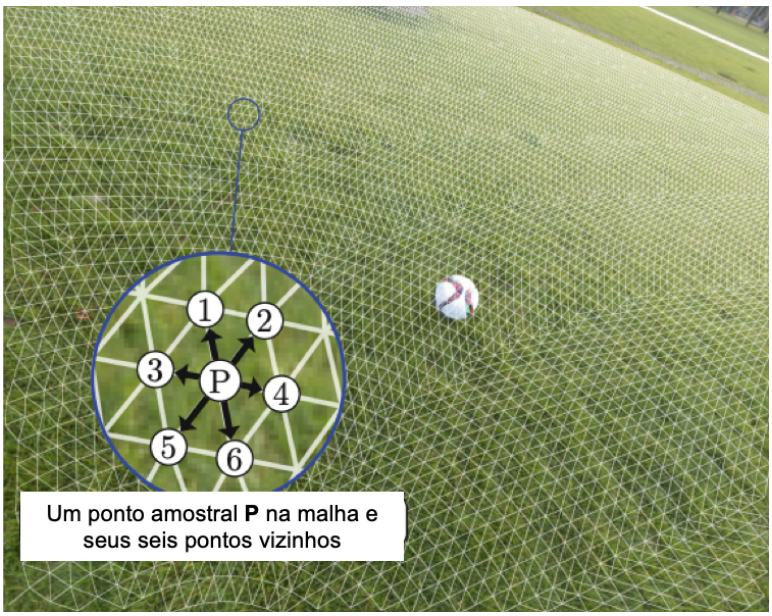
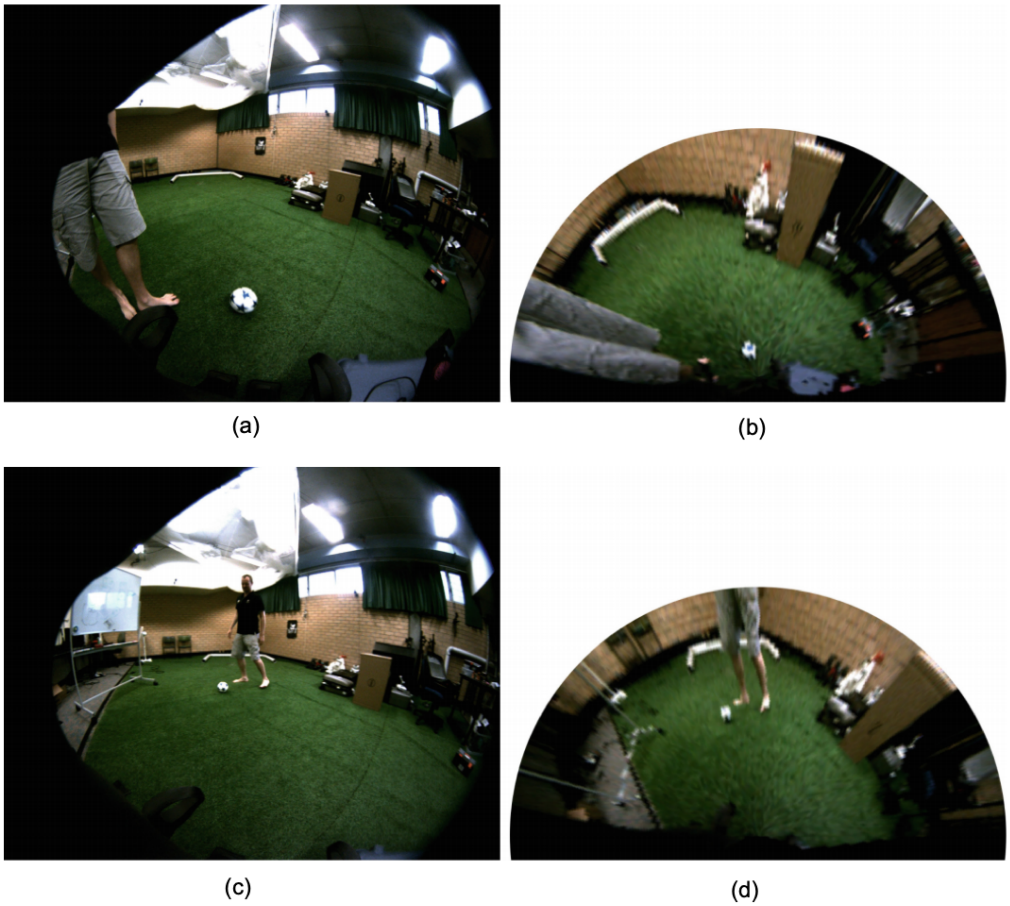


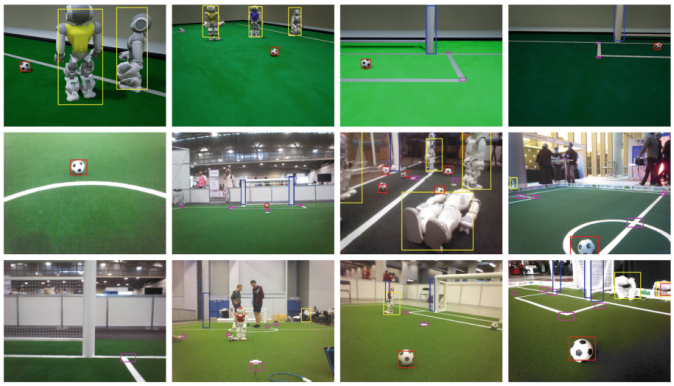
Figura 38 – Imagem pré-processada transformada em uma malha visual vista de uma nova perspectiva. Figura (a) e (c) representam a imagens originais e as Figuras (b) e (d) representam as imagens pré-processadas. Note que (b) e (d) permanecem com tamanhos similares. Fonte: (HOULISTON; CHALUP, 2019) adaptado.



Os autores utilizaram dados sintéticos em conjunto com dados reais para treinar diferentes CNNs como por exemplo a MobileNet. As CNNs foram executadas nas imagens sem pré-processamento, e comparadas com as Visual Mesh, redes que fazem o uso da malha visual avaliadas com diferentes densidades. O objeto de detecção foi a bola de futebol por ser mais facilmente modelada a partir do grafo. A melhor Visual Mesh obteve um tempo médio de execução de $2{,}44ms$ em CPU, comparado com $37ms$ da MobileNet e o resultado foi medido a partir de um gráfico de precisão e revocação, onde a Visual Mesh obteve resultados superiores. No entanto, constatam que as CNNs acusaram uma taxa maior de falsos positivos por conta da maioria dos *pixels* que não era bola de futebol, ser pixels de grama, o que gerou um sobreajuste no treinamento das CNNs.

Em contraste aos autores anteriores, Szemenyei e Estivill-Castro (2019b) apresenta um sistema de detecção de objetos para o robô *NAO v6* capaz de detectar os seguintes objetos para o futebol de robôs: bola, cruzamento de linha, trave e robô, simultaneamente. Para tal tarefa, a arquitetura de detecção de objetos ROBO foi proposta, baseada na arquitetura *Tiny YOLOv3*. O ROBO explora as características do ambiente de futebol de robôs para fornecer uma diminuição considerável (aproximadamente 35 vezes) no tempo de execução comparado ao método *Tiny YOLOv3* enquanto obtém precisão superior atingindo um mAP de 0.56 com 50% de IoU com dados reais. Os melhores resultados das detecções foram para a bola de futebol e a trave do gol, enquanto a classe com menor taxa de detecção foi a de robôs. Alguns exemplos de detecção podem ser visualizados na Figura 39.

Figura 39 – Ilustração de detecções da CNN ROBO. Fonte: (SZEMENYEI; ESTIVILL-CASTRO, 2019b)



Em um conceito mais amplo, (SZEMENYEI; ESTIVILL-CASTRO, 2019a) apresentam uma biblioteca implementada em C++ para rápida inferência em robôs do tipo *Nao*, chamada RoboDNN. Segundo os autores, o objetivo é entender as cenas do futebol de robôs. Esse en-

tendimento acontece pela segmentação semântica da imagem a partir da classificação de cada *pixel* nas seguintes classes: bola, robô, trave e linha de campo. A solução é composta por duas etapas principais: uma CNN para realizar uma segmentação semântica da imagem e outra para propagar rótulos de classes entre a sequência de quadros capturados pelo robô.

A base de dados utilizada para a segmentação semântica possui 5000 imagens sintéticas geradas com variações de 100 diferentes fatores ambientais e 570 imagens reais. Já para a rede de propagação de rótulos, a base possui 100 sequências de quadros que possuem pequenas variações na cena entre eles. Antes do treinamento das redes foi aplicado o conceito de aumento de dados nas duas bases considerando variações de brilho e contraste.

Nos conjuntos de dados, a proporção de *pixels* de fundo é de aproximadamente 93%. Os 7% restantes são distribuídos de maneira desigual entre as classes relevantes. Segundo os autores, esse conjunto de dados desigual geralmente complica a convergência de treinamento da rede e pode resultar em uma rede final fortemente enviesada para cometer erros do tipo falso negativo.

Para superar o desafio do desbalanceamento de *pixels*, uma versão ponderada da função de perda bidimensional de Probabilidade Negativa de Log (NLL), foi implementada. Segundo os autores, esta função incentiva a rede a enfatizar as categorias de objetos mais relevantes.

Foram testadas 7 variações de redes neurais profundas dentro da biblioteca RoboDNN, onde a melhor atingiu uma acurácia de 98%, porém com um tempo de execução de $380ms$.

O conceito de poda de pesos foi utilizado para melhorar o tempo de execução da rede, juntamente a um pós-processamento que permite a execução em até $166ms$, o que significa aproximadamente 6 quadros por segundo. Segundo os autores, esta velocidade de inferência é suficiente para processamento em tempo real.

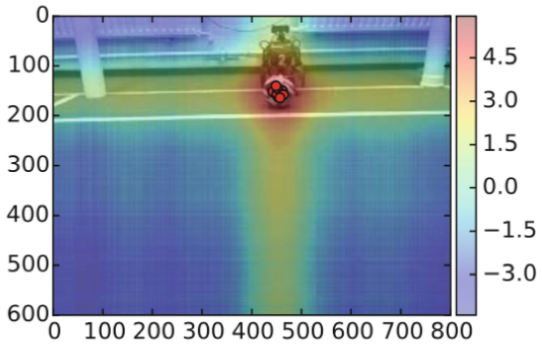
Um exemplo de resultado das previsões geradas por uma das redes da RoboDNN pode ser observado na Figura 40, onde a primeira imagem 40(a) ilustra a imagem original, seguida das imagens segmentadas semanticamente no primeiro quadro 40 (b), após 5 quadros 40 (c) e após 10 quadros 40 (d). Percebe-se que a qualidade da segmentação aumenta com o decorrer dos quadros.

Com um artigo do ano de 2016, mas que deu aos autores (SPECK et al., 2017) o prêmio de Contribuição à Engenharia na *RoboCup 2016: Robot World Cup XX*, a metodologia usada utiliza CNNs para detecção da bola de futebol. Para atacar o desafio de detecção da bola de futebol com uma abordagem inovadora foram desenvolvidas duas arquiteturas de redes neurais convolucionais treinadas com 1160 imagens. As CNNs tinham o objetivo de gerar a previsão

Figura 40 – Ilustração da sequência de classificação da RoboCNN. Imagem original (a) e classificação no primeiro quadro (b), após 5 quadros (c) e após 10 quadros (d). Fonte: (SZEMENYEI; ESTIVILL-CASTRO, 2019a) adaptada.

das coordenadas x e y relativas a uma distribuição normal da posição da bola no respectivo eixo. Desta forma, com a interseção das distribuições dos eixos pode-se gerar uma área mais provável da bola estar localizada, conforme exemplifica a Figura 41.

Figura 41 – Exemplo do mapa de probabilidades de localização da bola. Fonte: (SPECK et al., 2017) adaptado.



Apesar de inovadora e com precisão de 80% nos dados de teste com variações de mais ou menos 10 *pixels* para imagens complexas, os autores (SPECK et al., 2017) concluem que é necessário criar um conjunto de dados maior e mais diverso para que seja possível obter resultados mais acurados. Além disso, o poder computacional dos robôs foi um empecilho para criar redes mais profundas e mais acuradas.

Como método alternativo aos já apresentados, nos quais apenas o quadro atual é usado para a detecção de objetos, (KUKLEVA et al., 2019) desenvolveram um método que faz uso do histórico de quadros como entrada da CNN. O uso do histórico permite rastrear com eficiência a bola em situações em que a bola desaparece ou fica parcialmente oclusa em alguns dos *frames*. Os modelos de aprendizado utilizados foram três redes convolucionais temporais: TCN, ConvLSTM e ConvGRU. Tais redes foram treinadas para aprender mapas de calor de uma bola com base na sequência de quadros representando o histórico de seu movimento. Mais precisamente, se a data e a hora do quadro atual são t, e considerando h como o comprimento do histórico e p

é o comprimento da sequência prevista, a rede recebe como entrada os dados de $(t-h)$ a $(t-1)$ e a saída da rede é a sequência de mapas de calor do registro de data e hora t para $(t+p)$. A melhor rede atingiu 96,7% de acurácia nos dados de teste.

A Tabela 8 abaixo mostra um resumo dos trabalhos estudados e relacionados com detecção de objetos em conjuntos de dados do domínio do futebol de robôs e também outros trabalhos que avaliam o conjunto de dados COCO que possui diversas categorias de objetos.

Tabela 8 – Tabela resumo das principais publicações estudadas e correlatas. Fonte: Autor.

Autor	**Técnica**	**Objetos**
Vilão et al. (2014)	Segmentação, Transformada de Hough, HOG, SVM	Bola, traves e robô
Sanusi, Adiprawita e Mutijarsa (2015)	Transformada de Hough	Traves
Szegedy et al. (2015)	CNN	COCO Dataset
Redmon et al. (2016)	CNN	COCO Dataset
He et al. (2016)	CNN	COCO Dataset
Menashe et al. (2017)	Transformada de Hough, SVM, CNN	Bola
Cruz, Lobos-Tsunekawa e Ruiz-del-Solar (2017)	CNN	Robô
Andrew G Howard et al. (2017)	CNN	COCO Dataset
Redmon e Farhadi (2016)	CNN	COCO Dataset
Felbinger et al. (2019)	Algoritmo genético + CNN	Bola
Gabel et al. (2019)	CNN	Bola
Speck, Bestmann e Barros (2018)	CNN	Bola
Sandler et al. (2018)	CNN	COCO Dataset
Redmon e Farhadi (2018)	CNN	COCO Dataset
Houliston e Chalup (2019)	Visual Mesh + CNN	Bola
Kukleva et al. (2019)	LSTM, CNN	Bola
Poppinga e Laue (2019)	CNN	Robô
Szemenyei e Estivill-Castro (2019b)	CNN	Bola, linhas, trave, robô
Teimouri, Delavaran e Rezaei (2019)	CNN	Bola
Andrew Howard et al. (2019)	CNN	COCO Dataset
Szemenyei e Estivill-Castro (2019a)	CNN	Bola, linhas, trave, robô
Bochkovskiy, Wang e Liao (2020)	CNN	COCO Dataset

O presente trabalho se propõe a avaliar e comparar algumas redes avaliadas nos trabalhos correlatos e que obtiveram bons resultados e acrescentar à comparação a CNN YOLOv4, por ser uma rede recente e com poucos resultados na literatura. Além disso, a comparação de todas as redes em métricas para cada objeto avaliado e avaliação da evolução do treinamento das CNNs.

4 DETECÇÃO DE OBJETOS COM REDES NEURAIS CONVOLUCIONAIS

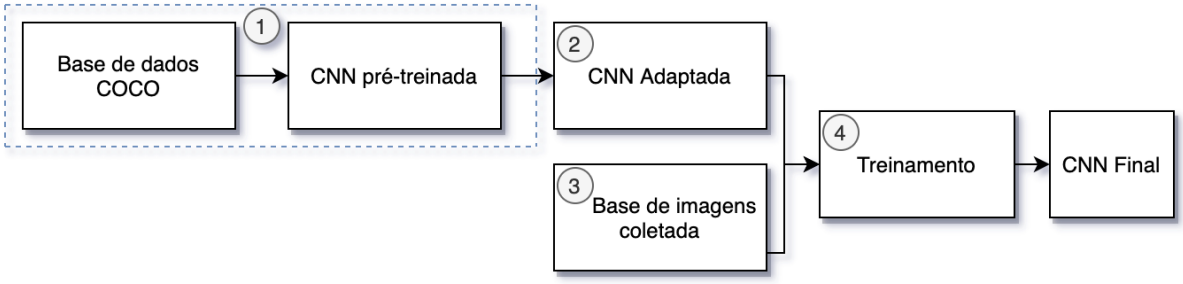
Este trabalho tem o objetivo de aplicar técnicas de aumento de dados e transferência de aprendizado a arquiteturas de CNNs já existentes na literatura, realizando adaptações de arquitetura para o contexto do futebol de robôs humanoides para detectar bolas de futebol e robôs.

Conforme Trabalhos Correlatos no capítulo 3, a classe das MobileNets já é utilizada em hardwares embarcados no contexto do futebol de robôs e traz resultados aplicáveis com um bom desempenho em termos tanto em termos de acurácia quanto em quadros por segundo na detecção dos objetos. Portanto, duas das quatro redes testadas nesta dissertação serão baseadas na arquitetura MobileNet.

A arquitetura de rede YOLOv3 usada por Szemenyei e Estivill-Castro (2019b) possui resultados promissores, porém a nova versão, YOLOv4 (BOCHKOVSKIY; WANG; LIAO, 2020), foi recentemente publicada e foi a escolhida para testes por superar a YOLOv3 nos testes dos autores.

Como complemento, as arquiteturas GoogLeNet e ResNet que foram vitoriosas no desafio ILSVRC no anos de 2014 e 2015, respectivamente, serão avaliadas e comparadas às MobileNets e à YOLOv4. E neste capítulo explora-se a metodologia por trás da adaptação e aplicação das redes mencionadas. Um resumo do passo a passo é apresentado na Figura 42. A mesma metodologia foi aplicada para todas as CNNs.

Figura 42 – Ilustração fluxo de trabalho adotado. Fonte: Autor.



Na Etapa 1 da Figura 42, foi feita a escolha de se utilizar redes pré-treinadas como alternativa a treinar uma CNN nova devido ao conceito de transferência de aprendizado. Como foi introduzido na seção 2.8, essa técnica de transferência de aprendizado se beneficia de uma rede previamente treinada para acelerar o processo de treinamento de uma rede, mesmo que inicialmente a CNN pré-treinada possua fins diferentes.

As MobileNets, ResNet e GoogLeNet foram pré-treinadas por Huang et al. (2017) no conjunto de dados COCO e disponibilizadas para *download* publicamente. A estrutura das CNNs possui a técnica SSD acoplada, o que as torna aptas para a tarefa de detecção de objetos. Já a YOLOv4 foi pré-treinada no mesmo conjunto de dados por Bochkovskiy, Wang e Liao (2020) e também disponibilizada publicamente para *download*.

O conjunto de dados COCO incorpora um grande número de classes, dentre elas, a classe de bolas esportivas, a qual contém 4431 imagens de bolas de futebol, tênis, *baseball*, entre outras. A Figura 43 mostra três exemplos de imagem.

Figura 43 – Imagens diversas da base de dados COCO com bola de baseball, futebol e tênis. Fonte: (LIN et al., 2014).

Em consequência do treinamento prévio das CNNs, os pesos presentes na rede já trazem uma habilidade inerente de detectar objetos redondos em ambientes esportivos com diversos cenários de fundo distintos, com variações de iluminação, brilho de imagem, tamanhos e sobreposições. Esse ponto é de extrema importância pois contribui para a boa generalização das CNNs quando forem testadas em diferentes campos de futebol, evitando que os pesos se adaptem apenas aos cenários de imagens de treinamento da CNN.

Com as CNNs escolhidas e os arquivos delas contendo os pesos pré-treinados para o conjunto de dados COCO, a Etapa 2 da Figura 42 contempla a adaptação dessas CNNs para o contexto da dissertação.

Para tal adaptação, as camadas de saída *Softmax* que realizam a previsão de oitenta classes de objetos na base de dados COCO foram modificadas. A modificação consistiu em remover essas camadas e adicionar outras camadas de *Softmax* com apenas três classes, sendo elas: a bola de futebol, o robô e uma terceira classe para indicar que nenhum objeto foi encontrado.

Desta forma, cada CNN manteve todos os pesos pré-treinados, exceto em sua camada final. Com isso, pode-se executar o processamento direto de imagens de entrada pela CNN, que irá produzir na saída um conjunto de anotações (c_i, s_i, b_i) de rótulos de classe c_i, as quais

podem assumir os valores representando (i) Robô, (ii) Bola de futebol ou (iii) Nenhum objeto. O valor de probabilidade s_i varia entre 0 e 100% e representa a confiança do algoritmo relação a classe c_i. As caixas delimitadoras b_i representam as coordenadas na imagem da classe c_i, que são dadas por (x, y, w, h), onde o par (x, y) representa as coordenadas do centro do objeto no caso da YOLOv4 e o ponto superior esquerdo da caixa delimitadora no caso das outras CNNs. A largura da caixa delimitadora prevista é representada por w e h representa a altura da caixa delimitadora.

Estabelecidas as redes e as suas saídas, as CNNs ainda necessitam da Base de dados com os novos objetos para serem treinadas com esses dados, fazendo com que seus pesos se adaptem ao novo problema. Esses processos são contemplados nas Etapa 3 e 4 da Figura 42.

4.0.1 BASE DE DADOS

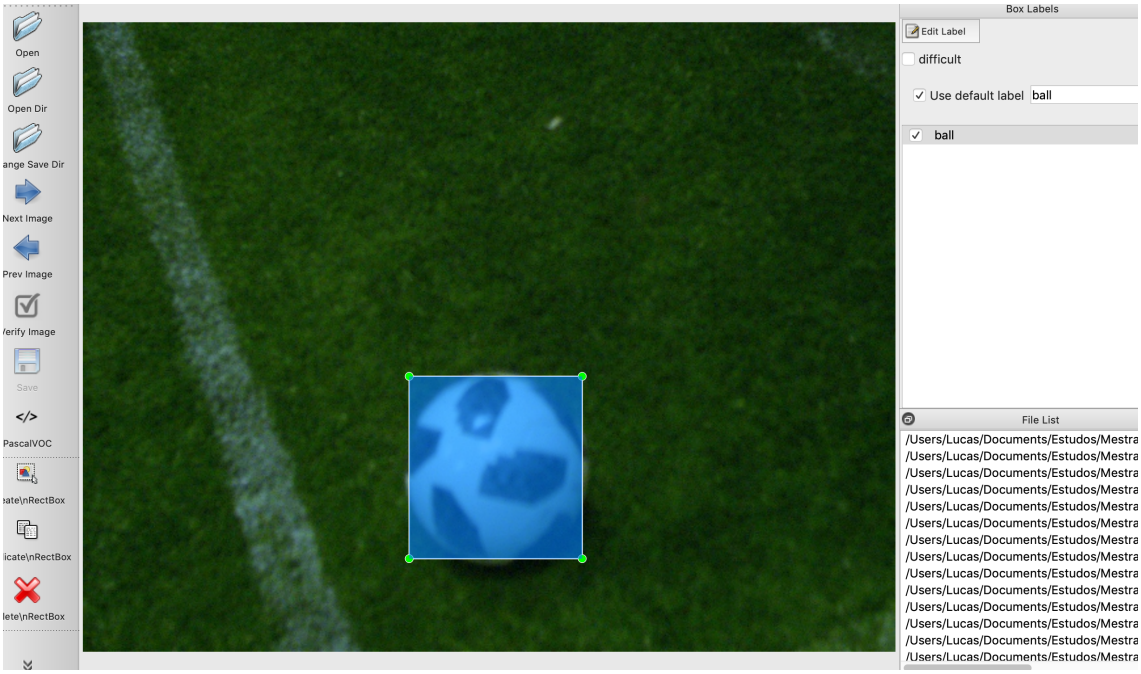
A terceira etapa do fluxo de trabalho adotado consiste em estabelecer uma base de dados para treinamento e teste das CNNs escolhidas. A plataforma ImageTagger (FIEDLER; BESTMANN; HENDRICH, 2019) foi utilizada para coletar imagens variadas de robôs e bolas de futebol em diferentes ambientes e consequentemente contendo variações de iluminação, presença ou não de plateia e diferentes bolas de futebol.

A ImageTagger no mês de setembro do ano de 2020 possui aproximadamente 1200 usuários com 241 conjuntos de dados totalizando aproximadamente 300 mil imagens de diversas situações e categorias de futebol de robôs. Apesar da abundância de imagens, a base carece de imagens com bolas de futebol e robôs em conjunto no mesmo quadro de forma que simule uma imagem de jogo real. A categoria de robô escolhida segundo as regras da RoboCup foi a *Kid Size* por conter maior volume de imagens que satisfaz o critério de imagens reais de jogo e presença de bolas de futebol.

Embora existam diversas imagens já anotadas (ou seja, foram criados arquivos contendo as caixas delimitadoras dos objetos presentes nas imagens) na ImageTagger, todas as imagens da base selecionada foram anotadas novamente, prezando pela qualidade das anotações.

Para criar essas anotações o software gratuito *LabelImg (TZUTALIN, 2020)* foi utilizado, conforme ilustrado na Figura 44. O software salva as anotações em arquivos XML no formato *Pascal VOC* (EVERINGHAM et al., 2014).

Figura 44 – Imagem do processo de anotação de uma das imagens contendo uma bola de futebol no software LabelImg. Fonte: Autor.



Para todas as imagens que continham a bola de futebol, a anotação foi feita conforme ilustrado na Figura 44. Na figura, pode-se observar que a caixa delimitadora cobre a bola por inteiro, deixando o mínimo de bordas da circunferência fora da caixa delimitadora.

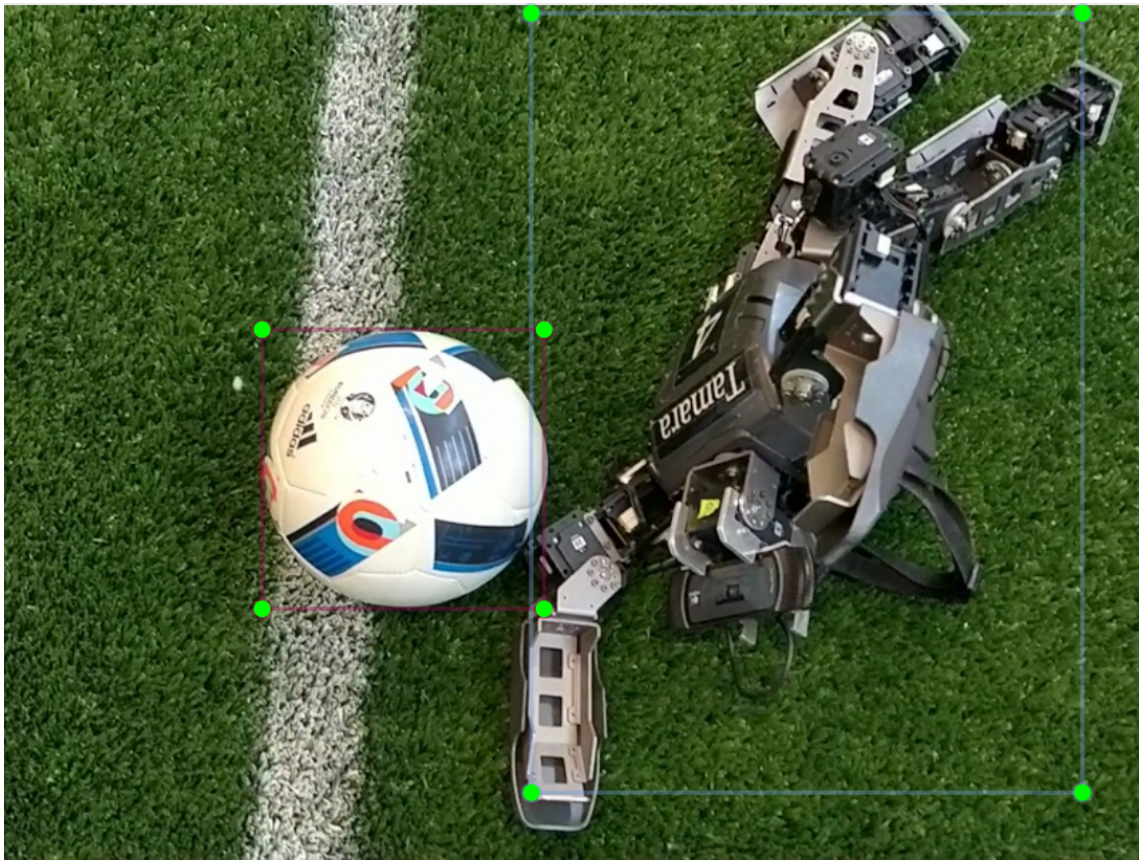
Já para os robôs, a estratégia de anotação foi posicionar a caixa delimitadora em volta de toda a estrutura do robô, mesmo que a quantidade e pixels de grama englobada seja significativa, conforme Figura 45.

Ainda sobre a base de imagens, ela é formada por 738 anotações de objetos sendo que 73% são de bolas de futebol e 27% de robôs, com pelo menos 13% das imagens contendo no mínimo dois objetos. O conjunto de imagens foi separado aleatoriamente em aproximadamente 80% das imagens em um conjunto de imagens de treino e 20% de teste.

Para auxiliar no treinamento e no poder de generalização em novos ambientes das CNNs, foram aplicadas nas imagens de treinamento as seguintes técnicas de aumento de dados em sequência e em combinações entre elas: espelhamento horizontal, cortes de seções aleatórias na imagem, ajuste de brilho e ajuste de contraste.

Scripts na linguagem de programação Python foram desenvolvidos para manipular os arquivos de anotações e adequá-las para a etapa seguinte, o treinamento das CNNs.

Figura 45 – Imagem do processo de anotação de uma das imagens contendo uma bola de futebol e um robô. Fonte: Autor.



4.0.2 TREINAMENTO DAS REDES

A etapa 4 da Figura 42 trata do treinamento das CNNs. Para esse treinamento, a plataforma de aprendizado de máquina de código aberto chamada *Tensorflow* (ABADI et al., 2015) foi utilizada. Contida nesta plataforma existe um módulo para Detecção de Objetos (HUANG et al., 2017) no qual existem CNNs previamente treinadas em diversos conjuntos de dados distintos, dentre as CNNs pré-treinadas estão as MobileNets, ResNet e GoogLeNet. Já para a YOLOv4, o código fonte é *open-source* e foi publicado pelos autores Bochkovskiy, Wang e Liao (2020) na plataforma GitHub, juntamente com a rede pré-treinada.

Como detalhado previamente na etapa 2, as redes escolhidas e pré-treinadas tiveram suas camadas de saída modificadas de modo a fornecerem um vetor com N detecções por imagem, representado por $v_d = [(c_0, s_0, b_0), (c_1, s_1, b_1), ..., (c_N, s_N, b_N)]$.

As CNNs contém hiperparâmetros que podem ser ajustados de acordo com o caso de uso para melhorar a detecção de objetos. A taxa de aprendizado é um hiperparâmetro que auxilia na magnitude com que a CNN atualiza seus parâmetros no processo de retropropagação. Um

baixo valor para a taxa de aprendizagem aumenta o tempo do processo. Por outro lado, um valor alto diminui o tempo do processo de aprendizagem, mas pode pode gerar grandes valores de gradientes e fazer com que a função de custo não atinja mínimos globais.

O hiperparâmetro que define o número de épocas, marca quantas iterações o algoritmo executou a operação de retropropagação por sobre todos os dados de treinamento.

Da mesma forma, é necessário escolher o valor do hiperpâmetro que define a quantidade de exemplos treinados por rodada de uma época, conhecido como tamanho do lote ou *batch size*. Este hiperparâmetro faz com que subconjuntos da base de imagens de treinamento seja processado a cada iteração. O tamanho do lote deve ser ajustado levando em consideração o espaço de memória tanto na GPU quanto na CPU.

O treinamento das redes consiste então em definir estes hiperparâmetros e realizar processamentos diretos (*foward pass*) com a base de dados de treino preparada pela CNN. Com isso, obtém-se o vetor de detecções v_d e compara-se com as anotações reais das imagens v_{real} e calcula-se o erro da rede com a sua respectiva equação de custo, as quais ponderam entre o erro de classificação e de localização das previsões da CNN.

Ao calcular esse custo, o algoritmo de retropropação é aplicado na rede para adaptar os pesos pré-treinados de modo a adaptá-los ao novo problema sendo resolvido.

Os dados de treinamento foram divididos em lotes de acordo com o hiperparâmetro e cada ciclo de *foward pass* e retropropação é definido como um passo ou *step*. Matematicamente, se um lote possui N imagens e o lote é de tamanho N_{lote}, então $\frac{N}{N_{lote}}$ representa um passo. Quando a iteração do treinamento da CNN passa por todos os lotes e consequentemente todas as imagens, diz-se que esse processo completou uma época (*epoch*).

Para treinamento das CNNs, uma máquina virtual da classe *Deep Learning AMI (Ubuntu 16.04) Version 27.0* foi alugada na *Amazon Web services* (AWS). A máquina contém quatro CPUs, uma GPU NVIDIA K80, 64 *gigabytes* de memória e tem custo aproximado de $3 (dólares) por hora utilizada. Mais detalhes para provisionamento e configuração da máquina virtual são descritos no *Anexo A - Configuração de Ambiente em Nuvem*. Além disso, todo o código desenvolvido e base de dados anotada para treinamento das redes está disponível no GitHub publicamente em Abreu (2020b) e Abreu (2020c) para a YOLOv4.

5 EXPERIMENTOS

Conforme apresentado no capítulo anterior, a base de dados foi definida e algumas arquiteturas de CNN foram escolhidas e tiveram suas camadas de saída adaptadas para resolver o problema de detecção de dois objetos no domínio do futebol de robôs humanoides. Neste capítulo, mais detalhes sobre o treinamento dessas redes será detalhado, juntamente com o acompanhamento da função de custo ou erro respectivo às CNNs.

A cada época de treinamento avalia-se a rede sendo treinada com os dados de teste e mensura-se a função de custo para esses dados. Idealmente, as funções de custo tanto de treinamento quanto de teste devem decrescer monotonicamente. No caso da função de treinamento ser decrescente e a função de teste ser crescente, pode ser um indício de que a rede está se adaptando aos dados de treinamento e não está generalizando bem para os dados de teste, e consequentemente errando mais.

A plataforma chamada TensorBoard (TENSORFLOW, 2020) foi utilizada para acompanhar o treinamento das CNNs, monitorando as funções de custo e as previsões dos conjuntos de treino e teste. Todos os experimentos foram monitorados a cada época e assim que a função de custo de treinamento se estabilizou, o treinamento foi parado manualmente.

Adicionalmente aos dados de custo, os experimentos avaliam como forma de desempenho das CNNs a métrica *mAP*, que leva em consideração o conceito de IoU, precisão e revocação e é utilizada em diversas competições e artigos de *benchmark* da área de detecção de objetos.

Alguns testes adicionais foram feitos para avaliar a rede em um vídeo de jogo de futebol com robôs humanoides com 10 segundos para mensurar a taxa de FPS e avaliar as redes em termos de velocidade de execução em CPU. Todos os FPS apresentados nas próximas seções consideram um processador 2.6 GHz Dual-Core Intel Core i5 com 8 GB de memória RAM, em linha com o hardware que geralmente é utilizado em robôs humanoides autônomos.

5.1 EXPERIMENTO 1 - MOBILENETV2

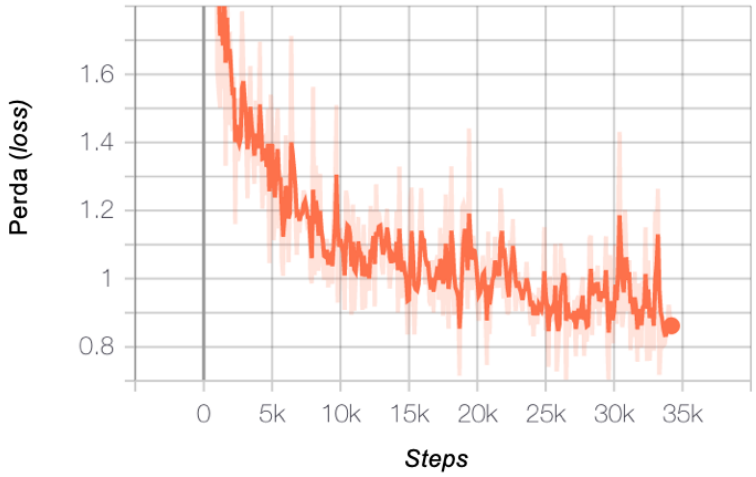
Como configurações para treinamento da MobileNetV2, um tamanho de lote de 64 imagens foi utilizado associado a um otimizador *RMSprop*, similar ao gradiente descendente com taxas de aprendizado adaptativas. A taxa inicial de aprendizado utilizada foi de 0.004, com fator

de decaimento exponencial de 0.95. O tamanho de entrada de imagens foi fixado em 300×300 pixels por ser o padrão utilizado no pré-treinamento da CNN.

Foram executados aproximadamente 34 mil passos (*steps*) na CNN com a base de dados de treinamento, durando aproximadamente 20 horas.

Como o tamanho de lote é de 64 imagens e existem 591 imagens de treinamento, cada $591/64 \approx 9$ *steps* equivalem a uma época. Então calculando $34000/9 \approx 3777$ épocas foram executadas. O gráfico da Figura 46 mostra o resultado da função de perda nos dados de treinamento. Para melhor visualização foi aplicada uma função de suavização no gráfico.

Figura 46 – Custo nos dados de treinamento do Experimento 1. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.



Como boa prática no treinamento de CNNs, o acompanhamento tanto do custo nos dados de treino, visto na Figura 46, quanto nos dados de teste foi realizado. Neste caso, aproximadamente a cada 300 *steps*, a CNN foi avaliada nos dados de teste conforme mostra o gráfico na Figura 47.

Pode-se perceber pela Figura 47 que a avaliação da CNN nos dados de teste tem um padrão de custo sempre decrescente até que se estabiliza. Assim que o custo se estabilizou por muitos passos seguidos, o treinamento foi finalizado manualmente.

Dado que obtivemos bons resultados de perda no treinamento e teste, uma próxima métrica para avaliarmos é o *mAP* com pelo menos 50% de IoU, apresentado nos dados de teste. A Figura 63 mostra a evolução da métrica ao longo dos *steps* até que estabiliza em ≈ 0.9953, sendo esse valor dividido em 0.9773 para a bola de futebol e 0.9546 para os robôs.

Figura 47 – Custo nos dados de teste do Experimento 1. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.

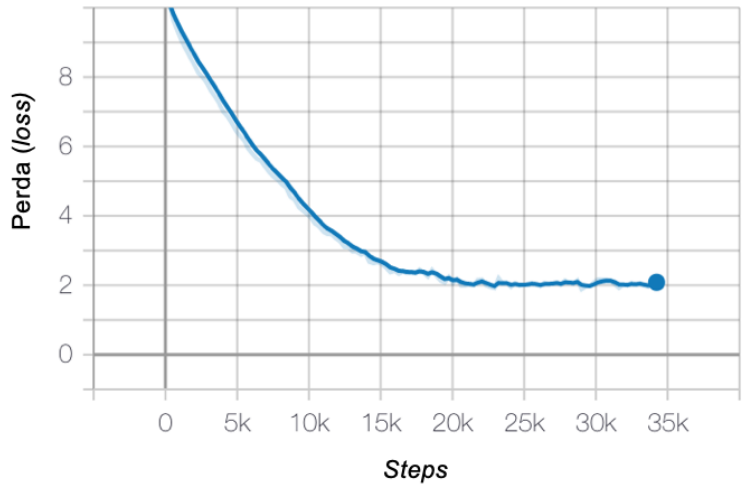
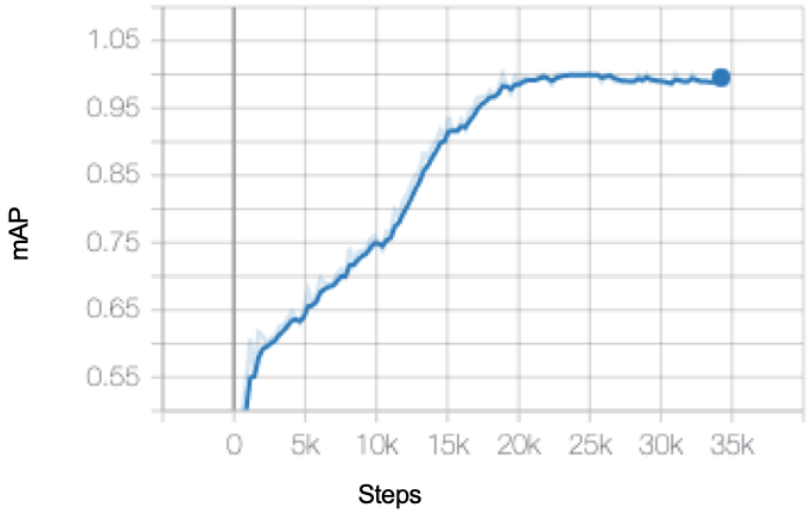


Figura 48 – Métrica *mAP* com 50% de IoU nos dados de teste do Experimento 1. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.



Nas configurações do treinamento do modelo foi estabelecido o parâmetro de avaliar 40 imagens dos dados de treinamento. Ao longo das épocas, essas imagens foram avaliadas e podem ser visualizadas e comparadas com as caixas delimitadoras verdadeiras no Tensorboard, conforme Figura 49 mostra avaliação no *step* 6075.

Também por meio da interface do Tensorboard é possível acompanhar a evolução da CNN ao longo dos *steps*. A Figura 50 ilustra essa evolução para a MobileNetV2.

Como complemento às métricas e avaliação nos dados de teste, a CNN foi executada em alguns vídeos que não estavam presentes nem nos dados de treino e nem nos dados de teste.

Figura 49 – Interface da avaliação do treinamento das CNNs (Tensorboard). É possível observar quatro imagens, onde as imagens da esquerda ilustram previsões da CNN e na direita as imagens reais que foram anotadas. Fonte: Autor.

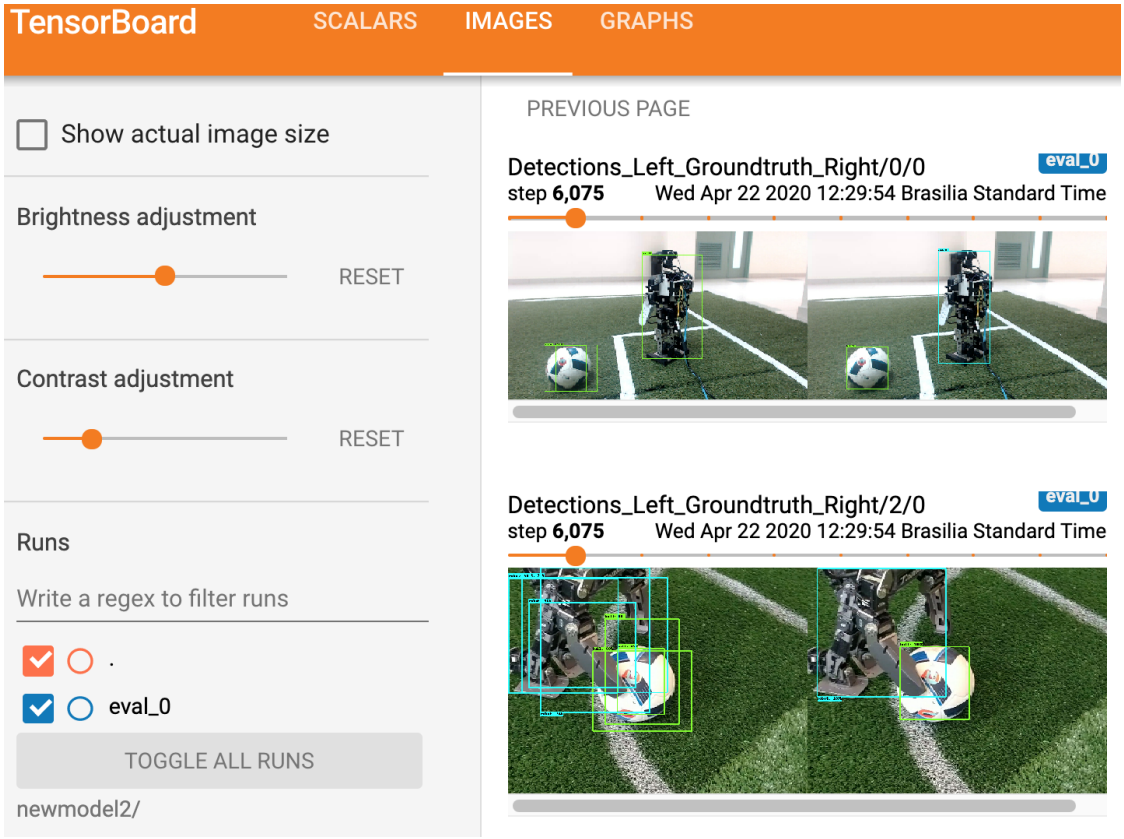
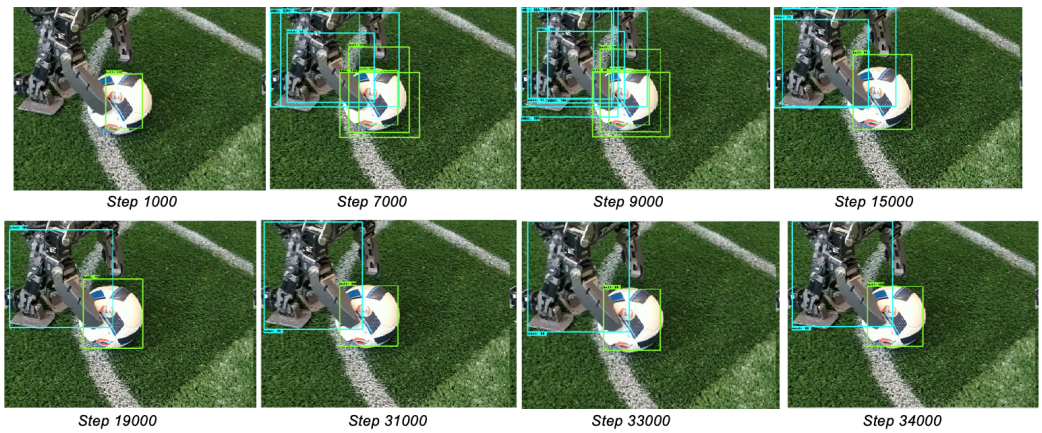
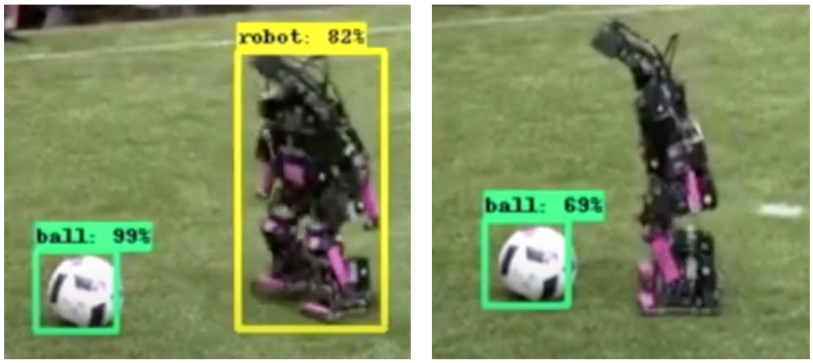


Figura 50 – Evolução das detecções da MobileNetV2. No *step* 1000 apenas uma parte da bola foi detectada. Já nos *steps* 7000 ao 15000, diversas previsões são geradas para os mesmos objetos, até que a partir do *step* 19000 as previsões de estabilizam. Fonte: Autor.



Um compilado desses vídeos pode ser visto em (ABREU, 2020a). Aos 13 segundos do vídeo, apresentado aqui na Figura 51, é possível perceber que quando o robô se posiciona lateralmente na imagem a CNN deixa de identificar o robô. A detecção na posição frontal, na Figura 51(a), acontece de forma correta. Para estes vídeos, o desempenho da CNN em termos de FPS médio foi de ≈ 14.0.

Figura 51 – Comparativo da (a) detecção frontal e (b) não-detecção lateral do robô. Fonte: Autor.

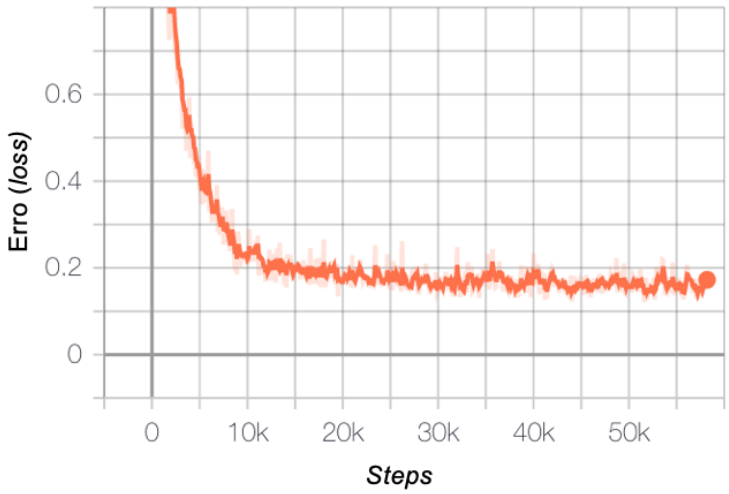


(a) (b)

5.2 EXPERIMENTO 2 - MOBILENETV3

Já para treinar a MobileNetV3, foi utilizado um tamanho de lote de 48 imagens e o otimizador *Momentum*, baseado no gradiente descendente, com uma taxa inicial de aprendizado de 0.4. Foram executados aproximadamente 58 mil passos (*steps*), e cada 12 *steps* aproximadamente equivalem a uma época. Sendo assim, calculando $58000/12 \approx 4833$ épocas foram executadas, durando aproximadamente 10 horas. O gráfico na Figura 56 mostra o custo nos dados de treinamento. O tamanho de entrada das imagens foi fixo em 320×320 por ser o padrão utilizado no pré-treinamento da CNN, assim como os parâmetros anteriores.

Figura 52 – Evolução do treinamento da rede no Experimento 2. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.



A CNN também foi avaliada nos dados de teste até que a curva se estabilizou e o treinamento foi finalizado manualmente, conforme mostra o gráfico na Figura 57.

Com o treinamento feito e curvas de erro decrescentes conforme esperado, a próxima métrica para avaliarmos é o mAP apresentado nos dados de teste. A Figura 63 mostra a evolução da métrica ao longo dos *steps* até que estabiliza em ≈ 0.9803, sendo dividido em 0.9975 para a bola de futebol e 0.9631 para o robô.

Para acompanhar a evolução da CNN ao longo dos steps a Figura 55 foi selecionada pois ilustra a evolução para a MobileNetV3 que finaliza, no último passo, com a não localização da bola de futebol.

Já para os vídeos, o desempenho da CNN em termos de FPS médio foi de ≈ 14.7

Figura 53 – *Loss* nos dados de teste do Experimento 2. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.

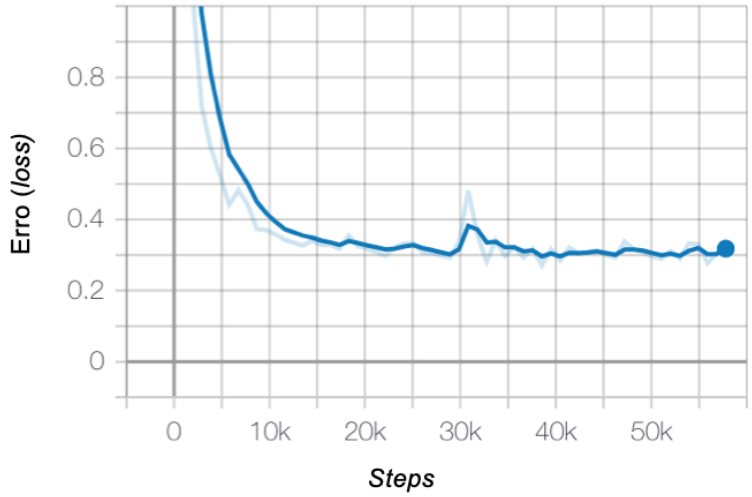


Figura 54 – Métrica *mAP* com 50% de IoU nos dados de teste do Experimento 1. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.

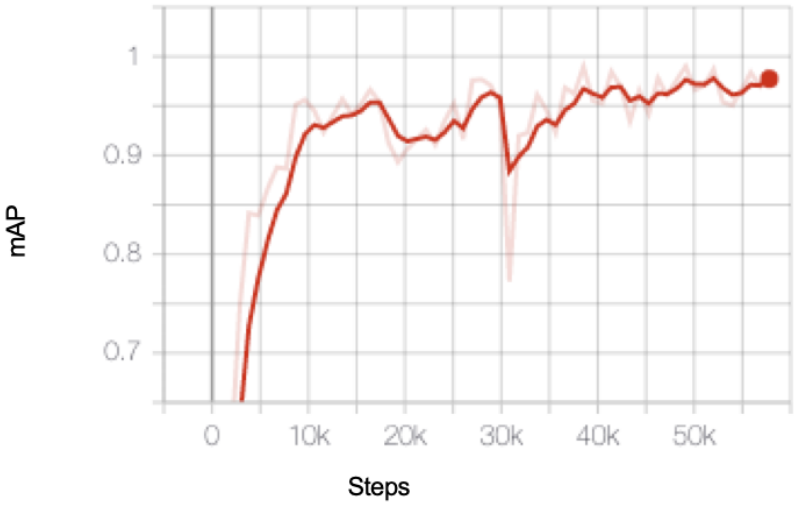
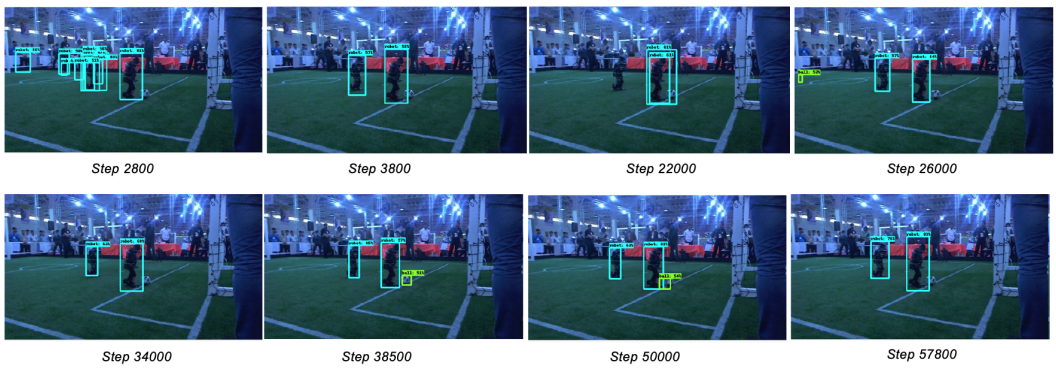


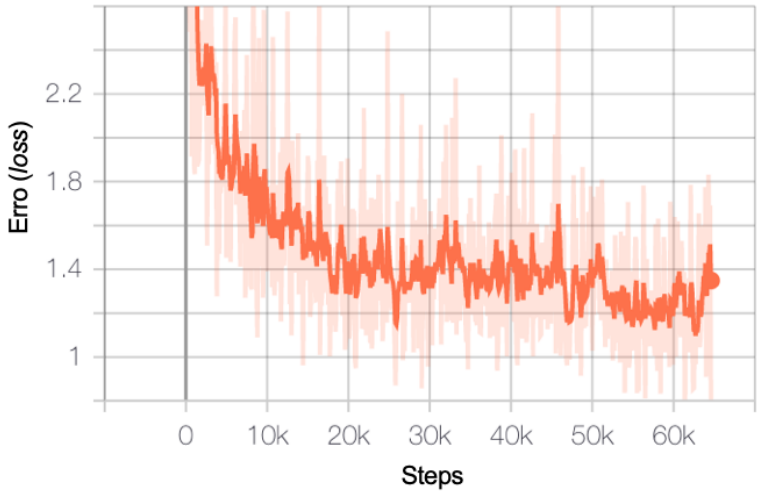
Figura 55 – Evolução das detecções da MobileNetV3 ao longo do treinamento. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.



5.3 EXPERIMENTO 3 - GOOGLENET

Para treinar a GoogLeNet, foi utilizado um tamanho de lote de 12 imagens e o otimizador *RMSprop.* com uma taxa inicial de aprendizado de 0.004 e fator de decaimento de 0.95. Foram executados aproximadamente 53 mil passos (*steps*) e a cada 50 *steps* aproximadamente equivalem a uma época. Sendo assim, calculando $53000/50 \approx 1060$ épocas foram executadas, durando aproximadamente 9 horas. O gráfico na Figura 56 apresenta o erro para os dados de treinamento. O tamanho de entrada de imagens foi fixo em 300×300 por ser o padrão utilizado no pré-treinamento da CNN.

Figura 56 – Evolução do treinamento da rede no Experimento 3. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.



A CNN também foi avaliada nos dados de teste até que a curva se estabilizou e o treinamento foi finalizado manualmente, conforme mostra o gráfico na Figura 57.

Para acompanhar a evolução da CNN ao longo dos steps as componentes de localização e classificação é ilustrada na Figura 62.

Com o treinamento feito e curvas de erro decrescentes conforme esperado, a próxima métrica para avaliarmos é o mAP apresentado nos dados de teste. A Figura 63 mostra a evolução da métrica tanto para a detecção do robô quanto para a bola de futebol ao longo dos *steps* até que estabiliza em ≈ 0.948 para a bola de futebol e ≈ 0.8503 para a bola de futebol. O mAP final foi de 0.8995.

Por fim, em termos de FPS médio, o valor medido foi de ≈ 7.2 para esta CNN.

Figura 57 – Acompanhamento da função erro da CNN para os dados de teste do Experimento 3. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.

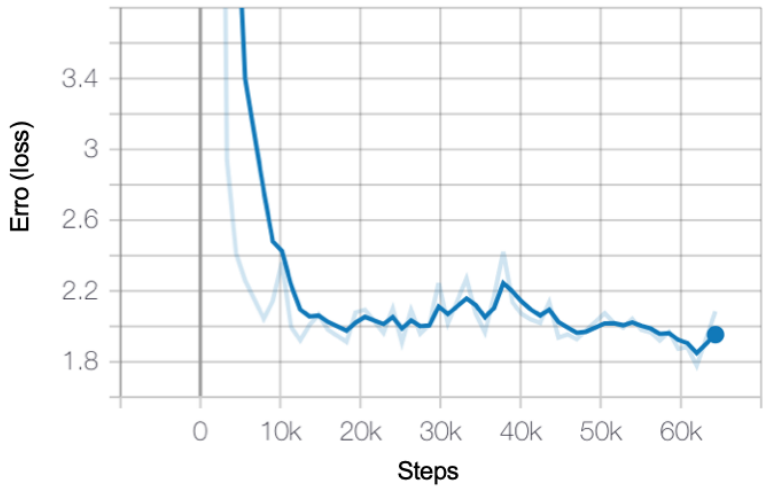


Figura 58 – Evolução das componentes de localização e classificação da GoogLeNet. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.

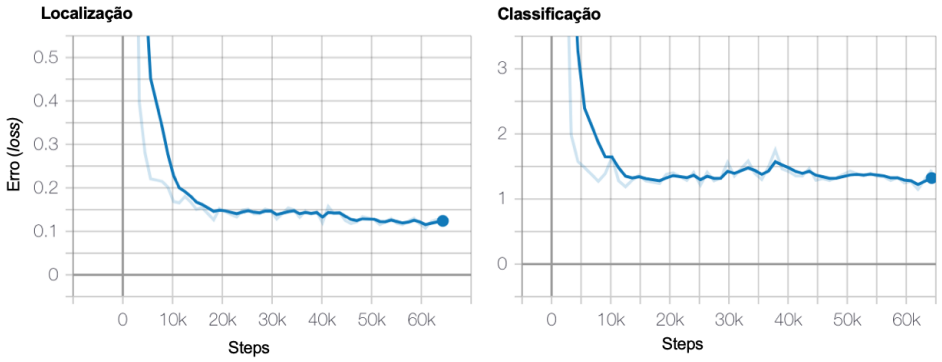
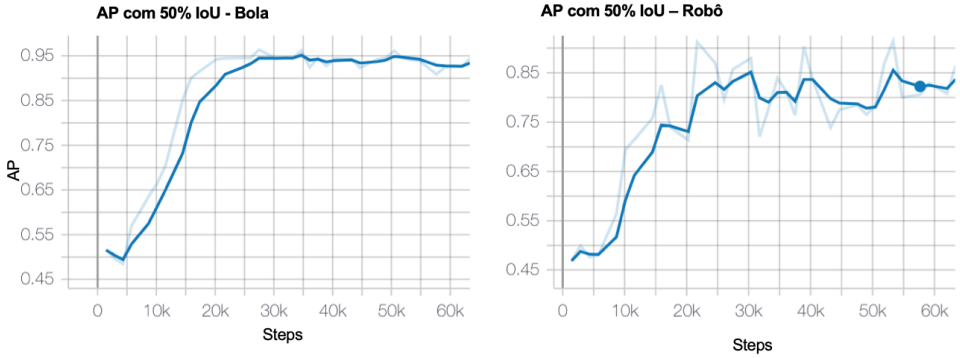


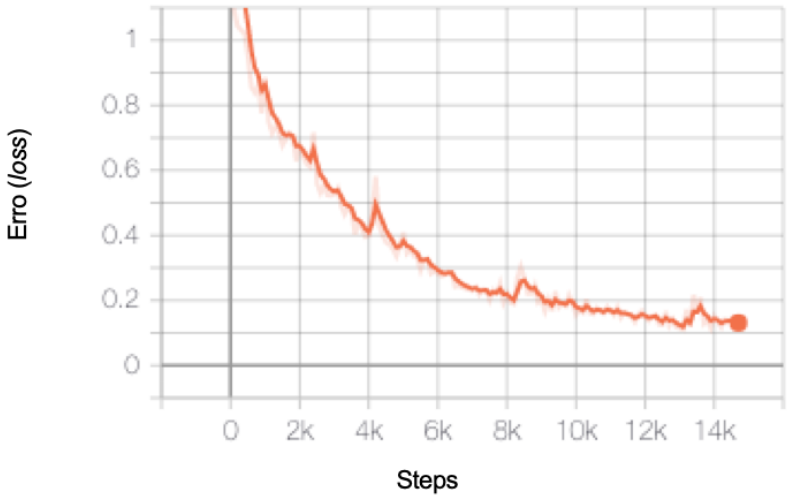
Figura 59 – Métrica *mAP* com 50% de IoU nos dados de teste do Experimento 3 separados por classes. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.



5.4 EXPERIMENTO 4 - RESNET

Para treinar a ResNet, foi utilizado um tamanho de lote de 16 imagens e o otimizador *RMSProp.* com uma taxa inicial de aprendizado de 0.004 e fator de decaimento de 0.95. Foram executados aproximadamente 14 mil passos (*steps*) e cada 38 *steps* aproximadamente equivalem a uma época. Sendo assim, calculando $53000/38 \approx 1395$ épocas foram executadas, durando aproximadamente 11 horas. O gráfico na Figura 60 apresenta a função de para dados de treinamento. O tamanho de entrada de imagens fixo em 300×300 por ser o padrão utilizado no pré-treinamento da CNN.

Figura 60 – Evolução do treinamento da rede no Experimento 4. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.



Conforme mostra o gráfico na Figura 61, a CNN também foi avaliada nos dados de teste e percebe-se alguns picos na função de erros. Apesar da função recuperar seu caminho descendente, pode ser um indício de algo sobreajuste nos dados de treino e que levaram a erros nos dados de teste.

Para acompanhar a evolução da CNN ao longo dos *steps* e investigar o pico na função de erro, as componentes de localização e classificação são apresentadas na Figura 62. Para os picos de erro, pode-se observar que eles aconteceram tanto em termos de classificação do objeto, quanto em termos de localização.

Com o treinamento feito e curvas de erro decrescentes conforme esperado, a próxima métrica para avaliarmos é o mAP apresentado nos dados de teste. A Figura 63 mostra a evolução

Figura 61 – *Loss* nos dados de teste do Experimento 4. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.

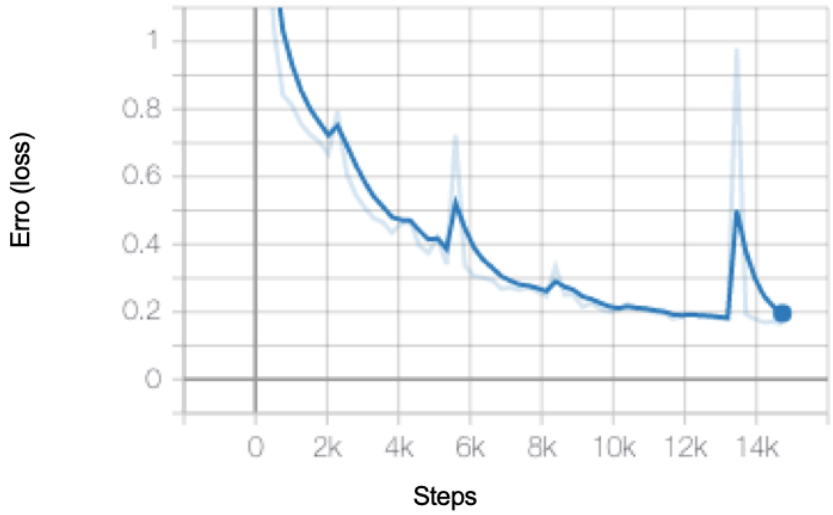
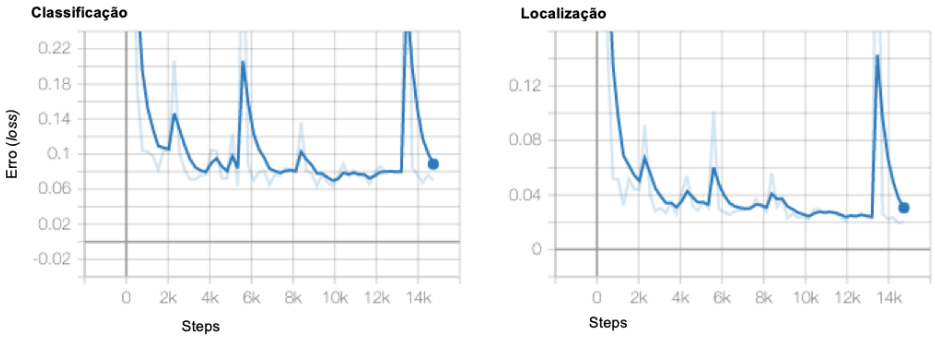


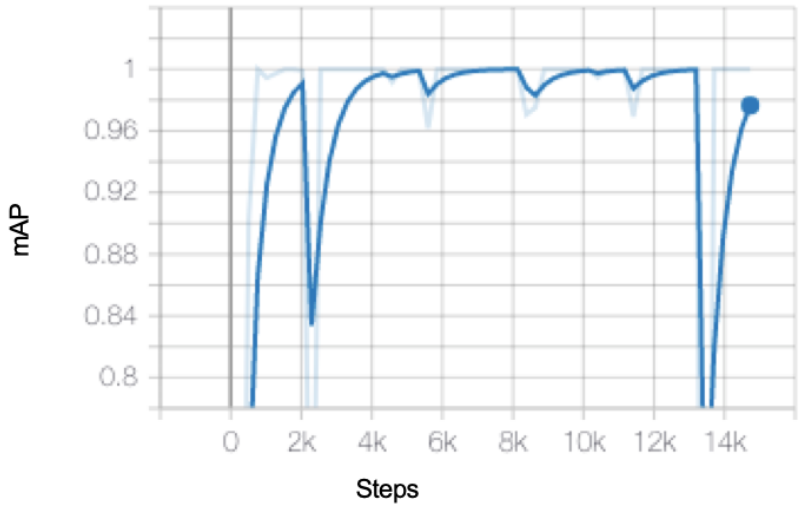
Figura 62 – Evolução das componentes de localização e classificação da ResNet. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.



da métrica ao longo dos *steps* até que estabiliza em ≈ 0.97658 distribuídos em 0.9859 para a bola e 0.9672 para o robô.

O desempenho da CNN em termos de FPS médio foi de apenas≈ 3.2.

Figura 63 – Métrica *mAP* com 50% de IoU nos dados de teste do Experimento 3. A linha azul clara se refere ao resultado real e a linha azul traz a função suavizada por uma média móvel. Fonte: Autor.

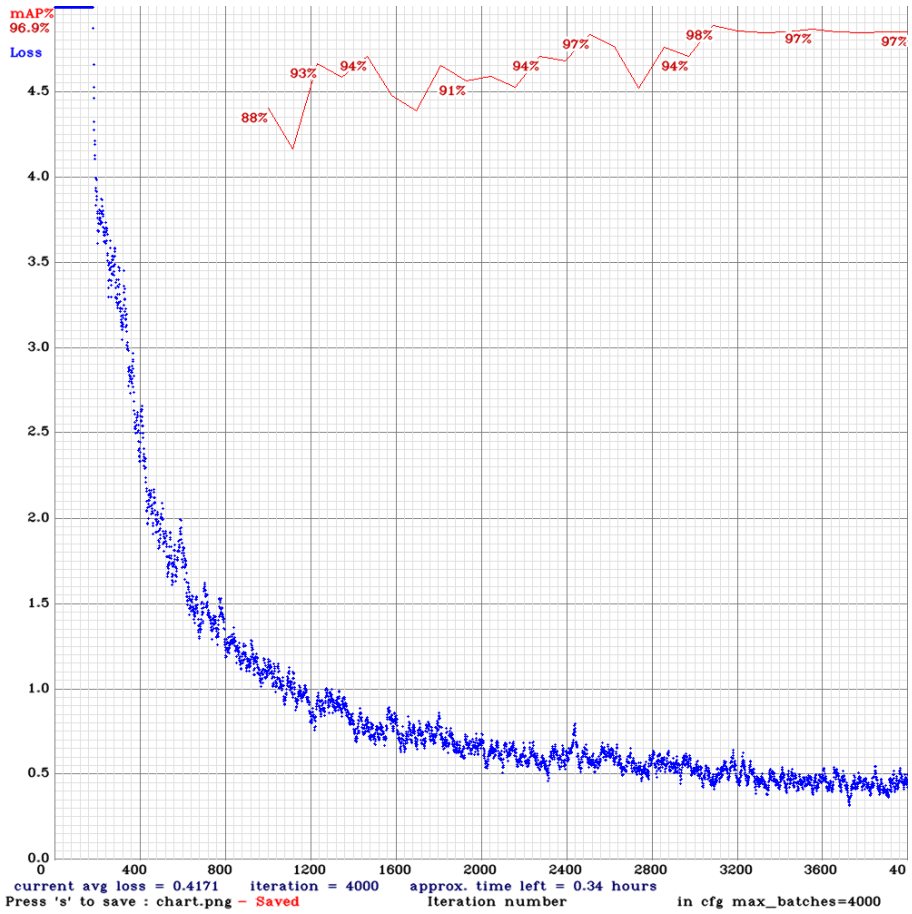


5.5 EXPERIMENTO 5 - YOLOV4

Para treinar a YOLOv4, foi utilizado um tamanho de lote de 64 imagens com uma taxa inicial de aprendizado de 0.001 e fator de decaimento de 0.0005. Foram executados aproximadamente 4000 mil passos (*steps*), cada 9 *steps* aproximadamente equivalem a uma época. Portanto, calculando $4000/9.23 \approx 444$ épocas foram executadas, durando aproximadamente 12 horas. O tamanho de entrada de imagens fixo em 412×412 por ser o padrão utilizado no pré-treinamento da CNN.

O gráfico na Figura 64 mostra a função de custo nos dados de treinamento, juntamente à métrica mAP com 50% de IoU que finalizou em 0.9733, dividida em 0.9815 para a bola de futebol e 0.9568 para o robô. O desempenho da CNN em termos de FPS médio foi de ≈ 5.6.

Figura 64 – Evolução da função de custo e mAP 50% de IoU da YOLOv4 da rede no Experimento 5. Fonte: Autor.



5.6 DISCUSSÃO

Os experimentos realizados tiveram objetivo de avaliar diferentes Redes Neurais Convolucionais para de detecção de objetos e coletar as métricas de desempenho relevantes. Para isto, foram executados cinco experimentos, utilizando um conjuntos de hiperparâmetros padrão das redes conforme respectivos autores utilizaram nos pré-treinamentos.

Cada experimento contou com uma análise cuidadosa tanto do comportamento da função de custo agregada quanto a função de custo segmentada entre Localização e Classificação de objetos. Como cada CNN possui estruturas próprias e funções de custo ligeiramente diferentes, estas funções não são comparáveis. O intuito foi a avaliação de que elas se comportavam de modo decrescente e que a função de custo de teste acompanhava o decrescimento.

Durante o treinamento de cada CNN, a mesma base de imagens com cenas de jogo do futebol de robôs humanoides em diferentes campos de futebol, condições iluminação e brilho de imagem foi utilizada para teste das CNNs e possibilitar a comparação entre elas. A comparação

foi feita com a coleta das seguintes métricas de desempenho: (i) *Frames* por segundo (FPS); (ii) mAP e (iii) AP por classe.

Um primeiro ponto de comparação, conforme pode ser observado na Tabela 9, é a quantidade de épocas que cada CNN foi executada até que a curva de custo se estabilizasse e o treinamento fosse finalizado. As MobileNets em um patamar por volta de 4000 épocas, enquanto a ResNet e a GoogLeNet finalizaram em um patamar próximo às 1000 épocas. A YOLOv4 foi a CNN com menor número de épocas necessárias para a finalização do treinamento.

Tabela 9 – Tabela final comparativa entre as CNNs para Épocas, mAP, AP por classe. Fonte: Autor.

CNN	**Épocas**	**mAP**	**mAP 50% IoU**	**AP bola50% IoU**	**AP robô50% IoU**
RMobileNetV2	3777	0.66	**0.995**	0.996	**0.974**
ResNet	1395	**0.83**	0.977	0.986	0.967
MobileNetV3	4833	0.58	0.980	**0.998**	0.963
YOLO	4000	-	0.973	0.982	0.957
GoogLeNet	1060	0.55	0.900	0.948	0.850

A Tabela 9 também mostra que as CNNs foram capazes de atingir excelentes níveis de precisão média mAP com pelo menos 50% IoU tanto agregada quanto nas métricas AP segmentadas por classe. A CNN com melhor mAP foi a MobileNetV2, atingindo um valor de 0.995, porém apenas 0.015 acima da MobileNetV3. As duas MobileNets também compartilham a melhor média por classe. A MobileNetV2 sendo capaz de atingir valores maiores de AP para a classe de robôs e a MobileNetV3 para a classe da bola de futebol. Analisando a mAP geral, a qual considera diferentes níveis de revocação, a MobileNetV2 também foi superior à MobileNetV3 atingindo 0.66 contra 0.58. Entretando, a houveram picos de mAP na MobileNetV3 de até 0.65 nas avaliações da base de teste durante o treinamento. Estes resultados estão em linha com trabalhos correlatos, como por exemplo Meneghetti et al. (2020), onde resultados da MobileNetV2 e MobileNetV3 quando comparadas em mAP, possuem uma diferença de 0.02.

Os níveis de *mAP* foram similares aos encontrados nos trabalhos correlatos que utilizaram a mesma métrica de avaliação. Comparativamente, Poppinga e Laue (2019) foram capazes de atingir $mAP = 0.591$ apenas para a detecção da bola de futebol. Um patamar similar ao atingido por todas as CNNs treinadas neste trabalho. Já Szemenyei e Estivill-Castro (2019b), atingiram $mAP = 0.56$ identificando as classes bola, cruzamento de linha, trave e robô, um resultado também em linha com os resultados atingidos neste trabalho.

Já em termos de FPS, conforme apresentado na Tabela 10, a CNN com a maior taxa de quadros por segundo foi a MobileNetV3, atingindo 14.7 quadros por segundo, comparado a

14 da segunda melhor rede, uma vantagem de aproximadamente 5%. Nos testes em algumas imagens específicas, as duas CNNs obtiveram resultados muito similares, exceto por algumas imagens com maior distância, como a ilustrada na Figura 55, onde a MobileNetV2 foi superior.

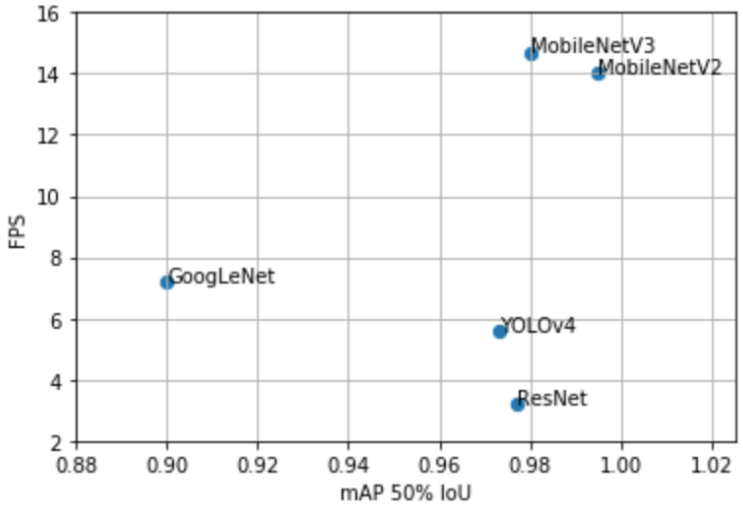
Tabela 10 – Tabela final comparativa entre as CNNs para a métrica de FPS considerando detecções para um vídeo com ambos objetos. Fonte: Autor.

CNN	**FPS**
MobileNetV2	14
ResNet	3.2
MobileNetV3	**14.7**
YOLO	5.6
GoogLeNet	7.2

Comparativamente à literatura, as MobileNets e a YOLOv4 obtiveram um desempenho inferior à JET-Net de Poppinga e Laue (2019) que atinge a taxa de 30 FPS. Já Szemenyei e Estivill-Castro (2019b) atingiram aproximadamente 13 FPS com a rede que detecta múltiplos objetos, o que está no mesmo patamar de FPS deste trabalho.

Por fim, a comparação em termos de mAP com 50% de IoU e FPS na Figura 65 traz a classe de MobileNets no canto superior direito, demonstrando a superioridade em desempenho e velocidade.

Figura 65 – Comparativo mAP e FPS entre todas as CNNs testadas. Fonte: Autor.



Em testes nos vídeos de partidas de futebol, os resultados foram bastante similares visualmente, inclusive com a mesma falha de detecção do robô em posição lateral da Figura 51. Este erro deve estar relacionado à baixa quantidade de imagens do robô de perfil na base de treinamento dos algoritmos.

Analisando a GoogLeNet e a ResNet estas foram as CNNs com pior desempenho tanto em termos de precisão quanto velocidade respectivamente. Este resultado está em linha com a literatura e fundamentação teórica estudadas. Tanto as MobileNets quanto a YOLOv4 implementam conceitos criados na ResNet e na GoogLeNet, que possuem uma importância histórica relevante por vitórias em competições e inovações trazidas ao mundo das Redes Neurais Convolucionais Profundas.

6 CONCLUSÃO

A *RoboCup* e a área robótica realmente geram desafios que incentivam a comunidade científica a evoluir e consequentemente desenvolver novas técnicas para resolução dos mais diversos problemas, sendo a visão computacional apenas um deles.

As CNNs se mostraram cada vez mais consolidadas no ramo da visão artificial, evoluindo tanto no quesito precisão quanto no quesito velocidade. A arquitetura mais promissora termos de velocidade e que manteve patamares excelentes de precisão foi a MobileNetV3, superando a YOLOv4 que é uma arquitetura estado da arte para diversas tarefas e conjuntos de dados e que é recente na literatura, publicada durante o desenvolvimento deste trabalho.

O objetivo de comparar arquiteturas em velocidade e precisão, avaliando-as com métricas *benchmark* difundidas na maioria dos estudos de detecção de objetos foi bem sucedido e obteve métricas comparáveis aos trabalhos relacionados. Dentre as MobileNets, a diferença de mAP entre a MobileNetV3 e a MobileNetV2 deve ser estudada mais profundamente para verificar se é uma diferença gerada por desvios padrão altos ou se realmente é uma diferença significativa que eleva MobileNetV2 a um patamar superior em termos de mAP. Contudo, a velocidade da MobileNetV3 é superior quando avaliada em CPU, conforme literatura estudada, sendo mais indicada para soluções embarcadas com baixa capacidade de processamento e que não possuem GPU.

Infelizmente, a pandemia do Covid-19 impediu o acesso presencial aos laboratórios do Centro Universitário FEI, prejudicando testes embarcados em robôs humanoides e a coleta de imagens adicionais para testes mais profundos quanto à distância de câmera e diferentes iluminações. O acesso a recursos computacionais como GPU com custo acessível para treinamento de diversos modelos com diferentes hiperparâmetros também foi um gargalo neste projeto por conta do elevado custo das máquinas virtuais alugadas.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

6.1 CONTRIBUIÇÕES

Durante este trabalho foi preparada uma base de dados com cenas de futebol de robôs humanoides com a presença de bolas de futebol e robôs. A base conta com variações de campos de futebol, ângulo de perspectiva das cenas, condições de brilho e iluminação. Todas as imagens

foram rotuladas manualmente pelo autor com o cuidado de seguir uma metodologia de anotação descrita neste trabalho. Tal metodologia certamente agregou em qualidade de desempenho nos modelos finais. A base viabiliza que outros trabalhos possam executar seus experimentos, inclusive com a anotação de traves e linhas do campo para trabalhos futuros.

Como contribuição, também foram disponibilizados no GitHub (ABREU, 2020b) (ABREU, 2020c) todos os códigos para realização dos experimentos. Além disso, o script de configuração de uma máquina virtual com GPU na nuvem AWS também foi disponibilizado no Anexo A.

É importante ressaltar que o trabalho avaliou a YOLOv4, publicada durante o desenvolvimento do trabalho e que trouxe bons resultados.

O presente trabalho também gerou uma publicação de resultados preliminares no Workshop de Visão Computacional (WVC) (ABREU; BIANCHI, 2019).

6.2 TRABALHOS FUTUROS

Novos estudos podem ser feitos com as redes neurais convolucionais profundas para o teste com mais objetos, como as bases das traves de futebol para identificação da direção do gol. Outro ponto interessante parece ser a aplicação de das CNNs para segmentação de objetos, um conceito mais amplo que classifica cada pixel da imagem. Bases de dados maiores e com boa metodologia de anotação são necessárias para continuação dos estudos nessa área. A agregação de quadros antecedentes ao mais recente no processamento das imagens, associada a redes como a *Long Short Term Memory* é uma alternativa para não apenas detectar os objetos estaticamente mas também para previsão de trajetórias. Apesar do teste em hardware com configurações similares a do robô humanoide autônomo, testes embarcados são importantes de serem feitos, juntamente a Sistemas Operacionais Robóticos, como o ROS (*Robot Operating System*).

Por fim, este trabalho avaliou imagens com diferentes iluminações e distâncias de câmera. Entretanto testes de mAP com marcações de distância e iluminação parametrizados e bem definidos também seriam interessantes.

REFERÊNCIAS

ABADI, Martin et al. **TensorFlow : Large-Scale Machine Learning on Heterogeneous Distributed Systems**. [S.l.: s.n.], jan. 2015.

ABREU, Lucas. **CNN para detecção de bolas e robôs no domínio do futebol de robôs**. Youtube. 2020. Disponível em: <https://www.youtube.com/watch?v=0pts6u_V8uw&feature=youtu.be>.

ABREU, Lucas. **Object Detection - MobileNets, ResNet, GoogleLeNet**. [S.l.: s.n.], 2020. https://github.com/lucasrabreu/obj_detection. Acesso em Novembro de 2020.

ABREU, Lucas. **Object Detection - YOLO**. [S.l.: s.n.], 2020. https://github.com/lucasrabreu/train_darknet. Acesso em Novembro de 2020.

ABREU, Lucas; BIANCHI, Reinaldo. Real-time Ball Detection for Robocup Soccer Using Convolutional Neural Networks. In: ANAIS do XV Workshop de Visão Computacional. São Bernardo do Campo: [s.n.], 2019. P. 103–108. Disponível em: <https://sol.sbc.org.br/index.php/wvc/article/view/7636>.

BOCHKOVSKIY, Alexey; WANG, Chien-Yao; LIAO, Hong-Yuan Mark. **YOLOv4: Optimal Speed and Accuracy of Object Detection**. [S.l.: s.n.], 2020.

BODLA, Navaneeth et al. Soft-NMS — Improving Object Detection with One Line of Code. **2017 IEEE International Conference on Computer Vision (ICCV)**, IEEE, out. 2017. http://dx.doi.org/10.1109/ICCV.2017.593.

CRUZ, Nicolás; LOBOS-TSUNEKAWA, Kenzo; RUIZ-DEL-SOLAR, Javier. Using Convolutional Neural Networks in Robots with Limited Computational Resources: Detecting NAO Robots while Playing Soccer. **CoRR**, abs/1706.06702, 2017. http://arxiv.org/abs/1706.06702.

D. COOK K. FEUZ, N. Krishnang. Transfer learning for activity recognition: a survey. **Knowledge and Information Systems**, v. 36, p. 537–556, 2013.

DENG, Jia et al. Imagenet: A large-scale hierarchical image database. In: IEEE. 2009 IEEE conference on computer vision and pattern recognition. [S.l.: s.n.], 2009. P. 248–255.

EVERINGHAM, Mark et al. The Pascal Visual Object Classes Challenge: A Retrospective. **International Journal of Computer Vision**, v. 111, 2014.

FELBINGER, Georg et al. Designing Convolutional Neural Networks Using a Genetic Approach for Ball Detection. In: [s.l.: s.n.], ago. 2019. P. 150–161.

FIEDLER, Niklas; BESTMANN, Marc; HENDRICH, Norman. ImageTagger: An Open Source Online Platform for Collaborative Image Labeling. In: ROBOCUP 2018: Robot World Cup XXII. [S.l.: s.n.], 2019. P. 162–169.

GABEL, Alexander et al. Jetson, Where Is the Ball? Using Neural Networks for Ball Detection at RoboCup 2017. In: HOLZ, Dirk et al. (Ed.). **RoboCup 2018: Robot World Cup XXII**. [S.l.]: Springer International Publishing, 2019. P. 181–192.

GÉRON, A. **Hands–On Machine Learning with Scikit–Learn and TensorFlow**. [S.l.: s.n.], 2019.

GLOROT, Xavier; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. **Journal of Machine Learning Research - Proceedings Track**, v. 9, p. 249–256, jan. 2010.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. [S.l.]: MIT Press, 2016. http://www.deeplearningbook.org.

HE, Kaiming et al. Deep residual learning for image recognition. In: PROCEEDINGS of the IEEE conference on computer vision and pattern recognition. [S.l.: s.n.], 2016. P. 770–778.

HOULISTON, Trent; CHALUP, Stephan K. Visual Mesh: Real-Time Object Detection Using Constant Sample Density. **Lecture Notes in Computer Science**, Springer International Publishing, p. 45–56, 2019. http://dx.doi.org/10.1007/978-3-030-27544-0_4.

HOWARD, Andrew et al. Searching for MobileNetV3. **2019 IEEE/CVF International Conference on Computer Vision (ICCV)**, IEEE, out. 2019. http://dx.doi.org/10.1109/ICCV.2019.00140.

HOWARD, Andrew G et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

HUANG, Jonathan et al. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. In: p. 3296–3297.

IOFFE, Sergey; SZEGEDY, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

KITANO, Hiroaki; ASADA, Minoru. The RoboCup humanoid challenge as the millennium challenge for advanced robotics. **Advanced Robotics**, Taylor & Francis, v. 13, n. 8, p. 723–736, 1998.

KITANO, Hiroaki et al. Robocup: The robot world cup initiative. In: PROCEEDINGS of the first international conference on Autonomous agents. [S.l.: s.n.], 1997. P. 340–347.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: ADVANCES in neural information processing systems. [S.l.: s.n.], 2012. P. 1097–1105.

KUKLEVA, Anna et al. Utilizing Temporal Information in Deep Convolutional Network for Efficient Soccer Ball Detection and Tracking. **Lecture Notes in Computer Science**, Springer International Publishing, p. 112–125, 2019. http://dx.doi.org/10.1007/978-3-030-35699-6_9.

LECUN, Yann; KAVUKCUOGLU, Koray; FARABET, Clément. Convolutional networks and applications in vision. **Proceedings of 2010 IEEE International Symposium on Circuits and Systems**, p. 253–256, 2010.

LIN, Min; CHEN, Qiang; YAN, Shuicheng. **Network In Network**. [S.l.: s.n.], 2013.

LIN, Tsung-Yi et al. Microsoft COCO: Common Objects in Context. In: v. 8693.

LIU, Wei et al. SSD: Single Shot MultiBox Detector. **Lecture Notes in Computer Science**, Springer International Publishing, p. 21–37, 2016. http://dx.doi.org/10.1007/978-3-319-46448-0_2.

MALLAT, Stéphane. Understanding Deep Convolutional Networks. **Philosophical transactions. Series A, Mathematical, physical, and engineering sciences**, v. 374 2065, p. 20150203, 2016.

MCCULLOCH, Warren S; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

MENASHE, Jacob et al. Fast and precise black and white ball detection for RoboCup soccer. In: SPRINGER. ROBOT world cup. [S.l.: s.n.], 2017. P. 45–58.

MENEGHETTI, Douglas De Rizzo et al. **Detecting soccer balls with reduced neural networks: a comparison of multiple architectures under constrained hardware scenarios**. [S.l.: s.n.], 2020.

NIELSEN, Michael A. **Neural networks and deep learning**. [S.l.]: Determination press San Francisco, CA, 2015. v. 2018.

PADILLA, R.; NETTO, S. L.; DA SILVA, E. A. B. A Survey on Performance Metrics for Object-Detection Algorithms. In: 2020 International Conference on Systems, Signals and Image Processing (IWSSIP). [S.l.: s.n.], 2020. P. 237–242.

POPPINGA, Bernd; LAUE, Tim. JET-Net: real-time object detection for mobile robots. In: SPRINGER. ROBOT World Cup. [S.l.: s.n.], 2019. P. 227–240.

REDMON, Joseph; FARHADI, Ali. **YOLO9000: Better, Faster, Stronger**. [S.l.: s.n.], 2016.

REDMON, Joseph; FARHADI, Ali. **YOLOv3: An Incremental Improvement**. [S.l.: s.n.], 2018.

REDMON, Joseph et al. **You Only Look Once: Unified, Real-Time Object Detection**. [S.l.: s.n.], 2016.

RICHARD HAHNLOSER, Rahul Sarpeshkar. Digital selection and analogue amplification coesist in a cortex inspired silicon circuit. **Nature**, Springer International Publishing, 2000.

ROBOCUP. [S.l.: s.n.], 2020. http://www.robocup.org. Acesso em Novembro de 2020.

ROBOCUP. **RoboCupSoccer Photos**. [S.l.: s.n.], 2019. https://www.robocup.org/photos.

ROSENBLATT, Frank. **The perceptron, a perceiving and recognizing automaton Project Para**. [S.l.]: Cornell Aeronautical Laboratory, 1957.

RUMELHART, David E; HINTON, Geoffrey E; WILLIAMS, Ronald J. Learning representations by back-propagating errors. **Nature**, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986.

RUSSAKOVSKY, Olga et al. Imagenet large scale visual recognition challenge. **International journal of computer vision**, Springer, v. 115, n. 3, p. 211–252, 2015.

SANDLER, Mark et al. Mobilenetv2: Inverted residuals and linear bottlenecks. In: PROCEEDINGS of the IEEE conference on computer vision and pattern recognition. [S.l.: s.n.], 2018. P. 4510–4520.

SANUSI, B. N.; ADIPRAWITA, W.; MUTIJARSA, K. Humanoid robosoccer goal detection using hough transform. In: 2015 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC). [S.l.: s.n.], 2015. P. 153–159.

SERVICES, Amazon Web. **Instâncias P3 do Amazon EC2**. [S.l.: s.n.], 2020. https://aws.amazon.com/pt/ec2/instance-types/p3/.

SMITH, Steven W. The Scientist and Engineer's Guide to Digital Signal Processing. In:

SPECK, Daniel; BESTMANN, Marc; BARROS, P. Towards Real-Time Ball Localization Using CNNs. In: ROBOCUP. [S.l.: s.n.], 2018.

SPECK, Daniel et al. Ball Localization for Robocup Soccer Using Convolutional Neural Networks. In: p. 19–30.

SZEGEDY, Christian et al. Going deeper with convolutions. In: PROCEEDINGS of the IEEE conference on computer vision and pattern recognition. [S.l.: s.n.], 2015. P. 1–9.

SZEMENYEI, Marton; ESTIVILL-CASTRO, Vladimir. Real-Time Scene Understanding Using Deep Neural Networks for RoboCup SPL. In: ROBOCUP 2018: Robot World Cup XXII. [S.l.]: Springer International Publishing, 2019. P. 96–108.

SZEMENYEI, Marton; ESTIVILL-CASTRO, Vladimir. ROBO: Robust, Fully Neural Object Detection for Robot Soccer. In: ROBOCUP 2019: Robot World Cup XXIII. [S.l.]: Springer International Publishing, 2019. P. 309–322.

TAN, Mingxing et al. MnasNet: Platform-Aware Neural Architecture Search for Mobile. **2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**, IEEE, jun. 2019. http://dx.doi.org/10.1109/CVPR.2019.00293.

TEIMOURI, Meisam; DELAVARAN, Mohammad Hossein; REZAEI, Mahdi. A Real-Time Ball Detection Approach Using Convolutional Neural Networks. In: ROBOCUP 2019: Robot World Cup XXIII. [S.l.]: Springer International Publishing, 2019. P. 323–336.

TENSORFLOW. **Tensorboard**. [S.l.: s.n.], 2020. https://github.com/tensorflow/tensorboard. Acesso em Novembro de 2020.

TZUTALIN. **LabelImg**. [S.l.: s.n.], 2020. https://github.com/tzutalin/labelImg. Acesso em Novembro de 2020.

VILÃO, C. O. et al. A Single Camera Vision System for a Humanoid Robot. In: 2014 Joint Conference on Robotics: SBR-LARS Robotics Symposium and Robocontrol. [S.l.: s.n.], 2014. P. 181–186.

WASIK, Zbigniew; SAFFIOTTI, Alessandro. Robust color segmentation for the robocup domain. In: IEEE. OBJECT recognition supported by user interaction for service robots. [S.l.: s.n.], 2002. v. 2, p. 651–654.

ZHAO, Zhong-Qiu et al. Object detection with deep learning: A review. **IEEE transactions on neural networks and learning systems**, IEEE, v. 30, n. 11, p. 3212–3232, 2019.

ANEXO A - CONFIGURAÇÃO DE AMBIENTE EM NUVEM

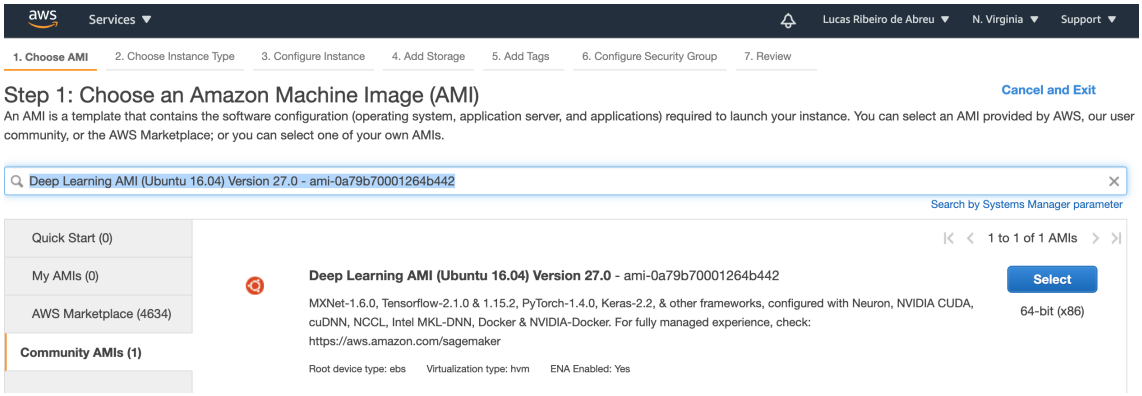
Este anexo tem o objetivo de demonstrar o passo a passo para o provisionamento e configuração de uma máquina virtual na plataforma de serviços de computação em nuvem AWS.

SISTEMA OPERACIONAL E SOFTWARES

O componente utilizado da Amazon é conhecido como EC2 (*Amazon Elastic Compute Cloud*). O EC2 permite que os usuários aluguem computadores virtuais nos quais rodam suas próprias aplicações de diversas naturezas, inclusive com a possibilidade de GPUs.

O tipo de EC2 de referência *"Deep Learning AMI (Ubuntu 16.04) Version 27.0 - ami-0a79b70001264b442"* foi selecionada no Console da AWS após registro no site, conforme Figura 66. Esta configuração de EC2 foi escolhida pelo fato de já ter pré-instalada algumas bibliotecas de código aberto instaladas, como o TensorFlow, PyTorch, Neuron, além de softwares (CUDA, cuDNN, NVIDIA) que integram essas bibliotecas com as GPUs.

Figura 66 – Interface da AWS para seleção de máquina virtual. Fonte: Autor.



HARDWARE

Em sequência, as configurações de hardware devem ser escolhidas. Segundo Services (2020), as instâncias P3 do Amazon EC2 fornecem computação de alta performance com até 8 GPUs para aprendizado de máquina. As instâncias possuem até 1 petaflop de desempenho de precisão mista para acelerar significativamente aplicações de aprendizado de máquina e os aplicativos de computação de alta performance. As instâncias P3 do Amazon EC2 demonstraram ser capazes de reduzir o tempo de treinamento de aprendizado de máquina de dias para

minutos. Com base nisso, as escolhas de hardware podem ser feitas com base na tabela abaixo na Figura 67, com valores de até $31 doláres por hora utilizada. E a escolhida neste trabalho foi a *p3.8xlarge* ou em momentos de testes a *p3.2xlarge*.

Figura 67 – Tabela de configurações e valores das instâncias P3. Fonte: (SERVICES, 2020).

Tamanho de instância	GPUs – Tesla V100	Peer-to-peer de GPUs	Memória de GPU (GB)	vCPUs	Memória (GB)	Largura de banda de rede	Largura de banda do EBS	Preço sob demanda/hora*	Instância reservada por 1 ano – por hora*	Instância reservada por 3 anos – por hora*
p3.2xlarge	1	N/D	16	8	61	Até 10 Gbps	1,5 Gbps	3,06 USD	1,99 USD	1,05 USD
p3.8xlarge	4	NVLink	64	32	244	10 Gbps	7 Gbps	12,24 USD	7,96 USD	4,19 USD
p3.16xlarge	8	NVLink	128	64	488	25 Gbps	14 Gbps	24,48 USD	15,91 USD	8,39 USD
p3dn.24xlarge	8	NVLink	256	96	768	100 Gbps	19 Gbps	31,218 USD	18,30 USD	9,64 USD

ACESSO

Com o hardware e o sistema operacional escolhidos e já pré-configurados, deve-se abrir o terminal e acessar a máquina virtual via comandos SSH com o IP da máquina e também a chave de acesso:

```
ssh -i "chave_acesso.pem" ubuntu@IP_VM
```

CONFIGURAÇÕES

Tensorflow - Detecção de objetos

Ao acessar a máquina, o script Shell abaixo pode ser executado para configurar o módulo de detecção de objectos do Tensorflow. O último passo do script testa diversas configurações e também a conexão da máquina virutal com a GPU.

```
source activate tensorflow_p36
git clone https://github.com/tensorflow/models.git
cd models/research/
wget -O protobuf.zip \
https://github.com/google/protobuf/releases \
/download/v3.0.0/protoc-3.0.0-linux-x86_64.zip
unzip protobuf.zip
./bin/protoc object_detection/protos/*.proto --python_out=.
```

```
pip install tf_slim
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
python3 object_detection/builders/model_builder_test.py
```

Métricas

Outro importante ponto é configurar as bibliotecas para cálculo de métricas de detecção de objetos. O script abaixo lida com a instalação desse módulo.

```
pip install pycocotools
git clone https://github.com/cocodataset/cocoapi.git
cd cocoapi/PythonAPI
make
cp -r cocoapi/PythonAPI/pycocotools/ models/research/
```