Universidade Federal de Pernambuco
Centro de Informática

Graduação em Engenharia da Computação

DETECÇÃO E RASTREAMENTO DE CAPACETES EM JOGADAS DE FUTEBOL AMERICANO UTILIZANDO FASTERRCNN, YOLOV5 E DEEPSORT

Juan Carlo Henrique dos Santos

TRABALHO DE GRADUAÇÃO

Recife
19 de Maio de 2022

Universidade Federal de Pernambuco
Centro de Informática

Juan Carlo Henrique dos Santos

DETECÇÃO E RASTREAMENTO DE CAPACETES EM JOGADAS DE FUTEBOL AMERICANO UTILIZANDO FASTERRCNN, YOLOV5 E DEEPSORT

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: *Prof. Dr. Tsang Ing Ren*

Recife
19 de Maio de 2022

Dedico este trabalho aos meus familiares e amigos que me apoiaram e incentivaram ao longo deste caminho, além de todos os professores, mentores e orientadores que fizeram este sonho possível.

AGRADECIMENTOS

Agradeço a minha família por me apoiar de todas as formas possíveis para que eu pudesse me concentrar nos estudos.

Agradeço aos amigos das turmas 2015.2 e 2016.1, por todas as experiências compartilhadas e pela ajuda nos momentos difíceis.

Agradeço a todos os professores que me incentivaram e me educaram, desde a educação fundamental até o ensino superior.

Agradeço ao professor Tsang por ter aceitado ser meu orientador nesta jornada, sua experiência foi inestimável.

Nothing is lost and nothing's gained,
It's the same old situation again,
No, I can't give or take anymore.

—STEELERS WHEEL (Gets So Lonely, 1972)

RESUMO

Com o advento e avanço das redes sociais e plataformas de *streaming*, o futebol americano ganhou grande popularidade nos últimos anos. A popularização do esporte, enfatiza para o mundo tanto os pontos positivos quanto os negativos do esporte. Um dos pontos negativos são os constantes contatos entre os capacetes de jogadores. Esses contatos podem causar concussões, que prejudicam não só a longevidade e qualidade na carreira profissional dos atletas, mas também nas suas vidas pessoais depois da aposentadoria. Este trabalho propõe a detecção de capacetes em imagens de jogadas de futebol americano através da comparação entre as técnicas de *Deep Learning*, FasterRCNN e Yolov5, além do rastreamento em vídeos de jogadas utilizando a técnica DeepSort, de forma a ajudar na identificação do contato entre capacetes. Ele mostra que a Yolov5 consegue um mAP 14% maior que a FasterRCNN e que o rastreamento com o DeepSort funciona melhor com classes de objetos com variação estética.

Palavras-chave: redes sociais, plataformas de streming, futebol americano, concussões, detecção de capacetes, Deep Learning, FasterRCNN, Yolov5, rastreamento de capacetes, DeepSort

ABSTRACT

With the advent and advancement of social media and streaming platforms, american football has gained great popularity in recent years outside of the United States. The popularization of sport emphasizes to the world both the positive and negative points of the sport. One of the negative points is the constant contact between the players' helmets. These contacts can cause concussions, which damage not only the longevity and quality of athletes' professional careers, but also their personal lives after retirement. This work proposes the detection of helmets in images of american football plays through the comparison between the DeepLearning techniques, FasterRCNN and Yolov5, in addition to tracking in videos of plays using the DeepSort technique, in order to help identify the contact between helmets. It shows that Yolov5 achieves a 14% higher mAP than FasterRCNN and that tracking with DeepSort works better with object classes with aesthetic variation.

Keywords: social media, streaming platforms, american football, concussions, helmet detection, DeepLearning, FasterRCNN, Yolov5, helmet tracking, DeepSort

SUMÁRIO

Capítulo 1—Introdução 1

1.1 Motivação . . . 1
1.2 Objetivos . . . 2
1.3 Estrutura . . . 2

Capítulo 2—Background e Estado da Arte 4

2.1 Redes Neurais Fully Connected e Redes Neurais Convolucionais (CNNs) . 4
2.1.1 Redes Neurais Fully Connected . . . 4
2.1.2 Redes Neurais Convolucionais (CNNs) . . . 5
2.2 Detecção e Rastreamento de Objetos . . . 8
2.2.1 Detecção . . . 8
2.2.2 Rastreamento . . . 8
2.3 Família FasterRCNN . . . 9
2.3.1 R-CNN . . . 9
2.3.1.1 Geração das Regiões Propostas . . . 10
2.3.1.2 Extração do Vetor de Características . . . 10
2.3.1.3 Classificação das Regiões Propostas . . . 10
2.3.2 FastRCNN . . . 11
2.3.2.1 Camada de ROI *Pooling* . . . 11
2.3.2.2 Camada de Saída . . . 12
2.3.3 FasterRCNN . . . 13
2.3.3.1 RPN . . . 13
2.4 Família Yolo . . . 16
2.4.1 Yolo . . . 17
2.4.2 Yolov2 . . . 18
2.4.3 Yolov3 . . . 19
2.4.4 Yolov4 . . . 21
2.4.4.1 Backbone . . . 21
2.4.4.2 Neck . . . 22
2.4.4.3 Head . . . 23
2.4.5 Yolov5 . . . 24
2.5 Família DeepSort . . . 24
2.5.1 SORT . . . 25
2.5.1.1 Detecção . . . 25
2.5.1.2 Estimação do Deslocamento . . . 25

2.5.1.3 Associação de Objetos a Tracks 25
2.5.1.4 Criação e Remoção de Tracks 25
2.5.2 DeepSort 26
2.5.2.1 Estimação do Deslocamento 26
2.5.2.2 Associação de Objetos a Tracks 26
2.5.2.3 Criação e Remoção de Tracks 27
2.6 Estado da Arte 27

Capítulo 3—Metodologia 30

3.1 Conjunto de Dados 30
3.2 Treinamento das Técnicas de Detecção 32
3.2.1 Treinamento da FasterRCNN 34
3.2.2 Treinamento da Yolov5 34
3.3 Treinamento da Técnica de Rastreamento 35
3.4 Avaliação dos Modelos - Mean Average Precision (mAP) 35
3.5 Ambiente 36

Capítulo 4—Experimentos e Resultados 37

4.1 Treinamento FasterRCNN 37
4.2 Treinamentos Yolov5 38
4.3 Comparação estatística entre FasterRCNN e Yolov5 40
4.4 Treinamento do DeepSort 42
4.5 Teste dos Detectores com Imagens do Conjunto de Dados 42
4.6 Teste dos Detectores com Outro Tipos de Capacetes 42
4.7 Demonstração com o DeepSort 45

Capítulo 5—Conclusões e Trabalhos Futuros 47

LISTA DE FIGURAS

2.1 Redes Neurais *fully connected*. Fonte: [15] 5
2.2 Filtro de convolução aplicado a uma imagem. Fonte: [16] 6
2.3 VGG16. Fonte: [18] 7
2.4 *Max Pooling* e *Average Pooling*. Fonte: [19] 7
2.5 Objetos e suas *bounding boxes*. Fonte: [20] 9
2.6 Funcionamento R-CNN. Fonte: [21] 10
2.7 Funcionamento FastRCNN. Fonte: [22] 11
2.8 Suposto mapa de características de uma imagem. Fonte: [25] 12
2.9 Seção do mapa de características de uma imagem correspondente a uma região proposta. Fonte: [25] 13
2.10 Divisão da região proposta em quatro sub-seções. Fonte: [25] 14
2.11 Funcionamento FasterRCNN. Fonte: [12] 14
2.12 Funcionamento RPN. Fonte: [12] 15
2.13 *Intersection Over Union*. Fonte: [27] 16
2.14 Funcionamento Yolo. Fonte: [28] 17
2.15 CNN Yolo. Fonte: [28] 18
2.16 CNN Yolov2, 20 classes. Fonte: [30] 19
2.17 CNN Yolov3. Fonte: [33] 21
2.18 DarkNet-53. Fonte: [32] 22
2.19 Mosaico. Fonte: [34] 23
2.20 PAN original [37], a esquerda e modificada a direita. Fonte: [34] 23
2.21 Arquitetura da Yolov5 (não oficial). Fonte: [38] 24
2.22 DeepSort com detector Yolo. Fonte: [41] 26

3.1 Exemplo de imagem com câmera atrás do gol. Fonte: [11] 30
3.2 Exemplo de imagem com câmera na linha lateral. Fonte: [11] 31
3.3 Movimentação da câmera em direção a lateral. Fonte: [11] 31
3.4 Movimentação da câmera em direção ao fundo. Fonte: [11] 32
3.5 Diagrama de treinamento da FasterRCNN. 34
3.6 Diagrama de treinamento da Yolov5. 35

4.1 Erro de *objectness* RPN FasterRCNN. 37
4.2 Erro de regressão RPN FasterRCNN. 38
4.3 Erro de classificação da camada de saída FasterRCNN. 38
4.4 Erro de regressão da camada de saída FasterRCNN. 39
4.5 mAP FasterRCNN. 39
4.6 Erro de *objectness* Yolov5. 40

4.7 Erro de regressão Yolov5. 40
4.8 mAP Yolov5. 41
4.9 Erro de classificação da ResNet50 para o DeepSort. 42
4.10 Exemplo da FasterRCNN com detecção dos bonés dos juízes. Fonte: [11] 43
4.11 Exemplo da Yolov5 sem detecção dos bonés dos juízes. Fonte: [11] 43
4.12 Exemplo da FasterRCNN com capacetes pequenos. Fonte: [11] 44
4.13 Exemplo da Yolov5 com capacetes pequenos Fonte: [11] 44
4.14 Exemplo da FasterRCNN com capacetes de hockey no gelo. Fonte: [53] . 45
4.15 Exemplo da Yolov5 com capacetes de hockey no gelo. Fonte: [53] 45
4.16 Exemplo da FasterRCNN com capacetes de patinação no gelo. Fonte: [54] 46
4.17 Exemplo da Yolov5 com capacetes de patinação no gelo. Fonte: [54] . . . 46

LISTA DE TABELAS

2.1 Darknet-19 com dimensões de entrada 448x448. Fonte: [29] 20

3.1 Porção da tabela com localidade e classe dos capacetes nas imagens. Fonte: [11] 33
3.2 Porção da tabela com localidade, classe e identificador único dos capacetes nas imagens. Fonte: [11] 33

4.1 Comparação entre erros e mAP dos modelos. 41
4.2 Número de TP, FP, FN e *Precision*, *Recall* 41
4.3 Acertos e erros de um modelo no outro. 42

CAPÍTULO 1

INTRODUÇÃO

1.1 MOTIVAÇÃO

A NFL (*National Football League*) [1] foi fundada em 1920, e desde lá o esporte vem se popularizando pelo mundo. O avanço na utilização das redes sociais e plataformas de *streaming* tem acelerado a divulgação e aceitação do esporte em um escopo mundial [2]. No Brasil não é diferente, a Liga BFA (Liga Brasil Futebol Americano) [3] foi fundada em 2017 e é a maior liga de futebol americano do país.

Essa popularização trouxe a atenção do grande público não só para os pontos positivos do esporte, mas também para os negativos. Um dos pontos negativos são os constantes contatos entre os capacetes dos jogadores. Esses contatos podem causar concussões, que se ocorrerem de forma repetida podem resultar em uma doença chamada Encefalopatia Traumática Crônica (ETC) [4]. Existem diversos relatos de jogadores e familiares de jogadores que perderam a vida de forma trágica devido ao ETC. Como o ETC é uma doença que só pode ser constatada depois da morte [5][6], não existia muita informação sobre sua manifestação em ex-jogadores de futebol americano. Porém nos últimos 15 anos, é possível ver uma progressão no número de estudos feitos na área [7].

A NFL vem então buscando alternativas para resolver os problemas das concussões. Uma delas são competições lançadas no *Kaggle* [8], que é uma plataforma de competições de ciência de dados e aprendizagem de máquina. A NFL lança todos os tipos de competição, envolvendo ou não processamento de imagens [9][10], mas o foco deste trabalho é a competição de detecção de impacto entre capacetes [11].

A detecção de objetos em imagens não é um tema novo, técnicas tradicionais de processamento de imagens e visão computacional eram utilizadas para essa tarefa. Quase todas essas técnicas envolvem fazer o *sliding window* do formato do objeto em questão na imagem, utilizando diferentes proporções e tamanhos, de forma a encontrar na imagem formas que se assemelhem a procurada, depois a forma é passada por um extrator de características, como HOG (*Histogram of Oriented Gradients*) ou SIFT (*Scale-Invarient Feture Transform*), e por último as características passam por um classificador, como SVM (*Support Vector Machine*). Estes tipos de técnicas funcionam, porém são muito demoradas, pelo fato de uma imagem ter que ser processada várias vezes. Por este motivo elas tem sido substituídas por técnicas de *Deep Learning*.

A detecção e rastreamento de capacetes em imagens e vídeos em esportes no geral, são problemas importantes e têm diversas aplicações práticas, como o rastreamento de jogadores em campo para saber o quanto cada jogador percorreu em distância, a velocidade média do jogador, além das áreas do gramado que ele ocupou; a detecção da intensidade do impacto do contato entre capacetes; tendências de cada jogador, como eles se comportam em cada situação; além de possíveis aplicações para identificar violações as regras do

jogo, como ocupação de áreas indevidas no gramado, mais jogadores em campo do que o permitido, entre outros.

1.2 OBJETIVOS

O objetivo **geral** deste trabalho é comparar duas técnicas de *Deep Learning*, mais especificamente as técnicas FasterRCNN (*Region Based Convolutional Neural Networks*) [12] e Yolov5 (*You Only Look Once*) [13], para verificar qual obtém um melhor desempenho na detecção de capacetes em jogadas de futebol americano. Depois da detecção, a melhor técnica será aplicada como uma demonstração, junto ao rastreamento dos objetos em vídeos de jogadas.

São objetivos **específicos** deste projeto:

- Realizar uma visão geral de conceitos básicos de redes neurais e *Deep Learning*;
- Definir as diferenças entre detecção e rastreamento de objetos;
- Explicar as técnicas utilizadas para detecção e rastreamento;
- Revisar o estado da arte para técnicas relacionadas com detecção e rastreamento de objetos;
- Executar uma análise exploratória do conjunto de dados;
- Performar treino das técnicas de detecção;
- Avaliar e comparar as técnicas de detecção utilizadas;
- Performar treino da técnica de rastreamento;
- Aplicar melhor técnica de detecção junto a técnica de rastreamento;
- Organizar demonstrações dos experimentos.

1.3 ESTRUTURA

O trabalho é estruturado da seguinte forma:

No Capítulo 2 são explicados alguns conceitos básicos de *Deep Learning*, as diferenças entre detecção e rastreamento de objetos, todas as técnicas utilizadas para detecção e rastreamento nesse trabalho, e por fim, é exposta uma visão sobre o estado da arte.

O Capítulo 3 explica a metodologia utilizada para a realização do trabalho, desde a análise exploratória dos dados, métodos de treinamento e avaliação dos modelos, até o ambiente de trabalho, isto é, *Hardware* e *Frameworks* utilizados em todas as etapas.

São relatados todos os experimentos realizados, como o treino dos modelos, os testes em diferentes conjuntos de dados, além dos resultados e da comparação entre os modelos e da demonstrações em imagens e vídeos no Capítulo 4.

Por fim, no Capítulo 5 são expostos numa conclusão, todos os acertos e erros do trabalho, além de sugestões de construção sobre o material que podem vir a ser contribuições interessantes para explorações futuras e melhorias em geral para o tralho atual.

CAPÍTULO 2

BACKGROUND E ESTADO DA ARTE

Para um melhor entendimento do trabalho, é necessário uma introdução a alguns conceitos das áreas de processamento de imagens e *Deep Learning*, além da revisão de algumas técnicas utilizadas. Este capítulo explica alguns conceitos importantes, como a diferença entre redes neurais *fully connected*(ou *feed forward*) e redes neurais convolucionais (CNNs), e entre detecção e rastreamento de objetos. Depois são abordadas as três técnicas utilizadas, a FasterRCNN [12], o Yolov5 [13] e o DeepSort [14].

2.1 REDES NEURAIS FULLY CONNECTED E REDES NEURAIS CONVOLUCIONAIS (CNNS)

As redes neurais são artifícios matemáticos que auxiliam na extração de características revelantes de conjuntos de dados. Elas podem ser utilizadas para regressão, classificação, e etc. O trabalho atual foca na utilização de redes neurais para detecção, o que envolve regressão e classificação.

2.1.1 Redes Neurais Fully Connected

Redes neurais *fully connected*, ilustradas na Figura 2.1, são o tipo de rede neural mais tradicional. Elas consistem de uma estrutura em camadas, cada camada possui nós chamados de neurônios, percebe-se no exemplo que os neurônios da Camada 1 são chamados $n11$, $n21$, $n31$, $n41$. As redes *fully connected* tem esse nome por elas serem uma estrutura completamente conectada, no exemplo todos os neurônios da Camada 1 influenciam nos neurônios da Camada 2, e todos os neurônios da Camada 2 influenciam nos neurônios da Camada de Saída. Outro elemento dos neurônios são os pesos. No exemplo, os pesos do neurônio $n11$ são chamados $w1$, $w2$, $w3$. Esses pesos são multiplicados por todas as entradas, no exemplo $a1$, $a2$, $a3$, somados a um bias, digamos $b11$, e por último passados por uma função de ativação. Portanto para cada neurônio temos equações do tipo:

$$n11 = activation(w1 * a1 + w2 * a2 + w3 * a3 + b11) \tag{2.1}$$

A função de ativação é escolhida de tal forma a incluir uma não linearidade na estrutura. No final, a saída dessas redes é o resultado do cálculo dessa ativação em cada neurônio, pois, como no exemplo, os valores de saída dos neurônios da Camada 1, são utilizados como valores de entrada na Camada 2, e os valores de saída dos neurônios da Camada 2, são utilizados como valores de entrada na Camada de Saída. Assim os valores fluem da entrada até a saída passando por cada neurônio da rede, e por essa razão estas redes também são chamadas de *feed forward*.

O maior artifício e qualidade das redes neurais é a possibilidade de elas serem treinadas em um determinado conjunto de dados. Ainda com o exemplo da Figura 2.1 em mente,

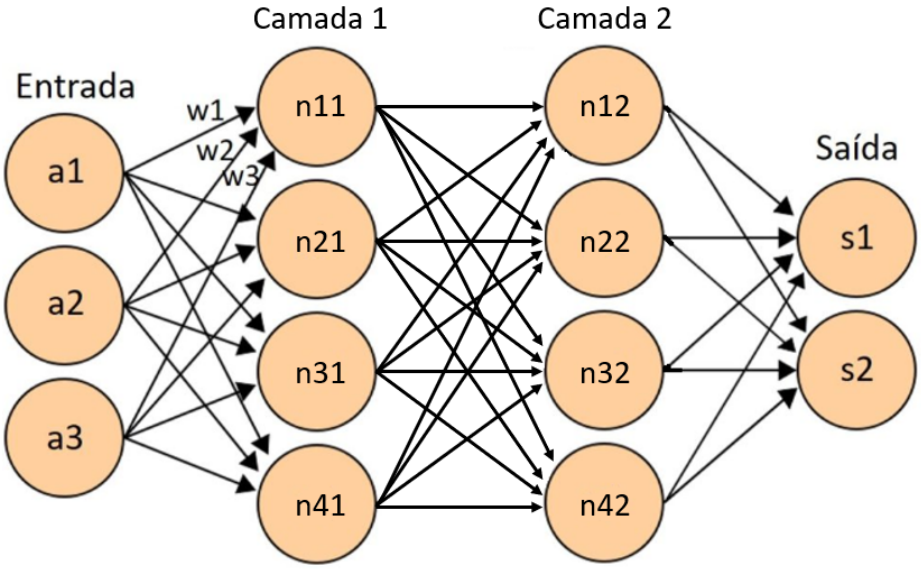

Figura 2.1: Redes Neurais *fully connected*. Fonte: [15]

suponha que tenhamos um conjunto de dados com 3 atributos, idade ($a1$), localização ($a2$) e valor monetário ($a3$), que diz respeito a casas que devam ou não ser demolidas. Uma regra simples seria passar os atributos pela rede de tal forma que $s1$ fique ativo quando uma casa do conjunto deva ser demolida e $s2$ fique ativo quando uma casa do conjunto deva ser preservada. Considerando que o conjunto de dados esteja organizado nesse formato, tudo que deve ser feito é passar os atributos pela rede e comparar a saída conhecida do conjunto de dados com a saída da rede e calcular se a rede errou a previsão ou não. Uma vez constatado o erro ou acerto, esse valor é quantizado (usualmente uma função de erro como entropia cruzada), e propagado de volta aos neurônios da rede, corrigindo os pesos e os bias para que na próxima passagem a rede reconheça, e efetivamente aprenda a prever quais casas devem ser demolidas e quais devem ser preservadas. Esse processo de propagação do erro de volta na rede é chamado de *backpropagation*.

2.1.2 Redes Neurais Convolucionais (CNNs)

As redes neurais convolucionais (CNNs), são utilizadas para o processamento de imagens pelo fato delas utilizarem convoluções para o processamento do conjunto de dados. A utilização de convoluções em imagens é vantajoso, pois a convolução leva em conta a dependência espacial dos pixels de uma imagem, pois pixels próximos tendem a descrever o mesmo objeto.

Observe na Figura 2.2, o funcionamento da convolução em uma imagem. Uma matriz de convolução, ou filtro (no caso um filtro de Sobel), é passado sobre uma imagem, executando a convolução. Numa CNN, cada filtro é um conjunto de pesos, onde cada posição da matriz é um peso diferente.

É comum que uma rede convolucional possua muitos filtros e muitas camadas, por

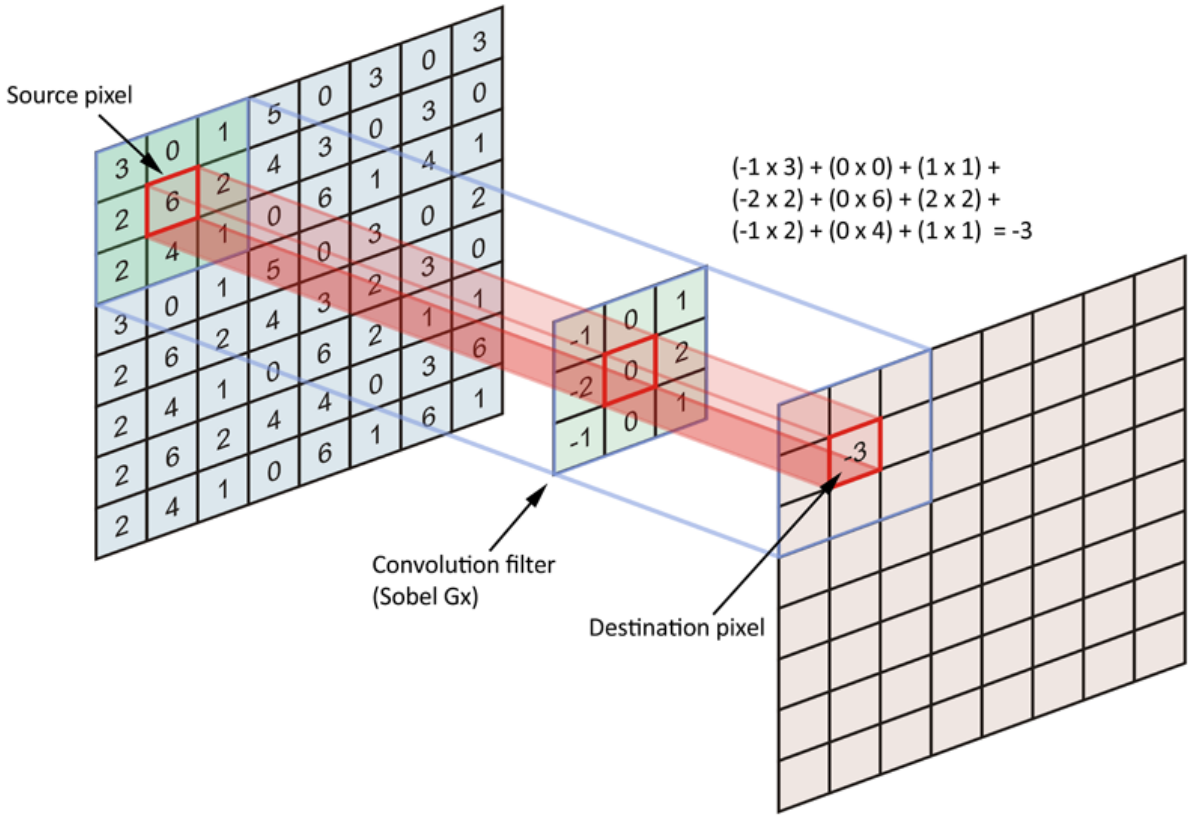

Figura 2.2: Filtro de convolução aplicado a uma imagem. Fonte: [16]

isso o nome de *Deep Learning*. Observe o exemplo da Figura 2.3. Esta é uma rede convolucional chamada VGG16 [17]. Percebe-se que ela tem três componentes (lado inferior direito): camadas de convolução + ReLU (função de ativação), camadas de *max pooling* e camadas de neurônios *fully connected*. As camadas de convolução $conv1$ possuem 64 filtros cada. A primeira atua sobre a imagem original. As camadas de convolução subsequentes atuam sobre os resultados das camadas anteriores, da mesma forma que uma rede neural *fully connected*. Logo depois de cada camada convolucional existe uma função de ativação dos resultados, para que seja inserida a não linearidade.

Outro elemento comum em redes convolucionais são as camadas de *pooling* (no caso *max pooling*). O exemplo da Figura 2.4, mostra como funciona o *max pooling* e o *average pooling*. Percebe-se que uma vez escolhida a janela de *pooling*, essa janela é passada sobre o resultado da convolução. No *max pooling* o máximo valor dentro da janela é o tomado como resultado para cada local da janela. No *average pooling* a média entre os valores da janela é tomado como resultado para cada local da janela. A janela pode andar com passos de tamanhos diversos, mas geralmente o tamanho da janela é igual ao tamanho do passo, assim no resultado do *pooling*, não haverá sobreposição nas posições assumidas pela janela.

O último elemento na VGG16 que é comum a redes convolucionais, são as camadas de neurônios *fully connected*. As redes convolucionais são utilizadas para extrair carac-

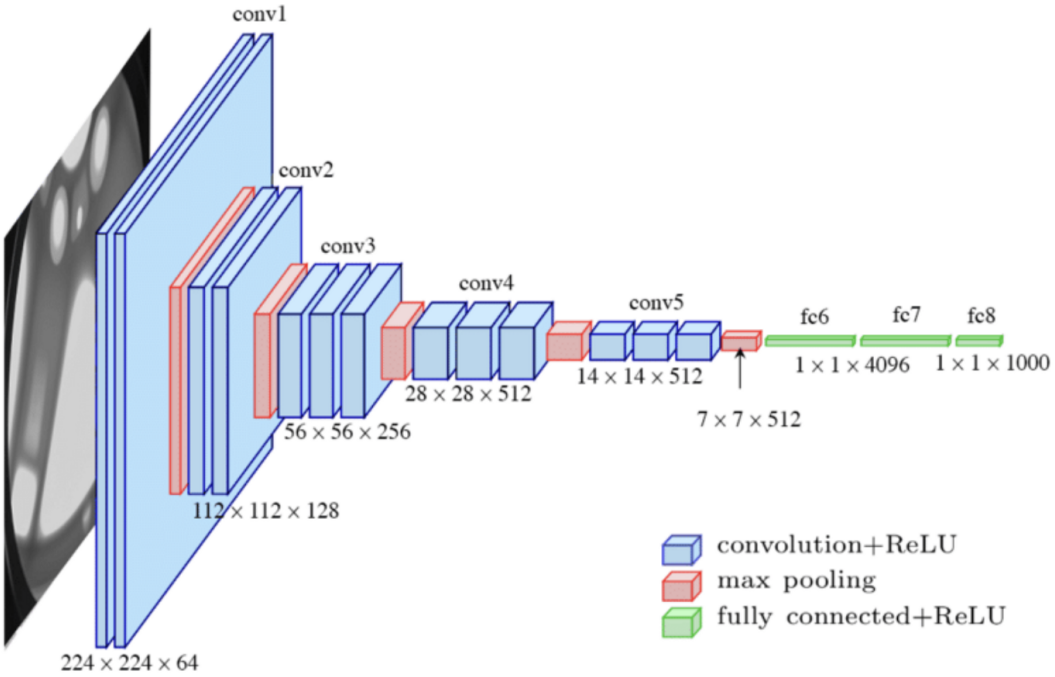

Figura 2.3: VGG16. Fonte: [18]

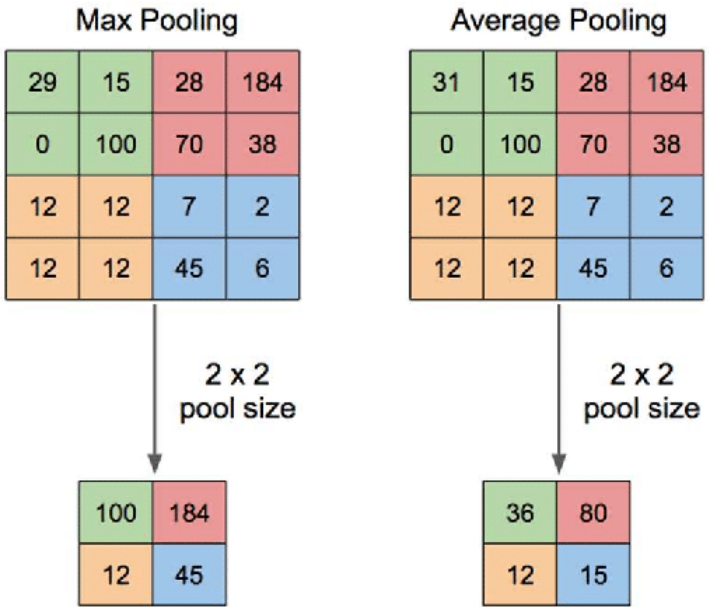

Figura 2.4: *Max Pooling* e *Average Pooling*. Fonte: [19]

terísticas dos objetos numa imagem, uma vez que o resultado das convoluções chega a uma quantidade de valores razoável, eles são colocados como entrada para uma rede *fully connected*, e ela é encarregada de fazer as decisões.

O treinamento de uma rede convolucional é similar ao treinamento de uma rede *fully*

connected. Também é utilizado o *backpropagation* para propagar o erro no resultado final de volta à rede de forma a corrigir os valores dos pesos nos filtros, para que a rede efetivamente aprenda o conjunto de dados.

2.2 DETECÇÃO E RASTREAMENTO DE OBJETOS

Os termos detecção e rastreamento de objetos são muitas vezes confundidos e utilizados intercambiavelmente. Esta seção acentua as diferenças entre detecção e rastreamento de objetos, e como uma parte utiliza a outra. É importante frisar que técnicas de detecção e rastreamento não necessariamente precisam incluir *Deep Learning* em suas composições.

2.2.1 Detecção

A detecção de objetos consiste de aplicar uma técnica em uma imagem que retorne como resultado a localização dos objetos representados na imagem, além do tipo dos objetos. O exemplo da Figura 2.5 ilustra o resultado de uma técnica de detecção de objetos. É possível perceber que a técnica localiza o objeto na imagem e retorna o tipo do objeto (cachorro, bicicleta, etc). A técnica também pode retornar erros, como a roda da bicicleta avaliada como um relógio.

2.2.2 Rastreamento

O rastreamento de objetos consiste de aplicar uma técnica a um vídeo que, não só detecta objetos nos quadros do vídeo, mas também identifica unicamente os objetos através dos quadros do vídeo. Então, se uma pessoa for detectada no quadro 1, ela deve ser acompanhada, ou rastreada, por todos os quadros subsequentes.

Técnicas de rastreamento de objetos podem ser divididas em algumas categorias, das quais as de interesse são: se a técnica rastreia um único objeto (*Single Object Tracking* ou SOT), se a técnica rastreia múltiplos objetos (*Multiple Object Tracking* ou MOT), se a técnica faz rastreamento em tempo real (*online*) ou se a técnica funciona de forma *offline* e só faz o rastreamento em vídeos gravados.

Técnicas que fazem rastreamento de um único objeto geralmente não utilizam técnicas de detecção e nem *Deep Learning*. No primeiro quadro do vídeo o usuário marca na imagem onde se encontra o objeto de interesse informando o seu *bounding box*, daí em diante são utilizadas estratégias tradicionais de processamento de imagens para predizer os movimentos do objeto de interesse através dos quadros do vídeo.

Técnicas MOT são mais complexas, pois não só rastreiam múltiplos objetos em vídeos, mas também rastreiam objetos de tipos diferentes. Essas técnicas geralmente utilizam *Deep Learning* junto a estratégias tradicionais de processamento de imagem para fazer o rastreamento. Também é utilizado a detecção antes do rastreamento em cada quadro, ou a cada x número de quadros.

Técnicas SOT e MOT podem ser *Online* ou *Offline*. Técnicas *Online* fazem o rastreamento dos objetos em tempo real. Essas técnicas são muito utilizadas para processar *streams* de câmeras. As técnicas *Offline* fazem o processamento de vídeos depois dos vídeos terem sido obtidos. Elas tendem a atingir mais precisão no rastreamento, porém

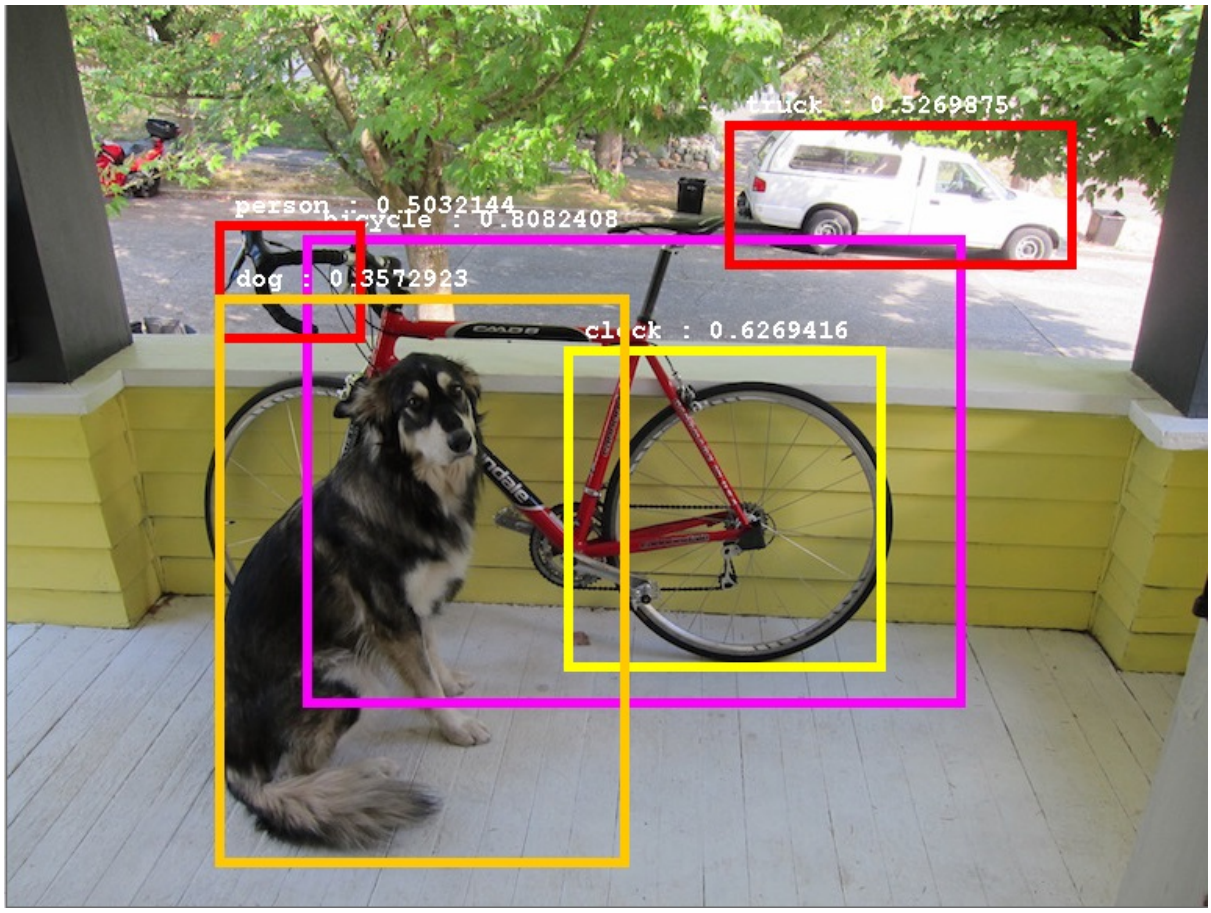

Figura 2.5: Objetos e suas *bounding boxes*. Fonte: [20]

ao custo da rapidez das predições.

2.3 FAMÍLIA FASTERRCNN

A FasterRCNN [12] é a terceira de uma linhagem de três técnicas de detecção de objetos em imagens utilizando CNNs. A primeira é a R-CNN [21] e a segunda é a FastRCNN [22]. Como os próprios nomes indicam, a FasterRCNN utiliza conceitos da FastRCNN, que por sua vez utiliza conceitos da R-CNN. Portanto, para um melhor entendimento da FasterRCNN é necessária uma visão geral sobre as técnicas R-CNN e FastRCNN.

2.3.1 R-CNN

A R-CNN é uma técnica de detecção de objetos em imagens que funciona em três passos. A Figura 2.6 os ilustra. O primeiro é a geração de regiões propostas. Essas regiões são candidatas a conter objetos. O segundo é a passagem de cada região por uma CNN para que seja extraído um conjunto valores, chamado de vetor de características, de cada uma das regiões propostas no passo anterior. É importante notar que os vetores

gerados pela CNN tem tamanho fixo. O terceiro passo consiste de passar os vetores de características de cada região proposta para um conjunto de SVMs (*Support Vector Machines*), em que cada SVM comporta uma classe de objeto, como na Figura 2.6 item 4. É importante notar que os passos 2 e 3 são extremamente parecidos com o processo de classificação de imagens mostrados com a VGG16 na Seção 2.1.2, substituindo o conjunto de SVMs por uma rede neural *fully connected*, pois essa é uma diferença importante entre a R-CNN e a FastRCNN.

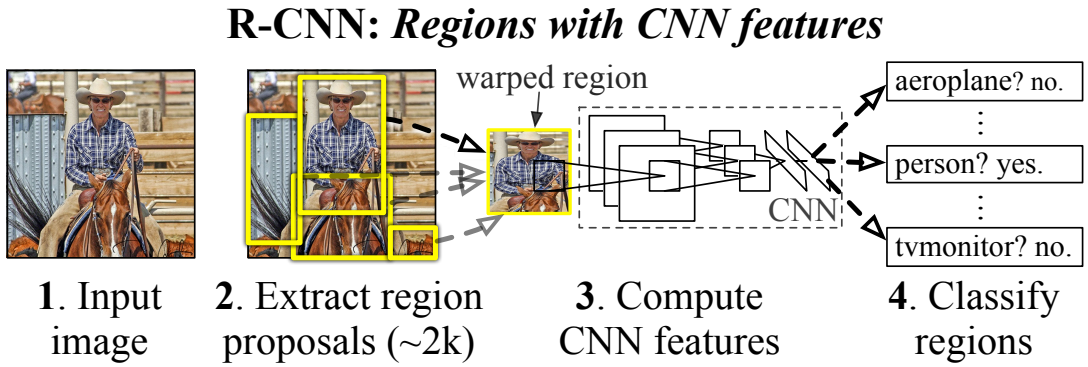

Figura 2.6: Funcionamento R-CNN. Fonte: [21]

2.3.1.1 Geração das Regiões Propostas

Qualquer algoritmo para extração de regiões propostas pode ser utilizada pela R-CNN por causa de sua construção modularizada. O algoritmo utilizado na R-CNN é o *Selective Search* [23]. *Selective Search* é um algoritmo que utiliza abordagens tradicionais de processamento de imagens, nomeadamente segmentação, para encontrar áreas na imagem que sejam candidatas a conter um objeto.

2.3.1.2 Extração do Vetor de Características

Para a extração dos vetores de características das regiões propostas encontradas na imagem com o algoritmo *Selective Search*, a técnica R-CNN utiliza a AlexNet [24]. Cada região proposta passa por essa rede. A única diferença é que a AlexNet foi treinada para classificar imagens em um conjunto de 1000 classes, portanto a última camada *fully connected* é eliminada e o tamanho do vetor de características fica 4096, que é a saída da penúltima camada *fully connected* na AlexNet.

2.3.1.3 Classificação das Regiões Propostas

A terceira e última etapa da R-CNN é a classificação. Nessa etapa são utilizadas SVMs individualmente treinadas para cada classe de objetos (como no item 4 da Figura 2.6).

2.3.2 FastRCNN

A R-CNN tem uma arquitetura modular, o que facilita na utilização de diferentes algoritmos em cada módulo. Essa arquitetura, no entanto, faz com que o treinamento dessa técnica tenha que ser feito de forma desconexa, em várias etapas, causando uma perda de desempenho e também de qualidade nas predições.

A Figura 2.7 ilustra o funcionamento da FastRCNN [22]. O primeiro passo da R-CNN ainda é executado, portanto o processo do exemplo assume que as regiões propostas já foram calculadas. Depois do cálculo das regiões propostas, a imagem é passada para uma CNN para extração de um mapa de características. No mapa extraído da imagem são mapeadas as coordenadas das regiões propostas, e para cada região proposta mapeada no mapa de características da imagem, é aplicada uma camada de ROI (*Region of Interest*) *pooling*. O resultado do *pooling* é passado para algumas camadas *fully connected* que geram um vetor de características. Esse vetor é passado a outras duas camadas *fully connected*. Uma é um classificador que tem saída igual ao número de classes mais uma, que é a classe *background*; esse classificador utiliza o algoritmo *softmax* para responder a confiança de que um objeto pertença a uma classe. A outra camada é um regressor que faz um refinamento nas coordenadas da região proposta em questão.

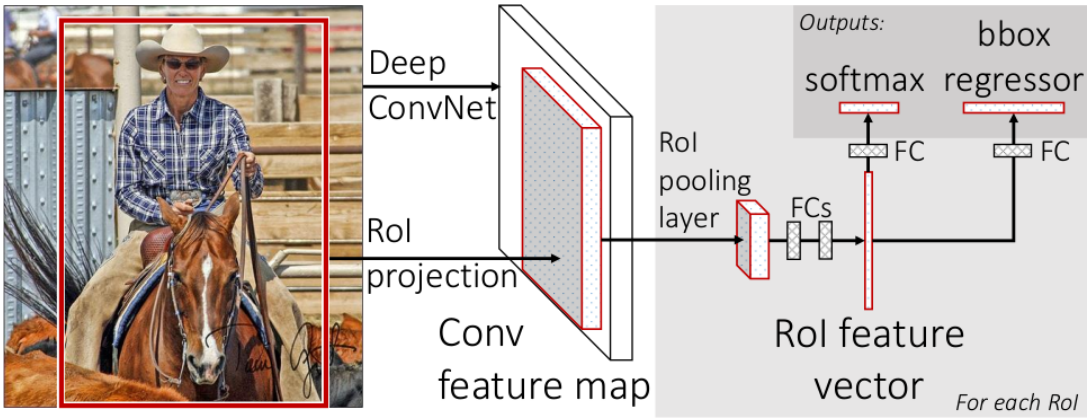

Figura 2.7: Funcionamento FastRCNN. Fonte: [22]

Antes de entrar em cada parte do funcionamento da FastRCNN, é importante destacar as diferenças arquiteturais em relação a R-CNN. A primeira é que na R-CNN era executada uma passagem na CNN para cada região proposta na imagem; na FastRCNN só é feita a passagem da imagem na CNN. Outra diferença é a camada de *pooling* aplicada a seção mapeada de cada região proposta no mapa de características da imagem. Por fim, em vez do conjunto de SVMs para classificação do vetor de características da região proposta, a FastRCNN tem um regressor para o refinamento da região proposta e um classificador *softmax* para classificação do objeto contido na região.

2.3.2.1 Camada de ROI *Pooling*

A camada de ROI *pooling* recebe uma seção do mapa de características correspondente

a uma região proposta. A Figura 2.8 ilustra um suposto mapa de características de uma imagem.

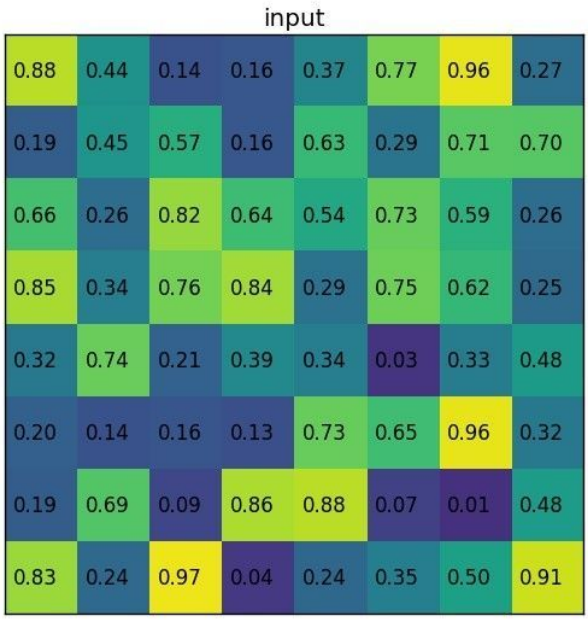

Figura 2.8: Suposto mapa de características de uma imagem. Fonte: [25]

A Figura 2.9 ilustra uma seção do mapa de características de uma imagem correspondente a uma região proposta. A camada de ROI *pooling* é encarregada de dividir essa seção em um número fixo de sub-seções (4 na Figura 2.10), e executar o *max pooling* em cada sub-seção.

Portanto, como a divisão da região proposta é sempre um número fixo, toda e qualquer região proposta que passe da camada de ROI *pooling* terá um número fixo de valores na saída que pode ser utilizado como entrada para as camadas *fully connected* e por consequência resultarem em um vetor de características.

2.3.2.2 Camada de Saída

Na saída, a FastRCNN substitui o conjunto de SVMs da R-CNN por duas camadas *fully connected*, uma para classificar a região proposta, e outra para refinar as coodenadas da região proposta na imagem. É importante notar que esse é um grande trunfo da FastRCNN em comparação a R-CNN, pois em vez de o trenamento da técnica precisar ser feito em etapas diferentes para cada módulo como na R-CNN, na FastRCNN é possível fazer o treinamento de forma fim a fim, pois é possível utilizar o *backpropagation* para propagar o erro da classificação e da regressão de volta para a rede, corrigindo os seus pesos.

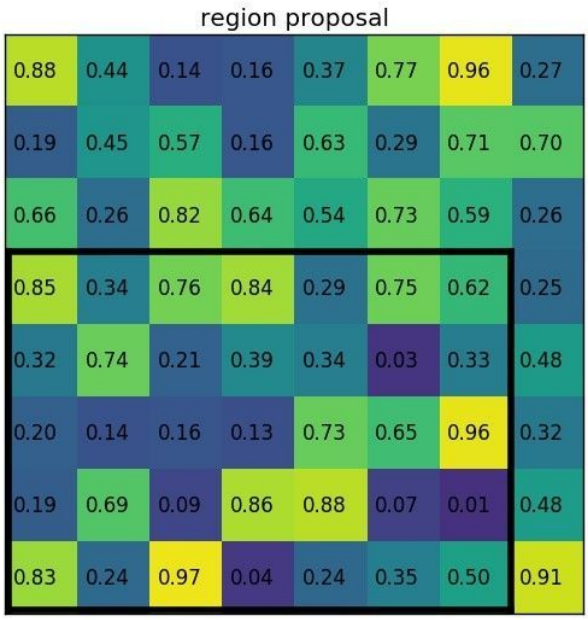

Figura 2.9: Seção do mapa de características de uma imagem correspondente a uma região proposta. Fonte: [25]

2.3.3 FasterRCNN

A FastRCNN tem uma arquitetura que possibilita o treinamento fim a fim com o *backpropagation*, por isso consegue bons resultados e bom desempenho no treino e no teste. Porém o passo inicial tanto da FastRCNN, quanto da R-CNN é baseado em um algoritmo que é o *Selective Search*, que se utiliza de abordagens tradicionais de processamento de imagens, nomeadamente segmentação, para encontrar regiões candidatas a conterem objetos na imagem. A FasterRCNN utiliza uma nova abordagem, chamada de RPN (*Region Proposal Network*), para encontrar essas regiões.

A Figura 2.11 ilustra o funcionamento da técnica FasterRCNN [12]. O processo de extração dos mapas de características da imagem com uma CNN e o processo da camada de ROI *pooling* e das camadas de classificação e regressão de saída são iguais aos do FastRCNN. O funcionamento segue três passos. No primeiro a imagem é passada à uma CNN (neste trabalho a ResNet50 [26]) para a extração dos mapas de características. No segundo passo a RPN utiliza os mapas de características para encontrar as regiões propostas. O terceiro passo é simplesmente rodar a FastRCNN com mapas de características e com as regiões propostas.

2.3.3.1 RPN

A Figura 2.12 ilustra o funcionamento da RPN. A RPN é uma rede totalmente con-

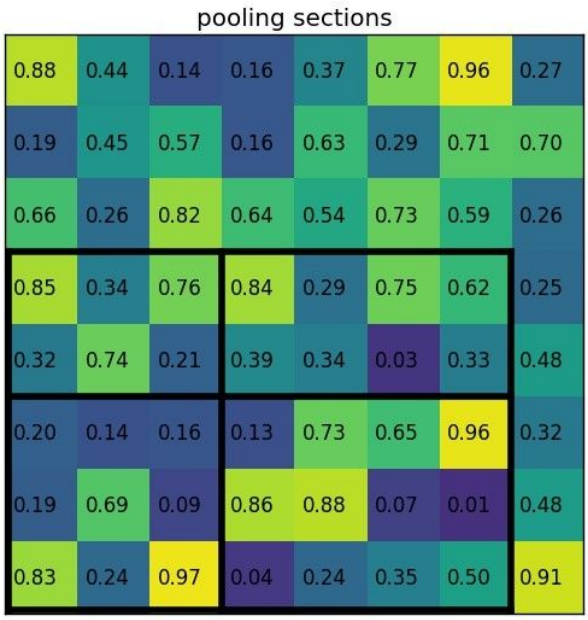

Figura 2.10: Divisão da região proposta em quatro sub-seções. Fonte: [25]

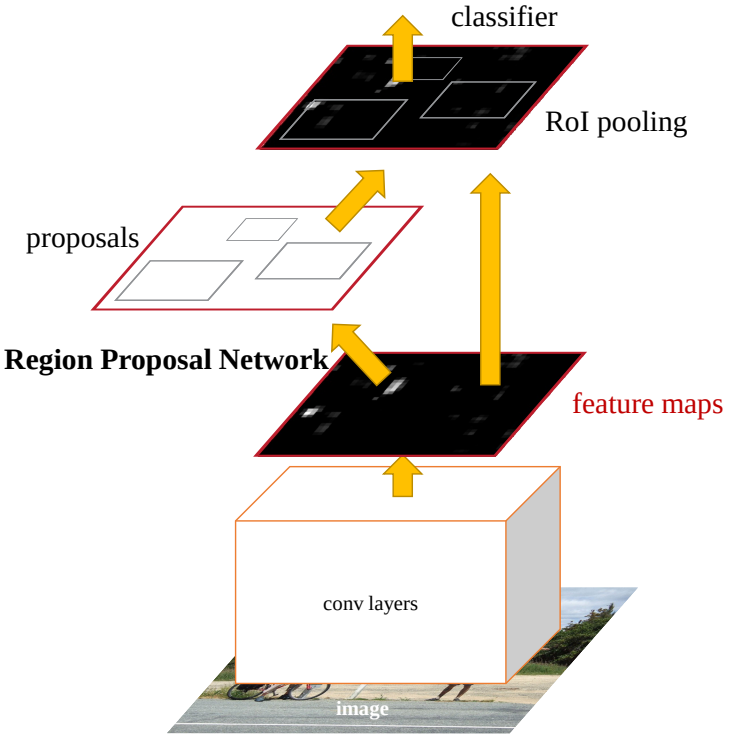

Figura 2.11: Funcionamento FasterRCNN. Fonte: [12]

volucional que é aplicada a cada elemento da imagem num formato de janela deslizante, que vai percorrendo espacialmente todo o mapa de características gerado pela ResNet50, portanto é importante enfatizar que todos os elementos do mapa utilizam a mesma RPN. Em cada elemento do mapa, uma convolução 3x3xd, em que d é a quantidade de canais do mapa de características, é executada e seguida por uma função de ativação ReLU. Essa convolução gera um vetor de dimensão 512 (no caso da VGG16 [17]).

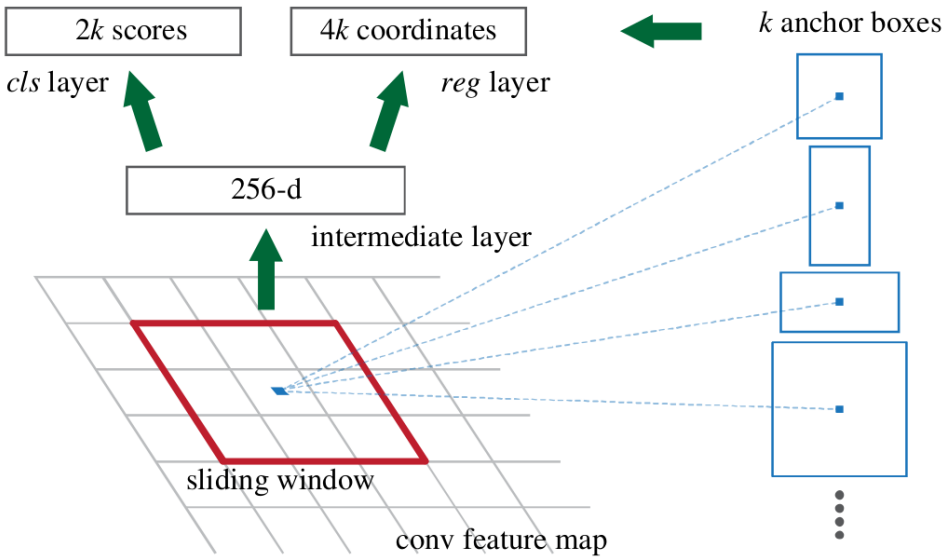

Figura 2.12: Funcionamento RPN. Fonte: [12]

Em cada elemento do mapa são gerados k (igual a 9 em [12]) regiões chamadas de *anchor boxes*. Essas *anchor boxes* tem tamanhos e proporções diferentes entre si, mas são iguais entre os elementos do mapa de características.

No treinamento, com as *anchor boxes* é calculado o IoU (*Intersection Over Union*) em relação as *ground truth boxes* (gt), em que as *gt boxes* são as regiões que é sabido do conjunto de dados que contêm objetos. O IoU é calculado dividindo a intersecção pela união entre uma região e outra, como ilustra a Figura 2.13. Se o IoU entre duas regiões for perto de 1, a sobreposição entre as regiões é alta, e se for perto de 0, a sobreposição é baixa. O IoU é calculado entre todas as *anchor boxes* e *gt boxes*. Se uma *anchor box* tem um IoU maior que $0,7$ com alguma *gt box*, ela é dita um exemplo positivo. Se uma *anchor box* tem um IoU menor que $0,3$ com alguma *gt box*, ela é dita um exemplo negativo. Toda e qualquer *anchor box* com IoU entre $0,3$ e $0,7$ é descartada para o treino. As *anchor boxes* positivas são parametrizadas em relação aos *gt boxes* que cada uma sobrepõe. Essa parametrização é feita pelo do regressor. O regressor é treinado de forma a retornar um *offset* que corresponde a diferença entre as coordenadas da *anchor box* e da *gt box*, mais especificamente, a diferença nas posições espaciais do centro (x, y) e da largura e altura (w, h). Simultaneamente é feita a classificação das *anchor boxes*. Essa classificação é chamada de *objectness*, e ela diz se uma *anchor box* contém ou não um objeto, mas não a classe do objeto. Para que não haja viés entre exemplos positivos e negativos no treino, são utilizados 128 exemplos positivos e 128 exemplos negativos, e caso não existam 128

exemplos positivos, o conjunto é preenchido com exemplos negativos, de forma que em cada passo do treino sejam utilizados sempre 256 exemplos.

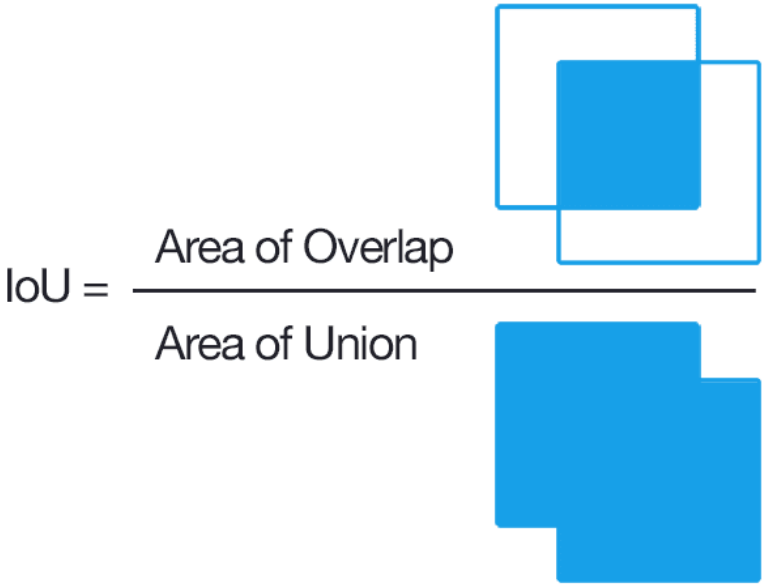

Figura 2.13: *Intersection Over Union*. Fonte: [27]

Portanto a saída da RPN são *anchor boxes* parametrizadas (com o regressor) e a confiança dela conter ou não um objeto. É importante notar que no treino a saída da RPN é controlada, porém em teste, o número de regiões propostas pela RPN pode ser muito grande para processar, além de que deve ocorrer um alto grau de sobreposição entre as regiões propostas. Para diminuir a quantidade de boxes e eliminar a sobreposição, é utilizado o algoritmo *Non-Max Supression*. Com as regiões propostas devidamente filtradas e com o mapa de características obtido anteriormente, o processo continua da mesma forma que com a FastRCNN.

2.4 FAMÍLIA YOLO

A técnica Yolov5 [13], assim como a FasterRCNN, pertence a uma linhagem de técnicas para detecção de objetos em imagens. Comparadas as técnicas da família FasterRCNN, as técnicas Yolo foram criadas de forma que fossem simples em complexidade e consequentemente rápidas. Elas enxergam a detecção de objetos como um problema diferente. A FasterRCNN propõe regiões que podem conter objetos e classifica os objetos que estão dentro dessas regiões, portanto essencialmente para a FasterRCNN a detecção de objetos é um problema de classificação. Já as técnicas Yolo enxergam a detecção de objetos como uma regressão. É uma única CNN que recebe a imagem e retorna as regiões que ela avaliou como contendo objeto, a probabilidade dela conter um objeto e a confiança de que o objeto pertença a uma determinada classe.

2.4.1 Yolo

A técnica Yolo [28] funciona como mostra a Figura 2.14. Primeiro divide a imagem de entrada em um *grid* de formato SxS. Cada célula desse *grid* prediz B *bounding boxes* (regiões propostas). A célula que conter o centro de um *gt box* é responsável por predizer tal *box*. Uma *bounding box* é definida por 5 valores: O centro (dois valores, x e y) relativos a célula, a largura, a altura e a probabilidade da *box* conter um objeto. Essa probabilidade é dada pela IoU entre a *box* predita e qualquer *gt box* que ela venha a sobrepor.

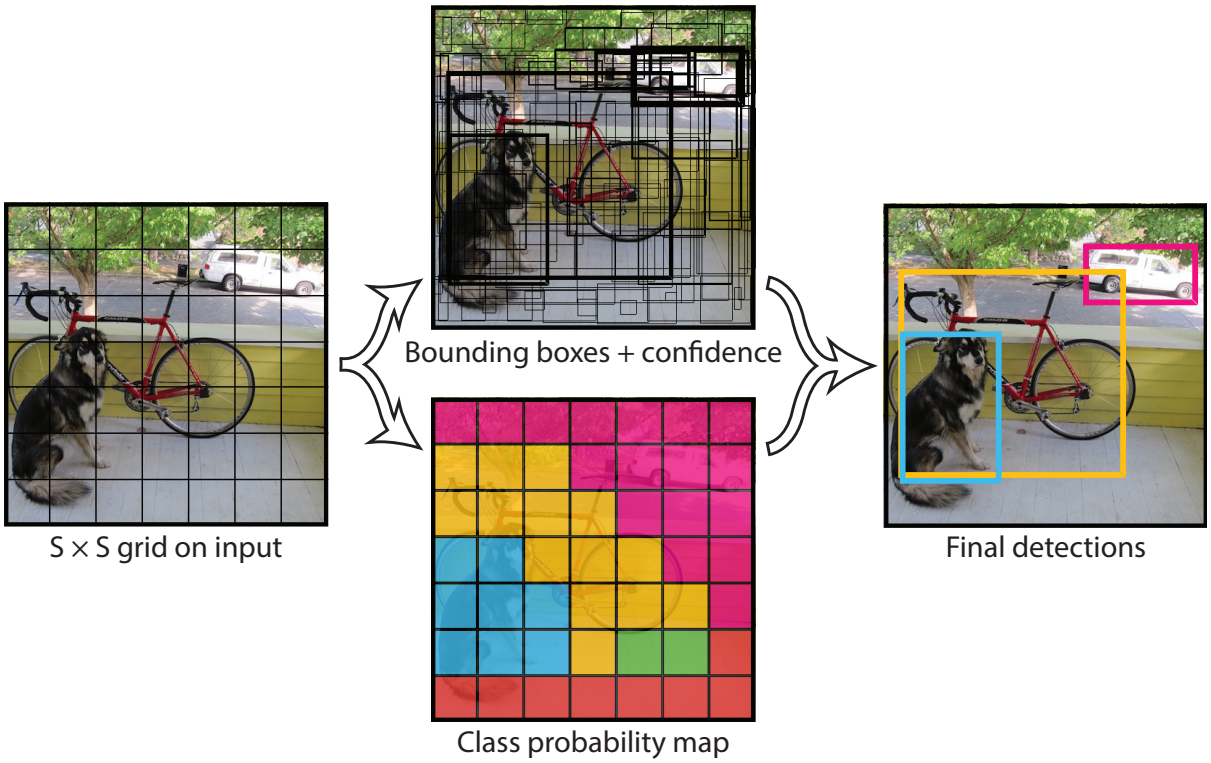

Figura 2.14: Funcionamento Yolo. Fonte: [28]

Os cinco valores mencionados dizem respeito as *bounding boxes* preditas por uma célula, mas além desses valores também são retornados mais C valores, onde C é o número de classes no conjunto de dados. Esses C valores são relativos a célula e eles indicam qual classe de objeto está sobreposta pela célula, ilustrado na Figura 2.14 centro inferior.

Então para cada célula são preditas B *bounding boxes*, cada *bounding box* predita gera 5 valores. Cada célula também gera mais C valores referentes ao número de classes do problema. Portanto a saída da CNN deve ser uma matriz de três dimensões no formato $S\text{x}S\text{x}(B*5+C)$. Tudo isso é feito por uma única CNN, como mostra a Figura 2.15. Essa CNN possui duas camadas *fully connected* no final. No treinamento a rede é primeiro treinada como um classificador, para que ela aprenda a extrair características e essas camadas do final dão lugar a outras camadas *fully connected* que retornem as classes do conjunto de classificação. Num segundo momento ela é treinada como um detector

e consequentemente as camadas *fully connected* da figura voltam a rede. É importante notar que no primeiro momento do treino, o classificador é treinado com uma resolução de imagem menor do que no segundo momento, quando a rede é treinada como um detector.

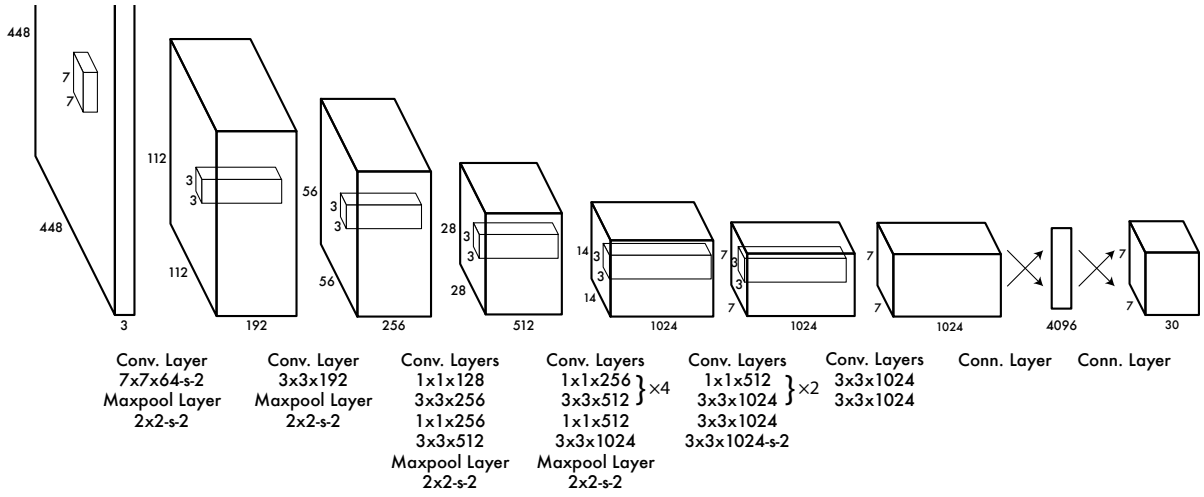

Figura 2.15: CNN Yolo. Fonte: [28]

2.4.2 Yolov2

Algumas melhorias acontecem da técnica Yolo [28] para a Yolov2 [29]. A primeira é a adição de camadas de *batch normalization* depois de toda camada convolucional; isso ajuda a manter os valores dos pesos controlados, o que por consequência evita o *overfitting*.

O treinamento continua tendo dois momentos, o primeiro trata a rede como um classificador, só que dessa vez a resolução das imagens de entrada são iguais a resolução das imagens de entrada do segundo momento, em que a rede é tratada como um detector. Isso faz com que a rede se acostume a uma resolução de imagem maior desde o início do treino.

A técnica Yolov2 utiliza *anchor boxes* para auxiliar nas predições. Diferentemente da FasterRCNN, essas *anchor boxes* são escolhidas baseadas nas *gt boxes* do conjunto de dados, pois no Yolo, o *grid* é fixo, então se escolhidas *anchor boxes* de tamanhos e proporções iguais para cada célula em diferentes conjuntos de dados, é provável que vários objetos sejam perdidos. Essas *boxes* são escolhidas aplicando o *k-means clustering* no conjunto de dados para avaliar quais são as formas mais comuns. No Yolo, são preditas 2 *boxes* por célula, segundo [28]. No Yolov2 são utilizadas 5 *anchor boxes* por célula, então são preditas informações para 5 *bounding boxes*. Com a utilização de *anchor boxes*, os valores preditos mudam em relação a Yolo. Os valores preditos são na Yolov2, *offsets* que parametrizam as *anchor boxes* nas *gt boxes*. É importante notar que na Yolov2 esses valores são preditos em relação a célula a qual a *anchor box* pertence, e que eles são mantidos entre 0 e 1 (depois da aplicação de uma sigmoid), pois isso é fundamental manter os valores que fluem na rede pequenos e comportados de forma que a rede fique

estabilizada e que convirja rápido no treinamento.

Percebe-se que a técnica Yolo tem antes das camadas *fully connected* um mapa de características no formato 7x7x1024, Figura 2.15. A Yolov2 tem um mapa no formato espacial 13x13 em vez de 7x7. Isso é feito por retirando a última camada de *max pooling* e diminuindo a resolução da imagem de entrada para 416x416. A intenção disso é sempre reduzir o tamanho da imagem de entrada em relação ao mapa de características por um fator de 32. Assim qualquer imagem de tamanho divisível por 32 pode ser uma entrada para a rede.

Outra mudança é que o mapa de características, além de contar com a saída da última camada convolucional já mencionada (13x13x1024), também conta com a saída de uma camada intermediária de dimensões 26x26x512, Figura 2.16. Isso ajuda na detecção de objetos menores, que podem se perder nas convoluções quando as dimensões do mapa de características são pequenas em relação as dimensões originais da imagem.

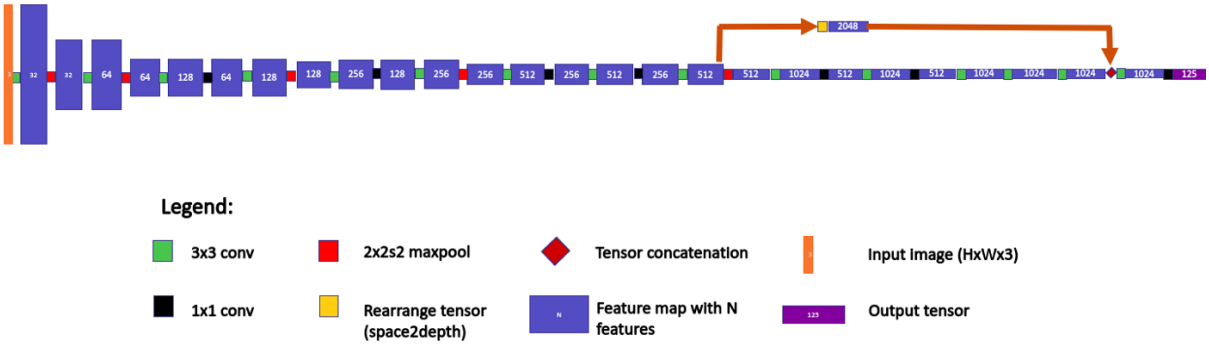

Figura 2.16: CNN Yolov2, 20 classes. Fonte: [30]

A CNN utilizada que tem as características mencionadas é a DarkNet-19, Tabela 2.1. Percebe-se que a saída da rede é uma camada convolucional com 1000 filtros, isso porque ela foi treinada no primeiro momento (classificador) com o ImageNet [31], que é um conjunto de dados com 1000 classes.

O treinamento da Yolov2 também é mais robusto. Como o mapa de características tem redução espacial de tamanho fixo em relação a imagem original (original dividida por 32), o treinamento é feito redimensionando as imagens de entrada por um fator de 32 a cada número fixo de épocas, fazendo assim com que o modelo fique mais acostumado a entradas de tamanhos diferentes.

2.4.3 Yolov3

A técnica Yolov3 [32] apresenta o conceito de *objectness* para as *bounding boxes* preditas. Este valor corresponde a confiança que a técnica tem que a *bounding box* contém um objeto (este valor é predito e não calculado). A partir desse conceito, só uma *bounding box* predita é atribuída a cada *gt box*, no caso a com maior *objectness*. Se uma *box* predita não for atribuída a qualquer *gt box*, ela é descartada e ela não contribui para os erros de

Tabela 2.1: Darknet-19 com dimensões de entrada 448x448. Fonte: [29]

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

localização (coordenadas preditas) e nem para o erro de classificação, mas só para o erro de *objectness*.

Outra mudança é na classificação. A técnica Yolov3 não utiliza mais o *softmax* para o cálculo das confianças das classes, pois o *softmax* supõe que o objeto só pertença a uma classe, quando na verdade muitos conjuntos de dados são *multi label* (várias classes por objeto).

A mudança mais importante, sem dúvidas é na arquitetura da Yolov3. Ela faz as predições em diferentes escalas, isto é, em diferentes pontos da CNN, como mostra a Figura 2.17. Percebe-se que predições são feitas em três pontos da rede, e que a segunda predição utiliza informações da primeira, e que a terceira utiliza informações da segunda e consequentemente da primeira. Em cada célula de cada escala são utilizadas 3 *anchor boxes* escolhidas da mesma maneira que a Yolov2. Portanto as dimensões do resultado

de cada escala é igual a NxNx$[3 * (4 + 1 + C)]$, em que N é a quantidade de células do *grid*, 4 é o número de predições de localização, 1 é o valor do *objectness* e C é o número de classes do conjunto de dados.

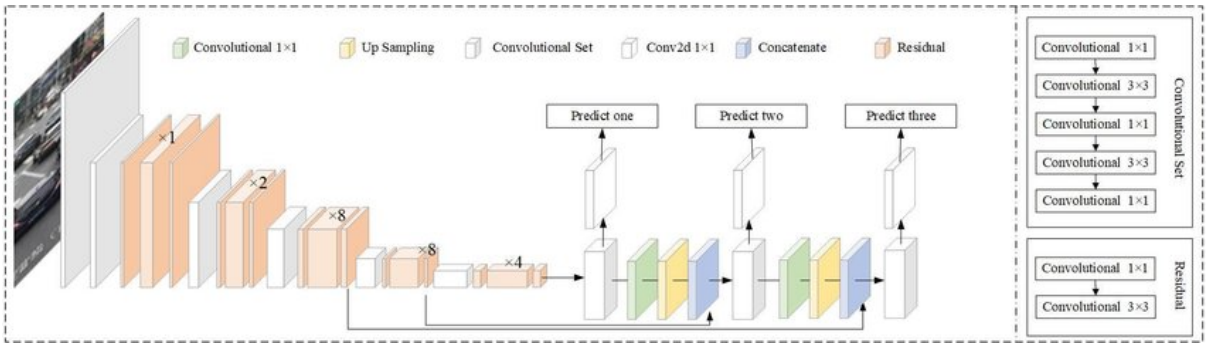

Figura 2.17: CNN Yolov3. Fonte: [33]

Para realizar estas tarefas, a técnica Yolov3 utiliza a DarkNet-53, suas camadas são ilustradas na Figura 2.18. Em comparação com a versão utilizada na Yolov2, ela tem 53 camadas convolucionais em vez de 19. Como mencionado a pouco, ela também tem resquícios residuais.

2.4.4 Yolov4

A técnica Yolov4 [34] é a versão que realiza mais atualizações e mudanças em relação as outras. A progressão entre as versões anteriores em utilizar mapas de características em momentos diferentes do extrator de características é mantida. São adicionados também mecanismos de regularização e normalização dos valores que fluem na rede, além de mecanismos novos de *data augmentation*. A última parte do Yolov3, que faz as predições baseadas no mapa de características resultante, continua ativa na Yolov4. A técnica Yolov4 pode ser dividida em três partes: *Backbone*, *Neck* e *Head*.

2.4.4.1 Backbone

O *Backbone* é a parte da rede que faz a extração de características da imagem, similiar ao que a DarkNet-53 faz na Yolov3. De fato, a rede utilizada para isso é uma versão modificada da DarkNet-53 chamada de CSPDarkNet-53. A principal diferença entre as duas é que a última utiliza dois tipos de blocos convolucionais novos, os blocos de CSP (*Cross Section Partial*), que como o nome indica faz um *bypass* da saída de uma parte da rede até outra parte a frente, ou seja, pula uma seção; o segundo tipo de bloco são os blocos residuais, a saída desses blocos são a sua entrada combinada com a sua entrada processada. A adição desses blocos é feita para que a rede consiga absorver e aprender objetos de diferentes tamanhos e proporções, que poderiam a ser perdidos pelas consequentes convoluções feitas sobre um único mapa de características.

No *Backbone* também são aplicadas algumas técnicas de *data augmentation*. A CutMix [35], corta pedaços de imagens e os colam sobre outras imagens, ou simplesmente

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 2.18: DarkNet-53. Fonte: [32]

colocam retângulos pretos aleatórios sobre as imagens, de forma a dar a ideia de um objeto sobreposto por algo ou outro objeto. Outra técnica é a de criar um mosaico com imagens, como ilustra a Figura 2.19.

2.4.4.2 Neck

O *Neck* é a parte da rede responsável por montar o mapa de características resultante que é utilizado pela última parte da Yolov4. Duas técnicas são utilizadas no *Neck*.

A primeira é o bloco SPP (*Spatial Pyramid Pooling*) [36], que fica entre o *Backbone* e a segunda técnica. Como o nome indica este bloco faz um *pooling* no mapa de características que vem da última camada do *Backbone*, com a intenção de fazer a manutenção das características espaciais que podiam ser perdidas se fossem feitas simples convoluções e *max pooling* no mapa de características.

A segunda é a PAN (*Path Aggregation Network*) [37]. A PAN original faz combinações em mapas de características para gerar novos mapas, mantendo certos aspectos dos antigos. Na verdade, a Yolov4 utiliza uma versão modificada da PAN, em que essas combinações entre os mapas são concatenações, como mostra a Figura 2.20. A PAN

Figura 2.19: Mosaico. Fonte: [34]

utiliza o mapa de características obtidos em vários momentos, gerados tanto pelo *Backbone*, quanto pelo bloco SPP e concatena eles de tal forma a manter as características espaciais dos objetos da imagem original. É vital notar a importância dada pela Yolov4 à manutenção dessas características, pois uma das falhas das versões antigas da Yolo era na predição de objetos pequenos ou parcialmente cobertos em imagens.

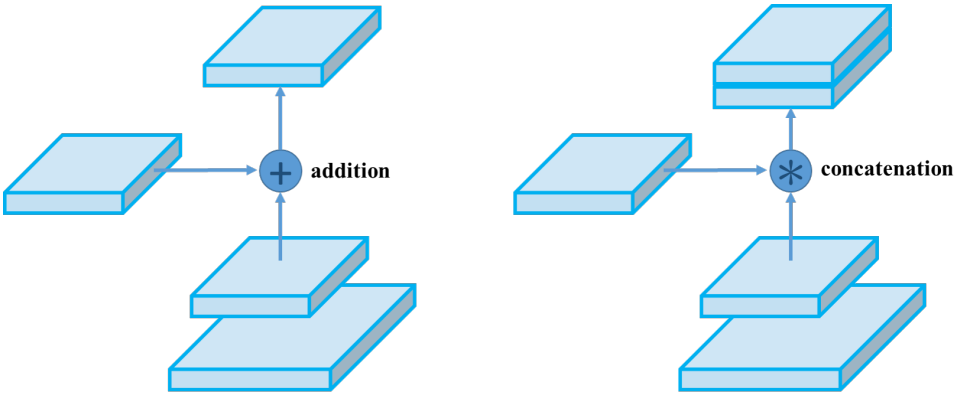

Figura 2.20: PAN original [37], a esquerda e modificada a direita. Fonte: [34]

2.4.4.3 Head

O *Head* é o mesmo esquema de predição dos *offsets* das boxes e das confianças das classes da técnica Yolov3.

2.4.5 Yolov5

Até a data de escrita deste trabalho não foi publicado nenhum artigo formal pelos autores explicando a técnica Yolov5 [13]. Existem muitas controvérsias com a nomeação da técnica de Yolov5, pois ela não foi criada pelos mesmos autores das versões anteriores.

De qualquer forma, a arquitetura da Yolov5 só difere da arquitetura da Yolov4 na questão de quantidade de modelos, o *framework* da Yolov5 oferece cinco opções: Yolov5n (*nano*), Yolov5s (*small*), Yolov5m (*medium*), Yolov5l (*large*) e Yolov5x (*extra large*). Como os nomes indicam, o que varia entre os modelos são seus tamanhos em quantidade de camadas e consequentemente parâmetros na rede.

A Figura 2.21 é uma representação não oficial, mas aprovada pelos autores, da arquitetura da Yolov5. É possível perceber os componentes citados anteriormente da Yolov4, e suas novas alocações na Yolov5. Também é possível perceber como o *Head* (*Output*) é igual ao da Yolov4, que por sua vez é igual ao da Yolov3, e faz a predição em três escalas.

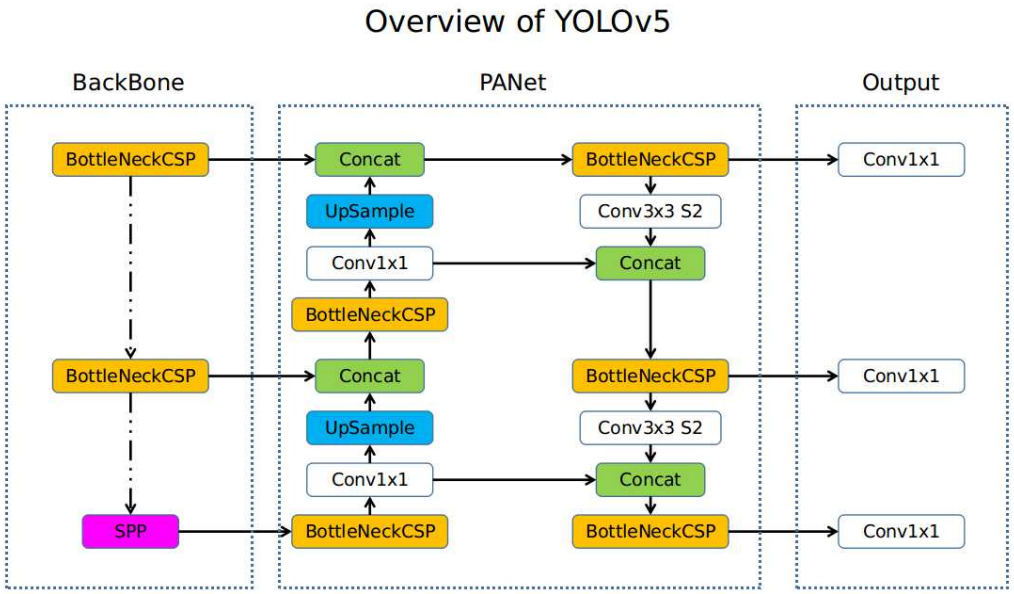

Figura 2.21: Arquitetura da Yolov5 (não oficial). Fonte: [38]

2.5 FAMÍLIA DEEPSORT

DeepSort (*Simple Online and Realtime Tracking with a Deep Association Metric*) [14] é uma técnica de rastreamento *online* que é construída sobre os conceitos da técnica SORT [39] (*Simple Online and Realtime Tracking*). A SORT foi criada pensando em uma abordagem simples, ingênua e extremamente rápida, comparada a outras técnicas

de rastreamento. Ela não tem mecanismos de oclusão, nem leva em conta fatores externos, como velocidade da câmera.

2.5.1 SORT

A técnica SORT [39] pode ser dividida em quatro componentes: a detecção dos objetos a serem rastreados, também chamados de *tracks*, em quadros do vídeo; a estimação da localidade dos *tracks* em quadros subsequentes; a associação das *bounding boxes* preditas e os *tracks* existentes; a criação e remoção de *tracks*.

2.5.1.1 Detecção

Na detecção é utilizada uma técnica de detecção que envolve uma CNN (FasterRCNN em [39]). Uma vez detectados, os objetos são atribuídos estados, em que cada estado é da forma $(u, v, s, r, \dot{u}, \dot{v}, \dot{s})$, sendo (u, v) as coordenadas do centro da *box*, s a área da *box*, r a proporção, e $\dot{u}, \dot{v}, \dot{s}$ as velocidades de aumento ou diminuição de cada valor.

2.5.1.2 Estimação do Deslocamento

Se um objeto predito no quadro atual do vídeo, for associado a um *track* já existente, o estado do *track* é atualizado com os valores da *box* do objeto predito, com os componentes de velocidade sendo calculados através do algoritmo filtros de Kalman [40]. Se um *track* não for identificado entre os objetos preditos no quadro atual, seu estado é atualizado utilizando os componentes de velocidade já existentes.

2.5.1.3 Associação de Objetos a Tracks

A associação é a parte mais elegante da técnica SORT. Ela é resolvida como um problema de otimização utilizando o algoritmo Húngaro. É sabido que esse algoritmo precisa de uma matriz de custo. A matriz de custo é obtida a partir do IoU entre cada *track* predita para o quadro atual do vídeo (utilizando as informações de deslocamento do estado do *track*) e as *boxes* preditas pelo detector. Portanto cada linha da matriz tem o IoU entre o *track* i e todas as *boxes* preditas pelo detector nas colunas.

2.5.1.4 Criação e Remoção de Tracks

Se um objeto predito pelo detector tem IoU menor que um valor mínimo estipulado, ele é adicionado a lista de *tracks* por um período experimental. Esse período exige que esse *track* seja associado a uma *box* predita pelo detector por uma quantidade estipulada de quadros seguidos. Se isso não acontece, ele é "esquecido".

Se um *track* que já passou com sucesso pelo período experimental, ele só é "esquecido"ou removido da lista depois que ele fica uma quantidade de quadros estipulada seguidos sem nenhuma associação.

O IoU mínimo, a quantidade de quadros do período experimental e a quantidade de quadros para a remoção dos *tracks*, são definidos experimentalmente, pois eles são fortemente relacionados ao conjunto de dados.

2.5.2 DeepSort

A técnica DeepSort [14] tem três melhorias notáveis em relação a SORT. A primeira é na estimação do deslocamento dos *tracks* com o filtro de Kalman. A segunda é na associação das *boxes* preditas pelo detector aos *tracks*. A terceira é na criação e remoção de *tracks*. A Figura 2.22 ilustra os componentes do DeepSort utilizando a técnica Yolo como detector.

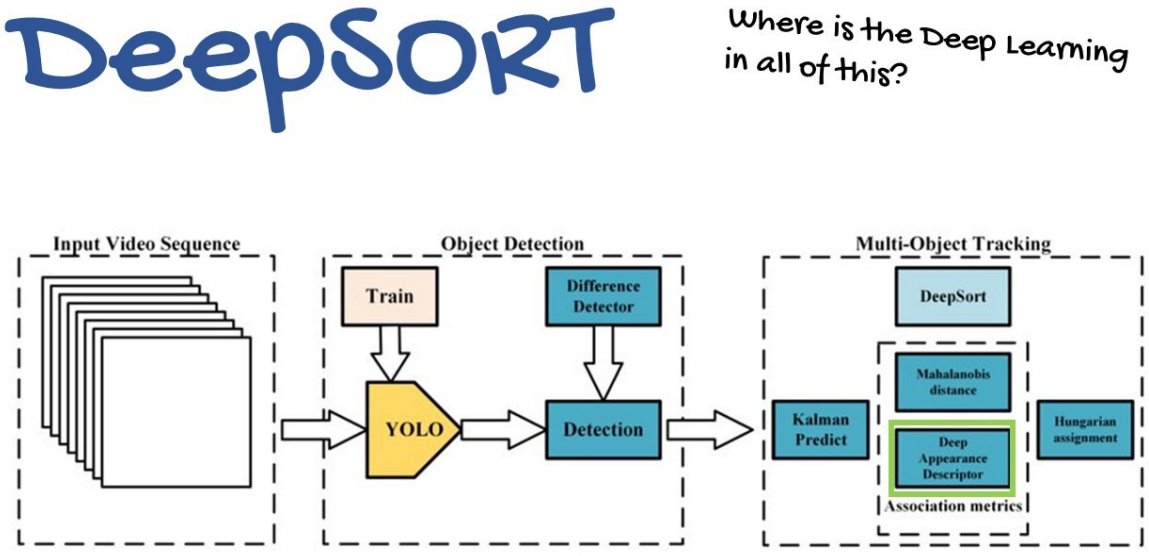

Figura 2.22: DeepSort com detector Yolo. Fonte: [41]

2.5.2.1 Estimação do Deslocamento

Novos valores foram adicionados aos estados das *boxes*, elas agora são definidos pelo formato $(u, v, \gamma, h, \dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$. Os valores (u, v) são o centro da *box*, γ é a proporção da *box* (a proporção no SORT é constante e não calculada com Kalman), h é a altura da *box* e o resto dos são as velocidades de aumento ou diminuição dos valores mencionados calculados por Kalman.

2.5.2.2 Associação de Objetos a Tracks

A associação de objetos preditos pelo detector a *tracks* ainda é visto como um problema de otimização e resolvido a partir da aplicação do algoritmo Húngaro a uma matriz de custos. O que muda é que a matriz de custos é formada de forma diferente. No SORT a matriz de custos era formada pelo IoU entre os *tracks* existentes e os *boxes* preditos pelo detector. Essa é uma medida válida, porém só leva em conta aspectos espaciais dos objetos, então se dois objetos diferentes tem tamanhos parecidos e estão perto entre quadros, existe uma grande possibilidade deles serem confundidos.

O nome DeepSort é justificado justamente neste momento. A matriz de custos agora é formada levando em conta duas métricas, a primeira é a distância de Mahalanobis, na verdade o quadrado dela. Essa distância é calculada entre os *tracks* e os *boxes* preditos pelo detector da mesma forma que o IoU no SORT, e ela leva em conta os aspectos espaciais dos objetos, assim como o IoU. A segunda métrica é a saída de uma CNN chamada em [14] de *Deep Appearance Descriptor*, o que nada mais é do que uma CNN que extrai características de um objeto. Assim a saída da CNN, e o que vai ser combinado com as distâncias de Mahalanobis para a formação da matriz de custos é uma métrica calculada sobre o mapa de características de cada objeto.

É importante notar que o treinamento do *Deep Appearance Descriptor* é feito como o treinamento de um classificador, porém as classes da rede são os objetos individuais do conjunto, e não a categoria do objeto. Portanto, se por exemplo for utilizado para o treino os quadros de um vídeo de pessoas passando em uma faixa de pedestres, cada pessoa que aparece no vídeo deve ser uma classe da rede, e não a categoria pessoa em geral.

2.5.2.3 Criação e Remoção de Tracks

Para a criação de *tracks*, o esquema do período experimental se mantém, mas o número de quadros seguidos é fixado para 3. Portanto se um objeto entra na lista de *tracks*, ele deve ser associado por 3 quadros seguidos para ganhar estadia.

Para a remoção de *tracks*, a mudança foi um pouco mais brusca. A cada *track* é associado um contador. Esse contador conta a quantidade de quadros desde a última associação ao *track* em questão. Ele é incrementado quando o cálculo das velocidades com o filtro de Kalman é feito, e zerado quando o *track* é associado com alguma *box* predita pelo detector. Em [14] se o contador chega a 30, o objeto é "esquecido".

2.6 ESTADO DA ARTE

As técnicas de detecção de objetos em imagens podem ser divididas em dois paradigmas: as técnicas de estágio único e as técnicas de dois estágios. As técnicas de estágio único enxergam o problema de detecção de forma mais ingênua. Elas executam a detecção de uma só vez, tipicamente com uma única CNN, como um problema de regressão, e portanto conseguem um desempenho alto e uma qualidade de predição baixa. As técnicas de dois estágios enxergam o problema de forma mais complexa. Elas primeiro extraem regiões que podem vir a conter objetos em uma imagem e depois classificam essas regiões, como um problema de classificação, e portanto conseguem um desempenho baixo e uma

qualidade alta.

A FasterRCNN [12] explicada na Seção 2.3 é uma técnica em dois estágios. Apesar de ser uma técnica antiga (c. 2015) ela é muito utilizada por ser uma rede que faz predições muito precisas. A Yolov5 [13] explicada na Seção 2.4 é uma técnica de um estágio. Ela é uma técnica recente (c. 2020), e dentre as técnicas derivadas da Yolo original [28] é a mais utilizada; por causa disso está em constante atualização.

A DETR (*Detector Transformer*) [42] é uma técnica que procura eliminar a necessidade da utilização explicita do conhecimento precedente para a detecção de objetos, como por exemplo na utilização de *anchor boxes* e na utilização de *Non-Max Supression*. Ela utiliza um *transformer encoder-decoder* como mecanismo de atenção para fazer o processamento das imagens, e na saída é utilizado um esquema chamado de *bipartite matching loss* para o treinamento, que prediz um número fixo de *boxes*, e verifica quais *boxes* preditas mais se assemelham a quais *gt boxes* utilizando o método Húngaro.

Em [43], os autores propõem um modelo chamado *Swim Transformer*, que utiliza as ideias de *transformers* para servir de *Backbone* (extrator de características) para técnicas de detecção. O conceito principal é tentar incorporar práticas utilizadas por *Backbones* com CNNs, no *Backbone* proposto. Isso é feito dividindo a imagem de entrada em um *grid* (chamado também de *patch*) e passar essas seções por *transformers* utilizando uma janela e deslizando-a por sobre esse *grid*. Esse processo é feito repetidas vezes e no final o resultado é um mapa de características, assim como um *Backbone* com CNNs.

A técnica EfficientDet [44] busca a melhora no paradigma dos detectores de um estágio atráves de uma proposta diferenciada na extração de características da imagem. Similarmente a Yolov5 [13] ela funciona em três etapas: extração de características, combinação dos mapas de características, e predição das regiões e confianças. A EfficientDet difere principalmente na parte da combinação dos mapas de características. Ela utiliza blocos chamados de BiFPN. Esses blocos misturam os mapas extraídos em diferentes momentos de tal forma que características não sejam perdidas e principalmente, que todos os mapas de características tenham importância igual no momento da predição.

O rastreamento de objetos pode ser feito em tempo real (*online*), em vídeos gravados (*offline*), com múltiplos objetos (MOT), com um único objeto (SOT), com a detecção separada do rastreamento ou com a detecção e rastreamento feitos em um único passo. O trabalho atual visa o rastreamento de múltiplos objetos (MOT) e em tempo real (*online*). O DeepSort [14] é uma técnica muito utilizada, explicada na Seção 2.5, que atende a essas requisições.

A MDNet [45] utiliza um pré-treinamento da sua rede, se aproveitando das informações independentes do domínio dos objetos e especificas ao domínio de seus objetos. Com a rede pré-treinada, um treinamento *online* é feito para que a rede aprenda a detectar os objetos. Depois do treino, a rede recolhe amostras ao redor das regiões preditas no quadro anterior, essas regiões são passadas pela rede, as coordenadas são refinadas e o resultado é uma média das cinco melhores predições para cada região. O IoU é utilizado para a associação de identificadores, e se o IoU for maior que um valor especificado para uma determinada *box*, parte da rede é re-treinada com os gradientes calculados para essa *box*. Assim, a rede aprende continuamente.

Na técnica DeepSort as detecções das regiões e a associação dessas regiões com objetos

rastreados são feitas em módulos separados. A técnica JDE (*Joint Detection and Embedding*) [46] faz as duas atividades em um só módulo. O que acontece é que a detecção de regiões e a predição das características dos objetos nessas regiões são feitas por uma única rede, se aproveitando da possibilidade do compartilhamento de recursos.

CAPÍTULO 3

METODOLOGIA

Neste capítulo são explicados todos os passos utilizados para o treinamento das técnicas de detecção e rastreamento, além de uma visão sobre o conjunto de dados, como ele foi manipulado para que os treinamentos fossem possíveis. Também é explicado o método de avaliação e comparação das técnicas de detecção, o mAP (*Mean Average Precision*). Por fim, são dadas as especificações do ambiente utilizado para todos os experimentos.

3.1 CONJUNTO DE DADOS

O conjunto de dados utilizado foi disponibilizado pela NFL para o problema do *Kaggle* [11], em que o desafio é detectar o impacto entre capacetes em vídeos de jogadas de futebol americano. Ele consiste de imagens e vídeos de jogadas, além de anotações dessas imagens e vídeos.

As imagens são retiradas de vídeos, então em termo de ângulos de câmera, os ângulos das imagens são iguais aos dos vídeos. Existem dois ângulos, o das linhas laterais do gramado e o por trás dos gols. A Figura 3.1 ilustra o ângulo atrás dos gol e a Figura 3.2 ilustra o ângulo da linha lateral.

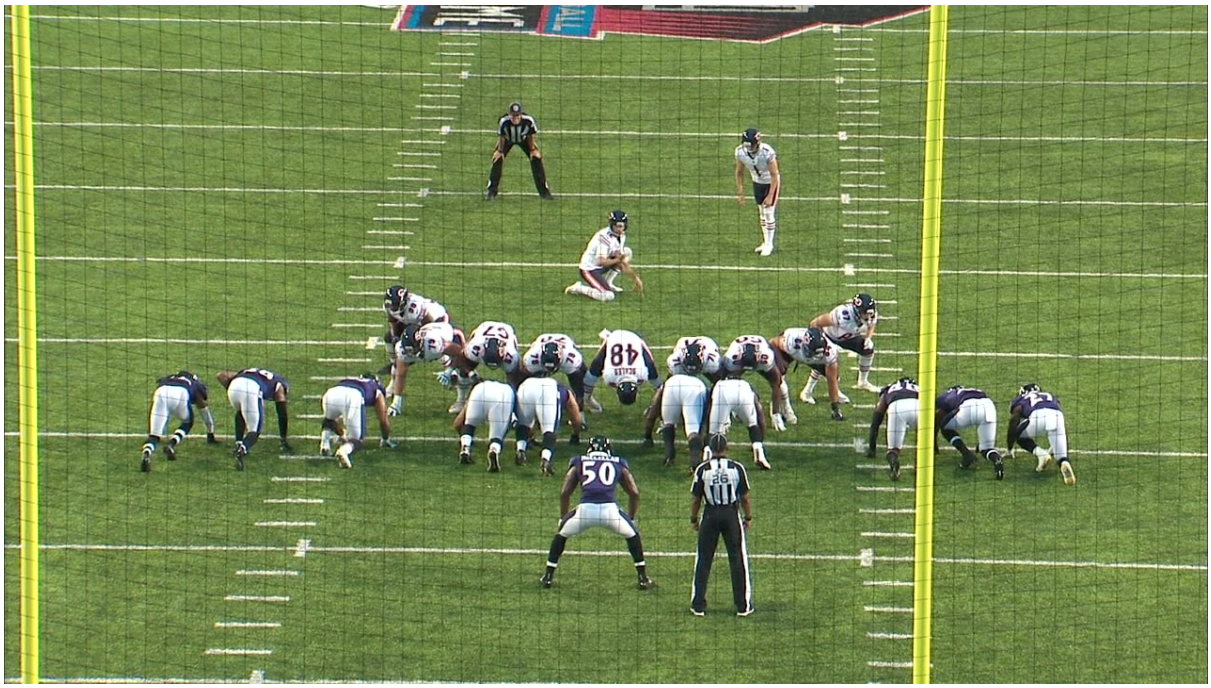

Figura 3.1: Exemplo de imagem com câmera atrás do gol. Fonte: [11]

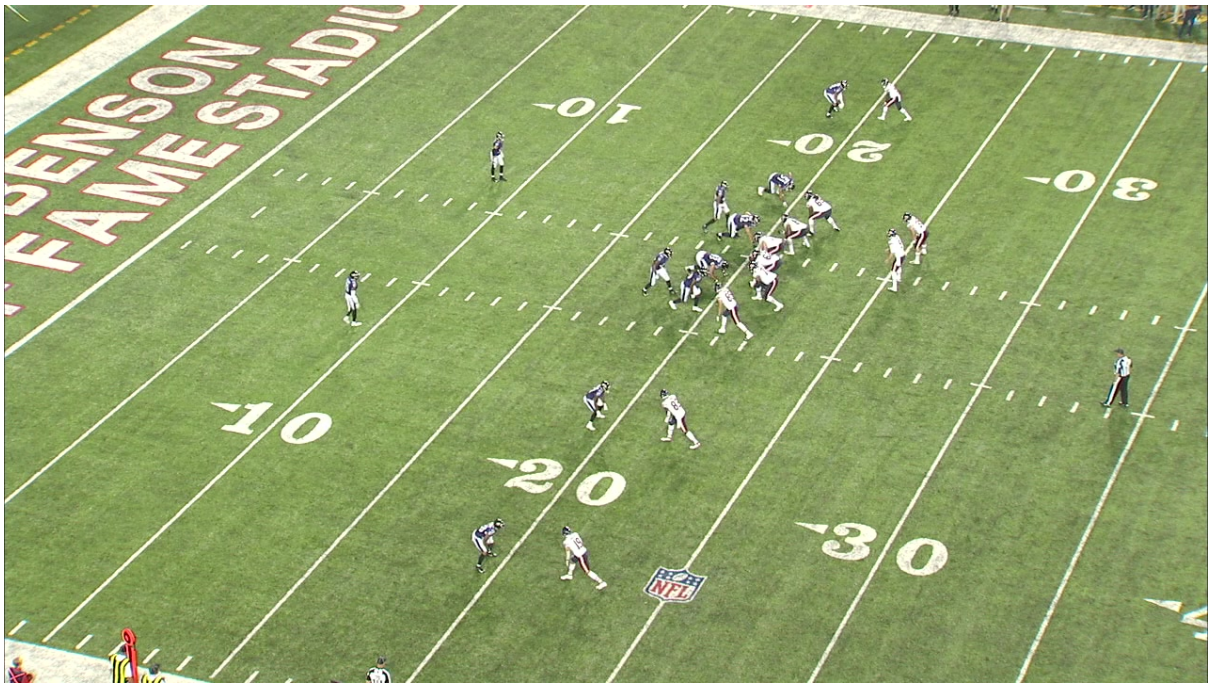

Figura 3.2: Exemplo de imagem com câmera na linha lateral. Fonte: [11]

Durante o jogo os jogadores se movimentam e as câmeras também, portanto o gramado é observado de diferentes maneiras, apesar de só existir dois ângulos de câmera. Isso fica evidente nas Figuras 3.3 e 3.4.

Figura 3.3: Movimentação da câmera em direção a lateral. Fonte: [11]

Figura 3.4: Movimentação da câmera em direção ao fundo. Fonte: [11]

Em todas as imagens mostradas é possível notar também a presença de jogadores nas laterais do gramado com o capacete na cabeça ou até mesmo nas mãos, todos eles são levados em conta.

As anotações da posição dos capacetes nas imagens são ilustradas na Tabela 3.1. A tabela contém da esquerda para direita: o nome do arquivo de imagem, a classe do objeto e o retângulo que localiza o objeto. Existem 5 classes de objetos no conjunto: *Helmet* (capacete), *Helmet-Blurred* (capacete embaçado), *Helmet-Sideline* (capacete visto da lateral), *Helmet-Partial* (capacete parcial) e *Helmet-Diffucult* (capacete difícil). O trabalho atual reduz todas as classes a capacete.

A Tabela 3.2 ilustra as anotações nos vídeos. A tabela contém da esquerda para direita: identificador único do jogo, identificador único da jogada, o nome do arquivo do vídeo, o quadro do vídeo, a classe do objeto e o retângulo que contém o objeto. No caso do vídeo um objeto deve ser identificado unicamente em todos os quadros, por isso cada classe identifica um objeto, e não a categoria do objeto. Todos os objetos são portanto capacetes, mas o objeto $H30$, por exemplo, é o capacete do jogador número 30 do time da casa ($H = Home$, $V = Visitor$).

Todas as imagens e vídeos são RGB com resolução 1280x720.

3.2 TREINAMENTO DAS TÉCNICAS DE DETECÇÃO

O conjunto de imagens contém 9947 imagens no total. Ele foi dividido em três sub-conjuntos, o conjunto de treino, o conjunto de validação e o conjunto de testes. A divisão foi feita escolhendo 70% das imagens para treino, 15% para validação e 15% para teste. As duas técnicas utilizam os mesmos sub-conjuntos para treino, validação e teste.

Tabela 3.1: Porção da tabela com localidade e classe dos capacetes nas imagens. Fonte: [11]

image	label	left	width	top	height
57503_000116_Endzone_frame443.jpg	Helmet	1099	16	456	15
57503_000116_Endzone_frame443.jpg	Helmet	1117	15	478	16
57503_000116_Endzone_frame443.jpg	Helmet	828	16	511	15
57503_000116_Endzone_frame443.jpg	Helmet	746	16	519	16
57503_000116_Endzone_frame443.jpg	Helmet	678	17	554	17
57503_000116_Endzone_frame443.jpg	Helmet	785	16	419	16
57503_000116_Endzone_frame443.jpg	Helmet	711	13	463	15
57503_000116_Endzone_frame443.jpg	Helmet	675	16	454	15
57503_000116_Endzone_frame443.jpg	Helmet	544	16	414	13
57503_000116_Endzone_frame443.jpg	Helmet	654	13	425	14
57503_000116_Endzone_frame443.jpg	Helmet	881	16	329	14
57503_000116_Endzone_frame443.jpg	Helmet	792	16	375	14
57503_000116_Endzone_frame443.jpg	Helmet	709	14	387	12
57503_000116_Endzone_frame443.jpg	Helmet	669	16	359	16
57503_000116_Endzone_frame443.jpg	Helmet	572	16	367	14

Tabela 3.2: Porção da tabela com localidade, classe e identificador único dos capacetes nas imagens. Fonte: [11]

gameKey	playID	view	video	frame	label	left	width	top	height
57583	82	Endzone	57583_000082_Endzone.mp4	1	H30	629	19	40	24
57583	82	Endzone	57583_000082_Endzone.mp4	1	V72	443	22	344	16
57583	82	Endzone	57583_000082_Endzone.mp4	1	V86	871	21	359	17
57583	82	Endzone	57583_000082_Endzone.mp4	1	V74	771	19	345	15
57583	82	Endzone	57583_000082_Endzone.mp4	1	V34	549	26	461	20
57583	82	Endzone	57583_000082_Endzone.mp4	1	H97	410	21	323	29
57583	82	Endzone	57583_000082_Endzone.mp4	1	H99	586	23	327	29
57583	82	Endzone	57583_000082_Endzone.mp4	1	V73	661	16	348	14
57583	82	Endzone	57583_000082_Endzone.mp4	1	V87	959	21	359	15
57583	82	Endzone	57583_000082_Endzone.mp4	1	H96	867	22	321	30
57583	82	Endzone	57583_000082_Endzone.mp4	1	V15	1071	25	298	21
57583	82	Endzone	57583_000082_Endzone.mp4	1	H59	961	20	233	29
57583	82	Endzone	57583_000082_Endzone.mp4	1	H90	245	21	327	29
57583	82	Endzone	57583_000082_Endzone.mp4	1	H56	723	22	228	27
57583	82	Endzone	57583_000082_Endzone.mp4	1	V68	551	8	333	12

Quanto ao *data augmentation*, foi tentado manter as mesmas operações para as duas técnicas. É importante lembrar que a Yolov5 utiliza algumas operações como o CutMix e Mosaico, essas não foram possíveis replicar para a FasterRCNN.

3.2.1 Treinamento da FasterRCNN

O *Framework* utilizado para o treinamento da FasterRCNN foi o PyTorch [47], pela sua simplicidade e desempenho. O modelo escolhido foi o fasterrcnn_resnet50_fpn [48] encontrado no módulo Torchvision do PyTorch. Esse modelo utiliza um extrator de características ResNet50 [26] com uma FPN [49].

Foi utilizada uma rede pré-treinada no conjunto de dados COCO [50]. O conjunto COCO tem 90 classes sem contar com o *background*, que na FasterRCNN deve ser contabilizada, portanto as últimas camadas da rede foram trocadas para conter o número correto de saídas para o conjunto utilizado neste trabalho (2 classes: *background* e *helmet*). O treinamento foi feito somente nas camadas trocadas da rede, com o resto da rede ficando "congelada". O processo de treinamento é ilustrado na Figura 3.5.

O tamanho do *batch* utilizado foi 16, pois era o máximo que cabia na GPU. O otimizador utilizado foi o SGD, com a taxa de aprendizagem $0,001$; *momentum* $0,9$ e *weight decay* $0,001$.

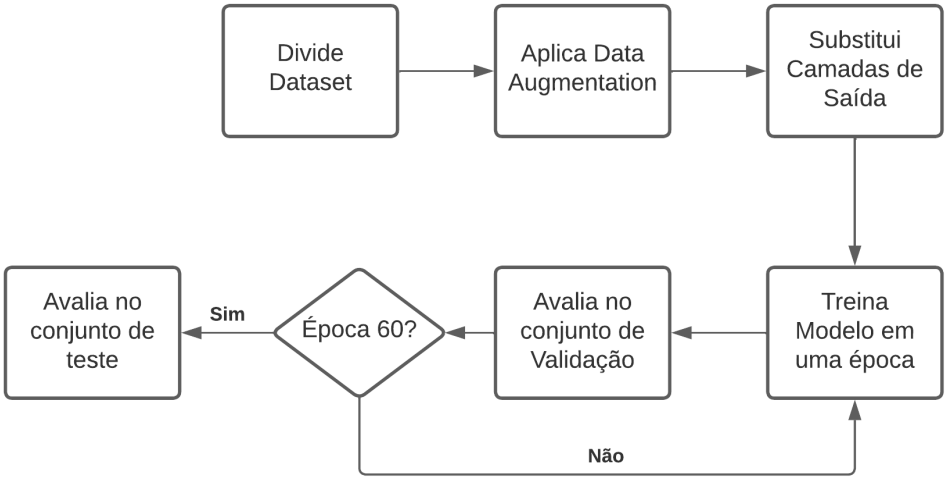

Figura 3.5: Diagrama de treinamento da FasterRCNN.

3.2.2 Treinamento da Yolov5

O treinamento da Yolov5 foi executado utilizando os *softwares* disponibilizados em [13]. O modelo escolhido foi o Yolov5m6, que é o modelo de tamanho médio. Foi utilizado um modelo pré-treinado no COCO [50] (assim como a FasterRCNN), com as imagens de entrada com dimensões 1280x1280. No treino o tamanho de entrada das imagens da rede foi reduzido para 640x640, para que o treinamento ocorresse de forma mais rápida. O tamanho do *batch* foi 32, que foi o máximo que cabia na GPU.

A quantidade de filtros nas camadas convolucionais de saída também são reduzidos para que correspondam a uma predição de apenas uma classe, similar ao que acontece com a FasterRCNN. A Figura 3.6 ilustra o processo.

O otimizador SGD foi utilizado com uma taxa de aprendizagem $0,01$; *momentum* $0,937$ e *weight decay* $0,0005$.

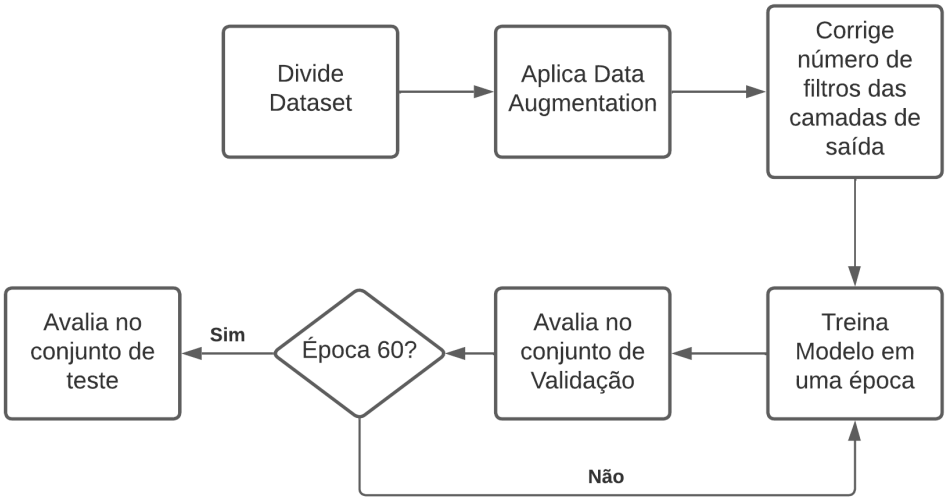

Figura 3.6: Diagrama de treinamento da Yolov5.

3.3 TREINAMENTO DA TÉCNICA DE RASTREAMENTO

A técnica DeepSort utiliza dois modelos de *Deep Learning*, um para a detecção e outro para a extração de características dos objetos. O modelo de detecção utilizado foi o Yolov5m6. O classificador escolhido foi a rede ResNet50 pré-treinada na ImageNet [51].

O treinamento desse classificador foi baseado nos vídeos disponíveis, e não nas imagens. Como mostrado na Seção 3.1, cada quadro de um vídeo possui as informações sobre todos os objetos no vídeo identificados unicamente. Foram utilizados 14 vídeos, 7 com ângulo do fundo do campo e 7 com o ângulo da lateral do gramado. Esses vídeos contêm 126 objetos diferentes, que foram divididos 88 para treino e 38 para validação. Ao todo existem 20513 exemplos desses 126 objetos, então em média são aproximadamente 162 exemplos para cada capacete. A rede foi ensinada a reconhecer cada um desses objetos, com esses 162 exemplos, aprendendo dessa forma quais os padrões e tendências dos movimentos de cada um dos capacetes em termos de estética e geometria.

Para esse treinamento foi utilizado o *Framework* TorchReID [52]. As dimensões da imagem de entrada na rede foi 128x256, com um tamanho do *batch* 32, a taxa de aprendizagem utilizada foi $0,0003$; com um otimizador Adam e um *learning rate scheduler* com tamanho do passo 20.

3.4 AVALIAÇÃO DOS MODELOS - MEAN AVERAGE PRECISION (mAP)

A métrica utilizada para avaliação dos modelos de detecção foi o mAP (*Mean Average Precision*). Para calcular o mAP, o AP (Average Precision) é calculado para cada classe

e a média é o mAP, mas como o problema atual só tem uma classe, o mAP será igual ao AP.

Primeiro são calculadas a quantidade de regiões que são *True Positive* (TP), *False Positive* (FP) e *False Negative* (FN).

- Uma *box* predita é TP se o IoU dela com alguma *gt box* for maior que um limiar previamente definido;
- Uma *box* predita é FP se o IoU dela com qualquer *gt box* não passar de um limiar previamente definido;
- FN é a quantidade de *gt boxes* que não foram preditas.

O limiar mencionado é definido para cada problema. O valor clássico escolhido e o utilizado no trabalho atual é de $0,5$; chamado de mAP@0,5. Porém alguns autores também utilizam uma média dos mAP como métrica, chamado mAP@0,5:0,95; isso quer dizer que o mAP é calculado com o limiar de $0,5$ a $0,95$ com incrementos de $0,05$.

Depois do calculo dos TP, FP e FN, são calculadas os *Precision* e *Recall*. O *Precision* é dada pela equação 3.1:

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

O *Precision* indica o quanto o modelo acerta entre o conjunto de *boxes* preditas por ele. O *Recall* é dado pela equação 3.2 e indica o quanto o modelo acerta entre todas as *gt boxes*.

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

O próximo passo é a construção da curva *Precision* x *Recall*. Essa curva é construída a partir dos TP e FP acumulados a cada objeto, por exemplo, no décimo objeto, o *Precision* é calculado considerando todos os objetos que vieram antes (se eles são TP ou FP). Dessa forma o AP é obtido como a área sob a curva.

3.5 AMBIENTE

O ambiente utilizado foi uma máquina na GCP (*Google Cloud Platform*), com 8 CPUs vituais, 30 GB de memória RAM e uma GPU NVIDIA Tesla K80. Todos os treinamentos foram executados nessa máquina, e algumas inferências foram executados no Google Colab.

CAPÍTULO 4

EXPERIMENTOS E RESULTADOS

Neste capítulo são mostrados todos os experimentos feitos, desde o treinamento até o teste em imagens e vídeos com outros tipos de objetos além de capacetes, para um maior entendimento sobre os modelos. Também é executada uma comparação estatística entre os modelos e discutida razões por trás das diferenças em suas performances.

4.1 TREINAMENTO FASTERRCNN

Como mencionado na Seção 2.3, a FasterRCNN é uma técnica de dois estágios, portanto existem quatro fontes de erro que podem ser utilizadas no *backpropagation*, os erros de *objectness* (classificação) e de regressão da RPN, e os erros de classificação e regressão da camada de saída do modelo.

A Figura 4.1 mostra o erro de *objectness* da RPN da FasterRCNN. É possível perceber uma grande variação especialmente na etapa inicial do treinamento. O valor na última época foi $0,06$.

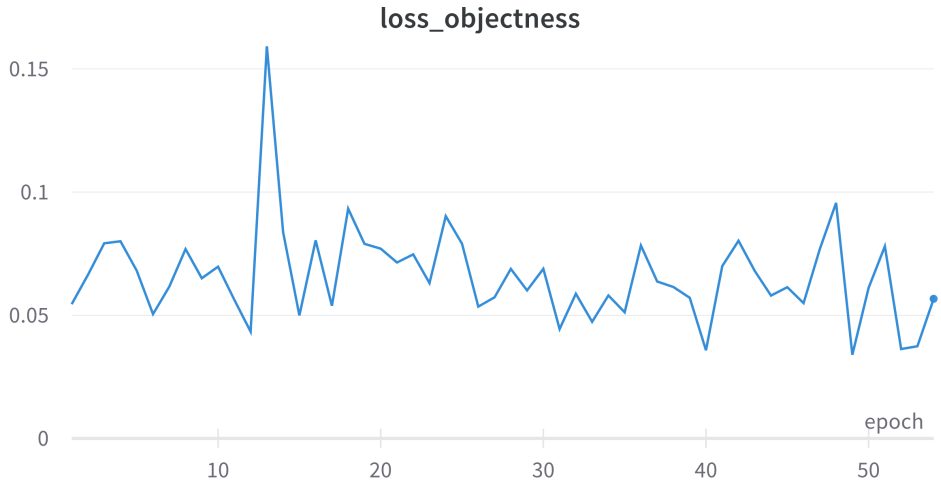

Figura 4.1: Erro de *objectness* RPN FasterRCNN.

O erro de regressão na RPN é mostrado na Figura 4.2. Uma variação do começo para o meio do treinamento também é observada. Na última época o valor do erro ficou $0,11$.

A Figura 4.3 mostra o erro de classificação da camada de saída da FasterRCNN. O valor do erro na última época ficou $0,16$.

O erro de regressão da camada de saída é ilustrado na Figura 4.4. O valor na última época foi $0,44$.

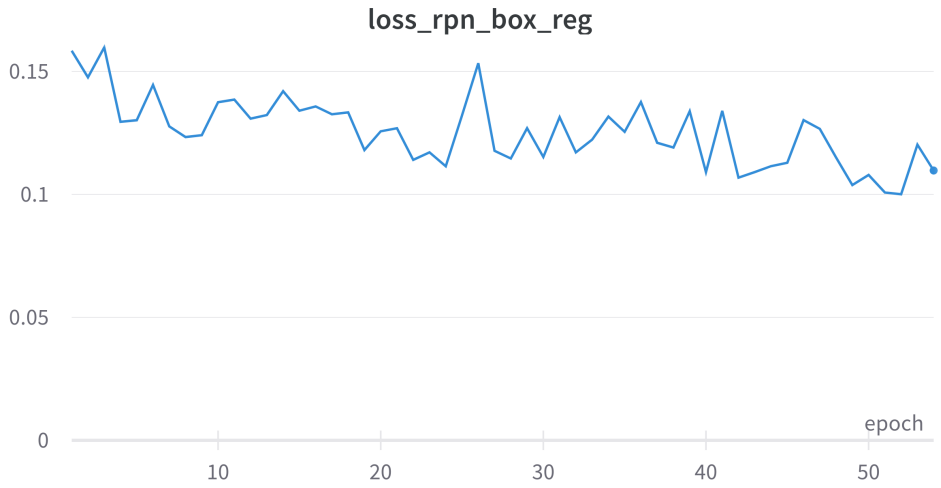

Figura 4.2: Erro de regressão RPN FasterRCNN.

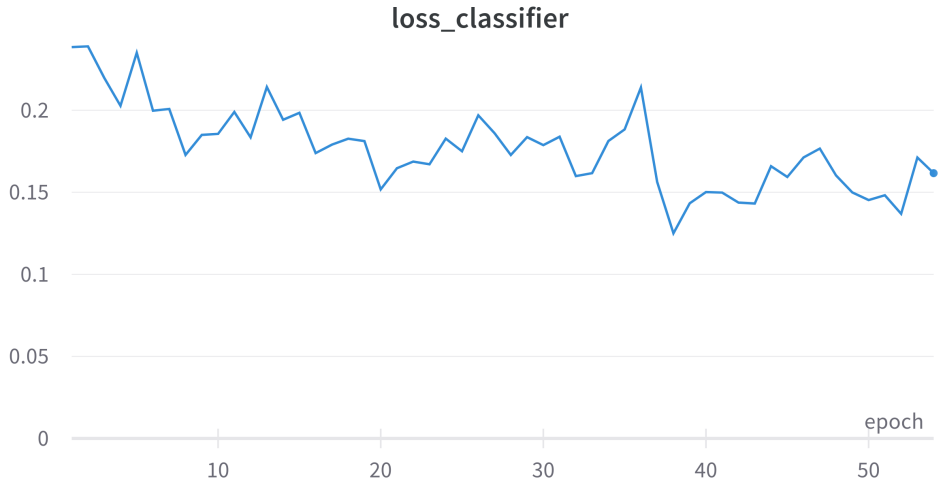

Figura 4.3: Erro de classificação da camada de saída FasterRCNN.

A progressão do mAP do conjunto de validação no treinamento é ilustrada na Figura 4.5. O valor na última época foi de $0,75$. Fica evidente por todos os gráficos do treinamento que o modelo poderia ter treinado mais, porém como é possível observar na progressão do mAP, o modelo estava aprendendo. Como a FasterRCNN é um modelo de dois estágios, ele é mais lento e um treinamento apropriado demanda mais tempo.

4.2 TREINAMENTOS YOLOV5

Diferentemente da FasterRCNN, a Yolov5 é uma técnica de detecção de um estágio, portanto só existem três tipos de erros, o erro de *objectness*, o erro de regressão e o erro de classificação. O erro de *objectness* difere do erro de classificação no fato de que *objectness*

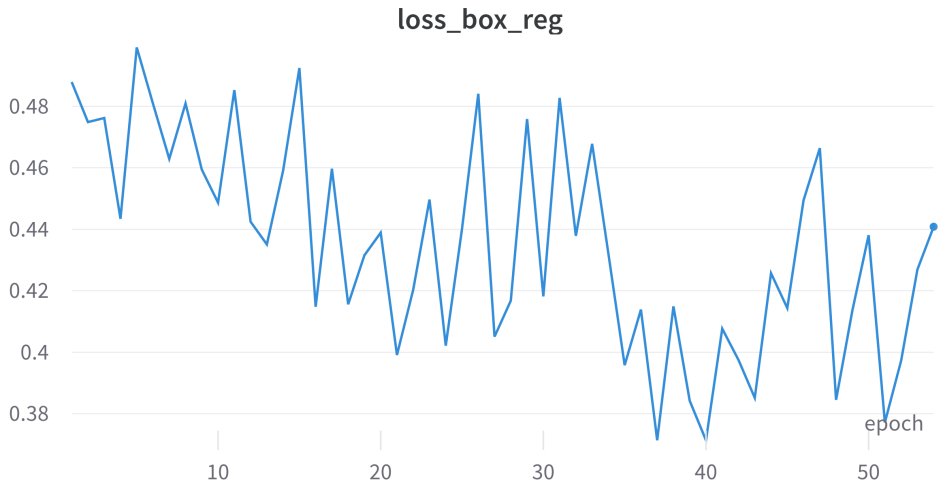

Figura 4.4: Erro de regressão da camada de saída FasterRCNN.

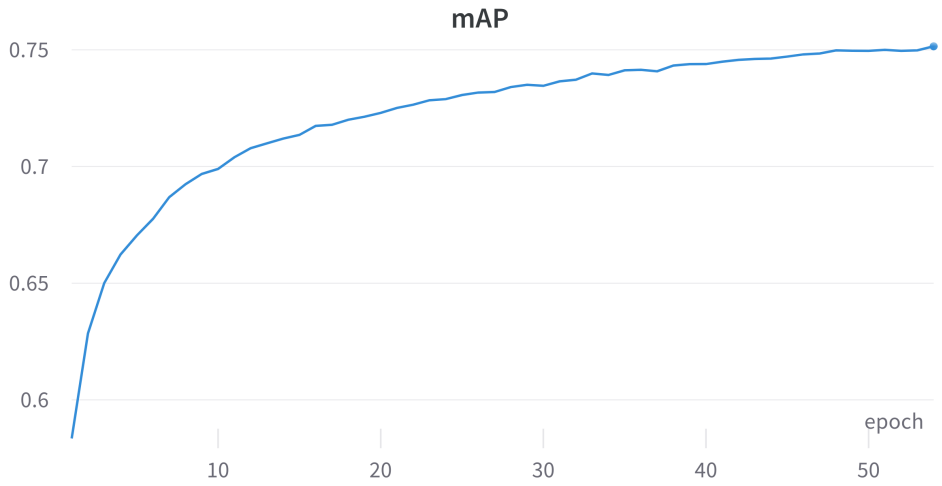

Figura 4.5: mAP FasterRCNN.

é a presença de objeto, enquanto classificação se trata da confiança da classe. Portanto, como o trabalho atual só lida com uma classe, o erro de classificação é sempre zero.

A Figura 4.6 ilustra o erro de *objectness* da Yolov5. Há uma variação bem menor no começo do treinamento em comparação com a FasterRCNN, mas no final das contas o valor do erro ficou um pouco maior $0,07$.

O erro de regressão é ilustrado na Figura 4.7. A Yolov5 tem menos variação em comparação com a FasterRCNN, e na última época o erro é consideravelmente menor $0,09$.

A mAP da Yolov5 é mostrada na Figura 4.8. A diferença entre os erros de regressão da Yolov5 e da FasterRCNN ficam notáveis quando as mAP das técnicas são comparadas.

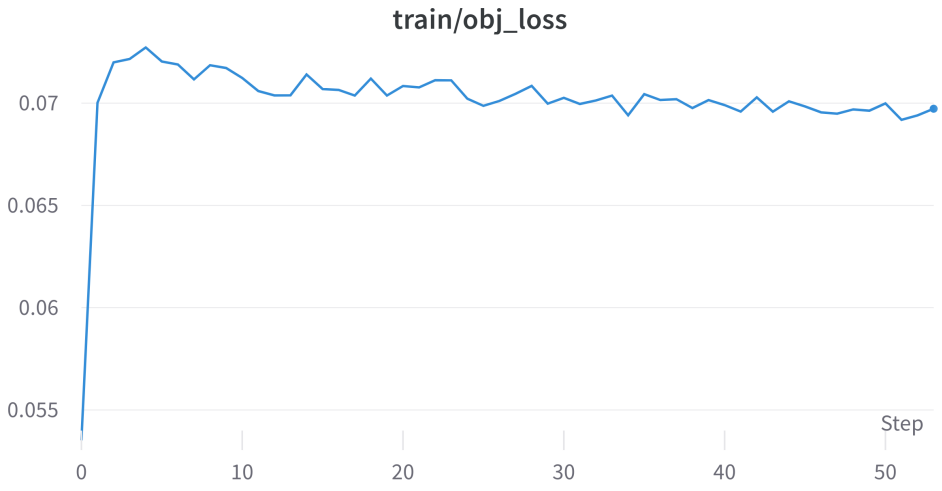

Figura 4.6: Erro de *objectness* Yolov5.

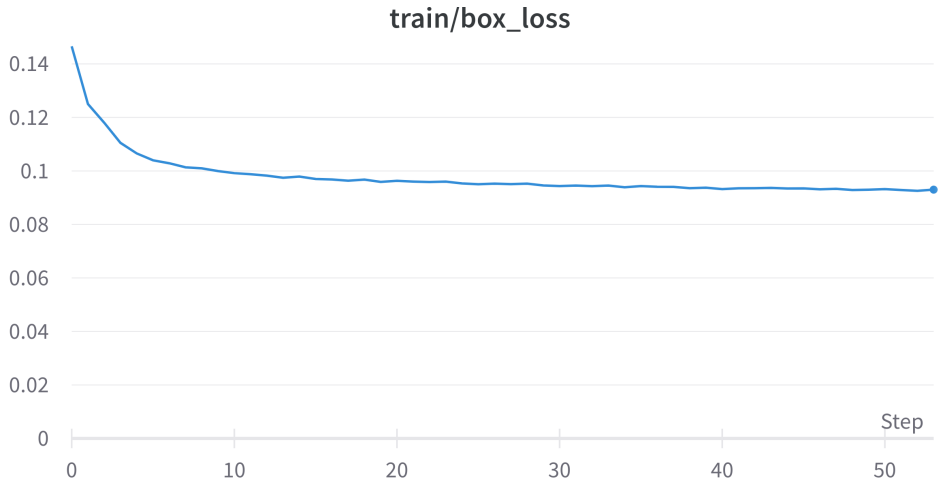

Figura 4.7: Erro de regressão Yolov5.

4.3 COMPARAÇÃO ESTATÍSTICA ENTRE FASTERRCNN E YOLOV5

A Tabela 4.1 mostra um resumo dos valores mostrados nas Seções 4.2 e 4.1. Enquanto a FasterRCNN consegue um erro de *objectness* um pouco melhor $(0,01)$, a Yolov5 consegue ser quase 5 vezes melhor na regressão das *boxes*, isso acaba causando uma diferença de 14% no mAP.

Existem 57348 objetos ao todo no conjunto de validação. A Tabela 4.2 mostra as quantidades de verdadeiros positivos, falsos positivos e falsos negativos, além do *Precision* e *Recall* dentre os objetos avaliados em cada modelo. O *Recall* dos dois modelos são similares, portanto os dois modelos acertam uma quantidade equivalente de *gt boxes*. Porém, o Yolov5 tem um *Precision* quase 4% maior que o FasterRCNN, isso indica que

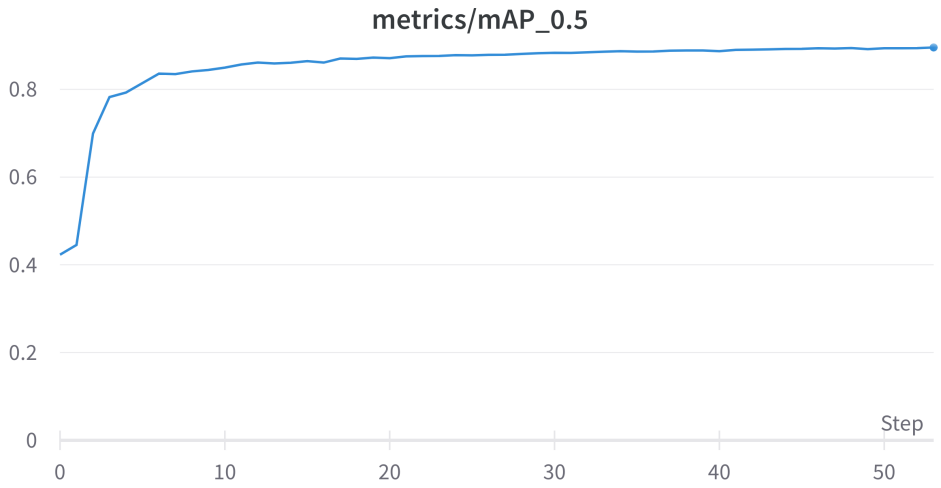

Figura 4.8: mAP Yolov5.

Tabela 4.1: Comparação entre erros e mAP dos modelos.

Modelos	Objectness	Regressão	mAP (%)
FasterRCNN	0,06	0,44	75
Yolov5	0,07	0,09	89

a FasterRCNN retorna mais resultados falsos positivos que a Yolov5.

Tabela 4.2: Número de TP, FP, FN e *Precision*, *Recall*

Modelos	TP	FP	FN	Precision (%)	Recall (%)
FasterRCNN	47917	5928	3503	88,99	93,19
Yolov5	50044	4006	3298	92,58	93,82

A próxima análise feita foi para identificar se os modelos erram objetos em comum, e quantos. Foi tomado como erro a união entre os conjuntos de Falsos Positivos e de Falsos Negativos para cada modelo. Os modelos erram 4975 objetos em comum. Isso equivale a aproximadamente $52,75\%$ dos erros da FasterRCNN e a aproximadamente $77,45\%$ dos erros da Yolov5. Isso é refletido pelo fato da FasterRCNN errar mais, no geral, do que o Yolov5.

Já que existem objetos que um modelo acerta e o outro erra, a próxima análise foi feita para identificar a quantidade desses objetos. A Tabela 4.3 mostra esses resultados. A FasterRCNN acerta cerca de $31,89\%$ dos objetos que o Yolov5 erra, enquanto o Yolov5 acerta cerca de $47,25\%$ dos objetos que a FasterRCNN erra.

Tabela 4.3: Acertos e erros de um modelo no outro.

Modelos	Acertados no outro modelo	Errados pelo outro modelo	(%)
FasterRCNN	2329	7304	31,89
Yolov5	4456	9431	47,25

4.4 TREINAMENTO DO DEEPSORT

O treinamento do DeepSort consiste do treinamento do extrator de características como um classificador. Uma ResNet50 foi o modelo escolhida para essa tarefa. A Figura 4.9 ilustra o erro de classificação do modelo. O modelo foi treinado por 60 épocas, e o erro da última época foi $0,77$

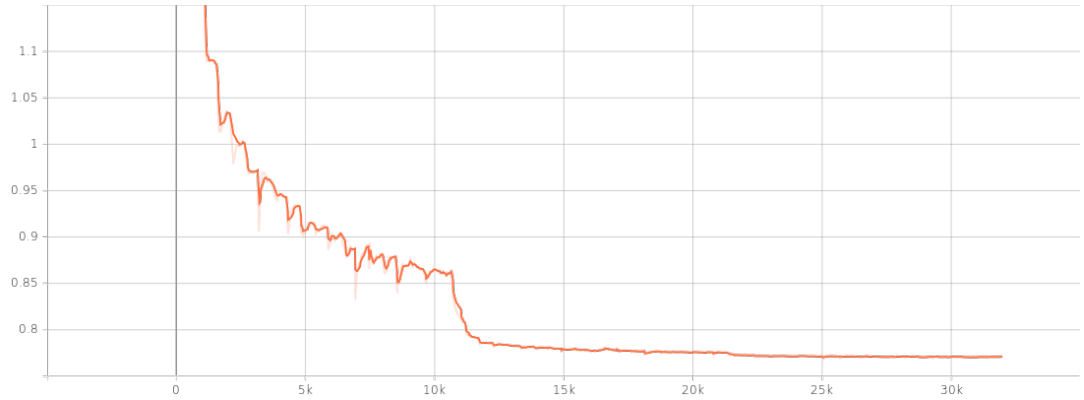

Figura 4.9: Erro de classificação da ResNet50 para o DeepSort.

4.5 TESTE DOS DETECTORES COM IMAGENS DO CONJUNTO DE DADOS

As Figuras 4.10 e 4.11 ilustram o porquê da FasterRCNN ter um *Precision* menor em relação a Yolov5. A FasterRCNN prediz muitos objetos que não são capacetes, como os bonés dos juízes e as cabeças das pessoas em geral, principalmente quando o tamanho do objeto em questão é pequeno na imagem.

Quando o tamanho dos capacetes diminui, a qualidade da predição das redes tendem a se igualar, como ilustram as Figuras 4.12 e 4.13. Mas mesmo assim, a Yolov5 ainda é melhor que a FasterRCNN.

4.6 TESTE DOS DETECTORES COM OUTRO TIPOS DE CAPACETES

Os modelos também foram testados em outros esportes que utilizam capacetes, mas de tipos diferentes. As Figuras 4.14 e 4.15 ilustram os modelos testados com capacetes de hockey no gelo. É notável que os modelos não se comportam bem, também porque os capacetes são bem diferentes em termos estéticos e geométricos.

Nos capacetes da patinação no gelo os modelos também não obtêm boa qualidade, como mostram as Figuras 4.16 e 4.17. Os capacetes da patinação no gelo são ainda

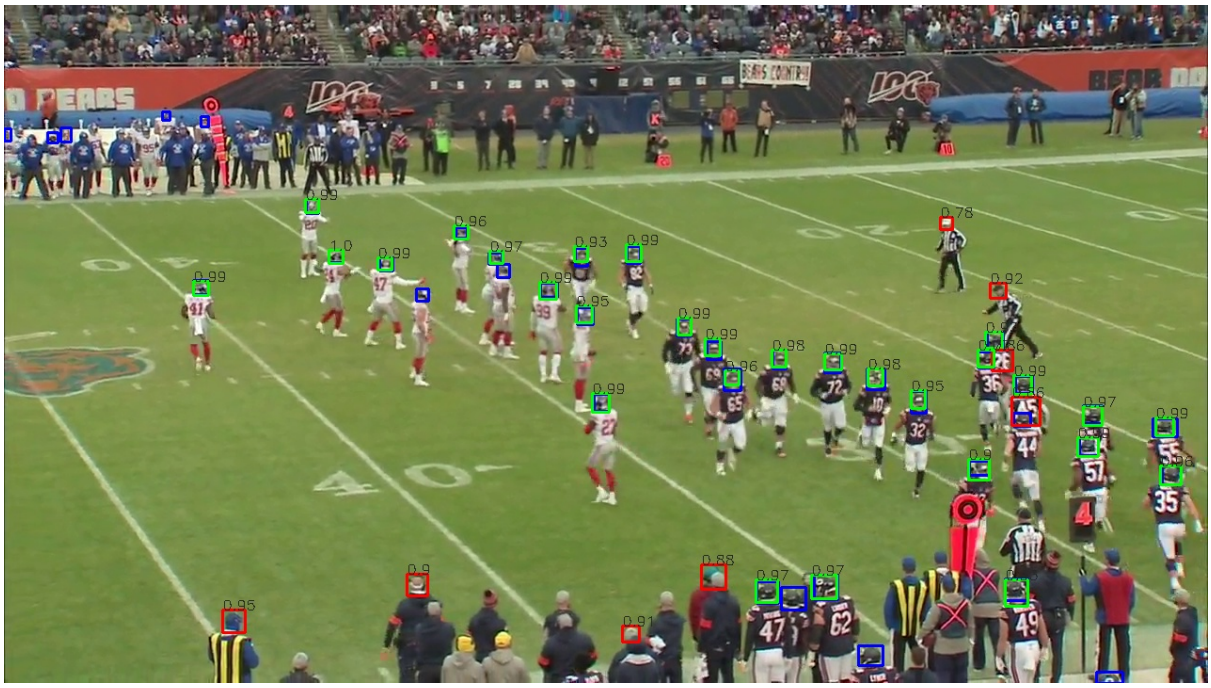

Figura 4.10: Exemplo da FasterRCNN com detecção dos bonés dos juízes. Fonte: [11]

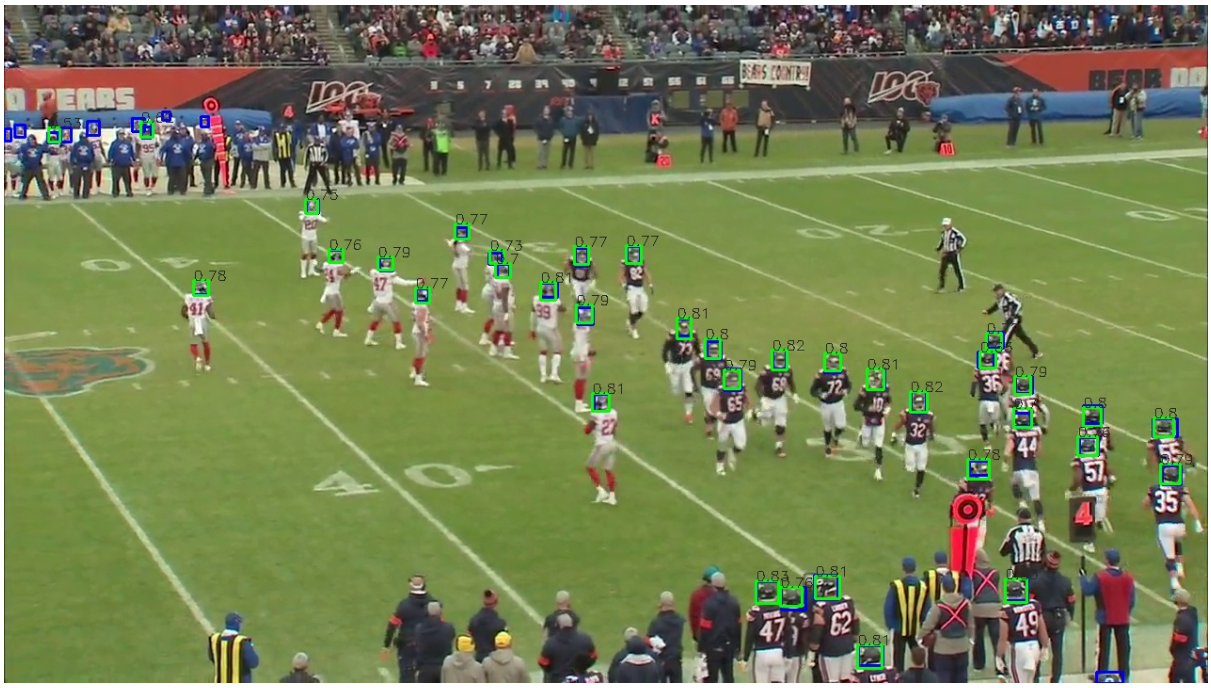

Figura 4.11: Exemplo da Yolov5 sem detecção dos bonés dos juízes. Fonte: [11]

mais diferentes dos capacetes do futebol americano. Não é um absurdo acreditar que os modelos comecem a ser capazes de reconhecer diferentes tipos de capacetes com um *fine tuning*

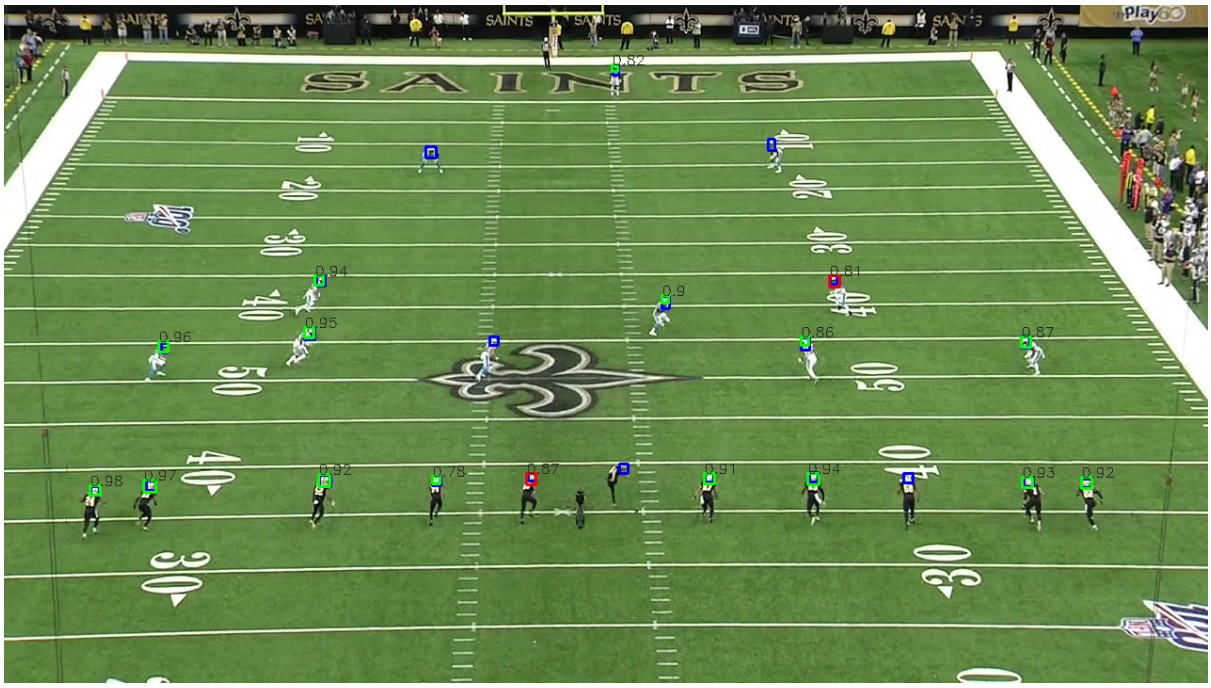

Figura 4.12: Exemplo da FasterRCNN com capacetes pequenos. Fonte: [11]

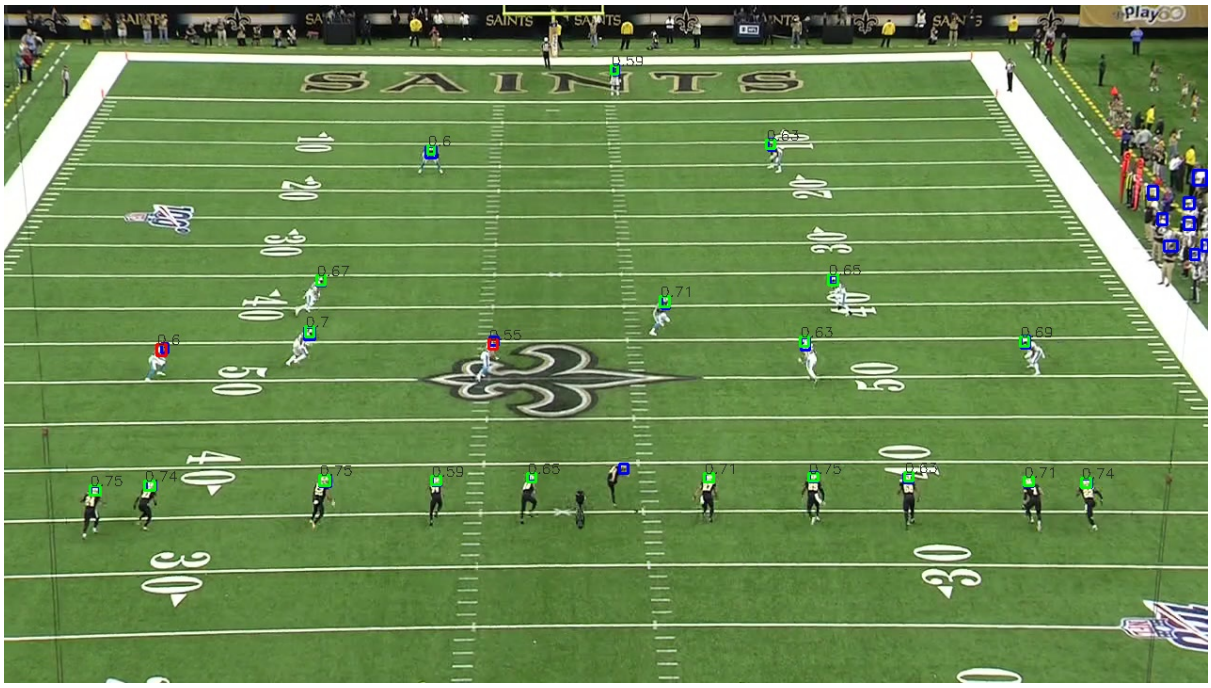

Figura 4.13: Exemplo da Yolov5 com capacetes pequenos Fonte: [11]

Figura 4.14: Exemplo da FasterRCNN com capacetes de hockey no gelo. Fonte: [53]

Figura 4.15: Exemplo da Yolov5 com capacetes de hockey no gelo. Fonte: [53]

4.7 DEMONSTRAÇÃO COM O DEEPSORT

Foram feitas algumas demonstrações com a Yolov5 e o DeepSort. O desempenho é variado, pois os capacetes são muito parecidos um com o outro, isso dificulta o trabalho do DeepSort, principalmente quando ocorre oclusão.

Todas as imagens utilizadas como exemplos de teste para a detecção, os vídeos no

Figura 4.16: Exemplo da FasterRCNN com capacetes de patinação no gelo. Fonte: [54]

Figura 4.17: Exemplo da Yolov5 com capacetes de patinação no gelo. Fonte: [54]

DeepSort e os modelos treinados se encontram em `https://drive.google.com/drive/folders/1SgnadDxg_d8D0sP263Y5_CCnFbBy4WML?usp=sharing`. Lá também se encontram os *scripts* utilizados para o treinamento da FasterRCNN e para inferência tanto da FasterRCNN quanto da Yolov5. Para o treinamento da Yolov5 visite [13]. Para inferência com o Yolov5 e o DeepSort visite [55].

CAPÍTULO 5

CONCLUSÕES E TRABALHOS FUTUROS

Em suma, ambas as técnicas de detecção atingem um bom desempenho considerando a métrica utilizada. A Yolov5 consegue ser melhor em relação a FasterRCNN, porque além de ser uma técnica mais recente, ela também é treinada de uma maneira muito mais completa, com mais artifícios de *data augmentation*, para que o modelo consiga ser mais robusto.

A FasterRCNN por ser uma técnica de dois estágios, acaba sendo mais lenta em termos de treinamento e de inferência, portanto um período mais longo de treino, mesmo sem os artifícios da Yolov5, pode ser benéfico e ajudar o modelo a ter uma melhor qualidade, principalmente em termos de *Precision*, isto é, não predizer tantas regiões que venham a ser falsas positivas, por exemplo, bonés dos árbitros, sombras no gramado e formas esféricas em geral.

Quanto a DeepSort, ela é uma técnica de rastreamento com ideias interessantes, e que podem funcionar bem para conjuntos de dados onde os objetos de uma mesma categoria tenham características diferentes. Se a categoria for Pessoa por exemplo, pessoas diferentes se vestem de forma diferente e têm tamanhos diferentes. No caso dos capacetes, todos os capacetes em campo têm a mesma dimensão e os capacetes de um mesmo time têm a mesma aparência. Isso acaba degradando o desempenho da técnica, pois ele tenta se aproveitar principalmente da descrição estética e geométrica desses objetos para fazer suas predições. Então na maioria das vezes em que há oclusão entre capacetes, os identificadores acabam trocando ou mesmo se perdendo. Por essas razões, para esse conjunto de dados, uma técnica de rastreamento um pouco mais robusta seria o mais indicado.

Algumas ideias para investigações futuras surgiram através da construção deste trabalho:

- As análises estatísticas do Capítulo 4, mostram que os modelos acertam regiões diferentes entre si com uma porcentagem considerável. Portanto seria interessante investigar se um *Ensemble* de modelos poderia funcionar. Invés de ter um modelo grande e complexo que detecte todos tipo de objeto (não em termos de classes, mas em termos de características), fazer dois ou mais modelos mais simples e leves que detectem conjuntos de objetos diferentes;

- A FasterRCNN é uma técnica que ainda é muito utilizada nos dias atuais, então seria válido utilizar um *Backbone* (extrator de características) mais recente que a ResNet50, e fazer um treinamento com mais épocas e mais robusto, como acontece na Yolov5, para que o modelo não fique limitado e consiga apresentar seu potencial completo;

- Também seria uma alternativa fazer as detecções com técnicas mais recentes que incorporam ideias de mecanismos de atenção e *transformers*;

- Por fim, o problema do *Kaggle* trata da detecção do impacto entre capacetes. Então uma extensão deste trabalho, utilizando e comparando técnicas mais robustas de rastreamento, e propondo um algoritmo, para não só a detecção do impacto, mas também a detecção da intensidade do impacto seria uma boa contribuição.

BIBLIOGRAFIA

[1] *National Football League*. endereço: https://www.nfl.com/ (acesso em 08/02/2022).

[2] N. Communications, *NFL and Amazon Prime Announce Streaming Partnership*. endereço: https://nflcommunications.com/Pages/NATIONAL-FOOTBALL-LEAGUE-AND-AMAZON-PRIME-ANNOUNCE-STREAMING-PARTNERSHIP-FOR-THURSDAY-NIGHT-FOOTBALL.aspx (acesso em 14/02/2022).

[3] *Liga BFA*. endereço: https://www.ligabfa.com/ (acesso em 08/02/2022).

[4] J. Huang, *Encefalopatia traumática crônica (ETC)*. endereço: https://www.msdmanuals.com/pt-br/profissional/dist%C3%BArbios-neurol%C3%B3gicos/delirium-e-dem%C3%AAncia/encefalopatia-traum%C3%A1tica-cr%C3%B4nica-etc (acesso em 14/02/2022).

[5] B. Resnick, *What a lifetime of playing football can do to the human brain*. endereço: https://www.vox.com/science-and-health/2018/2/2/16956440/super-bowl-2020-concussion-symptoms-cte-football-nfl-brain-damage-youth (acesso em 14/02/2022).

[6] B. Shpigel, *What to Know About C.T.E. in Football*. endereço: https://www.nytimes.com/article/cte-definition-nfl.html (acesso em 14/02/2022).

[7] C. E. Research, *NFL Concussions Fast Facts*. endereço: https://edition.cnn.com/2013/08/30/us/nfl-concussions-fast-facts/index.html (acesso em 14/02/2022).

[8] *Kaggle*. endereço: https://www.kaggle.com/ (acesso em 14/02/2022).

[9] T. N. F. League, *NFL Health and Safety - Helmet Assignment*. endereço: https://www.kaggle.com/c/nfl-health-and-safety-helmet-assignment/overview (acesso em 14/02/2022).

[10] ——, *NFL Punt Analytics Competition*. endereço: https://www.kaggle.com/c/NFL-Punt-Analytics-Competition/overview (acesso em 14/02/2022).

[11] ——, *NFL 1st and Future - Impact Detection*. endereço: https://www.kaggle.com/c/nfl-impact-detection/overview (acesso em 14/02/2022).

[12] S. Ren, K. He, R. B. Girshick e J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *CoRR*, v. abs/1506.01497, 2015. DOI: 10.48550/ARXIV.1506.01497. arXiv: 1506.01497. endereço: http://arxiv.org/abs/1506.01497.

[13] G. Jocher, A. Chaurasia, A. Stoken et al., *ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference*, versão v6.1, fev. de 2022. DOI: 10.5281/zenodo.6222936. endereço: https://doi.org/10.5281/zenodo.6222936.

[14] N. Wojke, A. Bewley e D. Paulus, "Simple Online and Realtime Tracking with a Deep Association Metric," *CoRR*, v. abs/1703.07402, 2017. DOI: 10.48550/ARXIV.1703.07402. arXiv: 1703.07402. endereço: http://arxiv.org/abs/1703.07402.

[15] D. TECH, *Introdução a Redes Neurais e Deep Learning*, 2022. endereço: https://didatica.tech/introducao-a-redes-neurais-e-deep-learning/.

[16] G. Software, *Matriz de convolução*, 2017. endereço: https://docs.gimp.org/2.8/pt_BR/plug-in-convmatrix.html.

[17] K. Simonyan e A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014. DOI: 10.48550/ARXIV.1409.1556. endereço: https://arxiv.org/abs/1409.1556.

[18] K. Le, *An overview of VGG16 and NiN models*, 2021. endereço: https://medium.com/mlearning-ai/an-overview-of-vgg16-and-nin-models-96e4bf398484.

[19] M. Yani, S. Irawan e C. Setianingsih, "Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail," *Journal of Physics: Conference Series*, v. 1201, p. 012 052, mai. de 2019. DOI: 10.1088/1742-6596/1201/1/012052.

[20] A. Zinoviev, *Object Detection with KotlinDL and Ktor*, 2022. endereço: https://blog.jetbrains.com/kotlin/2022/01/object-detection-with-kotlindl-and-ktor/.

[21] R. B. Girshick, J. Donahue, T. Darrell e J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, v. abs/1311.2524, 2013. DOI: 10.48550/ARXIV.1311.2524. arXiv: 1311.2524. endereço: http://arxiv.org/abs/1311.2524.

[22] R. B. Girshick, "Fast R-CNN," *CoRR*, v. abs/1504.08083, 2015. DOI: 10.48550/ARXIV.1504.08083. arXiv: 1504.08083. endereço: http://arxiv.org/abs/1504.08083.

[23] J. Uijlings, K. Sande, T. Gevers e A. Smeulders, "Selective Search for Object Recognition," *International Journal of Computer Vision*, v. 104, pp. 154–171, set. de 2013. DOI: 10.1007/s11263-013-0620-5.

[24] A. Krizhevsky, I. Sutskever e G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," em *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou e K. Weinberger, ed., vol. 25, Curran Associates, Inc., 2012. endereço: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[25] T. Grel, *Region of interest pooling explained*, 2017. endereço: https://deepsense.ai/region-of-interest-pooling-explained/.

[26] K. He, X. Zhang, S. Ren e J. Sun, "Deep Residual Learning for Image Recognition," *CoRR*, v. abs/1512.03385, 2015. DOI: 10.48550/ARXIV.1512.03385. arXiv: 1512.03385. endereço: http://arxiv.org/abs/1512.03385.

[27] A. Rosebrock, *Intersection over Union (IoU) for object detection*, 2016. endereço: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[28] J. Redmon, S. K. Divvala, R. B. Girshick e A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *CoRR*, v. abs/1506.02640, 2015. DOI: 10.48550/ARXIV.1506.02640. arXiv: 1506.02640. endereço: http://arxiv.org/abs/1506.02640.

[29] J. Redmon e A. Farhadi, "YOLO9000: Better, Faster, Stronger," *CoRR*, v. abs/1612.08242, 2016. DOI: 10.48550/ARXIV.1612.08242. arXiv: 1612.08242. endereço: http://arxiv.org/abs/1612.08242.

[30] U. Almog, *YOLO V3 Explained*, 2020. endereço: https://towardsdatascience.com/yolo-v3-explained-ff5b850390f (acesso em 14/02/2022).

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li e L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," em *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[32] J. Redmon e A. Farhadi, "YOLOv3: An Incremental Improvement," 2018. arXiv: 1804.02767 [cs.CV]. endereço: https://arxiv.org/abs/1804.02767.

[33] Q.-C. Mao, H.-M. Sun, Y.-B. Liu e R.-S. Jia, "Mini-YOLOv3: Real-Time Object Detector for Embedded Applications," *IEEE Access*, v. 7, pp. 133 529–133 538, 2019. DOI: 10.1109/ACCESS.2019.2941547.

[34] A. Bochkovskiy, C. Wang e H. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *CoRR*, v. abs/2004.10934, 2020. DOI: 10.48550/ARXIV.2004.10934. arXiv: 2004.10934. endereço: https://arxiv.org/abs/2004.10934.

[35] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe e Y. Yoo, "CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features," *CoRR*, v. abs/1905.04899, 2019. DOI: 10.48550/ARXIV.1905.04899. arXiv: 1905.04899. endereço: http://arxiv.org/abs/1905.04899.

[36] K. He, X. Zhang, S. Ren e J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," *CoRR*, v. abs/1406.4729, 2014. DOI: 10.1007/978-3-319-10578-9_23. arXiv: 1406.4729. endereço: http://arxiv.org/abs/1406.4729.

[37] S. Liu, L. Qi, H. Qin, J. Shi e J. Jia, *Path Aggregation Network for Instance Segmentation*, 2018. arXiv: 1803.01534 [cs.CV].

[38] seekFire, *Overview of model structure about YOLOv5*, 2020. endereço: https://github.com/ultralytics/yolov5/issues/280.

[39] A. Bewley, Z. Ge, L. Ott, F. Ramos e B. Upcroft, "Simple Online and Realtime Tracking," *CoRR*, v. abs/1602.00763, 2016. DOI: 10.1109/icip.2016.7533003. arXiv: 1602.00763. endereço: http://arxiv.org/abs/1602.00763.

[40] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME–Journal of Basic Engineering*, v. 82, n. Series D, pp. 35–45, 1960.

[41] R. Kanjee, *DeepSORT — Deep Learning applied to Object Tracking*, 2020. endereço: https://medium.com/augmented-startups/deepsort-deep-learning-applied-to-object-tracking-924f59f99104.

[42] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov e S. Zagoruyko, "End-to-End Object Detection with Transformers," *CoRR*, v. abs/2005.12872, 2020. DOI: 10.48550/ARXIV.2005.12872. arXiv: 2005.12872. endereço: https://arxiv.org/abs/2005.12872.

[43] Z. Liu, Y. Lin, Y. Cao et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows," *CoRR*, v. abs/2103.14030, 2021. DOI: 10.48550/ARXIV.2103.14030. arXiv: 2103.14030. endereço: https://arxiv.org/abs/2103.14030.

[44] M. Tan, R. Pang e Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," *CoRR*, v. abs/1911.09070, 2019. DOI: 10.48550/ARXIV.1911.09070. arXiv: 1911.09070. endereço: http://arxiv.org/abs/1911.09070.

[45] H. Nam e B. Han, "Learning Multi-Domain Convolutional Neural Networks for Visual Tracking," *CoRR*, v. abs/1510.07945, 2015. DOI: 10.48550/ARXIV.1510.07945. arXiv: 1510.07945. endereço: http://arxiv.org/abs/1510.07945.

[46] Z. Wang, L. Zheng, Y. Liu e S. Wang, "Towards Real-Time Multi-Object Tracking," *CoRR*, v. abs/1909.12605, 2019. DOI: 10.48550/ARXIV.1909.12605. arXiv: 1909.12605. endereço: http://arxiv.org/abs/1909.12605.

[47] A. Paszke, S. Gross, F. Massa et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," em *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox e R. Garnett, ed., Curran Associates, Inc., 2019, pp. 8024–8035. endereço: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[48] *fasterrcnn_resnet50_fpn*. endereço: https://pytorch.org/vision/main/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn.html#fasterrcnn-resnet50-fpn.

[49] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan e S. J. Belongie, "Feature Pyramid Networks for Object Detection," *CoRR*, v. abs/1612.03144, 2016. DOI: 10.48550/ARXIV.1612.03144. arXiv: 1612.03144. endereço: http://arxiv.org/abs/1612.03144.

[50] T. Lin, M. Maire, S. J. Belongie et al., "Microsoft COCO: Common Objects in Context," *CoRR*, v. abs/1405.0312, 2014. DOI: 10.48550/ARXIV.1405.0312. arXiv: 1405.0312. endereço: http://arxiv.org/abs/1405.0312.

[51] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li e L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," em *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[52] K. Zhou e T. Xiang, "Torchreid: A Library for Deep Learning Person Re-Identification in Pytorch," *arXiv preprint arXiv:1910.10093*, 2019.

[53] *Hóquei no gelo*. endereço: `https://pt.wikipedia.org/wiki/H%C3%B3quei_no_gelo`.

[54] *Patinação de velocidade em pista curta*. endereço: `https://pt.wikipedia.org/wiki/Patina%C3%A7%C3%A3o_de_velocidade_em_pista_curta`.

[55] M. Broström, *Real-time multi-camera multi-object tracker using YOLOv5 and DeepSort with OSNet*, `https://github.com/mikel-brostrom/Yolov5_DeepSort_OSNet`, 2022.