Instituto Tecnológico de Aeronáutica

CT-213: Inteligência Artificial para Robótica Móvel

Lab 9: Detecção de Objetos

Bruno Benjamim Bertucci - Turma 23.2

1 Introdução

No campo da visão computacional, existem algoritmos capazes de realizarem a detecção de objetos, o que é uma habilidade fundamental para robôs autônomos que se valem de câmeras para receberem informações do ambiente e decidirem as ações que eles executarão.

Um detector de objetos precisa ser capaz de perceber a presença de determinados objetos em uma imagem e, além disso, deve identificar a localização de cada objeto percebido nessa imagem. Como redes neurais convolucionais mostraram-se eficientes para uso em visão computacional, elas foram aplicadas na implementação de detectores de objetos.

2 Implementação de detector de objetos

Para a implementação de um detector de objetos adequado para aplicação no futebol de robôs, direcionado para a detecção de uma bola e das traves de um gol, foi utilizada uma implementação de redes neurais convolucionais baseada em um popular algoritmo de detecção de objetos denominado *YOLO* (*You Only Look Once*). A implementação consiste da rede neural em si, e de operações adicionais que processam a imagem de entrada em um formato adequado para a rede, e que processam os dados de saída gerados por ela para obter as detecções dos objetos desejados.

2.1 Rede neural

O modelo de rede neural convolucional utilizado possui uma arquitetura com um total de 10 camadas, além de uma camada de *input*. Os *inputs* têm formato de $(cols, rows, 3)$, onde $cols$ e $rows$ representam o número de colunas e de linhas, respectivamente, da matriz que representa cada uma das 3 camadas de cor da imagem de entrada. Após a camada de *input*, as duas próximas realizam operações de convolução 2D, normalização de *batch* e, finalmente, de *Leaky Rectified Linear Unit (Leaky ReLU)*. Nessas duas camadas, o número de filtros da operação de convolução 2D é 8.

O grupo constituído pelas próximas 4 camadas realizam as mesmas três operações das duas anteriores, mas adicionam uma operação de *Max Pooling 2D* ao final de cada uma, todas com tamanho de *pool* de 2 em ambas as dimensões e *stride* de $(2, 2)$ nas primeiras 3 camadas desse grupo, e de $(1, 1)$ na quarta camada. Em outras palavras, a operação avança um elemento da matriz a cada iteração, e a matriz de saída de cada camada tem metade do tamanho da entrada correspondente. Além disso, o número de filtros das operações de convolução 2D é de 16 para a primeira camada desse grupo, de 32 para a segunda, e de 64 para as duas últimas camadas do grupo. Devido às operações de *Max pooling* e ao número de filtros da última camada, a saída desse grupo de camadas tem formato $(15, 20, 64)$.

Nesse ponto, a linearidade da rede neural é quebrada, pois são criados dois grupos de camadas paralelas, sendo uma delas denominadas *skip connection*. A camada *skip connection* é constituída de uma operação de convolução 2D de 128 filtros, seguido de normalização de *batch*, e, finalmente, passando por uma operação de *Leaky ReLU*. O grupo de camadas paralelas a essa, é composto por duas camadas, sendo que cada uma realiza a sequência de operações: Convolução 2D, normalização de *batch* e *Leaky ReLU*, sendo que a convolução 2D da primeira camada desse grupo tem 128 filtros, e a segunda, 256 filtros.

Em seguida, os resultados dessas camadas paralelas são concatenados, e a saída resultante segue para a última camada, que realiza somente uma operação de convolução 2D com 10 filtros.

Além da camada *skip connection* e da última camada da rede, cujas operações de convolução 2D têm tamanho do *kernel* de $(1, 1)$ e *stride* de $(1, 1)$, todas as outras camadas utilizam *kernel* de $(3, 3)$ e *stride* de $(1, 1)$. Todas as operações de *Leaky ReLU* são executadas com *alpha* de 0,1. De todas as operações de convolução 2D realizadas, somente aquela da última camada utiliza *bias*. Um sumário da arquitetura dessa rede neural encontra-se na Figura 1.

```
Model: "ITA_YOLO"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_1 (InputLayer)         [(None, 120, 160, 3) 0
_________________________________________________________________________________________________
conv_1 (Conv2D)              (None, 120, 160, 8)  216         input_1[0][0]
_________________________________________________________________________________________________
norm_1 (BatchNormalization)  (None, 120, 160, 8)  32          conv_1[0][0]
_________________________________________________________________________________________________
leaky_relu_1 (LeakyReLU)     (None, 120, 160, 8)  0           norm_1[0][0]
_________________________________________________________________________________________________
conv_2 (Conv2D)              (None, 120, 160, 8)  576         leaky_relu_1[0][0]
_________________________________________________________________________________________________
norm_2 (BatchNormalization)  (None, 120, 160, 8)  32          conv_2[0][0]
_________________________________________________________________________________________________
leaky_relu_2 (LeakyReLU)     (None, 120, 160, 8)  0           norm_2[0][0]
_________________________________________________________________________________________________
conv_3 (Conv2D)              (None, 120, 160, 16) 1152        leaky_relu_2[0][0]
_________________________________________________________________________________________________
norm_3 (BatchNormalization)  (None, 120, 160, 16) 64          conv_3[0][0]
_________________________________________________________________________________________________
leaky_relu_3 (LeakyReLU)     (None, 120, 160, 16) 0           norm_3[0][0]
_________________________________________________________________________________________________
max_pool_3 (MaxPooling2D)    (None, 60, 80, 16)   0           leaky_relu_3[0][0]
_________________________________________________________________________________________________
conv_4 (Conv2D)              (None, 60, 80, 32)   4608        max_pool_3[0][0]
_________________________________________________________________________________________________
norm_4 (BatchNormalization)  (None, 60, 80, 32)   128         conv_4[0][0]
_________________________________________________________________________________________________
leaky_relu_4 (LeakyReLU)     (None, 60, 80, 32)   0           norm_4[0][0]
_________________________________________________________________________________________________
max_pool_4 (MaxPooling2D)    (None, 30, 40, 32)   0           leaky_relu_4[0][0]
_________________________________________________________________________________________________
conv_5 (Conv2D)              (None, 30, 40, 64)   18432       max_pool_4[0][0]
_________________________________________________________________________________________________
norm_5 (BatchNormalization)  (None, 30, 40, 64)   256         conv_5[0][0]
_________________________________________________________________________________________________
leaky_relu_5 (LeakyReLU)     (None, 30, 40, 64)   0           norm_5[0][0]
_________________________________________________________________________________________________
max_pool_5 (MaxPooling2D)    (None, 15, 20, 64)   0           leaky_relu_5[0][0]
_________________________________________________________________________________________________
conv_6 (Conv2D)              (None, 15, 20, 64)   36864       max_pool_5[0][0]
_________________________________________________________________________________________________
norm_6 (BatchNormalization)  (None, 15, 20, 64)   256         conv_6[0][0]
_________________________________________________________________________________________________
leaky_relu_6 (LeakyReLU)     (None, 15, 20, 64)   0           norm_6[0][0]
_________________________________________________________________________________________________
max_pool_6 (MaxPooling2D)    (None, 15, 20, 64)   0           leaky_relu_6[0][0]
_________________________________________________________________________________________________
conv_7 (Conv2D)              (None, 15, 20, 128)  73728       max_pool_6[0][0]
_________________________________________________________________________________________________
norm_7 (BatchNormalization)  (None, 15, 20, 128)  512         conv_7[0][0]
_________________________________________________________________________________________________
leaky_relu_7 (LeakyReLU)     (None, 15, 20, 128)  0           norm_7[0][0]
_________________________________________________________________________________________________
conv_skip (Conv2D)           (None, 15, 20, 128)  8192        max_pool_6[0][0]
_________________________________________________________________________________________________
conv_8 (Conv2D)              (None, 15, 20, 256)  294912      leaky_relu_7[0][0]
_________________________________________________________________________________________________
norm_skip (BatchNormalization) (None, 15, 20, 128)  512         conv_skip[0][0]
_________________________________________________________________________________________________
norm_8 (BatchNormalization)  (None, 15, 20, 256)  1024        conv_8[0][0]
_________________________________________________________________________________________________
leaky_relu_skip (LeakyReLU)  (None, 15, 20, 128)  0           norm_skip[0][0]
_________________________________________________________________________________________________
leaky_relu_8 (LeakyReLU)     (None, 15, 20, 256)  0           norm_8[0][0]
_________________________________________________________________________________________________
concat (Concatenate)         (None, 15, 20, 384)  0           leaky_relu_skip[0][0]
                                                              leaky_relu_8[0][0]
_________________________________________________________________________________________________
conv_9 (Conv2D)              (None, 15, 20, 10)   3850        concat[0][0]
=================================================================================================
Total params: 445,346
Trainable params: 443,938
Non-trainable params: 1,408
_________________________________________________________________________________________________
```

Figura 1: Sumário do modelo implementado

2.2 Algoritmo de detecção de objetos

O algorítmo de detecção utilizado executa os seguintes passos: primeiramente, ele processa a imagem passada usando *OpenCV*, adequando-a para a rede neural. Isso é feito pela redução do tamanho da imagem de $640x480$ pixels para $160x120$ pixels, seguido de conversão da matriz correspondente à imagem para um *numpy array*. Então, os valores de cor dos pixels são normalizados para estarem entre 0 e 1. Finalmente, o *array* é convertido para o formato $(1, 120, 160, 3)$, que é um formato adequado para um *input* da rede neural.

Em seguida, a imagem é levada à rede neural, sendo a saída dela constituída pelo vetor x: $x = [t_b, t_{xb}, t_{yb}, t_{wb}, t_{hb}, t_p, t_{xp}, t_{yp}, t_{wp}, t_{hp}]^T$

Esses valores são utilizados para calcular as posições e tamanhos dos *anchor boxes* correspondentes á bola e às traves em um campo de futebol de robôs. Os cálculos realizados encontram-se na Equação 1, onde p é a probabilidade de a célula conter a bola, se o parâmetro usado for t_b, e de conter uma trave, no caso de ser usado t_p, x e y são as coordenadas do centro do *anchorbox*, w e h representam a largura e altura, respectivamente, do *anchor box*, e a função σ é dada pela Equação 2. Nessa implementação, os parâmetros S_{coord} e S_{bb} valem 32 e 640, respectivamente.

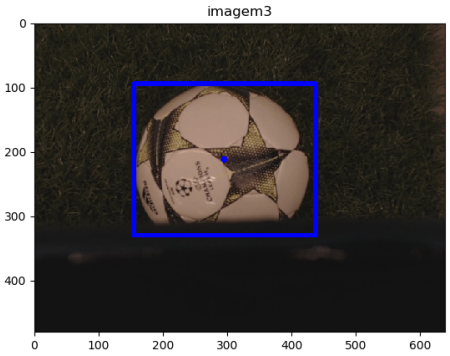
A célula correspondente à detecção da bola corresponde àquela com o maior valor de t_b, enquanto as duas traves são localizadas nas duas células com maior valor de t_p.

$$\begin{aligned} detection &= (p, x, y, w, h) \\ p &= \sigma(t) \\ x &= S_{coord}(j + \sigma(t_x)) \\ y &= S_{coord}(i + \sigma(t_y)) \\ w &= S_{bb} p_w e^{t_w} \\ h &= S_{bb} p_h e^{t_h} \end{aligned} \tag{1}$$

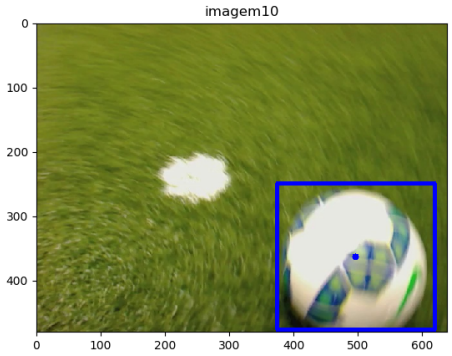
$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

3 Teste do detector de objetos

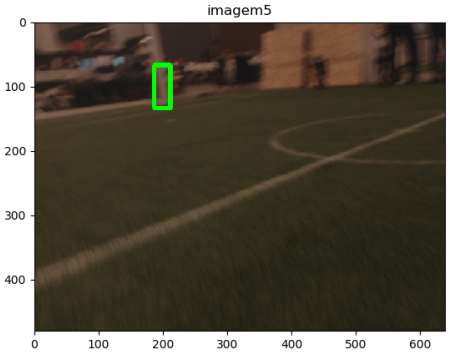
Dessa forma, o detector é capaz de receber uma imagem, buscar a presença de uma bola de futebol e de duas traves, e localizar a posição desses objetos usando *anchor boxes*. A Figura 2 mostra alguns exemplos do funcionamento do detector de objetos.



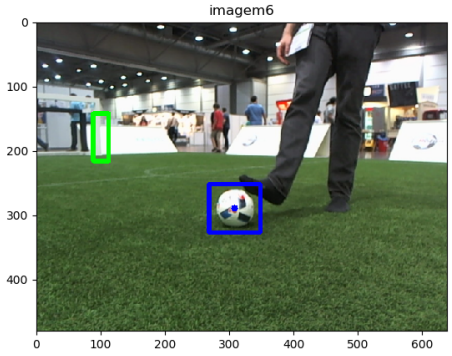
(a) Bola (agente estacionário).



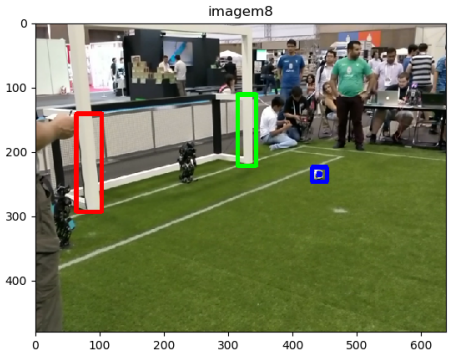
(b) Bola (agente em movimento).



(c) Somente uma trave.



(d) Bola e uma trave.



(e) Bola e duas traves.

Figura 2: Exemplos de utilização do detector de objetos.

Observando as figuras, é possível concluir que essa implementação de detecção de objetos é satisfatória, uma vez que ela é capaz de detectar a bola e as traves em diferentes situações em termos de iluminação, distância dos objetos buscados e estado de movimento da câmera.