Relatório do Lab 10 de CT-213

Trabalho 10 – Visão Computacional com Redes Neurais Convolucionais

Detecção de Objetos no futebol de robôs humanoides com o algoritmo YOLO (*You Only Look Once*)

Aluno:

Bruno Costa Alves Freire

Turma:

T 22.4

Professor:

Marcos Ricardo Omena de Albuquerque Máximo

Data:

01/06/2019

Instituto Tecnológico de Aeronáutica – ITA
Departamento de Computação

1. Implementando a arquitetura da rede neural

A implementação da RNC utilizada para processar a imagem e extrair as chamadas *features* (probabilidade de presença do objeto, posição do centro do objeto, dimensões da *bounding box* do objeto, para a bola e a trave) foi feita conforme a arquitetura do roteiro, utilizando o framework Keras. Na página 2 segue o sumário da arquitetura, compactado para caber numa página, gerado pelo *script* `make_detector_network.py`. A saída original, gerada pelo script consta também no arquivo `sumario.txt`.

2. Implementando a detecção de objetos com YOLO

Em vez de treinar a rede neural implementada, vamos apenas carregar uma rede de arquitetura idêntica já treinada para o propósito da detecção de objetos, no arquivo `yolo_ball_goalpost.hdf5`. Faremos isso pois o treinamento da rede neural requer muito poder computacional, além de dados de imagens, o que tomaria muito tempo e um dataset muito grande.

Vamos implementar somente o algoritmo de detecção, na classe `YoloDetector`. Essa classe conta com os métodos `preprocess_image`, `process_yolo_output`, e `detect`. O primeiro simplesmente recebe uma imagem carregada e a processa para que fique no formato da entrada da rede neural. Nesse processo, a resolução da imagem é reduzida a 25% do original, e a imagem é transformada num *array* 120x160, com 3 canais de cores, com valores de 0 a 255.

O método que processa a saída da rede neural toma um *array* de 3 dimensões (15, 20, 10), que conta com um vetor de *features* para cada célula da imagem. Dessas features, as 5 primeiras são informações sobre a detecção da bola, e as cinco últimas são sobre a trave. Essas 5 features são respectivamente a probabilidade da presença do objeto, as posição x e y do centro do objeto dentro da célula, e as dimensões de altura e largura da *bounding box*. Tomamos então a matriz com as features de probabilidade, e buscamos pela célula com maior probabilidade de conter a bola, e as duas células com maior probabilidade de conter uma trave. Uma vez encontradas essas células, geramos a informação sobre a detecção dos objetos na imagem, convertendo as features geradas pelo YOLO em posições e tamanhos na imagem original.

Por fim, o método `detect` é simplesmente o método que junta todas essas funcionalidades e retorna a saída para ser tratada pelo *script* externo `test_vision_detector.py`, que carrega e aplica a detecção dos objetos nas imagens, filtrando detecções com baixas probabilidades.

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

A seguir, as figuras de 1 a 10 exibem o resultado da detecção de objetos feita pela rede já treinada, com arquitetura idêntica à implementada aqui, comprovando o bom funcionamento da implementação.

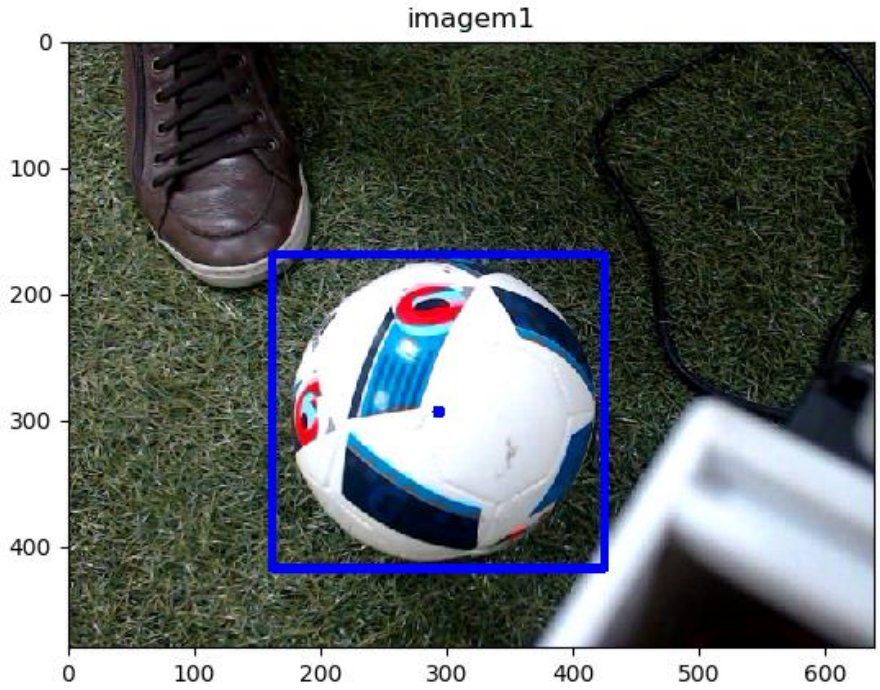

Figura 1: Imagem onde a rede neural detectou com sucesso uma bola.

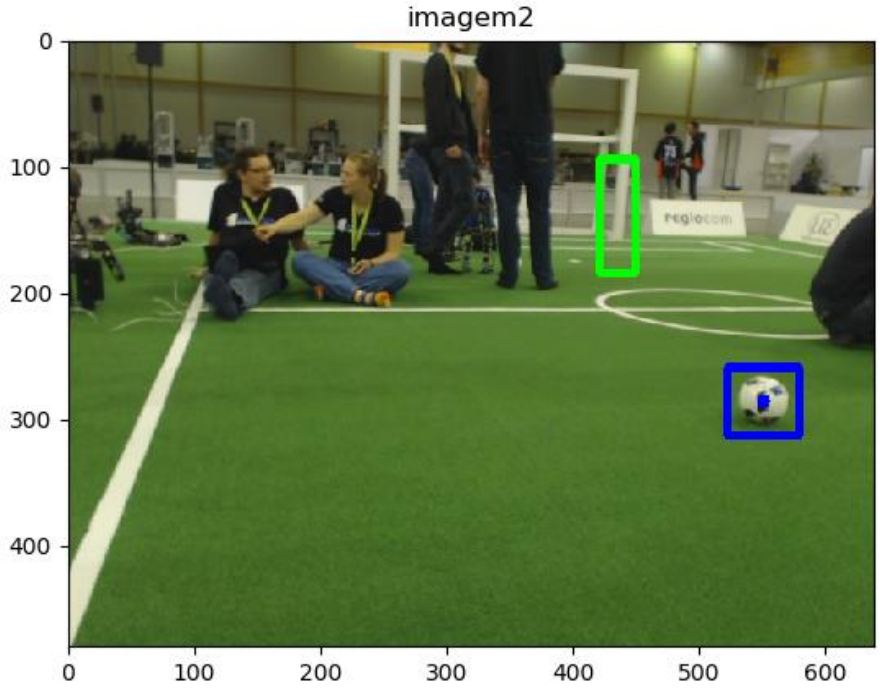

Figura 2: Imagem de um campo onde a rede detectou a bola e uma trave.

Note que na figura 2 a segunda trave não foi encontrada, pois a rede é treinada para detectar o encontro da trave com o chão, que estava coberto na imagem.

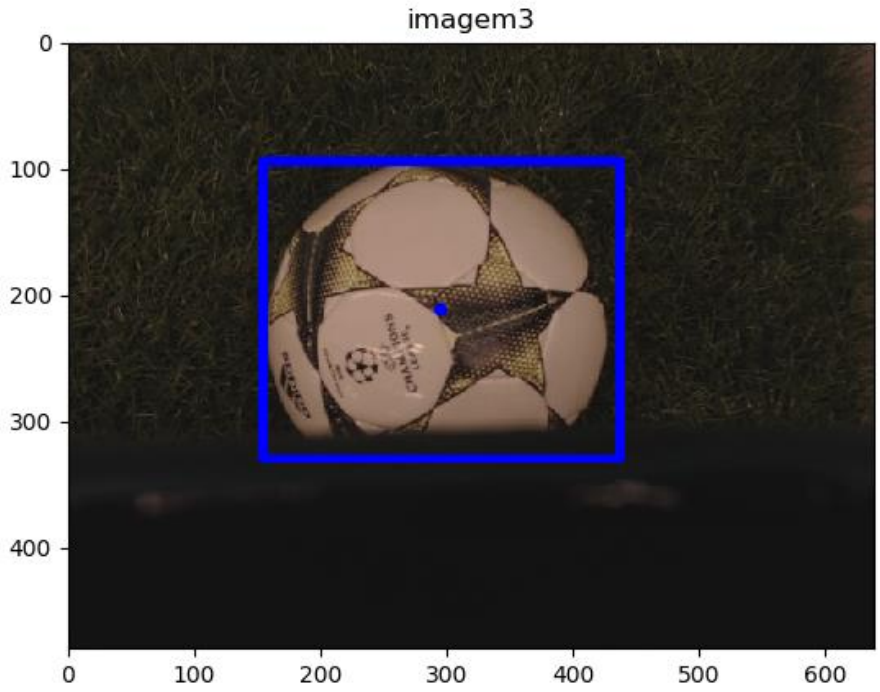

Figura 3: Aqui a rede detectou a bola mesmo com luminosidade mais baixa.

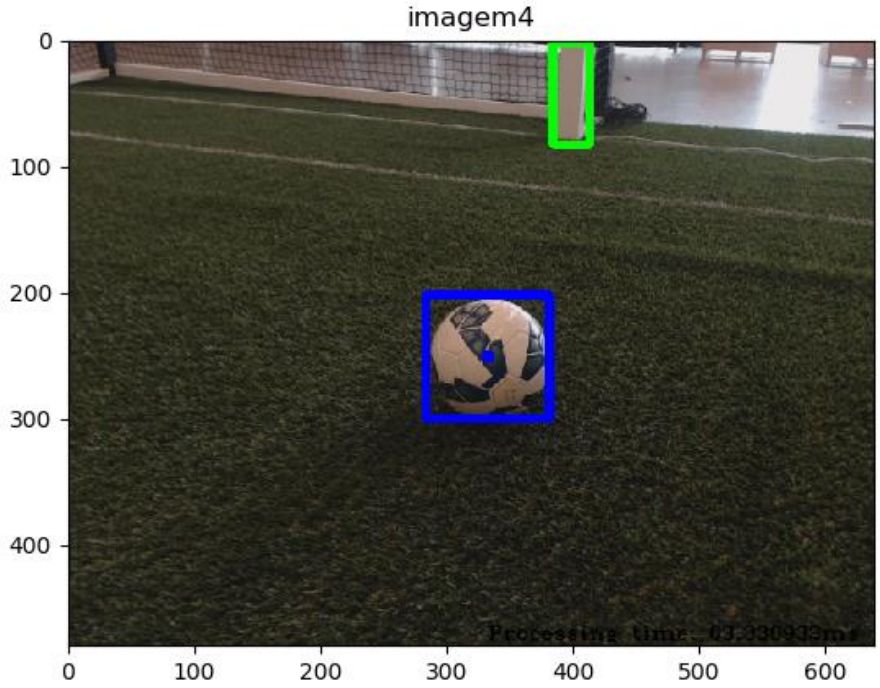

Figura 4: A rede detectou a primeira trave e a bola, apesar da mudança de luminosidade.

Pelas figuras 3 e 4 vemos que a rede sabe lidar bem com variação de luminosidade, graças ao seu treinamento rigoroso.

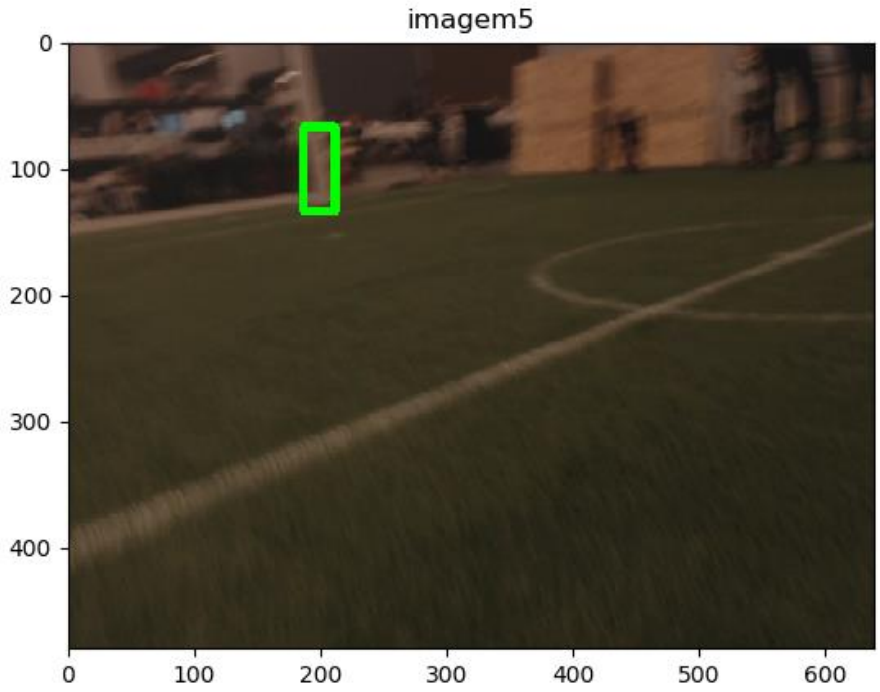

Figura 5: Exemplo onde a rede detectou uma trave apesar do efeito de *motion blur*.

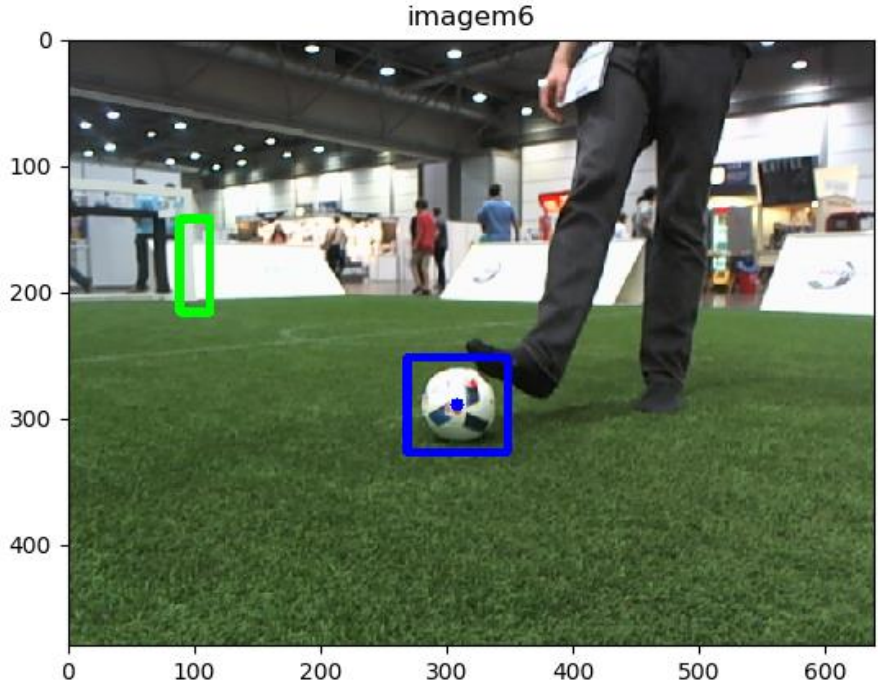

Figura 6: A rede foi capaz de detectar a bola e uma trave, apesar de haver um objeto branco por trás da trave.

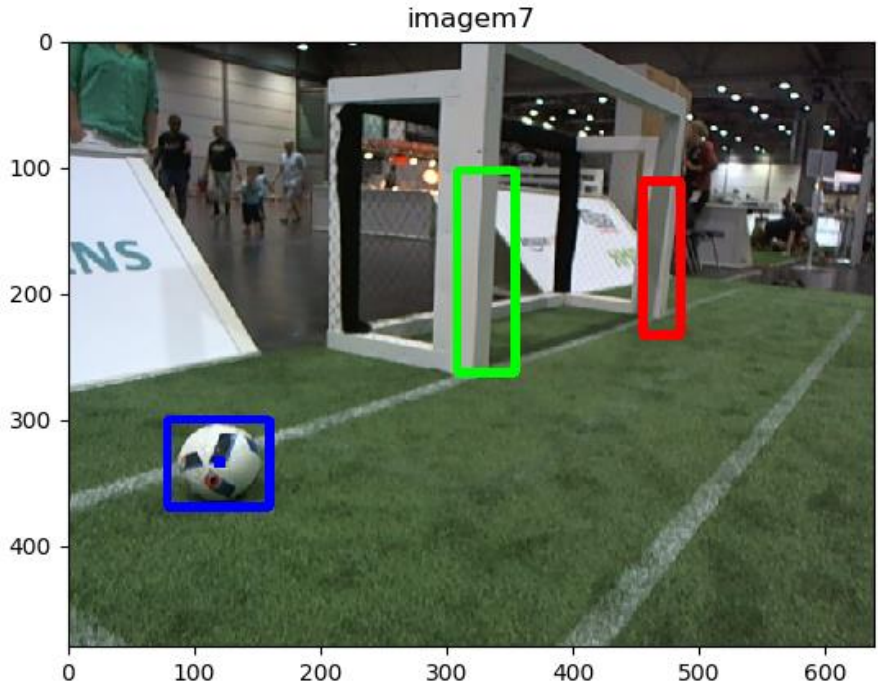

Figura 7: A rede foi capaz de detectar com sucesso as duas traves em suas posições corretas.

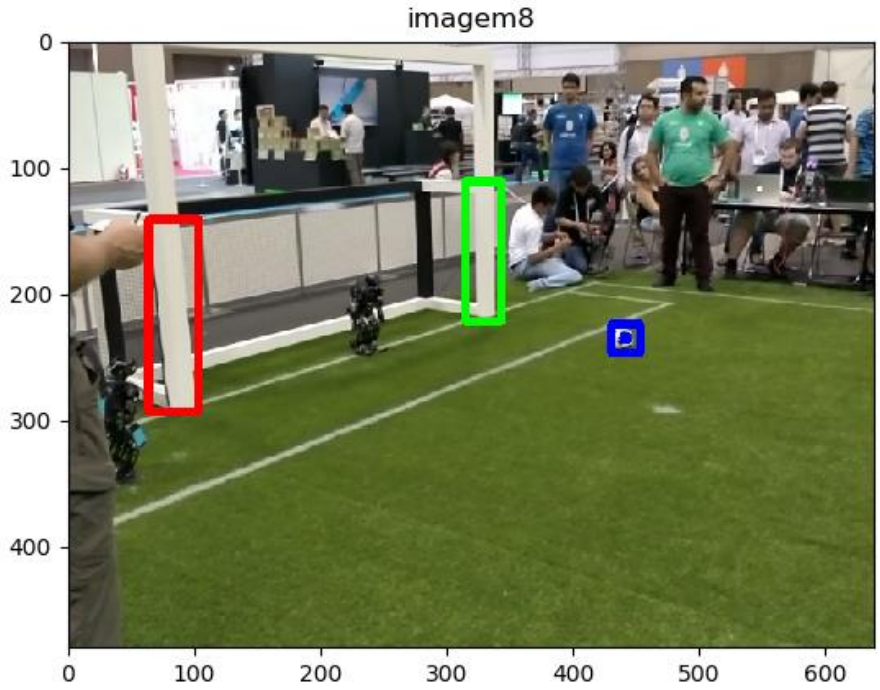

Figura 8: Novamente a rede conseguiu detectar as duas traves corretamente além da bola.

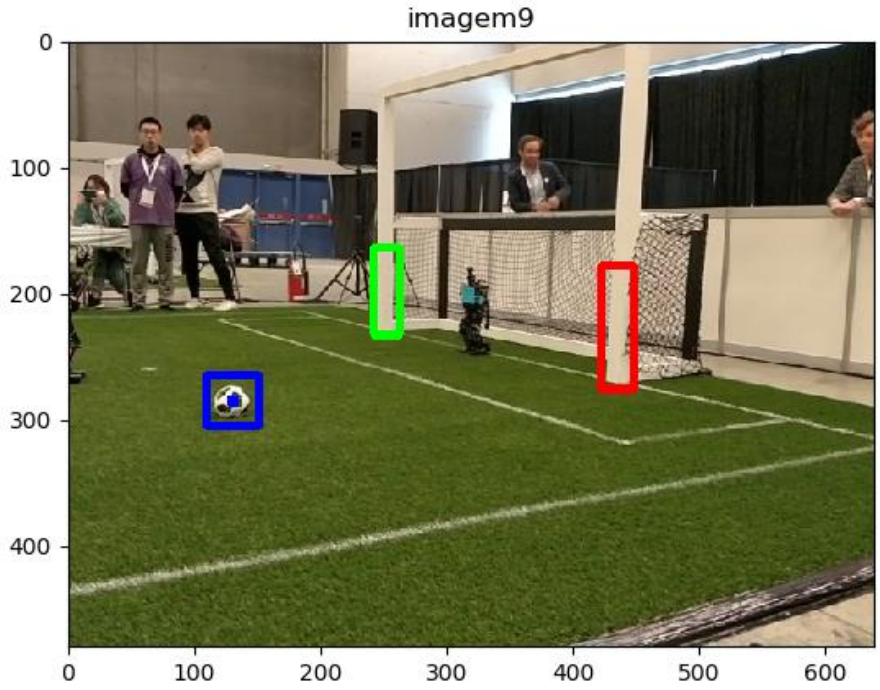

Figura 9: Mais uma detecção correta de todos os três objetos pela rede.

Pelas figuras 7, 8 e 9 vemos que a detecção de dois objetos idênticos (as duas traves) está funcionando corretamente, uma vez que todos os objetos foram identificados nos locais corretos.

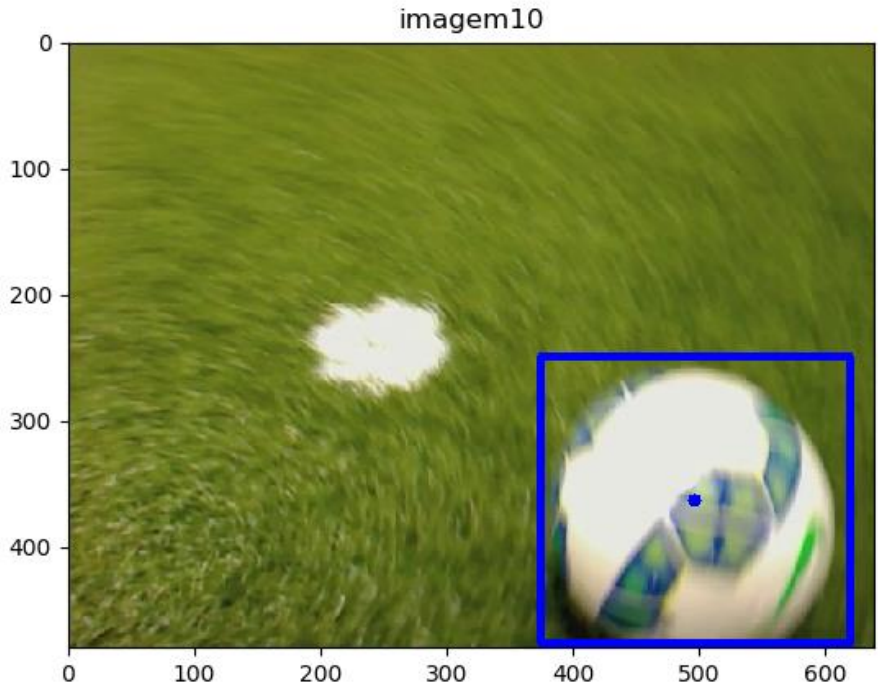

Figura 10: Mais uma detecção de sucesso da bola apesar do efeito de *motion blur*.