1 Introdução

O objetivo desse laboratório foi implementar (em *Python*) a rede neural convolucional *Yolo*. Tal rede é utilizada para processamento de imagens, no caso desse laboratório, para detectar a bola e as traves numa situação de futebol de robôs.

Foi pré-disponibilizado um código da equipe de futebol de robôs *ITA Androids* para a implementação de métodos específicos.

2 Montagem da rede neural

O roteiro disponibilizado pelo professor mostra o esquema da rede neural a ser montada (vide figura 1). Para isso, utilizou-se o *framework* do *Keras* para criação de camadas da rede (seguindo a tabela 1 do roteiro). O código para criação das *labels* dessa rede estão mostrados abaixo (todas as camadas).

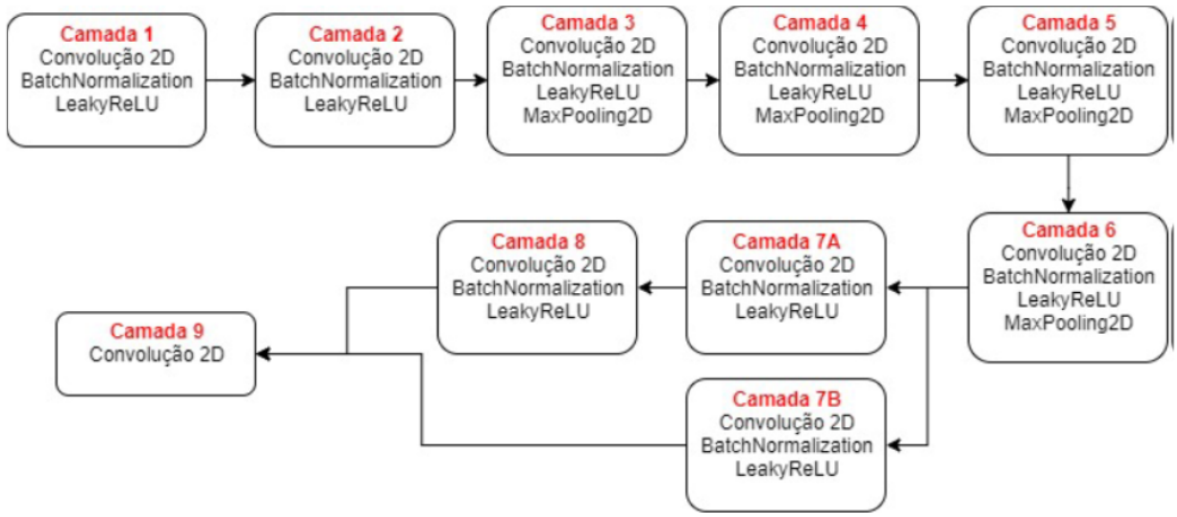

Figura 1: Arquitetura teórica da rede neural a ser implementada

```
# Input layer
input_image = Input(shape=(img_cols, img_rows, 3))

# Layer 1
layer = Conv2D(filters=8, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_1',
    use_bias=False)(input_image)
layer = BatchNormalization(name='norm_1')(layer)
layer = LeakyReLU(alpha=0.1, name='leaky_relu_1')(layer)

# Layer 2
layer = Conv2D(filters=8, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_2', use_bias=False)(
    layer)
layer = BatchNormalization(name='norm_2')(layer)
layer = LeakyReLU(alpha=0.1, name='leaky_relu_2')(layer)

# Layer 3
layer = Conv2D(filters=16, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_3', use_bias=False)(
    layer)
layer = BatchNormalization(name='norm_3')(layer)
layer = LeakyReLU(alpha=0.1, name='leaky_relu_3')(layer)

layer= MaxPooling2D(pool_size=(2,2), strides=(2,2),padding='same',
                    name='max_pool_3')(layer)
```

```
    # Layer 4
    layer = Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_4', use_bias=False)(
        layer)
    layer = BatchNormalization(name='norm_4')(layer)
    layer = LeakyReLU(alpha=0.1, name='leaky_relu_4')(layer)
    layer = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same',
                         name='max_pool_4')(layer)

    # Layer 5
    layer = Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_5', use_bias=False)(
        layer)
    layer = BatchNormalization(name='norm_5')(layer)
    layer = LeakyReLU(alpha=0.1, name='leaky_relu_5')(layer)
    layer = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same',
                         name='max_pool_5')(layer)

    # Layer 6
    layer = Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_6', use_bias=False)(layer)
    layer = BatchNormalization(name='norm_6')(layer)
    layer = LeakyReLU(alpha=0.1, name='leaky_relu_6')(layer)
    layer = MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding='same', name='max_pool_6')(layer)

    skip_connection = layer



    # Layer 7A
    layer = Conv2D(filters=128, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_7', use_bias=False)(
        layer)
    layer = BatchNormalization(name='norm_7')(layer)
    layer = LeakyReLU(alpha=0.1, name='leaky_relu_7')(layer)

    # Layer 7B
    skip_connection = Conv2D(filters=128, kernel_size=(1, 1), strides=(1, 1), padding='same', name='conv_skip',
        use_bias=False)(skip_connection)
    skip_connection = BatchNormalization(name='norm_skip')(skip_connection)
    skip_connection = LeakyReLU(alpha=0.1, name='leaky_relu_skip')(skip_connection)

    # Layer 8
    layer = Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_8', use_bias=False)(
        layer)
    layer = BatchNormalization(name='norm_8')(layer)
    layer = LeakyReLU(alpha=0.1, name='leaky_relu_8')(layer)

    # Concatenating layers 7B and 8
    layer = concatenate([skip_connection, layer], name='concat')

    # Layer 9 (last layer)
    layer = Conv2D(10, (1, 1), strides=(1, 1), padding='same', name='conv_9', use_bias=True)(layer)

    model = Model(inputs=input_image, outputs=layer, name='ITA_YOLO')
```

O que gera a seguinte rede neural (idêntica à tabela 1 do roteiro)

```
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
__________________________________________________________________________________________________
```

3 Métodos da *YOLO*

Nessa seção, o objetivo foi, recebendo os *outputs* da rede neural (já treinada), processar a imagem (redimensionar, transformar num *array* de *NumPy* e modificar esse *array*), processar os *outputs* da rede, com o redimensionamento e, finalmente retornar um vetor de *features* da seguinte forma:

$$[p\ x_0\ y_0\ w\ h], \tag{1}$$

em que, p é a probabilidade do objeto estar na imagem, x_0 e y_0 as coordenadas do centro da caixa do objeto e w e h a largura e altura da caixa, respectivamente. A implementação dos métodos está mostrada abaixo.

```
def detect(self, image):
    image=self.preprocess_image(image)
    ball_detection, post1_detection, post2_detection= self.process_yolo_output(
        self.network.predict(image))

    return ball_detection, post1_detection, post2_detection

def preprocess_image(self, image):
    image = mude o tamanho da imagem para (160,120)
    image=np.array(image)
    image = image/255.0
    image = mude o tamanho da imagem para (1, 120, 160, 3)
    return image

def process_yolo_output(self, output):
    coord_scale = 4 * 8
    bb_scale = 640
    output =  mude o formato de output para (15,20,10)

    ib,jb=0,0
    ip1,jp1=0,0
    ip2,jp2= 0,0
    #calculando as celulas i,j que contem a maior probabilidade
    for i in range(15):
        for j in range(20):
            if output[i][j][0]>output[ib][jb][0]:
                ib,jb=i,j
            if output[i][j][5]>output[ip1][jp1][5]:
                ip1,jp1=i,j
            elif output[i][j][5]>output[ip2][jp2][5]:
                ip2,jp2=i,j

    pb=sigmoid(output[ib][jb][0])
    xb=(jb+sigmoid(output[ib][jb][1]))*coord_scale
    yb=(ib+sigmoid(output[ib][jb][2]))*coord_scale
    wb=bb_scale*self.anchor_box_ball[0]*np.exp(output[ib][jb][3])
    hb=bb_scale*self.anchor_box_ball[1]*np.exp(output[ib][jb][4])

    pp1=sigmoid(output[ip1][jp1][5])
    xp1=(jp1+sigmoid(output[ip1][jp1][6]))*coord_scale
    yp1=(ip1+sigmoid(output[ip1][jp1][7]))*coord_scale
    wp1=bb_scale*self.anchor_box_post[0]*np.exp(output[ip1][jp1][8])
    hp1=bb_scale*self.anchor_box_post[1]*np.exp(output[ip1][jp1][9])

    pp2 = sigmoid(output[ip2][jp2][5])
    xp2 = (jp2 + sigmoid(output[ip2][jp2][6])) * coord_scale
    yp2 = (ip2 + sigmoid(output[ip2][jp2][7])) * coord_scale
    wp2 = bb_scale * self.anchor_box_post[0] * np.exp(output[ip2][jp2][8])
    hp2 = bb_scale * self.anchor_box_post[1] * np.exp(output[ip2][jp2][9])

    ball_detection = (pb,xb,yb,wb,hb)
```

```
    post1_detection = (pp1,xp1,yp1,wp1,hp1)
    post2_detection = (pp2,xp2,yp2,wp2,hp2)
    return ball_detection, post1_detection, post2_detection
```

As figuras 2, 3, 4, 5 e 6 mostram exemplos diversos de funcionamento da implementação. Veja que a rede neural foi capaz de identificar figuras apenas com a bola, apenas com a trave, com ambas e também em movimento, o que prova, para esses casos, a eficiência da rede.

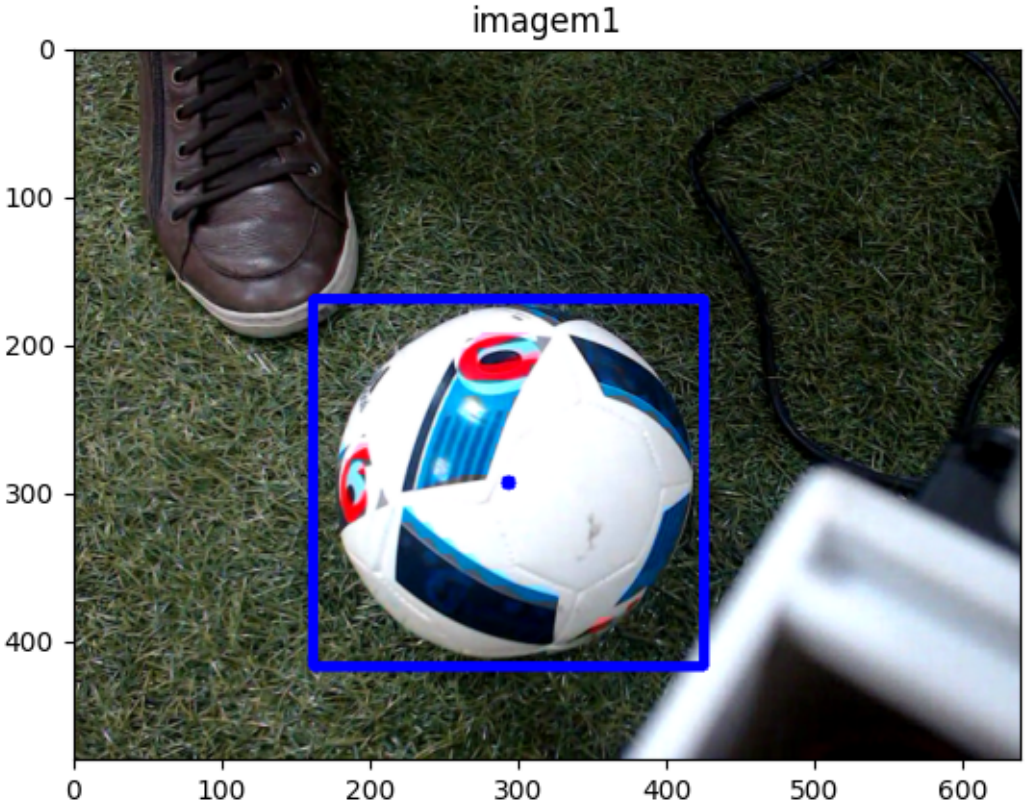

Figura 2: Detecção da bola a uma curta distância.

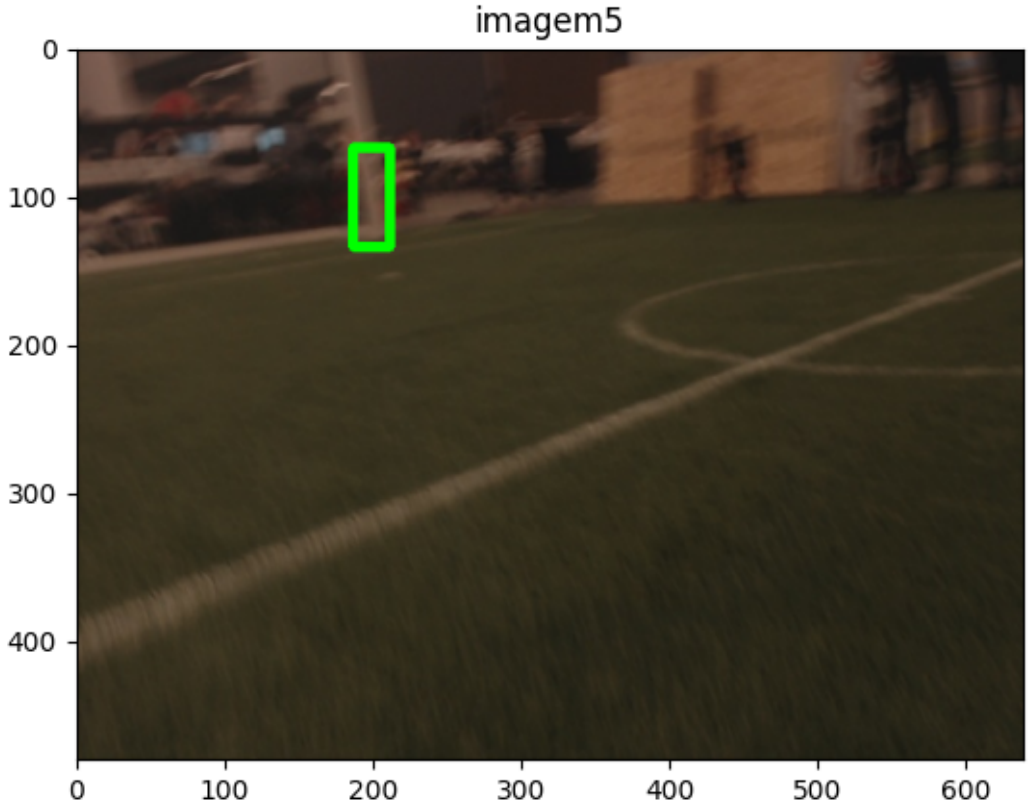

Figura 3: Detecção da trave há uma longa distância e em movimento.

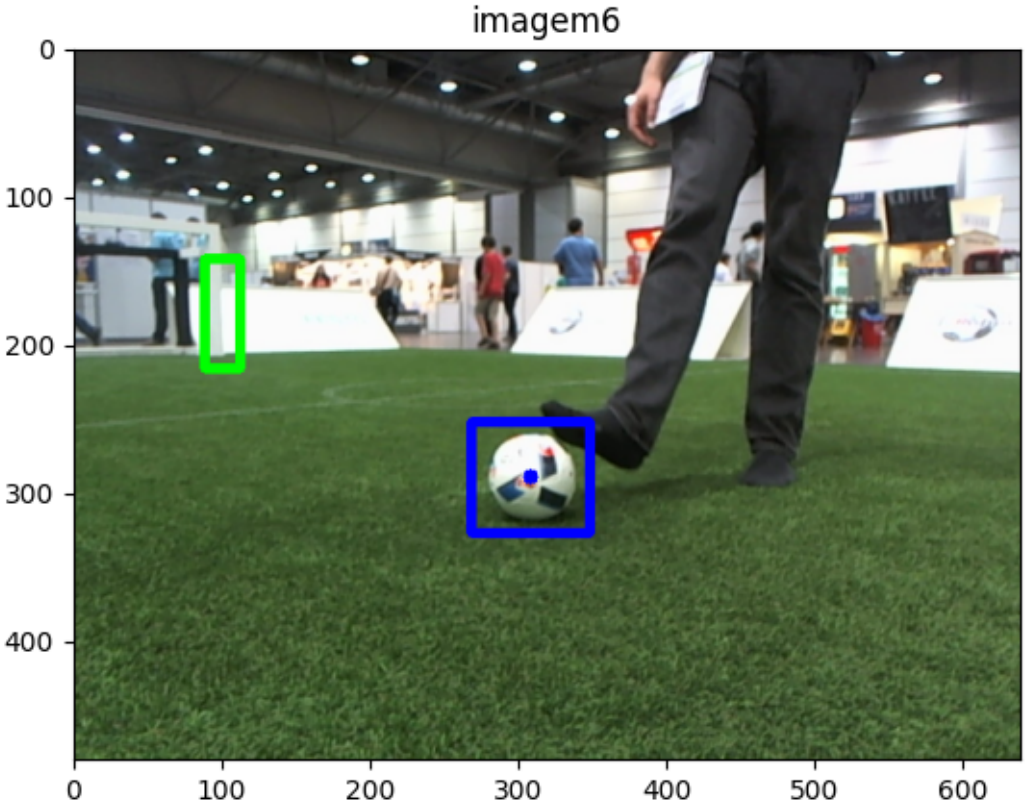

Figura 4: Detecção da bola e uma trave no fundo da imagem.

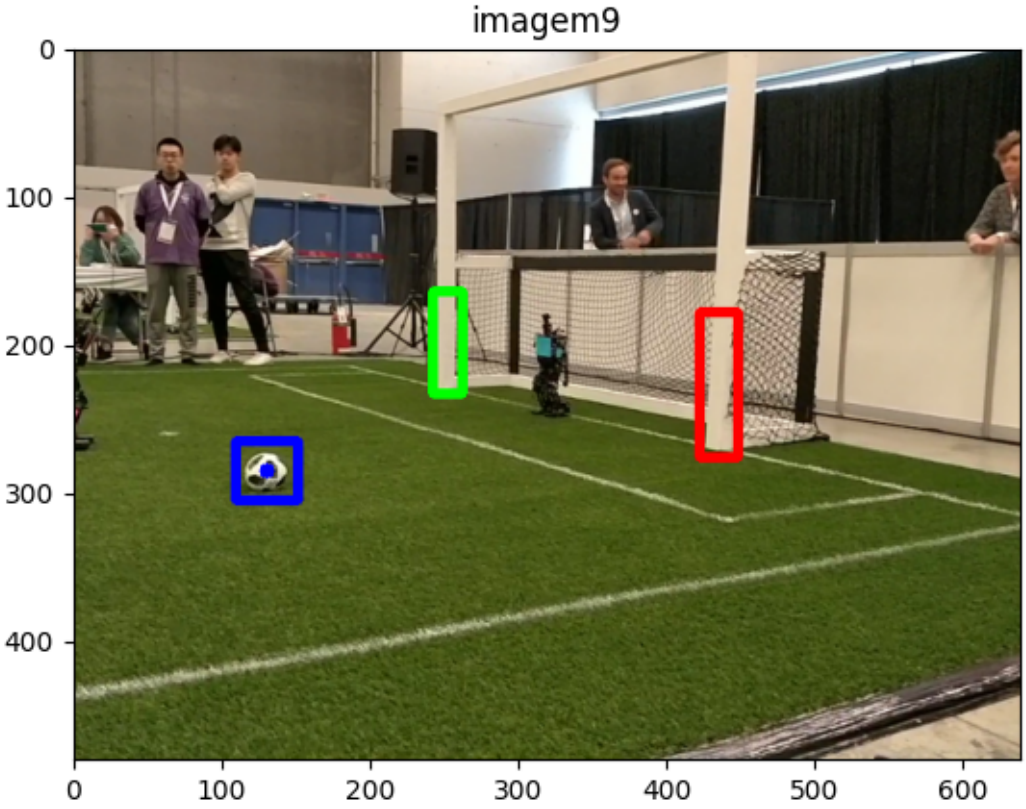

Figura 5: Detecção da bola e de ambas as traves numa mesma imagem.

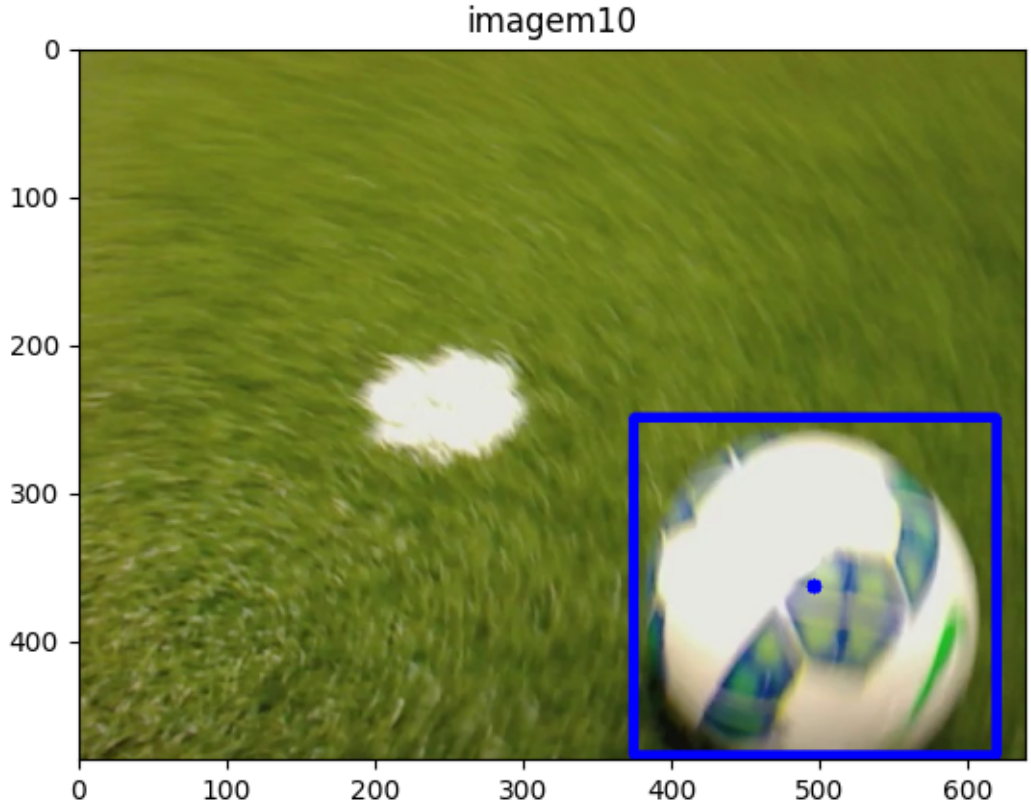

Figura 6: Detecção da bola durante o movimento.